

VAX Rdb/VMS

RDO and RMU Reference Manual Part 1

December 1990

This manual describes RDO and RMU language elements and statements. The material in this manual is not introductory.

Revision/Update Information: This manual is a revision and supersedes previous versions.

Operating System: VMS

Software Version: VAX Rdb/VMS Version 4.0

digital equipment corporation
maynard, massachusetts

The information in this document is subject to change without notice and should not be construed as a commitment by Digital Equipment Corporation.

Digital Equipment Corporation assumes no responsibility for any errors that may appear in this document.

Any software described in this document is furnished under a license and may be used or copied only in accordance with the terms of such license. No responsibility is assumed for the use or reliability of software or equipment that is not supplied by Digital Equipment Corporation or its affiliated companies.

Restricted Rights: Use, duplication, or disclosure by the U.S. Government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013.

© Digital Equipment Corporation 1989, 1990.

All rights reserved.
Printed in U.S.A.

The Reader's Comments forms at the end of this document request your critical evaluation to assist in preparing future documentation.

The following are trademarks of Digital Equipment Corporation: ALL-IN-1, CDD/Plus, DEC, DEC/CMS, DECdecision, DECdtm, DECforms, DECintact, DEC/MMS, DECnet, DECtp, DECtrace, DECwindows, MicroVAX, ULTRIX, UNIBUS, VAX, VAX ACMS, VAX Ada, VAX BASIC, VAX C, VAX CDD, VAXcluster, VAX COBOL, VAX DATATRIEVE, VAX DBMS, VAXELN, VAX FMS, VAX FORTRAN, VAX Pascal, VAX RALLY, VAX Rdb/ELN, VAX Rdb/VMS, VAX RMS, VAX SPM, VAXstation, VAX TEAMDATA, VIDA, VMS, VT, and the DIGITAL Logo.

MS-DOS is a registered trademark of Microsoft Corporation. OS/2 and IBM are trademarks of International Business Machines Corporation.

This document is available in printed and online versions.

This document was prepared using VAX DOCUMENT, Version 1.2.

Contents

Part 1

Preface	xiii
---------------	------

Technical Changes and New Features	xxi
--	-----

1 RDO Language Elements

1.1	The RDO Session	1-1
1.1.1	Beginning an RDO Session	1-1
1.1.2	Getting Information in RDO	1-2
1.1.3	Exiting from RDO	1-3
1.2	Prompts	1-3
1.3	Statements	1-3
1.3.1	Keywords	1-4
1.3.1.1	Required Keywords	1-4
1.3.1.2	Optional Keywords	1-5
1.3.2	User-Supplied Names	1-5
1.3.3	Literal Values	1-5
1.3.4	Expressions	1-6
1.3.5	Data Dictionary Path Names	1-7
1.3.6	File Specifications	1-8
1.3.7	Terminating Statements	1-9

2 Overview of Rdb/VMS Functions

2.1	Data Definition Statements	2-2
2.2	Data Manipulation Statements	2-5
2.3	Database Maintenance and Performance Statements	2-7
2.4	Statements for Interactive Control	2-8
2.5	VAX Data Distributor Statements	2-9

3 Value Expressions and Conditional Expressions

3.1	Value Expressions	3-2
3.1.1	Host Language Variables	3-4
3.1.2	Literals	3-6
3.1.2.1	Character String Literals	3-6
3.1.2.2	Numeric Literals	3-8
3.1.2.3	Date Literals	3-9
3.1.3	Statistical Expressions	3-11
3.1.3.1	The TOTAL Statistical Function	3-13
3.1.3.2	The COUNT Statistical Function	3-14
3.1.3.3	The AVERAGE Statistical Function	3-14
3.1.3.4	The MAX Statistical Function	3-15
3.1.3.5	Global Aggregates	3-16
3.1.4	Arithmetic Expressions	3-21
3.1.5	Concatenated Expressions	3-25
3.1.6	Missing Values (RDB\$MISSING Expression)	3-26
3.1.7	FIRST FROM Expression	3-31
3.1.8	Database Key	3-32
3.1.9	Segmented String Expressions	3-34
3.2	Conditional Expressions	3-36
3.2.1	Relational Operators	3-39
3.2.2	Logical Operators	3-47

4 The Record Selection Expression (RSE)

4.1	Format of the Record Selection Expression	4-2
4.2	The FIRST Clause: Restricting the Number of Records	4-3
4.3	The Relation Clause: Context Variables in Streams and Loops	4-6
4.4	The WITH Clause: Specifying Conditions for the Record Stream	4-8
4.5	The SORTED BY Clause: Sorting Records	4-9
4.6	The REDUCED TO Clause: Retaining Only Unique Field Values	4-11
4.7	The CROSS Clause: Joining Related Records	4-12
4.8	Views and the Record Selection Expression	4-15

5 Field Attributes in Relations and Views

5.1	Data Type Clause	5-5
5.2	Validity Clause	5-11
5.3	Missing Value Clause	5-13
5.4	DATATRIEVE Support Clauses	5-16
5.5	Collating Sequence Clause	5-19

6 RMU Command Syntax

6.1	Command Parameters	6-2
6.2	Qualifiers	6-2
6.3	RMU/ALTER Command	6-4
6.4	RMU/ANALYZE Command	6-5
6.5	RMU/ANALYZE/CARDINALITY Command	6-9
6.6	RMU/ANALYZE/INDEXES Command	6-13
6.7	RMU/ANALYZE/PLACEMENT Command	6-16
6.8	RMU/BACKUP Command	6-20
6.9	RMU/BACKUP/AFTER_JOURNAL Command	6-37
6.10	RMU/CLOSE Command	6-42
6.11	RMU/CONVERT Command	6-46
6.12	RMU/COPY_DATABASE Command	6-50
6.13	RMU/DUMP Command	6-54
6.14	RMU/DUMP/AFTER_JOURNAL Command	6-59
6.15	RMU/DUMP/BACKUP_FILE Command	6-62
6.16	RMU/DUMP/RECOVERY_JOURNAL Command	6-68
6.17	RMU/LOAD Command	6-70
6.18	RMU/MONITOR REOPEN_LOG Command	6-79
6.19	RMU/MONITOR START Command	6-80
6.20	RMU/MONITOR STOP Command	6-82
6.21	RMU/MOVE_AREA Command	6-85
6.22	RMU/OPEN Command	6-89
6.23	RMU/RECOVER Command	6-91
6.24	RMU/RECOVER/RESOLVE Command	6-100
6.25	RMU/RESOLVE Command	6-103
6.26	RMU/RESTORE Command	6-107
6.27	RMU/SET AUDIT Command	6-127
6.28	RMU/SHOW Command	6-142
6.28.1	RMU/SHOW AUDIT Command	6-143
6.28.2	RMU/SHOW STATISTICS Command	6-148
6.28.3	RMU/SHOW SYSTEM Command	6-154
6.28.4	RMU/SHOW USERS Command	6-155
6.28.5	RMU/SHOW VERSION Command	6-157
6.29	RMU/UNLOAD Command	6-158
6.30	RMU/VERIFY Command	6-161

7 RdbALTER Utility Command Syntax

7.1	AREA . . . PAGE Command	7-3
7.2	ATTACH Command	7-5
7.3	COMMIT Command	7-7
7.4	DEPOSIT Command	7-8
7.5	DEPOSIT FILE Command	7-14
7.6	DEPOSIT ROOT Command	7-16
7.7	DETACH Command	7-17
7.8	DISPLAY Command	7-18
7.9	DISPLAY FILE Command	7-25
7.10	DISPLAY ROOT Command	7-27
7.11	EXIT Command	7-28
7.12	HELP Command	7-29
7.13	LOG Command	7-30
7.14	MAKE_CONSISTENT Command	7-31
7.15	MOVE Command	7-33
7.16	NOLOG Command	7-35
7.17	PAGE Command	7-36
7.18	RADIX Command	7-37
7.19	ROLLBACK Command	7-38
7.20	UNCORRUPT Command	7-39
7.21	VERIFY Command	7-41

8 Rdb/VMS System Relations

8.1	RDB\$CONSTRAINTS System Relation	8-4
8.2	RDB\$CONSTRAINT_RELATIONS System Relation	8-5
8.3	RDB\$DATABASE System Relation	8-6
8.4	RDB\$FIELD_VERSIONS System Relation	8-9
8.5	RDB\$FIELDS System Relation	8-11
8.6	RDB\$INDEX_SEGMENTS System Relation	8-15
8.7	RDB\$INDICES System Relation	8-16
8.8	RDB\$RELATION_FIELDS System Relation	8-17
8.9	RDB\$RELATIONS System Relation	8-20
8.10	RDB\$VIEW_RELATIONS System Relation	8-23
8.11	RDBVMS\$COLLATIONS System Relation	8-24
8.12	RDBVMS\$INTERRELATIONS System Relation	8-25
8.13	RDBVMS\$PRIVILEGES System Relation	8-26
8.14	RDBVMS\$RELATION_CONSTRAINTS System Relation	8-27
8.15	RDBVMS\$RELATION_CONSTRAINT_FLDS System Relation	8-31
8.16	RDBVMS\$STORAGE_MAPS System Relation	8-31
8.17	RDBVMS\$STORAGE_MAP_AREAS System Relation	8-32
8.18	RDBVMS\$TRIGGERS System Relation	8-33

8.18.1	Clumplets That Can Be Used in RDBVMS\$TRIGGER_ACTIONS	8-36
--------	--	------

Part 2

9 VAX Rdb/VMS Statements

9.1	ANALYZE Statement	9-2
9.2	CHANGE DATABASE Statement	9-9
9.3	CHANGE FIELD Statement	9-27
9.4	CHANGE INDEX Statement	9-36
9.5	CHANGE PROTECTION Statement	9-43
9.6	CHANGE RELATION Statement	9-48
9.7	CHANGE STORAGE MAP Statement	9-61
9.8	COMMIT Statement	9-69
9.9	CREATE_SEGMENTED_STRING Statement	9-74
9.10	DCL Invoke (\$) Statement	9-81
9.11	DECLARE_STREAM Statement	9-82
9.12	DEFINE COLLATING_SEQUENCE Statement	9-85
9.13	DEFINE CONSTRAINT Statement	9-88
9.14	DEFINE DATABASE Statement	9-93
9.15	DEFINE FIELD Statement	9-112
9.16	DEFINE INDEX Statement	9-119
9.17	DEFINE PROTECTION Statement	9-132
9.18	DEFINE RELATION Statement	9-149
9.19	DEFINE SCHEDULE Statement (VAX Data Distributor)	9-164
9.20	DEFINE STORAGE MAP Statement	9-171
9.21	DEFINE TRANSFER Statement (VAX Data Distributor)	9-180
9.22	DEFINE TRIGGER Statement	9-204
9.23	DEFINE VIEW Statement	9-219
9.24	DELETE COLLATING_SEQUENCE Statement	9-225
9.25	DELETE CONSTRAINT Statement	9-228
9.26	DELETE DATABASE Statement	9-232
9.27	DELETE FIELD Statement	9-234
9.28	DELETE INDEX Statement	9-237
9.29	DELETE PATHNAME Statement	9-240
9.30	DELETE PROTECTION Statement	9-241
9.31	DELETE RELATION Statement	9-244
9.32	DELETE SCHEDULE Statement (VAX Data Distributor)	9-247
9.33	DELETE STORAGE MAP Statement	9-249
9.34	DELETE TRANSFER Statement (VAX Data Distributor)	9-251
9.35	DELETE TRIGGER Statement	9-253
9.36	DELETE VIEW Statement	9-255

9.37	EDIT Statement	9-257
9.38	END_SEGMENTED_STRING Statement	9-260
9.39	END_STREAM Statement	9-261
9.40	ERASE Statement	9-263
9.41	Execute (@) Statement	9-267
9.42	EXIT Statement	9-270
9.43	EXPORT Statement	9-271
9.44	FETCH Statement	9-276
9.45	FINISH Statement	9-280
9.46	FOR Statement	9-282
9.47	FOR Statement with Segmented Strings	9-287
9.48	GET Statement	9-291
9.49	HELP Statement	9-297
9.50	IMPORT Statement	9-299
9.51	INTEGRATE DATABASE Statement	9-325
9.52	INVOKE DATABASE Statement	9-329
9.53	MODIFY Statement	9-338
9.54	ON ERROR Clause	9-343
9.55	PLACE Statement	9-346
9.56	PRINT Statement	9-349
9.57	READY Statement	9-353
9.58	REINITIALIZE TRANSFER Statement (VAX Data Distributor)	9-357
9.59	ROLLBACK Statement	9-359
9.60	SET Statement	9-362
9.61	SHOW Statements	9-372
9.61.1	SHOW ALL Statement	9-375
9.61.2	SHOW COLLATING_SEQUENCE Statement	9-376
9.61.3	SHOW CONSTRAINTS Statement	9-377
9.61.4	SHOW DATABASES Statement	9-379
9.61.5	SHOW DATE_FORMAT Statement	9-381
9.61.6	SHOW DICTIONARY Statement	9-383
9.61.7	SHOW FIELDS Statement	9-384
9.61.8	SHOW INDEXES Statement	9-387
9.61.9	SHOW LANGUAGE Statement	9-391
9.61.10	SHOW PRIVILEGES Statement	9-392
9.61.11	SHOW PROTECTION Statement	9-395
9.61.12	SHOW RADIX_POINT Statement	9-397
9.61.13	SHOW RELATIONS Statement	9-398
9.61.14	SHOW STORAGE AREAS Statement	9-400
9.61.15	SHOW STORAGE MAPS Statement	9-404
9.61.16	SHOW STREAMS Statement	9-406
9.61.17	SHOW TRANSACTION Statement	9-407
9.61.18	SHOW TRANSFER Statement (VAX Data Distributor)	9-409
9.61.19	SHOW TRIGGERS Statement	9-413

9.61.20	SHOW VERSIONS Statement	9-416
9.62	START_SEGMENTED_STRING Statement	9-417
9.63	START_STREAM Statement, Declared	9-422
9.64	START_STREAM Statement, Undeclared	9-425
9.65	START_TRANSACTION Statement	9-431
9.66	START TRANSFER Statement (VAX Data Distributor)	9-453
9.67	STOP TRANSFER Statement (VAX Data Distributor)	9-456
9.68	STORE Statement	9-457
9.69	STORE Statement with Segmented Strings	9-464

A Rdb/VMS Reserved Words

B Rdb/VMS Error Message Explanation Files

C Components of Run-Time Only License

D Default, Minimum, and Maximum Values for Database Parameters

E Storage Area Parameter Default, Minimum, and Maximum Values

F RDO Statements Not Supported in Rdb/VMS Versions 3.1 and 4.0

F.1	BACKUP Statement	F-1
F.2	CLOSE Statement	F-3
F.3	CONVERT Statement	F-5
F.4	OPEN Statement	F-6
F.5	RECOVER Statement	F-7
F.6	REFRESH MONITOR LOG Statement	F-9
F.7	RESTORE Statement	F-10
F.8	SHOW MONITOR Statement	F-16
F.9	SHOW USERS Statement	F-17
F.10	SPOOL Statement	F-18
F.11	STOP MONITOR Statement	F-20

Index

Figures

1	A Sample Syntax Diagram	xviii
1-1	DEFINE CONSTRAINT—a Sample Syntax Diagram	1-4
3-1	Using a Statistical Expression to Group Records	3-17
3-2	A Statistical Expression Across Three Relations	3-20
5-1	A Record with a Segmented String Field	5-9

Tables

2-1	Rdb/VMS Data Definition Statements	2-2
2-2	Rdb/VMS Data Manipulation Statements	2-5
2-3	Rdb/VMS Database Maintenance and Performance Statements	2-8
2-4	RDO Statements for Interactive Control	2-8
2-5	VAX Data Distributor Statements	2-10
3-1	Quotation Marks in Character String Literals	3-7
3-2	Rdb/VMS Statistical Operators	3-12
3-3	RDO Relational Operators	3-39
3-4	Rdb/VMS Logical Operators—the AND Operator	3-47
3-5	Rdb/VMS Logical Operators—the OR Operator	3-47
3-6	Rdb/VMS Logical Operators—the NOT Operator	3-48
5-1	Rdb/VMS Data Types	5-7
6-1	Columns in a Database Table for Storing Security Audit Journal Records	6-71
6-2	Data Type Conversions Performed by Rdb/VMS	6-73
6-3	DACCESS Privileges for Database Objects	6-130
8-1	The RDB\$CONSTRAINTS System Relation	8-4
8-2	The RDB\$CONSTRAINT_RELATIONS System Relation	8-5
8-3	The RDB\$DATABASE System Relation	8-7
8-4	The RDB\$FIELD_VERSIONS System Relation	8-10
8-5	The RDB\$FIELDS System Relation	8-12
8-6	The RDB\$INDEX_SEGMENTS System Relation	8-16
8-7	The RDB\$INDICES System Relation	8-16
8-8	The RDB\$RELATION_FIELDS System Relation	8-18
8-9	The RDB\$RELATIONS System Relation	8-21
8-10	The RDB\$VIEW_RELATIONS System Relation	8-23
8-11	The RDBVMS\$COLLATIONS System Relation	8-24

8-12	The RDBVMS\$INTERRELATIONS System Relation	8-25
8-13	The RDBVMS\$PRIVILEGES System Relation	8-26
8-14	The RDBVMS\$RELATION_CONSTRAINTS System Relation	8-27
8-15	Values for RDBVMS\$CONSTRAINT_TYPE	8-30
8-16	The RDBVMS\$RELATION_CONSTRAINT_FLDS System Relation	8-31
8-17	The RDBVMS\$STORAGE_MAPS System Relation	8-32
8-18	The RDBVMS\$STORAGE_MAP_AREAS System Relation	8-32
8-19	The RDBVMS\$TRIGGERS System Relation	8-34
8-20	Trigger Type Values	8-36
9-1	CHANGE RELATION Options and Privileges	9-57
9-2	Defaults for DEFINE DATABASE Statement	9-96
9-3	Data Manipulation Access Rights	9-137
9-4	DDL and Administrative Statements Controlled by Database ACL	9-137
9-5	DDL and Administrative Statements Controlled by ACL for Each Relation, Local Field, or View in Statement	9-138
9-6	Task-Oriented Summary of Required Privileges	9-138
9-7	Privilege Override Capability	9-143
9-8	Record Data Updates	9-211
9-9	Logical Names for Internationalization of SET Statements	9-366
9-10	Rdb/VMS SHOW Statements	9-372
9-11	Defaults for the START_TRANSACTION Statement	9-433
9-12	VAX Rdb/VMS Share Modes	9-443
9-13	Comparison of Share Modes for Updates	9-443
C-1	RDO Statements Available in RTO License	C-1
D-1	Rdb/VMS Database-Wide Parameter Default Values and Minimum and Maximum Values	D-1
E-1	Rdb/VMS Multifile Storage Area Parameter Default Values and Minimum and Maximum Values	E-1

Part 1

Preface

VAX Rdb/VMS, often referred to as Rdb/VMS in this manual, is a general-purpose database management system based on the relational data model.

Note The version of Rdb/VMS previous to Version 4.0 is referred to throughout this manual as Version 3.1 or V3.1. The terms "Version 3.1" and "V3.1" refer to Version 3.1 of Rdb/VMS and any updates to Version 3.1; thus, for example, a reference to the "V3.1 software" refers to the Version 3.1 software and any of its updates. References to a specific update (for example, V3.1A or V3.1B) are made only where it is necessary to be precise.

Purpose of This Manual

This manual describes the syntax and semantics of all Rdb/VMS statements and language elements. It includes reference material for the Relational Database Operator (RDO) utility and the Rdb/VMS Management Utility (RMU).

Intended Audience

To get the most out of this manual, you should be familiar with data processing procedures, basic database management concepts and terminology, and the VMS operating system.

Operating System Information

Information about the versions of the operating system and related software that are compatible with this version of Rdb/VMS is included in the Rdb/VMS media kit, in the *VAX Rdb/VMS Installation Guide*.

For information on the compatibility of other software products with this version of Rdb/VMS, refer to the System Support Addendum (SSA) that comes with the Software Product Description (SPD). You can use the SPD/SSA to verify which versions of your operating system are compatible with this version of Rdb/VMS.

Structure

This manual is divided into two parts. Part 1 contains Chapters 1 through 8. Part 2 contains Chapter 9 plus the appendixes. Each part contains a complete Table of Contents and a complete Index. The Table of Contents and Index in each part encompass the material of both parts of the manual.

The following table shows the contents of the chapters and appendixes in this manual.

Chapter 1	Provides an introduction to the language elements of the Rdb/VMS Relational Database Operator (RDO) utility.
Chapter 2	Gives an overview of RDO functions, including data definition, data manipulation, and system maintenance.
Chapter 3	Describes value expressions, arithmetic expressions, conditional expressions, and statistical expressions.
Chapter 4	Describes the Rdb/VMS record selection expression (RSE).
Chapter 5	Shows how to describe the structure and optional elements of fields when you define relations.
Chapter 6	Describes the syntax and semantics of the Rdb/VMS Management Utility (RMU) commands. RMU commands, executed at DIGITAL Command Language (DCL) level, are used to monitor and display information about Rdb/VMS databases.
Chapter 7	Describes in detail the syntax and semantics of all RMU/ALTER commands available in the RdbALTER utility.
Chapter 8	Describes the Rdb/VMS system relations. The database system stores data about Rdb/VMS databases.
Chapter 9	Describes in detail the syntax and semantics of all Rdb/VMS statements. This chapter includes descriptions of Rdb/VMS data definition statements, data manipulation statements, database maintenance utility statements, interactive control commands, and VAX Data Distributor statements.

Appendix A	Lists the Rdb/VMS reserved words.
Appendix B	Describes the location of online error message explanation files for the RDB, RDMS, RDO, and DDAL facilities.
Appendix C	Contains a table that identifies the statements that are available using the run-time version of Rdb/VMS.
Appendix D	Contains a table that shows default, minimum, and maximum values for database parameters.
Appendix E	Contains a table that shows default, minimum, and maximum values for storage area parameters.
Appendix F	Provides information on RDO statements not supported in Rdb/VMS Version 3.1 and Version 4.0, and also explains the statement or command you should use in place of the unsupported statement.

Related Manuals

- *VAX Rdb/VMS Introduction and Master Index*
Introduces Rdb/VMS and explains major terms and concepts. Includes a glossary, a directory of Rdb/VMS documentation, and a master index that combines entries from all the Rdb/VMS manuals.
- *VAX Rdb/VMS Guide to Database Design and Definition*
Explains how to design a logical database and how to translate that design into a physical database using Rdb/VMS data definition statements.
- *VAX Rdb/VMS Guide to Database Maintenance and Performance*
Provides guidelines for maintaining good database performance and explains how to use the database maintenance utilities to perform backup and recovery operations, restore journals, and analyze the database.
- *VAX Rdb/VMS Guide to Database Tuning*
Introduces the concept of tuning, and explores how tuning the system, the database, and the application can affect database performance. Outlines a series of steps to follow in identifying, analyzing, isolating, and solving a performance problem, and in monitoring the resulting solution. Includes a set of decision trees that provide an organized approach to solving some common database tuning problems.
- *VAX Rdb/VMS Guide to Using RDO, RDBPRE, and RDML*
Describes how to use the features of Rdb/VMS to retrieve, store, change, and erase data. Shows how to write programs that use Rdb/VMS as a data access method; contains information on writing programs in high-level languages that are supported by Rdb/VMS preprocessors, including

Relational Data Manipulation Language (RDML); and describes Callable RDO, an interactive utility for languages without preprocessors.

- *VAX Rdb/VMS Guide to Using SQL*
Introduces the Rdb/VMS SQL (structured query language) interface, and shows how to retrieve, store, and update data interactively and through application programs.
- *VAX Rdb/VMS Guide to Using SQL/Services*
Describes how to develop application programs that use SQL/Services, a client/server software component of Rdb/VMS that allows programs from various remote computers running the Macintosh, MS-DOS, OS/2, ULTRIX, ULTRIX for RISC, or VMS operating systems, to access Rdb/VMS or VIDA databases on a VMS server system.
- *VAX Rdb/VMS Guide to Distributed Transactions*
Describes the two-phase commit protocol and distributed transactions, explains how to start and complete distributed transactions using SQL, RDBPRE, and RDML, and how to recover from unresolved transactions using RMU commands.
- *VAX Rdb/VMS SQL Reference Manual*
Provides reference material and a complete description of the statements, the interactive, dynamic, and module language interfaces, and the syntax for SQL, the structured query language interface for Rdb/VMS.
- *VAX Rdb/VMS SQL Quick Reference Guide*
Summarizes the information in the *VAX Rdb/VMS SQL Reference Manual*.
- *RDML Reference Manual*
Describes the syntax and use of the Relational Data Manipulation Language (RDML), which can be embedded in VAX C or VAX Pascal programs to access Rdb/VMS or Rdb/ELN databases.
- *VAX Rdb/VMS Installation Guide*
Describes how to install Rdb/VMS.
- *VAX Rdb/VMS Release Notes*
Describes new features, problems and problems fixed, restrictions, and other information related to the current release of Rdb/VMS. Contains information about SQL and other Rdb/VMS interfaces and utilities.

Syntax Diagrams

This manual presents the syntax of Rdb/VMS statements using syntax diagrams. Syntax diagrams graphically portray optional, required, and repeating characteristics of any Rdb/VMS statement. You can learn the syntax of a command or statement by reading that statement's syntax diagram.

To read a syntax diagram, start from the left and follow the arrows until you exit from the diagram at the right. When you come to a branch in the path, choose the branch that contains the option you want. If you want to omit an option, choose the path with no language elements. If a diagram occupies more than one horizontal line, the arrow returns to the left margin. Syntax diagrams can contain:

- Names of syntax diagrams

If a diagram is named, the name appears in lowercase type above and to the left of the diagram. Syntax diagrams can refer to each other by name. The equal sign (=) indicates that the name is equivalent to the diagram and that the diagram can be substituted wherever the name appears.

If the diagram contains the name of another diagram, substitute that other diagram where the name appears. Such a substitution is similar to putting the name of a field where "field-name" appears. Most named syntax diagrams appear as subdiagrams following the main diagram.

- Keywords

Keywords appear in uppercase type. If a keyword is underlined, you must include it in the statement. A keyword without underlining is optional, but the keyword makes the statement more readable. Omitting an optional keyword does not change the meaning of a statement.

- Punctuation marks

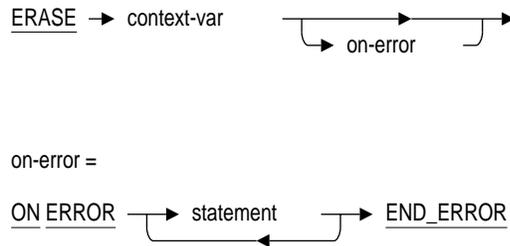
Punctuation marks are included in the diagram when required by the syntax of the command or statement.

- User-supplied elements

User-supplied elements appear in lowercase type. These elements can include names, expressions, and literals. They are usually defined in text following the diagram.

Figure 1 shows the syntax diagram for the Rdb/VMS ERASE statement.

Figure 1 A Sample Syntax Diagram



ERASE	Is in uppercase and underlined on the main line of the diagram. Therefore, you must supply the keyword (which can usually be abbreviated).
context-var	Is in lowercase on the main line of the diagram. Therefore, you must supply a substitute for context-var. In the reference section, the commentary following the diagram describes the possible values and the function for context-var.
on-error	Is in lowercase on a branch. Because it parallels an empty branch, the on-error clause is optional. The subdiagram expands the definition of on-error.
statement	Is in lowercase on a main branch. The on-error clause is optional, but if you include it, you must have ON ERROR, at least one statement, and END_ERROR.

Conventions

In examples, an implied carriage return occurs at the end of each line, unless otherwise noted. You must press the RETURN key at the end of a line of input.

Often in examples the prompts are not shown. Generally, they are shown where it is important to depict an interactive sequence exactly; otherwise, they are omitted in order to focus full attention on the statements or commands themselves.

The section explains the conventions used in this manual:

CTRL/x	This symbol in examples tells you to press the CTRL (control) key and hold it down while pressing the specified letter key.
RETURN	This symbol in examples indicates the RETURN key.

<code>TAB</code>	This symbol in examples indicates the TAB key.
.	A vertical ellipsis in an example means that information not directly related to the example has been omitted.
...	A horizontal ellipsis in an RMU command syntax diagram means you can enter additional information, such as parameters or qualifier arguments. A horizontal ellipsis in statements or commands means that parts of the statement or command not directly related to the example have been omitted.
e, f, t	Index entries in the printed manual may have a lowercase e, f, or t following the page number; the e, f, or t is a reference to the example, figure, or table, respectively, on that page.
Color	In printed manuals, color in examples shows user input.
< >	Angle brackets enclose user-supplied names.
[]	Brackets enclose optional clauses from which you can choose one or none.
\$	The dollar sign represents the DIGITAL Command Language prompt. This symbol indicates that the DCL interpreter is ready for input.

References to Products

The VAX Rdb/VMS documentation set to which this manual belongs often refers to products by their abbreviated names:

- DECdecision software is referred to as DECdecision.
- DECdtm software is referred to as DECdtm.
- DEC RdbExpert for VMS software is referred to as RdbExpert.
- DECtrace for VMS software is referred to as DECtrace.
- The SQL interface to VAX Rdb/VMS is referred to as SQL. The SQL interface is Digital Equipment Corporation's implementation of the SQL standard ANSI X3.135-1989, ISO 9075:1989, commonly referred to as ANSI/ISO.
- VAX ACMS software is referred to as ACMS.
- VAX BASIC software is referred to as BASIC.
- VAX C software is referred to as C.
- VAX CDD/Plus software is referred to as CDD/Plus or the data dictionary.
- VAX COBOL software is referred to as COBOL.
- VAX Data Distributor software is referred to as Data Distributor.

- VAX DATATRIEVE software is referred to as DATATRIEVE.
- VAX EDT software is referred to as EDT.
- VAX RALLY software is referred to as RALLY.
- VAX Rdb/VMS software is referred to as Rdb/VMS. Version 4.0 of VAX Rdb/VMS software is often referred to as V4.0.
- VAX TDMS software is referred to as TDMS.
- VAX TEAMDATA software is referred to as TEAMDATA.
- VAX TPU software is referred to as VAXTPU.
- VIDA software referred to as VIDA.

Technical Changes and New Features

This section presents a list of some of the new Rdb/VMS Version 4.0 features and technical changes that are described in this manual. See the Version 4.0 *VAX Rdb/VMS Release Notes* for more information about all the new Version 4.0 features and technical changes. Also see the Release Notes for reports of current limitations or restrictions.

The Version 4.0 major new features and technical changes described in this manual include:

- **Distributed transactions**
You can use the `START_TRANSACTION` statement to specify a distributed transaction that your transaction will be part of.
- **The ability to change Rdb/VMS default protection**
The `DEFINE PROTECTION` statement now allows you to specify a default protection that will be assigned to new relations and views created in the database. This feature allows you to specify protection that is different than the protection normally assigned to these objects under the default Rdb/VMS protection scheme.
- **Role-oriented privileges**
The `RDO ADMINISTRATOR` and `OPERATOR` privileges are role oriented in Rdb/VMS Version 4.0. Users who have these privileges are able to perform tasks for which they do not have explicit privileges. This is because users with the `ADMINISTRATOR` and `OPERATOR` privileges implicitly receive other privileges. See the `DEFINE PROTECTION` statement for more information.
- **CDD/Plus compatibility**
The `RDO SHOW FIELDS` and `SHOW DATABASES` statements have been enhanced to provide full path name displays.

- **Compressed indexes**
The DEFINE INDEX statement has been enhanced to improve the functionality of Rdb/VMS sorted and hashed indexes. The new SIZE IS n clause of the DEFINE INDEX statement allows you to provide smaller index nodes.
- **Enhancements to the EXPORT statement**
You can now use the EXPORT statement to export a database without the data. This new feature of the EXPORT statement simplifies the task of creating a duplicate copy of a database, and it allows a database administrator to experiment with storage areas and storage maps. The new NODATA option can be specified with either the EXTENSIONS or NOEXTENSIONS option.
- **Enhancements to the IMPORT statement**
The new NODATA option of the IMPORT statement enables you to import a database without the data. This new feature of the IMPORT statement works on an existing exported database.
- **Multiple storage area segmented strings**
You can now use the DEFINE STORAGE MAP statement to define multiple storage areas for segmented strings, specifying some areas as read, some as write, and some as read/write.
- **The following Rdb/VMS Management Utility (RMU) commands:**
 - **RMU/ALTER**
The RMU/ALTER command invokes the RdbALTER utility, which allows you to patch a database or move it from one device to another. For a description and syntax for each RdbALTER command, see Chapter 7.
 - **RMU/BACKUP**
The RMU/BACKUP command accepts the /OWNER, /TAPE_EXPIRATION, and /PROTECTION qualifiers. These new qualifiers make tape label handling more consistent with VMS tape label handling.
The /INCREMENTAL qualifier has been enhanced so you can specify a complete or by-area incremental backup operation.
The semantics of the /EXCLUDE and /INCLUDE qualifier have changed. You can no longer specify both /EXCLUDE=area-list and /INCLUDE=area-list in a single RMU/BACKUP command.
 - **RMU/COPY_DATABASE**
The RMU/COPY_DATABASE command can be used to create a duplicate database.

- RMU/DUMP/BACKUP_FILE
The RMU/DUMP/BACKUP_FILE command now accepts the /LABEL qualifier. The /LABEL qualifier makes tape label handling more consistent with VMS tape label handling.
- RMU/DUMP/USERS
When you use the /STATE=BLOCKED qualifier, this command generates a list of unresolved distributed transactions.
- RMU/MOVE_AREA command
The RMU/MOVE_AREA command permits you to relocate one or more areas, and optionally the root file, to different disks.
- RMU/RECOVER/RESOLVE
This command allows you to modify the state of records in the after-image journal file that are associated with unresolved distributed transactions. The RMU/RECOVER/RESOLVE command allows you to either commit or abort unresolved transactions in the after-image journal file.
- RMU/RESOLVE
This command resolves all unresolved transactions for the specified database. You can either commit or abort unresolved transactions using the RMU/RESOLVE command.
- RMU/RESTORE
The RMU/RESTORE command now accepts the /LABEL qualifier. The /LABEL qualifier makes tape label handling more consistent with VMS tape label handling.
- RMU/SET AUDIT
The RMU/SET AUDIT command enables Rdb/VMS security auditing to send security alarm messages and to make entries in the database security audit journal whenever specified audit events are detected by Rdb/VMS.
- RMU/SHOW AUDIT
The RMU/SHOW AUDIT command displays the set of security auditing characteristics that have been established with the RMU/SET AUDIT command.

- Additions and modifications to the following RMU commands:
 - RMU/CLOSE—accepts the /PATH qualifier.
 - RMU/DUMP/USERS—accepts the /STATE=BLOCKED qualifier.
 - RMU/LOAD—accepts the /AUDIT qualifier
 - RMU/OPEN—accepts the /PATH qualifier.

RDO Language Elements

The Relational Database Operator (RDO) is the interactive utility that you can use to define and access an Rdb/VMS database. When you run RDO and type statements at the RDO> prompt, Rdb/VMS executes the statements immediately. This chapter shows you how to get started using RDO. It also gives a brief introduction to the elements of the RDO language. RDO statements allow you to define and manipulate Rdb/VMS databases and database entities interactively. Use RDO to:

- Define a database and its components
- Administer and maintain the database
- Learn how to use Rdb/VMS data manipulation statements
- Test and debug your database program logic

1.1 The RDO Session

This section tells you how to run RDO. The *VAX Rdb/VMS Guide to Using RDO, RDBPRE, and RDML* includes a more complete sample terminal session that illustrates all the RDO features.

1.1.1 Beginning an RDO Session

Invoke RDO by typing:

```
$ RUN SYS$SYSTEM:RDO
```

RDO responds with the following prompt:

```
RDO>
```

You may want to define a DIGITAL Command Language (DCL) symbol to save typing. You can place this command in your login command file to define it whenever you log in.

```
$ RDO ::= $RDO
```

After you make this symbol assignment, you can run RDO by typing:

```
$ RDO
```

When you see the RDO> prompt, you can begin typing RDO statements. RDO interprets and executes the statements as you type them.

After running RDO, the first step is usually to invoke a database:

```
RDO> INVOKE DATABASE FILENAME 'DISK2:[DEPT3]PERSONNEL'
```

You can also use an initialization command file with RDO. When RDO runs, it looks for a file called RDOINI.RDO. If this file exists, RDO executes the commands in the file first, before displaying the prompt and accepting your input. If you have defined the logical name RDOINI to point to a general initialization file, RDO uses this file. Otherwise, it looks for RDOINI.RDO in the current default directory.

For example, perhaps you always use the PERSONNEL database:

```
$ DEFINE RDOINI DISK2:[FORESTER]RDOINI.RDO
$
$ TYPE DISK2:[FORESTER]RDOINI.RDO
INVOKE DATABASE FILENAME 'DOCD$:[RDMSDOC.TEST]PERSONNEL'
PRINT " "
PRINT "Rdb/VMS is ready to do your bidding, using a"
SHOW DATABASES
PRINT " "
$
$ RDO

Rdb/VMS is ready to do your bidding, using a
Database with filename DOCD$:[RDMSDOC.TEST]PERSONNEL
RDO>
```

1.1.2 Getting Information in RDO

You can type HELP to get help text explaining any Rdb/VMS statement or concept while using RDO. See Chapter 9 for a discussion of HELP, SHOW, and SET statements that provide information, display RDO parameters, and modify settings in your RDO sessions.

1.1.3 Exiting from RDO

You end an RDO session by entering one of the following:

- EXIT
- CTRL/Z

Entering either of these ends a session and normally returns you to the VMS DCL-level prompt. For example:

```
RDO> EXIT
$
```

If you have changed the database without finishing the transaction, RDO does not automatically exit. Rather, it allows you to commit the changes before exiting.

1.2 Prompts

Prompts help you keep track of your status during your interactive RDO session. The RDO prompts are:

- RDO> RDO command-level prompt. This prompt shows that you can enter any RDO statement.
- cont> The statement continuation prompt. This prompt indicates that you have not yet entered a complete statement.

1.3 Statements

When you type a statement at the RDO command level (indicated by the RDO> prompt), Rdb/VMS executes the statement immediately and displays the results on the current output device. RDO statements consist of:

- Keywords
- User-supplied names
- Literal values
- Expressions
- Data dictionary path names
- File specifications
- The statement terminator (RETURN) and the continuation character (hyphen)

The sections that follow describe the components briefly. Later chapters describe them in detail.

1.3.1 Keywords

Keywords can be used only as shown in the statement syntax diagrams. For many keywords, Rdb/VMS will accept a unique abbreviation. All keywords are shown in uppercase letters. You cannot use some keywords as user-supplied names. Appendix A lists the Rdb/VMS keywords that you cannot use as user-supplied names. These keywords are also called *reserved words*.

The two types of keywords are:

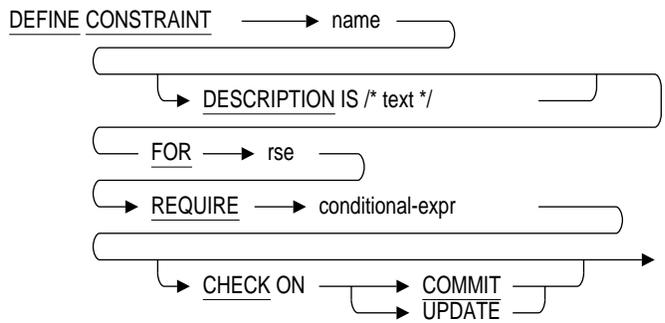
- Required
- Optional

Note *Some statements contain underscores. Rdb/VMS considers hyphens and underscores to be equivalent. That is, the results are the same, whether you type READ_ONLY or READ-ONLY.*

Rdb/VMS does not distinguish between uppercase and lowercase letters in statements. That is, READ_ONLY, read_only, and READ_only are equivalent.

1.3.1.1 Required Keywords Required keywords are words you must use in statements. In syntax diagrams, these keywords are underlined. Figure 1-1 shows an example of a syntax diagram.

Figure 1-1 DEFINE CONSTRAINT—a Sample Syntax Diagram



In this syntax diagram, `DEFINE`, `CONSTRAINT`, `DESCRIPTION`, `FOR`, `REQUIRE`, `CHECK`, `COMMIT`, and `UPDATE` are required keywords. If you choose to use the `CHECK` clause, you must type the word `CHECK` and either `COMMIT` or `UPDATE`. However, because there is an empty path parallel to the `CHECK` clause, you do not have to use the word `CHECK` or either of the qualifiers. Likewise, there is an empty path parallel to the `DESCRIPTION` clause, showing that it is an optional clause. In the reference section, the text following the diagram explains defaults.

Note All punctuation marks shown in the diagrams are required.

1.3.1.2 Optional Keywords Optional keywords are words that do not change the meaning of a statement. These words make the statements similar to natural language and, therefore, easy to remember. In syntax diagrams, optional keywords are uppercase but not underlined. Figure 1–1 contains the optional keyword ON. If you use the CHECK clause, you must type CHECK, but you need not type ON.

1.3.2 User-Supplied Names

You must supply names to satisfy the syntax of most RDO statements. In statement syntax diagrams, user-supplied names are shown in lowercase letters. Examples of user-supplied names are:

- name
- relation-name
- index-name

User-supplied names must:

- Be no more than 31 characters long
- Contain only uppercase or lowercase letters (A, a), digits, dollar signs (\$), ampersands (&), hyphens (-), and underscores (_)
- End with a letter or digit

Note Rdb/VMS considers hyphens and underscores to be equivalent. That is, the results are the same whether you use EMPLOYEE-ID or EMPLOYEE_ID as a user-supplied name.

Rdb/VMS also considers uppercase and lowercase letters in names to be the same. That is, EMPLOYEE_ID, employee_id, and EMPLOYEE_id are equivalent.

1.3.3 Literal Values

Many RDO clauses require literal values, which can be either numeric or nonnumeric.

A numeric literal value can be:

- An integer (signed or unsigned). For example:
-325 or 70953
- A fixed-point number. For example:
638.41

- A floating-point number. For example:

76E+03

A nonnumeric literal is a character string enclosed by matched pairs of quotation marks, either single (') or double ("). For example:

Valid:

```
FOR E IN TYPES WITH
  E.LABEL = "ALPHA" OR
  E.LABEL = 'ZETA'
```

Invalid:

```
FOR E IN TYPES WITH E.LABEL = "GAMMA'
```

You can include the quotation mark as part of the literal by using two consecutive quotation marks. For example:

This string:	Is displayed as:
' "A LITERAL WITH QUOTES" '	"A LITERAL WITH QUOTES"
" 'ANOTHER ONE' "	'ANOTHER ONE'
"RICHARD " "RICK" " SMITH"	RICHARD "RICK" SMITH

1.3.4 Expressions

Many Rdb/VMS statements require you to specify an expression. The three types of expressions are:

- Value expressions

A **value expression** is a symbol or string of symbols used to calculate a value. When you use a value expression in a statement, Rdb/VMS calculates the value associated with the expression and uses that value when executing the statement. See Chapter 3 for the complete description of value expressions.
- Conditional expressions

A **conditional expression** expresses the relationship between two value expressions. The value of a conditional expression is either true, false, or missing. See Chapter 3 for more information about conditional expressions.
- Record selection expressions (RSE)

A **record selection expression** is a phrase that specifies a group of records called a record stream. The RSE can include:

 - The relation or view from which the records come

- The conditions that records must meet before they are included in the stream
- The sorting order

See Chapter 4 for the complete description of the record selection expression.

1.3.5 Data Dictionary Path Names

Rdb/VMS allows you to perform database operations with or without using VAX CDD/Plus data dictionary software. Because an Rdb/VMS database stores both database definitions and the actual data values in the database file itself, you can use Rdb/VMS without storing data definitions in the dictionary.

Rdb/VMS makes it possible for you to enforce use of the data dictionary whenever data definitions are updated. Digital Equipment Corporation recommends that you store all database definitions in the dictionary as well as in the database file for the following reasons:

- **Compatibility with other Digital information management products**

If you use an Rdb/VMS database with other information management products such as DATATRIEVE, ACMS, or VAX DBMS, you are already storing Rdb/VMS database definitions in the dictionary. When you define an Rdb/VMS database and the dictionary is installed on your system, the DEFINE DATABASE statement automatically stores the definition of the database in the dictionary. Any other database entity definitions such as fields, relations, indexes, constraints, and views will be stored in the dictionary only if you invoke the database using the dictionary path name where the database resides.

If you define database entities after invoking the database with the file name, data definitions will be stored in the database file, but not in the dictionary. When the database definitions and the dictionary copy of those definitions are not identical, applications using other Digital products and attempting to access the database through the dictionary may fail.

You can regain consistency between the database file and the dictionary by using the RDO INTEGRATE DATABASE statement. You can use the INTEGRATE DATABASE statement to copy definitions stored in the database to the data dictionary. You can also use the INTEGRATE DATABASE statement to copy definitions from the data dictionary to the database file.

Once the dictionary is consistent with the database file definitions, you should plan to invoke the database using the dictionary path name when you change, delete, or add database definitions. You can enforce use of the data dictionary for all data definition changes by specifying that no definition changes may be made when the database is invoked by file name. Storing all database definitions in the dictionary ensures consistency and

compatibility with other Digital information management products that use VAX CDD/Plus in your applications.

- Compatibility with future data dictionary requirements for Digital information management software

Digital supports the data dictionary as a central source for all data definitions in its information management software products. Therefore, Digital recommends procedures that routinely and consistently place all data definitions in the dictionary. By storing all Rdb/VMS data definitions in the dictionary, you ensure that your applications will remain compatible with future versions of Rdb/VMS and Digital dictionary products.

You can identify a particular database in an RDO statement with a data dictionary path name. A dictionary path name can be:

- A full DMU path name such as CDD\$TOP.JONES.RDB.PERSONNEL
- A full CDO path name such as DISK1:[DICTIONARY]CORP.PERSONNEL

You can use the Common Dictionary Operator (CDO) naming convention to access dictionary objects that were created with the CDO utility, or dictionary objects that were converted from the Dictionary Management Utility (DMU) format to the CDO format.

- A relative path name

A relative path name consists of the portion of the full path name that follows the current default dictionary directory. For example, assume that you have used the SET DICTIONARY statement to set the current dictionary directory to CDD\$TOP.JONES.RDB. Then you can use the relative path name PERSONNEL in place of the full path name CDD\$TOP.JONES.RDB.PERSONNEL. By default, Rdb/VMS sets the current dictionary directory to the directory defined by the CDD\$DEFAULT logical name. See Chapter 9 for the description of the SET DICTIONARY statement.

- A logical name for a full or relative path name

See the *VAX CDD/Plus Common Dictionary Operator Reference Manual* for a complete description of data dictionary path names and directories.

1.3.6 File Specifications

You can also identify a database with a VMS file specification. A full file specification includes:

- Network node name
- Device name
- Directory name or list

- File name
- File type
- File version number

The following example illustrates a full file specification:

```
NODE3::DISK$DEPT3:[LICENSES]APPLICANTS.RDB;18
```

Note A file specification must be 255 characters or less.

The system supplies default values for omitted fields in the file specification if it can. See the VMS documentation or HELP for a complete description of file specifications.

In most cases, a full file specification is not necessary, and, in some cases, it may cause problems. If you are invoking a database using its file name, do not include the version number or the file type. If you do, there may be a mismatch between the version numbers of the database file and the snapshot file.

On the other hand, when you are applying an after-image journal file to a database, you should use the full file specification to make sure you are using the correct journal.

It is a good idea to enclose the file name or data dictionary path name in either single or double quotation marks, although this is not required in interactive RDO. The preprocessors and Callable RDO require quoted file names and path names.

If you are using a remote database, that is, a database on another node in a DECnet-VAX network, you must add the node specification to the file specification. Note that the Rdb/VMS installation procedure creates a special account for remote database access called RDB\$REMOTE. See the *VAX Rdb/VMS Installation Guide* for details.

1.3.7 Terminating Statements

If you type a complete statement on one line and press the RETURN key, your carriage return terminates the statement, and RDO executes it. However, RDO statements may span several lines. If you press the RETURN key before typing a complete statement, RDO prompts you for the rest of the statement with the cont> prompt. RDO continues to parse your statement until it encounters a statement terminator, such as END_FOR. For example:

```
RDO> FOR E IN EMPLOYEES
cont> WITH E.EMPLOYEE_ID = "00345"
cont> PRINT E.EMPLOYEE_ID, E.LAST_NAME END_FOR
```

Some Rdb/VMS statements do not have explicit terminators. These statements are terminated by a carriage return. When you use an optional clause on the end of such a statement, it is possible for Rdb/VMS to treat a partial statement as though it were complete. An example is the `START_TRANSACTION` statement. If you include the following statement in a command file or an editing buffer, RDO will treat the first line as a complete statement and start a transaction. Then RDO will process the second line, which is not a valid statement, and return an error message.

```
START_TRANSACTION READ_WRITE  
RESERVING EMPLOYEES FOR PROTECTED WRITE
```

There are two ways to handle this problem:

- Make sure that each line of the statement is incomplete, so that RDO will wait for the next line before processing:

```
START_TRANSACTION READ_WRITE RESERVING  
EMPLOYEES FOR PROTECTED WRITE
```

- Use the continuation character (hyphen):

```
START_TRANSACTION READ_WRITE -  
RESERVING EMPLOYEES FOR PROTECTED WRITE
```

See the *VAX Rdb/VMS Guide to Using RDO, RDBPRE, and RDML* for more complete, tutorial information on using RDO.

Overview of Rdb/VMS Functions

VAX Rdb/VMS provides several methods of retrieving and updating information in a database. You can use:

- RDO, the interactive Rdb/VMS utility
RDO provides an interactive interface to Rdb/VMS. Using RDO, you can learn how the Rdb/VMS statements work, test your data manipulation statements, perform data definition, and submit simple queries. VAX Data Distributor users can enter statements that control the distribution of Rdb/VMS databases throughout VAX nodes in a DECnet-VAX network.
- High-level language preprocessed programs
With minor adjustments, the statements you develop using RDO can be included in programs. Several preprocessors are available; together, they provide support for VAX C, VAX BASIC, VAX COBOL, VAX FORTRAN, and VAX Pascal programs that contain embedded data manipulation language (DML) statements. See the *VAX Rdb/VMS Guide to Using RDO*, *RDBPRE*, and *RDML* and the *RDML Reference Manual* for details on the preprocessors.
- Callable RDO, the high-level call interface
The Rdb/VMS language can be passed to the interpretive interface, Callable RDO, using simple calls from any language that adheres to the VAX Calling Standard.
- SQL, an industry-standard interface
SQL is included with the Rdb/VMS kit. See the *VAX Rdb/VMS Guide to Using SQL* and the *VAX Rdb/VMS SQL Reference Manual* for more information about SQL.

In addition, you can use VAX DATATRIEVE, VAX RALLY, VAX TEAMDATA, and DECdecision to access an Rdb/VMS database for queries, reports, and graphic output.

This chapter introduces you to the functional aspects of Rdb/VMS. The tables describe the following types of Rdb/VMS statements:

- Data definition statements
- Data manipulation statements
- Database maintenance utility statements
- Statements for controlling the interactive terminal session

In addition, Table 2-5 summarizes the VAX Data Distributor statements that are executed in RDO and control the distribution of Rdb/VMS databases throughout nodes in a DECnet-VAX network.

2.1 Data Definition Statements

Rdb/VMS data definition statements let you determine the structure of the database and its components. You can also use the data definition statements to change and delete the definitions. Table 2-1 lists the statements you use to define data. For a complete description of each statement, see Chapter 9. For a step-by-step guide to using the data definition statements, see the *VAX Rdb/VMS Guide to Database Design and Definition*.

Table 2-1 Rdb/VMS Data Definition Statements

Statement	Description
CHANGE DATABASE	Changes the definition for an existing database. You use this statement to specify an after-image journal (AIJ) file for the database or to change a previous AIJ file specification. You can also adjust other database and storage area parameters with the CHANGE DATABASE statement.
CHANGE FIELD	Changes a field definition. You can change a named field definition, whether the field was defined with a DEFINE FIELD statement or within a DEFINE RELATION statement.

(continued on next page)

Table 2-1 (Cont.) Rdb/VMS Data Definition Statements

Statement	Description
CHANGE INDEX	Changes the description text for the index and the storage map associated with the index. Changes the physical structure of a sorted index definition.
CHANGE PROTECTION	Changes the protection (access rights) for a user identifier within an access control list.
CHANGE RELATION	Changes a relation definition. You can delete, modify, and insert fields within relation definitions.
CHANGE STORAGE MAP	Changes a storage map definition. You can change how records will be partitioned across storage areas and whether data will be compressed or uncompressed.
DEFINE COLLATING_SEQUENCE	Allows you to specify a collating sequence that has been defined using the VMS National Character Set (NCS) utility, including both the predefined and user-defined NCS sequences.
DEFINE CONSTRAINT	Creates a constraint definition that defines the set of conditions that restrict the values permitted in a relation.
DEFINE DATABASE	Creates a database. When you use this statement, RDO creates: <ul style="list-style-type: none"> ■ A database (RDB) file, in which Rdb/VMS stores the database root file information ■ One or more storage area (RDA) files, in which Rdb/VMS stores data, if you create a multifile database ■ One or more snapshot (SNP) files that provide temporary storage for read-only data retrieval ■ An entry for the database in the data dictionary, if VAX CDD/Plus is installed on your system ■ Default protection for the database

(continued on next page)

Table 2-1 (Cont.) Rdb/VMS Data Definition Statements

Statement	Description
DEFINE FIELD	Creates a field definition. A field definition created with the DEFINE FIELD statement can be used in a relation definition.
DEFINE INDEX	Defines a field or set of fields in a relation definition as a single or multisegment index key for a relation. You can define a sorted index or a hashed index. You can also create a storage map for the index.
DEFINE PROTECTION	Creates an access control list entry for a database or relation. An access control list entry contains a user identifier and a list of access rights granted to that user.
DEFINE RELATION	Creates a relation definition. This definition includes all of the relation's field definitions. Rdb/VMS also creates a default access control list for the newly created relation.
DEFINE STORAGE MAP	Creates a storage map definition for a relation. The storage map associates a relation with a particular storage area or areas. You can specify how records will be partitioned across storage areas.
DEFINE TRIGGER	Creates a trigger for a relation. A trigger is a mechanism that causes one or more specified actions to occur whenever a particular type of update operation is performed on a relation.
DEFINE VIEW	Creates a view definition. Rdb/VMS also creates a default access control list for the newly created view. You cannot change a view definition directly. You must first delete the existing view definition and then create a new definition.
DELETE COLLATING_SEQUENCE	Deletes the specified collating sequence.
DELETE CONSTRAINT	Deletes one or more constraint definitions.
DELETE DATABASE	Deletes a database.
DELETE FIELD	Deletes one or more field definitions, if the field definitions are not currently in use by a relation or view.
DELETE INDEX	Deletes one or more index definitions.

(continued on next page)

Table 2-1 (Cont.) Rdb/VMS Data Definition Statements

Statement	Description
DELETE PATHNAME	Deletes the data dictionary path name where the current database definitions reside.
DELETE PROTECTION	Deletes the protection (access rights) from one entry within an access control list for a database or relation.
DELETE RELATION	Deletes one or more relation definitions.
DELETE STORAGE MAP	Deletes a storage map definition.
DELETE TRIGGER	Deletes one or more trigger definitions.
DELETE VIEW	Deletes one or more view definitions.

2.2 Data Manipulation Statements

The following Rdb/VMS data manipulation statements can be typed at the terminal in an RDO session or included in high-level language programs. Note, however, that the GET statement can only be used in programs, while the PRINT statement can only be used in RDO. Table 2-2 introduces the data manipulation statements.

Table 2-2 Rdb/VMS Data Manipulation Statements

Statement	Description
AT END	Specifies the statements Rdb/VMS performs in a host language program when a FETCH statement reaches the end of a record stream. Used only in programs.
COMMIT	Ends a transaction, makes permanent all changes to the database made during that transaction, and empties all buffers.
CREATE_SEGMENTED_STRING	Initializes a segmented string, allowing you to store segments.
DATABASE	Provides an alternative form of the INVOKE statement. (The INVOKE keyword is optional.) Declares a database, either to RDO or in a host language program.

(continued on next page)

Table 2-2 (Cont.) Rdb/VMS Data Manipulation Statements

Statement	Description
DECLARE_STREAM	Declares a record stream, and associates a stream name with its record selection expression.
END_SEGMENTED_STRING	Marks the end of a block that starts with a CREATE_SEGMENTED_STRING or a START_SEGMENTED_STRING statement.
END_STREAM	Closes the specified record stream but does not affect any other streams that you have open.
ERASE	Erases one or more records from an existing relation.
FETCH	Advances the pointer for a record stream to the next record. The FETCH statement does not retrieve a record. The FETCH statement is used with a record stream formed by the START_STREAM statement.
FINISH	Detaches your process from a database.
FOR	Executes a statement, or group of statements, once for each record in a record stream formed by a record selection expression. There is also a form of the FOR statement that applies to segments in a segmented string field.
GET	Retrieves field values from an Rdb/VMS record stream and assigns them to variables in a program. Used only in RDBPRE programs.
INVOKE	Declares a database, either to RDO or in a host language program.
MODIFY	Modifies one record in an existing relation.
ON ERROR	Specifies the statements Rdb/VMS performs if an error occurs during the execution of a data manipulation statement. Supported only for preprocessed programs.
PLACE	Returns the database key of a specified record. Can only be used in RDO or Callable RDO.

(continued on next page)

Table 2–2 (Cont.) Rdb/VMS Data Manipulation Statements

Statement	Description
PRINT	Retrieves field values from an Rdb/VMS record stream and assigns them to the current output device. Used only in RDO.
READY	In programs, explicitly declares your intention to access one or more databases and causes an attach to the database. Not available in RDO.
ROLLBACK	Terminates a transaction and undoes any changes made during the transaction.
START_SEGMENTED_STRING	Starts a stream of segmented string segments within a record stream.
START_STREAM	Declares and opens a record stream. When you use the START_STREAM statement, your program controls the loop that processes the records. You must use the FETCH statement to point to each succeeding record in the stream. There are two forms of this statement: one for declared streams, and one for undeclared streams.
START_TRANSACTION	Initiates a transaction, which is a series of data manipulation or data definition statements or both that Rdb/VMS executes as a unit. The START_TRANSACTION statement determines several characteristics of the transaction; for instance, the kind of lock you will place on the data you are going to manipulate.
STORE	Stores one record in an existing relation.

2.3 Database Maintenance and Performance Statements

The following statements let you perform system administration tasks, such as storage analysis, recovery, and journaling. Table 2–3 introduces the Rdb/VMS maintenance utilities.

Table 2-3 Rdb/VMS Database Maintenance and Performance Statements

Statement	Description
EXPORT	Makes a copy of a database in an intermediate form. You can then use the IMPORT statement to build the Rdb/VMS database.
IMPORT	Makes a new version of a database by copying the database contained in the exported copy. You can also supply new values for database parameters with the IMPORT statement.
INTEGRATE	Creates the data dictionary definitions from the database definitions stored in an existing database, or creates the database definitions from data dictionary definitions.

2.4 Statements for Interactive Control

Table 2-4 introduces the statements that let you control the characteristics of your terminal session, get information about Rdb/VMS and your database, and compose RDO statements easily.

Table 2-4 RDO Statements for Interactive Control

Statement	Description
EDIT	Invokes VAX EDT or VAXTPU and uses the current default initialization file. Rdb/VMS places previous RDO statements in the editing buffer for you to edit. When you make changes to the statements and exit from the editor, RDO automatically executes the statements.
HELP	Displays information about RDO topics. You can specify a topic or choose from the HELP menu.

(continued on next page)

Table 2-4 (Cont.) RDO Statements for Interactive Control

Statement	Description
SET	<p>Changes the characteristics of your RDO environment. Optional qualifiers allow you to:</p> <ul style="list-style-type: none"> ■ Control execution of data manipulation statements (SET COMMAND) ■ Change the display format for date and time values (SET DATE_FORMAT) ■ Change your default dictionary directory (SET DICTIONARY) ■ Change the language to be used for date and time input and display (SET LANGUAGE) ■ Control the line length for RDO output (SET LINE_LENGTH) ■ Route output to a terminal or disk file (SET OUTPUT) ■ Change the character representing the radix point in output displays (SET RADIX_POINT) ■ Control display of command file contents (SET VERIFY) ■ Control the number of statements that RDO places in the editing buffer (EDIT *)
SHOW	<p>Displays information about a database and database elements, such as:</p> <ul style="list-style-type: none"> ■ The definitions of fields, relations, constraints, indexes, views, user access rights, storage maps, storage areas, and triggers ■ The currently active databases and streams ■ The current default directory in the data dictionary

2.5 VAX Data Distributor Statements

Data Distributor users can execute the following statements in the interactive RDO environment and control the distribution of Rdb/VMS databases throughout the nodes in their DECnet-VAX network. Table 2-5 summarizes the Data Distributor statements that are detailed in two documents:

- *The VAX Data Distributor Handbook*
- Chapter 9 of this manual

Table 2-5 VAX Data Distributor Statements

Statement	Description
DEFINE SCHEDULE	Enters a schedule definition associated with a particular transfer. Options let you specify when the transfer starts, how often it is executed thereafter, and, in the event a transfer fails, how many times to retry.
DEFINE TRANSFER	Enters a transfer definition that causes Data Distributor to move selected data from a source database to a target database. Transfer types are extraction, extraction rollup, or replication. You can also specify target database parameters.
DELETE SCHEDULE	Deletes a schedule associated with a particular transfer.
DELETE TRANSFER	Deletes a transfer definition stored in the transfer database.
REINITIALIZE TRANSFER	Causes Data Distributor to create a replication database in a specified target directory again. This statement has the same effect as an initial execution of a replication transfer.
SHOW TRANSFER	Displays information about the transfer, its schedule, and its state.
START TRANSFER	Signals to Data Distributor that the named transfer can be executed either according to an existing schedule or on demand.
STOP TRANSFER	Stops transfer execution or prevents a transfer from beginning execution.

3

Value Expressions and Conditional Expressions

This chapter describes the value expressions and conditional expressions you can use with Rdb/VMS.

A **value expression** is a symbol or string of symbols used to calculate a value. When you use a value expression in a statement, Rdb/VMS calculates the value associated with the expression and uses that value when executing the statement. Section 3.1 describes Rdb/VMS value expressions.

A **conditional expression**, sometimes called a Boolean expression, expresses the relationship between two value expressions. The value of a conditional expression is either true, false, or null. Section 3.2 describes conditional expressions.

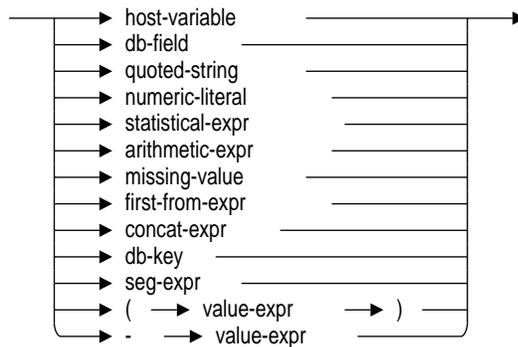
Value Expressions

3.1 Value Expressions

The following format statement lists the types of value expressions available in Rdb/VMS. The subsequent sections describe these types.

Format

value-expr =



Arguments

host-variable

A host or substitution variable, used to communicate with a calling (host) program. A host program declares a host language variable and then assigns database values to it. Host language variables are not used in RDO. For more information, see Section 3.1.1 and the *VAX Rdb/VMS Guide to Using RDO, RDBPRE, and RDML*.

db-field

The name of a field in the database, consisting of a context variable and field name.

Value Expressions

db-field =

→ context-var . field-name →

context-var

A temporary name that you associate with a relation. You define a context variable in a record selection expression. See Chapter 4 for more information on context variables and record selection expressions.

field-name

The name of a field in a relation. For example, once you have defined E as the context variable for the EMPLOYEES relation, E.LAST_NAME is a value expression that refers to a value from the LAST_NAME field of the EMPLOYEES relation.

quoted-string

A character string enclosed in quotation marks. See Section 3.1.2.

numeric-literal

A numeric literal. See Section 3.1.2.

statistical-expr

A statistical expression. Statistical expressions use functions to calculate values based on a value expression for every record in a record stream. The Rdb/VMS statistical functions are AVERAGE, MAX, MIN, TOTAL, and COUNT. See Section 3.1.3.

arithmetic-expr

An arithmetic expression. Arithmetic expressions include other value expressions and arithmetic operators. See Section 3.1.4.

missing-value

An expression that Rdb/VMS evaluates to determine the missing value for a field. See Section 3.1.6.

first-from-expr

An expression that returns the first value from the record stream formed by a record selection expression. See Section 3.1.7.

Value Expressions

concat-expr

An expression that combines two other value expressions by joining the second to the end of the first. See Section 3.1.5.

db-key

An expression that returns a pointer to a specific record. See Section 3.1.8.

seg-expr

A segmented string expression that allows you to refer to the segments of a segmented string. See Section 3.1.9.

You can enclose any value expression in parentheses to perform arithmetic operations on it, or to group it with other value expressions. You can precede any value expression with the unary minus sign (-).

3.1.1 Host Language Variables

A host or substitution variable is a program variable used to communicate with a calling (host language) program. After your program has declared a host language variable, you can use the GET statement to assign database values to the variable.

You can also use host language variables as names to represent databases and database elements. These names are called **handles**. The three types of handles are:

- Database handles

A **database handle** is a variable name you use to refer to a database. You can use a database handle to distinguish between two different active databases or to distinguish between two different attaches to the *same* database. You must use a database handle when you invoke more than one database. A database handle is a variable name declared as a longword in a host language program.

If you have more than one database active at a time, you must use the database handle in these statements and clauses to distinguish among the active databases in data manipulation statements. However, you cannot refer to more than one database in a single record selection expression. In particular, you cannot cross relations from different databases.

Value Expressions

Note *Host language variables are not permitted in RDO. However, you can use a database handle in RDO. When you do, Rdb/VMS declares the variable internally.*

For more information on database handles, see the *VAX Rdb/VMS Guide to Using RDO, RDBPRE, and RDML*.

- Transaction handles

A **transaction handle** is a host language variable you associate with a transaction. If you do not declare the transaction handle explicitly, Rdb/VMS attaches an internal identifier to the transaction.

Note *Normally, you do not need to declare transaction handles. The ability to declare a transaction handle is provided for compatibility with other database products and future releases of Rdb/VMS.*

For more information on using transaction handles, see the *VAX Rdb/VMS Guide to Using RDO, RDBPRE, and RDML*.

- Request handles

A **request handle** is a host language variable that points to the internal representation of a query. Rdb/VMS compiles requests for statements that contain record selection expressions. By using request handles, an application can cause the database system to use this internal representation again, thus reducing the overhead associated with repeatedly executing a query.

Note *Request handles are not permitted in RDO or RDB\$INTERPRET. You can only use request handles in programs.*

You must explicitly declare each request handle in your program as a longword integer.

A request handle is only valid for a single database attachment, and only for as long as the database is attached. A FINISH statement detaches you from the database. If you want to use the same request handle with a later attachment to a different database, you must set the request handle to zero after the FINISH statement and before you attempt to perform the request again. Otherwise, you receive a BAD_REQ_HANDLE error message.

For more information on using request handles, see the *VAX Rdb/VMS Guide to Using RDO, RDBPRE, and RDML*.

Value Expressions

3.1.2 Literals

You can use a literal as a value expression. A **literal** is either a character string literal, numeric literal, or date literal. For example:

```
FOR E IN EMPLOYEES WITH E.LAST_NAME = "Toliver"  
  PRINT E.EMPLOYEE_ID  
END_FOR
```

|
character string literal

```
FOR S IN SALARY_HISTORY WITH S.SALARY_AMOUNT > 40000  
  PRINT S.EMPLOYEE_ID  
END_FOR
```

|
numeric literal

```
FOR S IN SALARY_HISTORY WITH S.SALARY_START = "9-APR-1980"  
  PRINT S.EMPLOYEE_ID  
END_FOR
```

|
date literal
ZK-0929A-GE

3.1.2.1 Character String Literals

A character string literal is a string of printable characters. A character string literal must be enclosed in quotation marks. The maximum length of a character string is 65,536 characters. The printable characters consist of:

- Uppercase alphabetic characters:
A – Z
- Lowercase alphabetic characters:
a – z
- Numerals:
0 – 9
- Special characters:
! @ # \$ % ^ & * () - _ = + \ ` ~

Value Expressions

[] { } ; : \ " \ | / ? > < . ,

Use a pair of quotation marks, either single or double, to enclose a character string literal. When using quotation marks, follow these rules:

- Begin and end a character string literal with the same type of quotation mark.
- To include a quotation mark of one type in a character string literal, enclose the literal in quotation marks of the other type. For example, to include double quotation marks in a character string literal, enclose the character string in single quotation marks.
- If a quotation mark appears in a character string literal enclosed by quotation marks of the same type, use two consecutive quotation marks for every one you want to include in the literal. This technique is necessary if you want to include quotation marks of both types in a single quoted string.

Table 3-1 shows how to use quotation marks in character string literals.

Table 3-1 Quotation Marks in Character String Literals

Character String Value Expression	Value
"JONES"	JONES
'JONES'	JONES
"JONES'	[invalid]
"''''"	''''
"'''''	[invalid]
'My name is "Lefty".'	My name is "Lefty".
'My ''handle'' is "Lefty".'	My 'handle' is "Lefty".

Rdb/VMS usually treats uppercase and lowercase forms of the same letter as the same character. However, it preserves the case distinction when comparing character strings. That is, NAME = "JONES" and NAME = "Jones" yield different results.

Note that these rules apply only to RDO. For language-specific rules, refer to the *VAX Rdb/VMS Guide to Using RDO, RDBPRE, and RDML* or your programming language manual.

Value Expressions

One side effect of the date and time support discussed in Section 3.1.2.3 is that the literals `YESTERDAY`, `TODAY`, and `TOMORROW` (and equivalent words in other languages) are accepted in queries.

This has an advantage for SQL (interactive and dynamic), RDO, and `RDB$INTERPRET` because you do not need to specify the exact date in your query. The following query can be packaged in a command procedure and will execute correctly every day.

```
RDO> PRINT COUNT OF O IN ORDERS WITH O.SHIP_DATE = 'TOMORROW'
```

The unfortunate side effect is that these strings are translated at compile time by `RDBPRE`, `SQL$MOD`, and `SQL$PRE`. This means a query in an application program has tomorrow's date saved in the query. In other words, if your application program is run on 24-FEB-1989, the literal `TOMORROW` is interpreted as 25-FEB-1989, and will continue to be interpreted as 25-FEB-1989 until the application is recompiled. Similarly, the literal `TODAY` is interpreted as 24-FEB-1989 and the literal `YESTERDAY` is interpreted as 23-FEB-1989 until the application is recompiled. This may not be the behavior you expect.

The `YESTERDAY`, `TODAY`, and `TOMORROW` strings are not parsed by the language processors, but are simply dispatched to the VMS Run-Time Library date conversion routine for translation.

Note Digital Equipment Corporation recommends that the `YESTERDAY`, `TODAY`, and `TOMORROW` literals not be used in precompiled applications. These literals are not re-evaluated at run time, their use in compiled applications is not supported by Rdb/VMS, and you use them at your own risk and design. The existence of these strings is a side effect of the extended date support; they are not intended for use in language processors.

Digital Equipment Corporation also does not guarantee that the current behavior of the `YESTERDAY`, `TODAY`, and `TOMORROW` strings will remain the same in future releases of Rdb/VMS.

3.1.2.2 Numeric Literals

A numeric literal is a string of digits that Rdb/VMS interprets as a decimal number. A numeric literal may be:

- A decimal string that consists of digits and an optional decimal point. The maximum length, not counting the decimal point, is 19 digits.

Value Expressions

- A decimal number in scientific notation (E-format) that consists of a decimal string mantissa and a signed integer exponent, separated by the letter E or by the letter D (for G-floating).

Rdb/VMS allows flexibility in numeric expressions. You can use unary plus and minus, you can use any form of decimal notation, and you can embed spaces in E notation. The following are valid numeric strings:

```
123
34.9
-123
.25
123.
0.33889909
6.03 E+23
6.03 E -23
```

If you use a numeric literal to assign a value to a field or a variable, the data type of the field or variable determines the maximum value you can assign.

Table 5-1 in Chapter 5 indicates the maximum size allowed for a numeric literal assigned to each Rdb/VMS data type.

Because a period at the end of a data definition statement line terminates the line, do not use a decimal point to terminate a number if you want to include more data definition clauses in the statement. The period terminates the line in the following data definition statement:

```
COMPUTED BY X*2.
```

If you want to include more data definition clauses, include a zero after the decimal point or place the value expression in parentheses:

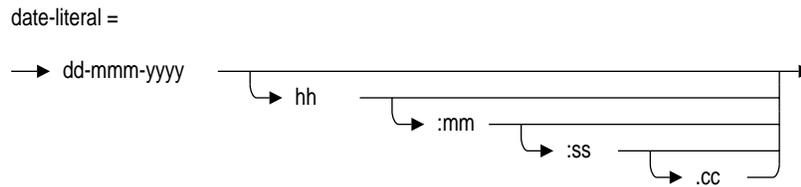
```
COMPUTED BY X*2.0
COMPUTED BY (X*2.)
```

3.1.2.3 Date Literals

A date literal is a string that represents a date. To refer to the DATE data type with a literal in an RDO statement, the default form is dd-mmm-yyyy enclosed in single or double quotation marks. Rdb/VMS does not distinguish between uppercase and lowercase characters in date literals.

Value Expressions

You can include an optional time component in addition to the day, month, and year. The time component can be complete or partial. The following diagram shows the default format for a complete date literal with a time component:



If you include a time component, you must separate the date and time portion of a date literal with a space. You can truncate the time portion of a date literal from the right. If you omit all or part of the time component, Rdb/VMS uses 00:00:00.00 as the default value.

The following examples show the default formats for date literals:

```
WITH SALARY_START = "14-JAN-1983"  
WITH SALARY_START = '14-Jan-1983'  
  
WITH SALARY_START = '14-Jan-1983 12:30:00.21'  
WITH SALARY_START = '14-Jan-1983 12:2'
```

In addition to these default formats, you can specify alternate formats for the output display format of date values using the SET DATE_FORMAT statement. See Section 9.60 for complete information on the SET DATE_FORMAT statement.

Note that SET DATE_FORMAT changes *only* the date format for output.

To change the date format for input, use the logical name LIB\$DT_INPUT_FORMAT. See the *VMS RTL Library (LIB\$) Manual* for information on the LIB\$DT_INPUT_FORMAT logical name.

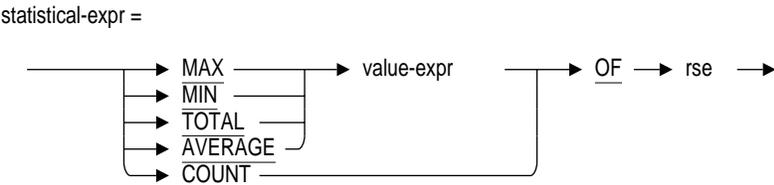
3.1.3 Statistical Expressions

Statistical expressions calculate values based on a value expression for every record in a record stream. Statistical expressions are sometimes called aggregate expressions because they calculate a single value for a collection of records. When you use a statistical expression (except for COUNT), you specify a value expression and a record selection expression (RSE). Rdb/VMS first evaluates the value expression for each record in the record stream formed by the RSE. Then it calculates a single value based on the results of the first step.

You can also use a value expression to group records within a relation and then use a statistical expression to calculate a single value for each group. This operation is often called a global aggregate because you can group records by a value in any relation in the database. For example, you can use the DEPARTMENT_CODE field in the DEPARTMENTS relation to group records in the SALARY_HISTORY relation in order to get the average salary for each department.

Table 3-2 explains the Rdb/VMS statistical operators.

Format



Arguments

value-expr
An Rdb/VMS value expression. See Section 3.1.

rse
An Rdb/VMS record selection expression. See Chapter 4.

Value Expressions

Table 3–2 Rdb/VMS Statistical Operators

Function	Value of Function	If Record Stream Is Empty:	If Field Value Is Null:
AVERAGE	The average of the values specified by the value expression for all the records specified by the RSE. The value expression must be a numeric data type.	MISSING (= null)	Not counted
COUNT	The number of records in the stream specified by the RSE.	= 0	Not applicable
MAX	The largest of the values specified by the value expression for all the records specified by the RSE.	MISSING (= null)	Not counted
MIN	The smallest of the values specified by the value expression for all the records specified by the RSE.	MISSING (= null)	Not counted
TOTAL	The sum of the values specified by the value expression for all the records specified by the RSE. The value expression must be a numeric data type.	= 0	Not counted

If the value for a statistical function evaluates to null, then the value is omitted from the calculation. In the following example, if any SH.SALARY_AMOUNT evaluates to missing, the calculation to compute the average does not include the missing value:

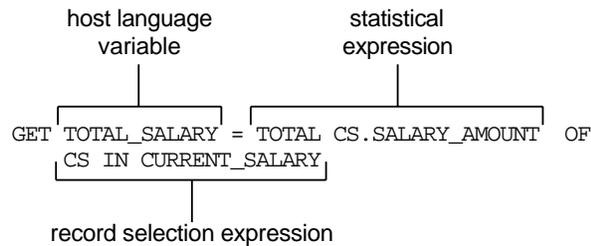
```
RDO> PRINT AVERAGE SH.SALARY_AMOUNT OF SH IN SALARY_HISTORY
```

The following sections provide examples of the various statistical functions.

3.1.3.1 The TOTAL Statistical Function

Assume you want to find the total annual payroll of the company:

```
&RDB& START_TRANSACTION READ_ONLY
&RDB&     GET TOTAL_SALARY = TOTAL CS.SALARY_AMOUNT OF
&RDB&     CS IN CURRENT_SALARY
&RDB&     END_GET
&RDB& COMMIT
```



ZK-0899A-GE

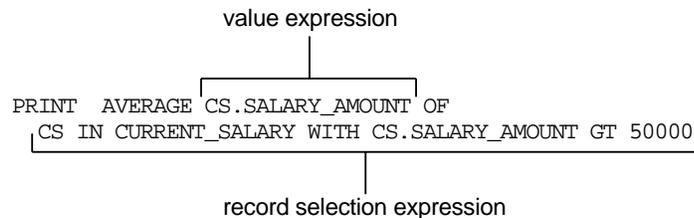
This statement calculates the value represented by the statistical expression and assigns that value to the host language variable.

Rdb/VMS uses the RSE to form a record stream of all the salary records in the CURRENT_SALARY view. Then it totals all the SALARY_AMOUNT values from each record. This total becomes the value of the statistical expression, which is the object of the GET statement. The GET statement assigns this value to the host language variable.

The next example shows the RDO statement used to display a total of all salaries for current mechanical engineers (that is, where the JOB_CODE value equals "MENG"):

```
PRINT TOTAL SH.SALARY_AMOUNT OF SH IN SALARY_HISTORY CROSS
  JH IN JOB_HISTORY OVER EMPLOYEE_ID WITH
  JH.JOB_CODE = "MENG" AND
  JH.JOB_END MISSING
```


Value Expressions



ZK-0898A-GE

In this RDO example:

- Rdb/VMS uses the RSE to form a record stream from the CURRENT_SALARY view that consists of the records where the SALARY_AMOUNT field is greater than 50,000.
- Rdb/VMS calculates the value for the statistical expression, which is equal to the average of the values in the SALARY_AMOUNT field for the records in the record stream.

3.1.3.4 The MAX Statistical Function

The following example finds the highest paid employee in the company:

```
START_TRANSACTION READ_ONLY  
  
FOR SAL IN CURRENT_SALARY WITH SAL.SALARY_AMOUNT =  
  MAX CURR.SALARY_AMOUNT OF  
  CURR IN CURRENT_SALARY  
  PRINT SAL.EMPLOYEE_ID,  
  SAL.LAST_NAME,  
  SAL.SALARY_AMOUNT  
END_FOR  
COMMIT
```

This RDO statement uses the statistical expression as the object of the WITH clause in the RSE to determine the record stream. Note that you must use two context variables. In effect, Rdb/VMS compares each value in a field of the relation to a single value from the same field.

Value Expressions

- The first context variable, SAL, refers to the relation as a whole.
- The second context variable, CURR, is used to derive the maximum value of SALARY_AMOUNT to compare with each salary amount from the relation.

3.1.3.5 Global Aggregates

A statistical expression can operate on a group of records within a record stream. This operation is often called a global aggregate function because you can group records by a value in any relation in the database. For example, you can use the DEPARTMENT_CODE field in the DEPARTMENTS relation to group records in another relation, SALARY_HISTORY, to get the average salary for each department.

If you need to find the average salary for all employees in a corporation, you could use the RDO statements in the following example:

```
PRINT AVERAGE SH.SALARY_AMOUNT OF SH IN SALARY_HISTORY WITH
      SH.SALARY_END MISSING
```

Here, the AVERAGE function operates on all the records in the record stream formed by the RSE. However, a statistical function also can operate on groups of records within the record stream.

To find out how many employees currently work in each department:

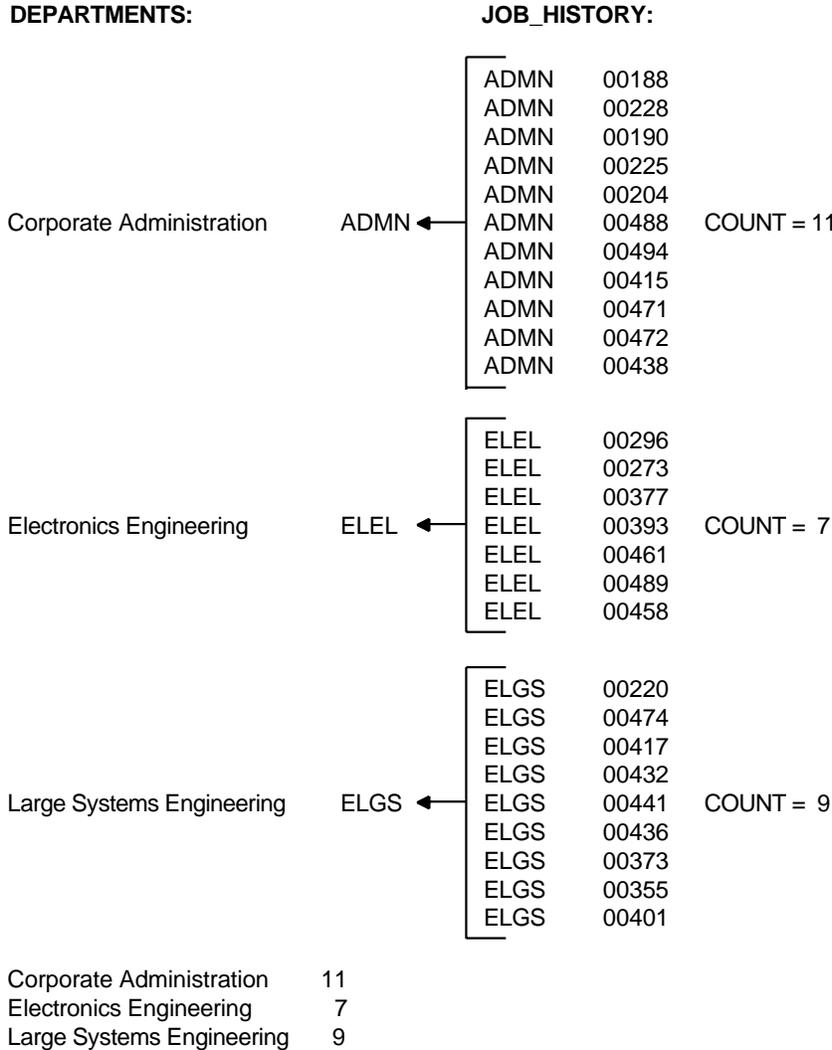
```
FOR D IN DEPARTMENTS
  PRINT D.DEPARTMENT_NAME ,
        COUNT OF JH IN JOB_HISTORY WITH
          JH.DEPARTMENT_CODE = D.DEPARTMENT_CODE AND
          JH.JOB_END MISSING
END_FOR
```

Here, instead of returning a value for each record in the record stream, a global aggregate function returns only a single value for the whole group.

Figure 3–1 illustrates how the global aggregate function works, using a simplified subset of the two relations.

Value Expressions

Figure 3-1 Using a Statistical Expression to Group Records



ZK-7381-GE

Value Expressions

In the preceding example:

- FOR D IN DEPARTMENTS sets up an outer loop. The query performs a PRINT statement for each record in the DEPARTMENTS relation.
- The COUNT statistical expression is the object of the PRINT statement. The statistical expression includes its own looping function: it loops through the records specified by the RSE and counts them.
- The RSE links the inner loop, formed by the COUNT function, to the outer loop, formed by the FOR statement. The following steps take place:
 - Rdb/VMS finds the first record in the DEPARTMENTS relation and reads the DEPARTMENT_CODE field. Assume that this value is ADMN.
 - Rdb/VMS finds the records in the JOB_HISTORY relation where the DEPARTMENT_CODE is ADMN. For each record in the JOB_HISTORY relation where DEPARTMENT_CODE is ADMN, it adds one to the count.
 - When Rdb/VMS has found all the matches in the JOB_HISTORY relation for ADMN, it displays the DEPARTMENT_NAME value that goes with ADMN and the count of JOB_HISTORY records.
 - Then Rdb/VMS returns to the outer loop of the query and finds the next DEPARTMENT_CODE value.

This type of query always has the following three features:

- The outer loop establishes the value by which to group records.
- The statistical expression creates the inner loop.
- The WITH clause links the two loops together.

Note The WITH clause in the inner loop must refer to the context variable established in the outer loop.

You can also use a conditional expression to place a restriction on the outer loop.

Value Expressions

The next example counts the number of employees in each job code group:

```
FOR J IN JOBS
PRINT J.JOB_TITLE,
      COUNT OF JH IN JOB_HISTORY WITH
      J.JOB_CODE = JH.JOB_CODE AND
      JH.JOB_END MISSING
END_FOR
```

This example works in a similar fashion to the example illustrated in Figure 3-1. Specifically:

- FOR J IN JOBS forms the outer loop.
- The COUNT statistical expression forms the inner loop.
- The WITH clause links the inner loop with the outer loop.

The following example is more complicated, due to the joining of multiple relations. The example finds the total payroll for each department:

```
FOR D IN DEPARTMENTS
PRINT D.DEPARTMENT_NAME,
      TOTAL SH.SALARY_AMOUNT OF
      SH IN SALARY_HISTORY CROSS
      JH IN JOB_HISTORY OVER EMPLOYEE_ID WITH
      JH.DEPARTMENT_CODE = D.DEPARTMENT_CODE AND
      JH.JOB_END MISSING AND
      SH.SALARY_END MISSING
END_FOR
```

The preceding example is complicated by having no direct link between the department information in the DEPARTMENTS relation and the salary information in the SALARY_HISTORY relation. The query uses the JOB_HISTORY relation as the link because it contains the common fields, DEPARTMENT_CODE and EMPLOYEE_ID, and is interpreted as follows:

- FOR D IN DEPARTMENTS sets up the outer loop.
- The TOTAL statistical function starts the inner loop.
- In this case, the query requires an extra join to link the salary information in SALARY_HISTORY through JOB_HISTORY to DEPARTMENTS. The SALARY_HISTORY and JOB_HISTORY relations share only one field: EMPLOYEE_ID.

Value Expressions

- JH.DEPARTMENT_CODE = D.DEPARTMENT_CODE links the inner loop to the outer loop.
- The final two elements in the WITH clause limit the record stream to only those records that correspond to current SALARY_HISTORY and JOB_HISTORY records. *They are both necessary.* If you limit the record stream to current SALARY_HISTORY records, there will be one current salary record for each JOB_HISTORY record with a matching EMPLOYEE_ID. The result looks correct but is actually inflated.

Figure 3-2 shows how the preceding query works with a subset of the data.

Figure 3-2 A Statistical Expression Across Three Relations

	From DEPARTMENTS		From JOB_HISTORY		From SALARY_HISTORY
Corporate Administration	ADMN	ADMN	00188	00188	21093
		ADMN	00228	00228	85150
		ADMN	00190	00190	34976
		ADMN	00225	00225	8687
		ADMN	00204	00204	87143
					↓
					TOTAL
Electronics Engineering	ELEL	ELEL	00296	00296	20770
		ELEL	00273	00273	44264
		ELEL	00377	00377	15646
					↓
					TOTAL

ZK-7382-GE

Value Expressions

In the next example, the PRINT statement includes two value expressions:

- The field D.DEPARTMENT_NAME
- The arithmetic expression, which includes two statistical expressions

Otherwise, this query is identical to previous statistical expression examples in this chapter.

```
FOR D IN DEPARTMENTS
  PRINT D.DEPARTMENT_NAME, ((TOTAL SH.SALARY_AMOUNT OF
    SH IN SALARY_HISTORY CROSS
    JH IN JOB_HISTORY OVER EMPLOYEE_ID WITH
      JH.DEPARTMENT_CODE = D.DEPARTMENT_CODE AND
      JH.JOB_END MISSING AND
      SH.SALARY_END MISSING) / TOTAL SH.SALARY_AMOUNT OF
    SH IN SALARY_HISTORY WITH
      SH.SALARY_END MISSING) * 100
END_FOR
```

Because SALARY_AMOUNT is defined as a longword integer in the preceding example, the precision of the arithmetic expression is limited to whole numbers. If you include this query in a program, you can remove the calculation of the percentage from the Rdb/VMS query and use your program's host language calculation instead. For example, you can retrieve the SALARY_AMOUNT value, insert it into your program, and convert it to F-floating before you calculate the percentage.

3.1.4 Arithmetic Expressions

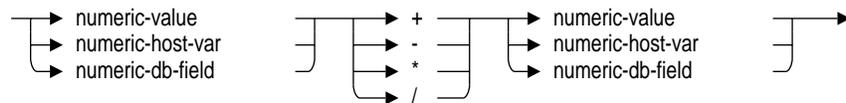
An **arithmetic expression** combines numeric values, numeric host language variables, and numeric database fields with arithmetic operators. When you use an arithmetic expression in a statement, Rdb/VMS calculates the value associated with the expression and uses that value when executing the statement. Therefore, an arithmetic expression must be reducible to a value. However, Rdb/VMS does not permit arithmetic operations on fields defined with the DATE data type.

If either operand of an arithmetic expression is a null value, the resulting value is also null.

Value Expressions

Format

arith-expr =



The arithmetic operators and their functions are:

- + Add
- Subtract
- * Multiply
- / Divide

You do not have to use spaces to separate arithmetic operators from value expressions, except in one case: if an Rdb/VMS name precedes a minus sign, you must separate the name and the minus sign by a space. Otherwise, a language that allows hyphens within names interprets the minus sign as a hyphen.

You can use parentheses to control the order in which Rdb/VMS performs arithmetic operations. Rdb/VMS follows the normal rules of precedence. That is, it evaluates arithmetic expressions in the following order:

- 1 Numeric values, numeric host language variables, or numeric database fields or both in parentheses
- 2 Multiplication and division, from left to right
- 3 Addition and subtraction, from left to right

Examples

Example 1

You can use an arithmetic expression in a data definition statement:

```
DEFINE VIEW DEDUCT OF
  E IN EMPLOYEES CROSS
  S IN SALARY_HISTORY OVER EMPLOYEE_ID WITH
  S.SALARY_END MISSING.
  E.LAST_NAME.
  E.FIRST_NAME.
  AMOUNT COMPUTED BY
    ( ( S.SALARY_AMOUNT / 52 ) * 0.05 ).
END DEDUCT VIEW.

FOR FIRST 5 D IN DEDUCT PRINT D.* END_FOR
LAST_NAME      FIRST_NAME      AMOUNT
Toliver        Alvin            4.972307692307692E+001
Smith          Terry            1.122692307692308E+001
Dietrich       Rick              1.778557692307692E+001
Kilpatrick     Janet            1.683653846153846E+001
Nash           Norman           3.101346153846154E+001
```

This example defines a view that calculates a payroll deduction for health insurance:

- The record selection expression limits the records in the view to current salary records.
- The view fields include the employee's name and a weekly deduction field, calculated using a complex arithmetic expression from the annual salary for each employee (5 percent of the weekly salary).

Example 2

You can use an arithmetic expression to store a value:

```
START_TRANSACTION READ_WRITE
!
! Get the current record
!
FOR OLD_SAL IN SALARY_HISTORY WITH
  OLD_SAL.EMPLOYEE_ID = "00164" AND
  OLD_SAL.SALARY_END MISSING
!
```

Value Expressions

```
! Modify the old salary, giving it an end date
!
MODIFY OLD_SAL USING
    OLD_SAL.SALARY_END = "14-APR-1984"
END_MODIFY
!
! Store the new salary, giving a 10% raise
!
STORE NEW_SAL IN SALARY_HISTORY USING
    NEW_SAL.EMPLOYEE_ID = OLD_SAL.EMPLOYEE_ID;
    NEW_SAL.SALARY_START = "14-APR-1984";
    NEW_SAL.SALARY_AMOUNT = ( OLD_SAL.SALARY_AMOUNT * 1.10 )
END_STORE
!
END_FOR

!
! Check the results
!
FOR S IN SALARY_HISTORY WITH S.EMPLOYEE_ID = '00164' PRINT
    S.EMPLOYEE_ID,
    S.SALARY_START,
    S.SALARY_END,
    S.SALARY_AMOUNT
END_FOR
00164      5-JUL-1980 00:00:00.00      2-MAR-1981 00:00:00.00      26291.00
00164      2-MAR-1981 00:00:00.00      21-SEP-1981 00:00:00.00      26291.00
00164     21-SEP-1981 00:00:00.00      14-JAN-1983 00:00:00.00      50000.00
00164     14-JAN-1983 00:00:00.00      14-APR-1984 00:00:00.00      51712.00
00164     14-APR-1984 00:00:00.00      17-NOV-1858 00:00:00.00      56883.20
COMMIT
```

This RDO example defines a FOR loop to include the current salary record for an employee. Then it modifies the salary in three steps:

- 1 Modifies the old record by setting the SALARY_END field to the date the old salary ends (14-APR-1984 in the example)
- 2 Stores a new record, using the old EMPLOYEE_ID field and the date the old salary ends
- 3 Calculates the new SALARY_AMOUNT field using the old SALARY_AMOUNT in an arithmetic expression

3.1.5 Concatenated Expressions

A **concatenated expression** is a value expression that combines two value expressions by joining the second to the end of the first. Concatenated expressions use the concatenate operator (|). You can combine value expressions of any kind, including numeric expressions, string expressions, and literals.

You can use concatenated expressions to simulate edit strings.

Format

```
concat-expr =  
    → value-expr  → |  → value-expr  →
```

Examples

Example 1

You can use a concatenated expression as a field in a view:

```
DEFINE VIEW MAIL OF  
E IN EMPLOYEES.  
NAME COMPUTED BY  
    E.FIRST_NAME | ' ' | E.MIDDLE_INITIAL | ' ' | E.LAST_NAME.  
E.ADDRESS_DATA_2.  
E.CITY.  
E.POSTAL_CODE.  
END MAIL VIEW.
```

This statement creates a field for the view by combining the three name fields from the EMPLOYEES relation.

Value Expressions

Example 2

A concatenated expression can simulate an edit string:

```
DEFINE VIEW DOLLARS OF S IN SALARY_HISTORY.  
  S.EMPLOYEE_ID.  
  S.SALARY_START.  
  S.SALARY_END.  
  SAL COMPUTED BY "$" | S.SALARY_AMOUNT.  
END.
```

When you display the fields of the view DOLLARS, a dollar sign appears in front of the salary amount.

3.1.6 Missing Values (RDB\$MISSING Expression)

A missing value for a field in a relation has no value associated with it. The missing value is an *attribute* of a field rather than a *value* stored in a field.

The **null flag** refers to the information that Rdb/VMS maintains about a field to determine whether or not it is null, or missing. Each field in the database has information associated with it using the null flag to mark the field as either missing or not missing. Value expressions can evaluate to a value or to missing. Conditional expressions can evaluate to true, false, or missing. Missing information also affects the evaluation of statistical functions, sorting, and projections.

If the source of the assignment is a database field and that database field has a missing value defined for it, then its missing value is used. Otherwise, the missing value is determined by the data type. For fixed-length strings, the string is blank filled. For variable-length strings, the length is set to zero. All numeric data types are set to zero. For the DATE data type, the missing value of the VMS zero date is used. This represents the date 17-NOV-1858 00:00:00.00.

When you assign a value to a database field, the null flag for that field is set according to the value in the assignment statement. If the value of the assignment is the missing value defined for that field, the null flag is set for that field. Otherwise, the null flag is not set. If no missing value is defined, the null flag for that field is not set.

Value Expressions

When you define a field, choose a missing value for that field that will never occur as an actual value for the field. If you change the definition of the missing value for a field, you change the incoming value that sets the null flag and you change the value produced when reading a field with the null flag set.

Rdb/VMS evaluates the expression RDB\$MISSING to determine the missing value for a field. There are two ways to declare that a field is missing:

- Omit the field in the STORE statement.
When you omit the field from the STORE statement, Rdb/VMS marks the field as null.
- Explicitly store the value associated with RDB\$MISSING for the field, or use RDB\$MISSING to signal the field's value to Rdb/VMS. Rdb/VMS does not actually store what you specify; rather, it stores nothing and marks the field's value as missing or null.

If you know the missing value for a field, you can include this value in the STORE or MODIFY statement. For example, assume you want to make the ADDRESS_DATA field missing for a particular employee. The definition for ADDRESS_DATA is as follows:

```
DEFINE FIELD ADDRESS_DATA
    DATA_TYPE IS TEXT SIZE IS 20
    MISSING_VALUE IS 'Not available'.
```

To make the field missing, you use a MODIFY statement:

```
FOR E IN EMPLOYEES WITH E.EMPLOYEE_ID = "00175"
    MODIFY E USING
        E.ADDRESS_DATA = 'Not available'
    END_MODIFY
END_FOR
```

When this statement executes, Rdb/VMS marks the field as null; and when you retrieve the field, Rdb/VMS returns the value "Not available", the missing value. Note, however, that to find all the records in which the ADDRESS_DATA field is marked as null, use the MISSING operator in a conditional expression, as in the following example:

```
FOR E IN EMPLOYEES WITH E.ADDRESS_DATA MISSING
    PRINT E.*
END_FOR
```

Value Expressions

If instead of the MISSING operator, you use the missing value in the conditional expression, Rdb/VMS displays nothing. The following query does *not* find records in which the ADDRESS_DATA field is marked as null:

```
FOR E IN EMPLOYEES WITH E.ADDRESS_DATA = "Not available"
  PRINT E.*
END_FOR
```

If you want to make a field missing and you do not know the literal missing value, you can use the expression RDB\$MISSING, which returns the missing value. For example, you might want to write general update programs that do not depend on a particular missing value. You can use RDB\$MISSING instead of an explicit value in a STORE or MODIFY statement.

Note *In order to use RDB\$MISSING, you must have defined a missing value explicitly for the field in the DEFINE FIELD statement. If you have not, RDB\$MISSING is undefined for that field.*

You can define a missing value that is longer than the length specified for the field in the field definition. However, when the missing value is displayed, it will be truncated to the length specified for the field in the field definition. In the following example, the missing value "Long" in the field SEX_FIELD is truncated to one character (the length specified for the field in the definition of SEX_FIELD).

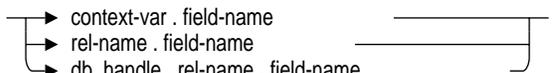
```
RDO> !
RDO> ! Define field SEX_FIELD:
RDO> !
RDO> DEFINE FIELD SEX_FIELD
cont> DATATYPE IS TEXT 1
cont> MISSING_VALUE "Long".
RDO> !
RDO> ! Define relation TEST_RELATION:
RDO> !
RDO> DEFINE RELATION TEST_RELATION.
cont> SEX_FIELD.
cont> EMPLOYEE_ID BASED ON ID_NUMBER.
cont> END RELATION.
RDO> !
RDO> ! Store a single record in TEST_RELATION, providing a value
RDO> ! for EMPLOYEE_ID, but not for SEX_FIELD. This creates a
RDO> ! missing value for SEX_FIELD in the record:
RDO> !
RDO> STORE TR IN TEST_RELATION USING
cont> TR.EMPLOYEE_ID = "00175";
cont> END_STORE
RDO> !
```

Value Expressions

```
RDO> ! The missing value in SEX_FIELD appears as one character
RDO> ! when the record is displayed:
RDO> !
RDO> FOR TR IN TEST_RELATION
cont> PRINT TR.*
cont> END_FOR
SEX_FIELD    EMPLOYEE_ID
L            00175
```

Format

missing-value =

→ RDB\$MISSING → () →

Usage Notes

The default value for a field ("Rdb default" in displays by the SQL statement SHOW TABLE table-name) is *not* the same as the missing value that you can specify with RDO. If you do not specify a value for a field (column) with a default value, the default value is actually stored in the database. This is true whether you are using RDO or SQL.

An RDO missing value is not actually stored in the database. If you use RDO to specify a missing value for a field, that missing value is displayed by RDO when the field has no value stored and the internal null flag is set. SQL does not recognize any missing value specified by RDO; if the field has no value stored and the null flag is set, then SQL displays "NULL" for the column, regardless of whether you specified any missing value with RDO.

One implication of the way in which Rdb/VMS handles default values is that if you change the default value for a column, it has no effect on any existing data in the database; that is, rows stored with columns containing the "old" default values are not changed. By contrast, changing the missing value *does* change what is displayed by RDO-based applications for columns that have no value stored and that have the null flag set.

For more information on specifying default values, see the sections on the CREATE DOMAIN, ALTER DOMAIN, CREATE TABLE, and ALTER TABLE statements in the *VAX Rdb/VMS SQL Reference Manual*.

Value Expressions

Examples

Example 1

The following example uses RDB\$MISSING in a STORE statement:

```
.
.
.
*
*   If the date in the input file contains zeros,
*   use the GET statement with RDB$MISSING to retrieve
*   the missing value and assign it to a variable.
*
      IF END-DATE = '000000' THEN
&RDB&   GET END-DATE = RDB$MISSING(SALARY_HISTORY.SALARY_END)
&RDB&   END_GET
      END-IF.
*

*   Store the field values in the relation.
*   If the date in the input file contained zeros,
*   the missing value is passed to Rdb/VMS, and
*   the SALARY_END field is marked as missing.
*
&RDB&   STORE S IN SALARY_HISTORY USING
&RDB&       S.EMPLOYEE_ID = EMP-ID;
&RDB&       S.SALARY_START = START-DATE;
&RDB&       S.SALARY_END = END-DATE;
&RDB&       S.SALARY_AMOUNT = SALARY;
&RDB&   END_STORE
```

This program fragment shows how your program might test for a missing field in an input file and make the field missing in the database. This program assumes that a missing value has been defined for the field. The program does not need to know what the missing value is, and the program needs only one STORE statement.

3.1.7 FIRST FROM Expression

The FIRST FROM expression works in two steps:

- 1 It forms the record stream as specified by the record selection expression.
- 2 If at least one record matches the RSE, Rdb/VMS uses the values stored in the first record of the record stream to evaluate the value expression.

Format

first-from-expr

→ FIRST → value-expr → FROM → rse →

Usage Notes

If there are no records in the record stream, FIRST FROM returns the RDB-E-FROM-NO-MATCH error message instead of simply returning no records. You must use the ON ERROR clause in your programs to prevent this error from terminating your program abnormally when no records are found.

Examples

Example 1

You can use the FIRST FROM expression to look up a value in a relation.

Assume you are storing a JOB_HISTORY record, and you know the job title but not the job code. The JOB_CODE and JOB_TITLE fields in the JOBS relation have a one-to-one relationship. You can use the FIRST FROM expression to look up the code corresponding to the title.

Value Expressions

```
STORE JH IN JOB_HISTORY USING
  JH.EMPLOYEE_ID = "00164";
  JH.JOB_CODE = FIRST J.JOB_CODE
    FROM J IN JOBS WITH
      J.JOB_TITLE = "Electrical Engineer";
  JH.JOB_START = "13-APR-1984";
  JH.SUPERVISOR_ID = "00175";
  JH.DEPARTMENT_CODE = "ELEL"
END_STORE
```

In this example, the `FIRST FROM` expression finds the first record in the `JOBS` relation that meets the conditions of the `WITH` clause. Because the `JOB_CODE` and `JOB_TITLE` fields are both unique in `JOBS`, only one record meets those conditions. Therefore, the `JOB_CODE` field in the record must correspond to the value you specified for `JOB_TITLE`.

3.1.8 Database Key

You can retrieve a value called the database key from an Rdb/VMS relation. The **database key** is a pointer or address that indicates a specific record in the database. You can retrieve this key as though it were a field in the record. For a relation, the database key is 8 bytes. It may be longer for a view. Once you have retrieved the database key, you can use it to retrieve its associated record directly, in a `FETCH` statement or in a record selection expression. The database key enables you to keep track of a subset of records in the database and retrieve these records repeatedly, without using data manipulation syntax. For example, your program could form the equivalent of a `DATATRIEVE` collection with database keys.

Note In RDBPRE and RDML preprocessed programs (BASIC, COBOL, and FORTRAN), you can obtain the database key of a record just stored by using the `GET . . . RDB$DB_KEY` expression in a `STORE . . . END_STORE` block. See the section on `STORE` in Chapter 9 for details. Also see the discussion on `DBKEY SCOPE` in the `INVOKE` section of Chapter 9 for important related information.

Format

db-key =
 → context-var → . → RDB\$DB_KEY →

Examples

Example 1

You can use a database key to retrieve records in a BASIC program:

```

      N = 0
&RDB&   FOR E IN EMPLOYEES
          CROSS J IN JOB_HISTORY
          WITH J.DEPARTMENT_CODE = "ELEL" AND
          J.JOB_END MISSING
          N = N + 1
&RDB&   GET KEY(N) = E.RDB$DB_KEY
&RDB&   END_FOR

      DO 10 I=1,N
&RDB&   FOR E IN EMPLOYEES WITH E.RDB$DB_KEY = KEY(I)
      .
      .
      .
      [ Here your program can process the record stream ]
      .
      .
&RDB&   END_FOR
      10 CONTINUE
  
```

The first FOR loop in this fragment retrieves the database key values for the current EMPLOYEES records in the Electronics Engineering department. The second loop uses the stored database keys to access those records directly.

Example 2

The following FORTRAN program fragment shows how the database key of a record just stored can be retrieved into a host language variable, using GET . . . RDB\$DB_KEY, in a STORE . . . END_STORE block. For this complete program, see the INVOKE DATABASE examples section in Chapter 9.

Value Expressions

```
&RDB& DATABASE FILENAME 'PERSONNEL' DBKEY SCOPE IS FINISH
&RDB& START_TRANSACTION READ_WRITE

&RDB&   STORE P IN EMPLOYEES USING P.EMPLOYEE_ID =15231;
&RDB&   P.LAST_NAME = "Santoli";
&RDB&   GET MY_DB_KEY = P.RDB$DB_KEY;
&RDB&   END_GET
&RDB&   END_STORE

&RDB& COMMIT

&RDB& START_TRANSACTION READ_WRITE
&RDB&   for J IN EMPLOYEES with J.rdb$db_key = my_db_key
&RDB&     get
&RDB&       1      JC = J.EMPLOYEE_ID;
&RDB&       2      JT = J.LAST_NAME;
&RDB&       3 end_get
&RDB&     4end_for
&RDB&     write (6,60020) JC, JT
&RDB&     .
&RDB&     .
&RDB&     .
```

3.1.9 Segmented String Expressions

The segmented string expressions let you retrieve separately the value and length of each segment in a segmented string.

Rdb/VMS defines a special name to refer to the segments of a segmented string. This value expression is equivalent to a field name; it names the *fields* or segments of the string. Furthermore, because segments can vary in length, Rdb/VMS also defines a name for the length of a segment. You must use these value expressions to retrieve the length and value of a segment. These names are:

RDB\$VALUE	The value stored in a segment of a segmented string
RDB\$LENGTH	The length in bytes of a segment

For more details on segmented strings, see Figure 5-1 and its accompanying text.

Value Expressions

Format

seg-expr =

context-var . RDB\$VALUE
context-var . RDB\$LENGTH

Examples

Example 1

The following example shows how to retrieve segments in a segmented string. Note that the sample personnel database has no data loaded in the RESUMES relation, so only the RDO prompt is displayed:

```
RDO> FOR R IN RESUMES WITH R.EMPLOYEE_ID = "00164"  
cont> PRINT R.EMPLOYEE_ID  
cont> FOR RES IN R.RESUME  
cont> PRINT RES.RDB$VALUE, RES.RDB$LENGTH  
cont> END_FOR  
cont> END_FOR  
RDO>
```

This example demonstrates the nested loop structure of a segmented string statement:

- The RSE specifies a record for retrieval.
- Within the outer loop, Rdb/VMS retrieves the EMPLOYEE_ID field from the specified record.
- Within the inner loop, you associate a context variable with the segmented string field, RESUME, and print the value of each segment of the string using the RDB\$VALUE expression, and the length of each segment using RDB\$LENGTH.
- You must finish each loop with an END_FOR clause.

Conditional Expressions

3.2 Conditional Expressions

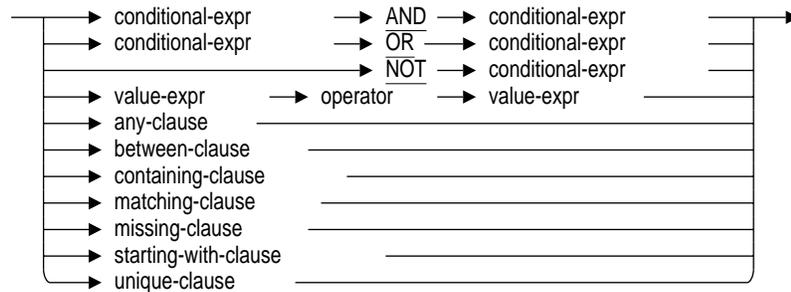
You use conditional expressions in Rdb/VMS as objects for the WITH clause of the record selection expression or the VALID IF clause in field definitions. A conditional expression has a value of true, false, or missing. If no value is stored in a field of a record, then the relationship of that field to others is unknown. Therefore, the results of comparisons that use that field are neither true nor false, but rather are considered missing. See Chapter 4 for more information about record selection expressions.

Conditional expressions consist of value expressions, relational operators, and logical operators:

- Relational operators let you compare value expressions.
- Logical operators let you join two or more conditional expressions and reverse the value of a conditional expression.

Format

conditional-expr =

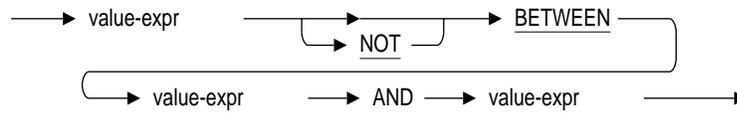


any-clause =

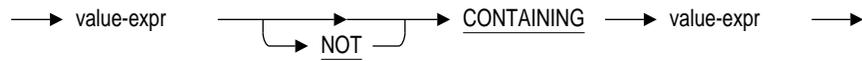


Conditional Expressions

between-clause =



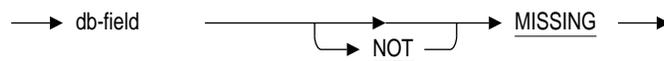
containing-clause =



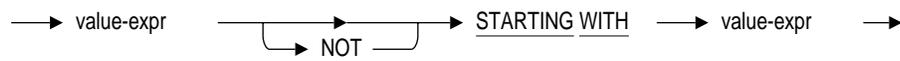
matching-clause =



missing-clause =

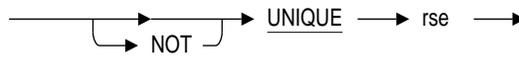


starting-with-clause =



Conditional Expressions

unique-clause =



Arguments

value-expr

A value expression. See Section 3.1. If either value expression in a conditional expression evaluates to null, the conditional expression evaluates to null.

operator

One of the following relational operators:

= EQ
<> NE
> GT
>= GE
< LT
<= LE

In all cases, if either operand in a conditional expression is null, the value of the conditional expression is null. Table 3–3 shows all the Rdb/VMS relational operators.

Note *The NOT operator applies to conditional expressions. Therefore, the following statement is not legal:*

WITH S.SALARY_AMOUNT NOT = 30000

Use one of the following alternatives:

WITH NOT (S.SALARY_AMOUNT = 30000)

WITH S.SALARY_AMOUNT NE 30000

WITH S.SALARY_AMOUNT <> 30000

rse

A record selection expression. Chapter 4 explains record selection expressions in detail.

Conditional Expressions

match-expr

An expression in quotation marks that is used to match a pattern. The MATCHING operator is not case sensitive. The MATCHING operator uses the following special symbols:

- * Matches any string of zero or more characters
- % Matches any single character

3.2.1 Relational Operators

Relational operators specify the relationship of value expressions. These operators perform the following kinds of operations:

- Comparing a value with a range
- Matching a pattern
- Testing for missing fields

Table 3–3 lists the Rdb/VMS relational operators.

Table 3–3 RDO Relational Operators

Permitted Symbols	Relational Operation
EQ =	True if the two value expressions are equal.
NE <>	True if the two value expressions are not equal.
GT >	True if the first value expression is greater than the second.
GE >=	True if the first value expression is greater than or equal to the second.
LT <	True if the first value expression is less than the second.
LE <=	True if the first value expression is less than or equal to the second.
BETWEEN	True if the first value expression is equal to or between the second and third value expressions.

(continued on next page)

Conditional Expressions

Table 3-3 (Cont.) RDO Relational Operators

Permitted Symbols	Relational Operation
ANY	True if the record stream specified by the RSE includes at least one record. If you add NOT, the condition is true if no records are in the record stream.
MATCHING	True if the second expression matches a substring of the first value expression. MATCHING uses these special characters: * Matches any string in that position % Matches any character in that position
MISSING	True if the value expression is missing.
UNIQUE	True if the record stream specified by the RSE includes only one record. If you add NOT, the condition is true if more than one record is in the record stream or the record stream is empty.
CONTAINING	True if the string specified by the second string expression is found within the string specified by the first. Not case sensitive.
STARTING WITH	True if the first characters of the first string expression match the second string expression. Case sensitive.

The following rules apply to the Rdb/VMS relational operators:

- Rdb/VMS compares character string literals according to the rules of the specified collating sequence. The default is the ASCII collating sequence if you have not specified a collating sequence. With the ASCII collating sequence, Rdb/VMS considers lowercase letters to have a greater value than uppercase letters and considers the letters at the beginning of the alphabet to have a lesser value than those at the end:

“a” > “A”
“a” > “Z”
“a” < “z”
“A” < “Z”

Conditional Expressions

- If you use a zero length string for the second expression with either the CONTAINING, STARTING WITH, or MATCHING operator, all records will be selected. The following example would return *all* records in the database:

```
E.LAST_NAME MATCHING ""
```

- When you compare numeric data and text data, the numeric data is converted to text before Rdb/VMS performs the comparison.

Examples

Example 1

The GT (>) operator compares a value with a range:

```
SH.SALARY_AMOUNT > 50000
```

This expression is true if the value in the SALARY_AMOUNT field of the SALARY_HISTORY record is greater than 50,000. When Rdb/VMS evaluates this expression, it examines the relationship between the two value expressions, SH.SALARY_AMOUNT and 50,000.

Example 2

The CONTAINING operator tests whether a specific substring is included in another substring:

```
FOR E IN EMPLOYEES
WITH
E.LAST_NAME CONTAINING "VER      "
!6 trailing spaces
PRINT E.LAST_NAME
END_FOR
  LAST_NAME
  Toliver
  Silver
!
!
FOR E IN EMPLOYEES
WITH
E.LAST_NAME CONTAINING "VER      "
!7 trailing spaces
PRINT E.LAST_NAME
END_FOR
  LAST_NAME
  Silver
```

Conditional Expressions

This expression is true in both cases if the E.LAST NAME context variable contains the string

```
"VER          " .
```

In the first case the expression

```
"VER          "  
!6 trailing spaces
```

is contained in two 14-character LAST_NAME values,

```
"Toliver          "  
!7 trailing spaces
```

and

```
"Silver          "  
!8 trailing spaces.
```

In the second case the expression

```
"VER          "  
!7 trailing spaces
```

is contained in a single 14-character LAST_NAME value,

```
"Silver          "  
!8 trailing spaces.
```

Note that the value expression on the left side of the CONTAINING operator has to include the value expression on the right hand side. The CONTAINING operator is case insensitive.

Example 3

The MATCHING operator matches a pattern. This example is the log file from an RDO session:

```
!  
! Find the names in which the letters 'on' come last  
! in the name. Because LAST_NAME is 14 characters long,  
! the matching pattern specifies 7 spaces after the  
! sequence 'on'. Therefore, the asterisk accounts for
```

Conditional Expressions

```
! the first 5 characters. RDO returns a 7-character
! last name. By reducing the number of explicit spaces
! in the matching pattern, RDO returns longer last
! names ending with the letters 'on'.
!
FOR FIRST 5 E IN EMPLOYEES WITH
    E.LAST_NAME MATCHING "*on      "
    REDUCED TO E.LAST_NAME
PRINT
E.LAST_NAME
END_FOR
    LAST_NAME
    Clinton
    Jackson
    Johnson
!
!
! Find the names where 'on' comes after the first
! character in the name:
!
FOR FIRST 5 E IN EMPLOYEES WITH
    E.LAST_NAME MATCHING "%on*"
    REDUCED TO E.LAST_NAME
PRINT
E.LAST_NAME
END_FOR
    LAST_NAME
    Connolly
    Lonergan
!
! MATCHING also works with numeric data types. Find
! the salaries that begin with the number 3. This
! is another way of finding all the salaries in the
! range BETWEEN 30,000 AND 39,999:
!
FOR FIRST 5 S IN SALARY_HISTORY WITH
    S.SALARY_AMOUNT MATCHING "3*"
    PRINT S.SALARY_AMOUNT
END_FOR
    SALARY_AMOUNT
    32254.00
    30598.00
    30880.00
    32589.00
    33944.00
!
```

Conditional Expressions

```
! Find the salaries where the number 87 follows the
! first digit:
!
FOR FIRST 5 S IN SALARY_HISTORY WITH
  S.SALARY_AMOUNT MATCHING "%87*"
  PRINT S.SALARY_AMOUNT
END_FOR
  SALARY_AMOUNT
    48797.00
    38758.00
    78743.00
!
```

Example 4

The ANY operator tests for the existence of records that meet a certain condition:

```
FOR E IN EMPLOYEES WITH
  ANY D IN DEGREES WITH
    D.EMPLOYEE_ID = E.EMPLOYEE_ID AND
    D.COLLEGE_CODE = "STAN"
PRINT E.EMPLOYEE_ID, E.LAST_NAME
END_FOR
```

This RDO query finds all the employees who have degree records stored in the DEGREES relation. You might translate this query, “Find all the EMPLOYEES records such that there exists a corresponding DEGREES record indicating the employee received a degree from ‘STAN’.”

Example 5

The MISSING operator retrieves records in which a field value is missing:

```
RDO> PRINT COUNT OF SH IN SALARY_HISTORY WITH
cont> SH.SALARY_AMOUNT > 20000 AND
cont> SH.SALARY_END MISSING
```

Note, however, that the display may be misleading. Consider the following set of queries:

```
PRINT COUNT OF SH IN SALARY_HISTORY WITH
  SH.SALARY_AMOUNT > 20000 AND
  SH.SALARY_END MISSING

PRINT COUNT OF SH IN SALARY_HISTORY WITH
  SH.SALARY_AMOUNT < 20000 AND
  SH.SALARY_END MISSING

PRINT COUNT OF SH IN SALARY_HISTORY WITH
  SH.SALARY_AMOUNT = 20000 AND
  SH.SALARY_END MISSING
```

Conditional Expressions

```
PRINT COUNT OF SH IN SALARY_HISTORY WITH
  SH.SALARY_END MISSING
```

Now assume that you store a new record in the SALARY_HISTORY relation with no current salary amount specified:

```
STORE S IN SALARY_HISTORY
  USING
    S.EMPLOYEE_ID = "00164";
    S.SALARY_START = "11-FEB-1984"
END_STORE
```

The new record's SALARY_AMOUNT field is empty. Therefore, it is not less than, greater than, or equal to 20,000. Its relationship to 20,000 is unknown. For this reason, the record would not be included in the record stream (or the COUNT) for any of the first three queries. The record exists, however, so it would be included in the final COUNT. The results might look like this:

```
RDO> PRINT COUNT OF SH IN SALARY_HISTORY WITH
cont>  SH.SALARY_AMOUNT > 20000 AND
cont>  SH.SALARY_END MISSING
    61
RDO> PRINT COUNT OF SH IN SALARY_HISTORY WITH
cont>  SH.SALARY_AMOUNT < 20000 AND
cont>  SH.SALARY_END MISSING
    38
RDO> PRINT COUNT OF SH IN SALARY_HISTORY WITH
cont>  SH.SALARY_AMOUNT = 20000 AND
cont>  SH.SALARY_END MISSING
    1
RDO> PRINT COUNT OF SH IN SALARY_HISTORY
    101
```

Note that the total count of the first three queries adds up to 100, because you have stored one record where the field is missing.

Rdb/VMS does not use the missing value for comparisons when a field is missing. When there is no salary amount stored, SALARY_AMOUNT < 20000 is evaluated as *missing*, not as *true*, no matter what missing value you have defined for the SALARY_AMOUNT field.

Because Rdb/VMS uses this method of handling null fields, you should be careful when using relational operators. For example, consider the following query:

```
RDO> PRINT COUNT OF E IN EMPLOYEES WITH
cont> NOT (E.MIDDLE_INITIAL EQ "V")
    59
```

Conditional Expressions

This count does *not* include those records where the MIDDLE_INITIAL value is missing, because when the value is missing, Rdb/VMS does not include it in the comparison. The following queries show the remaining records:

```
RDO> PRINT COUNT OF E IN EMPLOYEES WITH
cont> E.MIDDLE_INITIAL EQ "V"
5
RDO> !
RDO> ! Print the number of employees with no middle
RDO> ! initial entered in the database.
RDO> !
RDO> PRINT COUNT OF E IN EMPLOYEES WITH
cont> "E.MIDDLE_INITIAL MISSING"
36
RDO> PRINT COUNT OF E IN EMPLOYEES
100
```

If you want to include in the record stream all the records where the middle initial is not equal to "V" or is missing, you must specify this in the RSE:

```
RDO> PRINT COUNT OF E IN EMPLOYEES WITH
cont> NOT (E.MIDDLE_INITIAL EQ "V") OR
cont> E.MIDDLE_INITIAL MISSING
95
```

Example 6

For performance reasons, you should not define a SORTED index until after the data has been loaded into the relation. If the index is intended to enforce unique values for particular fields during the load, then a query can be run from RDO after the database load to detect duplicates. The duplicates can then be removed, or the key values modified and the unique index defined. The following query lists all the non-unique values for the fields specified. The index can then be defined using REFERENCE_NUMBER and FACILITY_NUMBER with no duplicates allowed.

```
PRINT "Find Duplicate Id Numbers"
FOR P1 IN PAYMENT_DETAILS WITH
  NOT UNIQUE P2 IN PAYMENT_DETAILS WITH
    P1.REFERENCE_NUMBER = P2.REFERENCE_NUMBER AND
    P1.FACILITY_NUMBER = P2.FACILITY_NUMBER
  PRINT P1.REFERENCE_NUMBER, P1.FACILITY_NUMBER
END_FOR
```

3.2.2 Logical Operators

Conditional expressions can be linked together with logical operators. The result of such a combination is also a conditional expression. The Rdb/VMS logical operators are AND, OR, and NOT.

The following truth tables (Table 3–4, Table 3–5, and Table 3–6) show how Rdb/VMS evaluates conditional expressions linked by logical operators. For example, if conditional expression A is true and B is missing, then the conditional expression A AND B is evaluated as missing.

Table 3–4 Rdb/VMS Logical Operators—the AND Operator

A	B	A AND B
True	False	False
True	True	True
False	False	False
True	Missing	Missing
False	Missing	False
Missing	Missing	Missing

Table 3–5 Rdb/VMS Logical Operators—the OR Operator

A	B	A OR B
True	False	True
True	True	True
False	False	False
True	Missing	True
False	Missing	Missing
Missing	Missing	Missing

Conditional Expressions

Table 3-6 Rdb/VMS Logical Operators—the NOT Operator

A	NOT A
True	False
False	True
Missing	Missing

Note the interaction of conditional expressions when one is evaluated as *missing*. For example, consider the following query:

```
RDO> PRINT COUNT OF E IN EMPLOYEES WITH
cont> E.STATE = "MA" OR E.MIDDLE_INITIAL = "A"
```

For a given record, if STATE is “MA”, then the record is included in the count, whether the MIDDLE_INITIAL value is present or not.

Examples

Example 1

The following example uses the AND logical operator:

```

      ┌────────────────── record selection expression ───────────────────┐
FOR  E IN EMPLOYEES WITH
      ┌──────────┐ ┌──────────┐
      │ E.STATE EQ HOME │ AND │ E.BIRTHDAY LT LIMIT │
      └─── cond 1 ───┘ └─── cond 2 ───┘
      ┌──────────────────┐
      │                    │
      └─── cond 3 ───┘

```

ZK-0901A-GE

This example combines two conditional expressions using the AND operator. If, for a given record, *cond1* is true and *cond2* is true, that record becomes part of the record stream defined by the FOR loop. That is:

```

IF      the value in the STATE field of the EMPLOYEES relation equals the value
        of the host language variable HOME
AND     the value in the BIRTHDAY field is less than (earlier than) the limit specified
        by the value of the LIMIT variable,
THEN    Rdb/VMS includes the record in the record stream

```

Conditional Expressions

If, for a given record, there is no value in the STATE field (E.STATE MISSING is true), then *cond1* is evaluated as null. For that same record, if *cond2* is true, then *cond3* also evaluates to null. If *cond2* is false, then *cond3* is false.

The Record Selection Expression (RSE)

This chapter describes the record selection expression as used in Rdb/VMS. A **record selection expression** is a phrase that defines specific conditions that individual records must meet before Rdb/VMS includes them in a record stream.

A **record stream** is a temporary group of related records that satisfy the conditions you specify in the record selection expression. A record stream can consist of:

- All the records in one or more relations
- A subset of the records in one or more relations

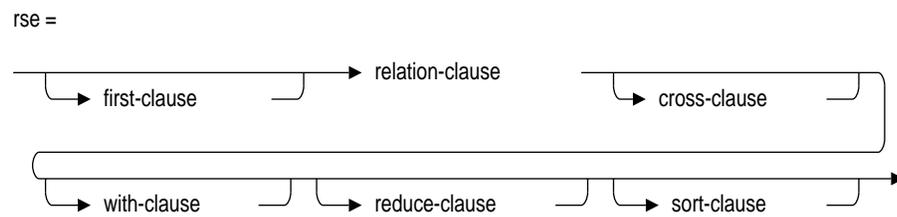
The following sections describe how to use the record selection expression to form record streams.

Format of the Record Selection Expression

4.1 Format of the Record Selection Expression

The following diagram represents the skeleton syntax of the Rdb/VMS record selection expression (RSE).

Format



Subsequent sections explain the RSE components.

The FIRST Clause: Restricting the Number of Records

4.2 The FIRST Clause: Restricting the Number of Records

The *first-clause* specifies how many records are in the record stream formed by the record selection expression. If you include the element FIRST *n*, the record stream has no more than *n* records. The argument *n* should be a positive integer or a value expression that evaluates to a positive integer. Note the following rules:

- Rdb/VMS does not guarantee the order of records in a record stream unless you specify a sort order. For this reason, when you use FIRST *n* in the RSE, the actual records in the record stream are unpredictable. Therefore, you should always use SORT when you use FIRST *n*.
- If *n* evaluates to zero or a negative number, the record stream will be empty.
- If you specify FIRST *n*, and *n* is greater than the number of records that satisfy those conditions, the record stream consists of all records that meet the conditions of the record selection expression.
- If *n* is not an integer, Rdb/VMS rounds any fractional part of the value and uses the resulting integer as the number of records in the record stream.

Format

first-clause =
 → FIRST → value-expr →

Usage Notes

Rdb/VMS syntax allows spaces in the specification of floating-point literals. As a result, it is possible to construct a query using the FIRST clause containing a context variable name that Rdb/VMS will interpret as a floating-point literal. For example, the following query includes a FIRST 10 clause followed by a legal context variable, E37; however, RDO interprets the expression (10 E37) as a floating-point literal.

The FIRST Clause: Restricting the Number of Records

```
FOR FIRST 10 E37 IN REL PRINT E37.* END_FOR
```

The same interpretation results in the preceding example if the context variable name is D37.

Thus, when you specify the FIRST clause, do not use a context variable name that starts with the letters D or E followed by a number. The D or E and number are interpreted to be part of the FIRST number; if the context variable name contains letters following the number (for example, E37ABC), the letters will be interpreted as the context variable (for example, ABC).

Examples

Example 1

You can print the first ten records in a relation:

```
RDO> FOR FIRST 10 E IN EMPLOYEES
cont>   SORTED BY E.LAST_NAME
cont> PRINT E.*
cont> END_FOR
```

The SORTED BY clause ensures that the first ten records are the first ten in an alphabetical list.

Example 2

You can use the FIRST and SORTED BY clauses to find the maximum values for a field. You could use the MAX statistical function to find the highest paid employee, but you can use the following statement to find the five highest paid employees:

```
RDO> FOR FIRST 5 C IN CURRENT_SALARY
cont>   SORTED BY DESCENDING C.SALARY_AMOUNT
cont> PRINT
cont>   C.FIRST_NAME,
cont>   C.LAST_NAME
cont> END_FOR
```

The FIRST Clause: Restricting the Number of Records

Example 3

The following example uses a calculated value expression in the FIRST clause to generate a random sample, consisting of 10 percent of the employees:

```
&RDB& GET TENTH = ( COUNT OF E IN EMPLOYEES / 10 )
&RDB& START_TRANSACTION READ_WRITE
&RDB& FOR FIRST TENTH E IN EMPLOYEES
        SORTED BY E. LAST_NAME
.
.
.
```

In this example, the value expression following the FIRST clause is calculated, using an Rdb/VMS statistical expression. If the value of TENTH is not an integer, Rdb/VMS rounds it to the nearest integer. Because 338 records are in the sample database, TENTH here equals 33.8, and the record stream consists of the first 34 records.

The Relation Clause: Context Variables in Streams and Loops

4.3 The Relation Clause: Context Variables in Streams and Loops

In the *relation clause*, you declare context variables for a stream or a loop. A **context variable** is a temporary name that identifies the record stream to Rdb/VMS.

Once you associate a context variable with a relation, you use the context variable to refer to fields from that relation. In this way, Rdb/VMS always knows which field you are referring to.

You must use a context variable in every data manipulation statement and in every data definition statement that uses an RSE.

If you access several record streams at once, the context variable lets you distinguish between fields from different record streams, even if different fields have the same name.

In all programs except RDBPRE, RDML, SQL\$PRE, and SQL\$MOD programs, context variables must be unique within a request.

Format

relation-clause =
→ context-var → IN ———→ relation-name →
 └─ db-handle ──┬─┘

Examples

Example 1

The following example erases a record:

```
RDO> FOR E IN EMPLOYEES WITH E.EMPLOYEE_ID = "00374"  
cont> ERASE E  
cont> END_FOR
```

This statement declares a context variable E, uses it to identify the field names in the EMPLOYEES relation, and then erases the record of the employee with the identification number "00374".

The Relation Clause: Context Variables in Streams and Loops

Example 2

The following example gives an employee a raise:

```
&RDB& FOR SH IN SALARY_HISTORY
&RDB&     WITH SH.EMPLOYEE_ID = ID-NUMBER
&RDB&     AND SH.SALARY_END MISSING
&RDB&     MODIFY SH USING
&RDB&         SH.SALARY_AMOUNT =
&RDB&         ( SH.SALARY_AMOUNT + ( 1000 * RATING ) )
&RDB&     END_MODIFY
&RDB& END_FOR
```

This statement:

- Declares a context variable SH, associated with the SALARY_HISTORY relation.
- Uses the context variable to qualify field names in the SALARY_HISTORY relation. This associates the field name SALARY_AMOUNT with the relation.
- Modifies the salary of the employee represented by the host language variable ID-NUMBER.

Example 3

The following example stores a value in the DEPARTMENTS relation:

```
STORE D IN DEPARTMENTS USING
    D.DEPARTMENT_NAME = "Recreation";
    D.DEPARTMENT_CODE = "RECR";
    D.MANAGER_ID = "00445"
END_STORE
```

Example 4

You can use the context variable to distinguish different fields that might have the same name.

```
&RDB& FOR E IN EMPLOYEES CROSS
&RDB&     J IN JOB_HISTORY WITH
&RDB&     E.EMPLOYEE_ID = J.EMPLOYEE_ID
```

See the *VAX Rdb/VMS Guide to Using RDO, RDBPRE, and RDML* for additional information on the use of context variables.

The WITH Clause: Specifying Conditions for the Record Stream

4.4 The WITH Clause: Specifying Conditions for the Record Stream

The *WITH clause* contains a conditional expression that allows you to specify conditions that must be true for a record to be included in a record stream. See Section 3.2 for a complete discussion of conditional expressions.

Format

with-clause =
 → WITH → conditional-expr →

Argument

conditional-expr

A record becomes part of a record stream only when the conditional expression is true. That is, its values must satisfy the conditions you specify in the conditional expression. If the conditional-expr evaluates to false or missing for a record, that record is not included in the record stream.

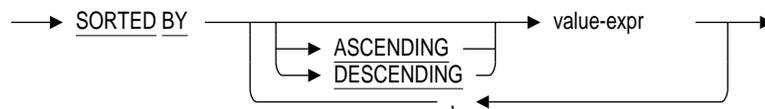
The SORTED BY Clause: Sorting Records

4.5 The SORTED BY Clause: Sorting Records

The *sort-clause*, or SORTED BY clause, of the record selection expression lets you sort the records in the record stream by the values of specific fields. You can sort the records according to a value expression called a sort key. The **sort key** determines the order in which Rdb/VMS returns the records in the record stream.

Format

sort-clause =



Argument

value-expr

A value expression specifying the sort value. This value is called the sort key.

Examples

Example 1

You can sort records by ID number:

```
FOR E IN EMPLOYEES SORTED BY E.EMPLOYEE_ID
```

Example 2

If you do not specify the sort order with the first sort key, the default order is ASCENDING:

```
FOR E IN EMPLOYEES SORTED BY E.BIRTHDAY
```

Because this example did not specify the sort order, Rdb/VMS automatically sorts the EMPLOYEES records in ASCENDING order by BIRTHDAY.

The SORTED BY Clause: Sorting Records

Example 3

You can sort by multiple keys. If you do not specify ASCENDING or DESCENDING for the second or subsequent sort keys, Rdb/VMS uses the order you specified for the preceding sort key:

```
FOR E IN EMPLOYEES
  SORTED BY DESCENDING E.STATUS_CODE,
            ASCENDING E.LAST_NAME, E.EMPLOYEE_ID
```

When you use multiple sort keys, Rdb/VMS treats the first field or value expression in the list of sort keys as the major sort key and successive field or value expressions as minor sort keys. That is, it first sorts the records into groups based on the first field or value expression. Then it uses the second field or value expression to sort the records within each group, and so on.

In the preceding example, Rdb/VMS sorts first by STATUS_CODE in descending order. Within each STATUS_CODE group, it sorts the records by LAST_NAME in ascending order. Finally, within groups of employees with the same last name, it sorts by EMPLOYEE_ID. The order for this last sort is also ascending because it adopts the order from the previous sort key.

If a field has a specific missing value defined for it, the missing value is included in the field values displayed. Missing values are assigned the highest value in all collating sequences. Records sorted by a field with a defined missing value appear first when the sort order is DESCENDING and last when the sort order is ASCENDING. For example, suppose the missing value for a field is -1. If the field is sorted in ASCENDING order, the records with fields that contain missing values appear at the end of the display. Therefore, you may see the following sorted list of values:

1,1,3,3,'missing','missing' as 1,1,3,3,-1,-1.

The REDUCED TO Clause: Retaining Only Unique Field Values

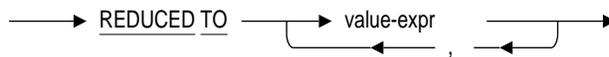
4.6 The REDUCED TO Clause: Retaining Only Unique Field Values

The *reduce-clause*, or the REDUCED TO clause, of the record selection expression lets you eliminate duplicate values for fields in a record stream. You can use this expression to eliminate redundancy in the results of a query and to group the records in a relation according to unique field values. Rdb/VMS does not guarantee the order of the resulting record stream. To ensure a specific order, use the SORTED BY clause.

If you use the REDUCED TO clause, you should retrieve only the fields that you specify in the list of value expressions. If you retrieve other fields, the results are unpredictable.

Format

reduce-clause =



Examples

Example 1

The following statement lists all the currently active job codes:

```
FOR J IN JOB_HISTORY
  WITH J.JOB_END MISSING
  REDUCED TO J.JOB_CODE
  SORTED BY J.JOB_CODE
PRINT J.JOB_CODE
END_FOR
```

The CROSS Clause: Joining Related Records

4.7 The CROSS Clause: Joining Related Records

The *cross-clause* of the record selection expression lets you combine records from two or more record streams. You can join these records in combinations based on the relationship between the values of fields in the separate record streams. This combining of records is called a **relational join**.

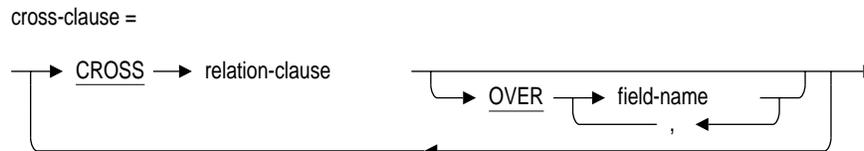
The CROSS expression with no additional qualifiers forms the cross product of relations. A **cross product** is the result of matching *each* record of one relation with *all* the records of the other relation. In most cases, the cross product is not useful. You will normally want to limit the returned records by using one of the following clauses of the record selection expression:

- first-clause (Section 4.2)
- with-clause (Section 4.4)
- sort-clause (Section 4.5)
- reduce-clause (Section 4.6)

See the *VAX Rdb/VMS Guide to Using RDO, RDBPRE, and RDML* for more information on joining relations.

Note You cannot cross relations from different databases. A record selection expression may reference only one database at a time. Instead, use a nested FOR loop to combine data from different databases.

Format



The CROSS Clause: Joining Related Records



A join operation can combine data from any number of relations. For example, you can use **CROSS** to join two relations over a common field. The result is a temporary *relation* that combines records from both. You can then join this composite relation with a third, using a field shared by the second and third relations. Using this process, you can relate data in any relation to data in any other relation (up to 32 relations), as long as you can find a chain of common fields. You can make this process efficient by defining index keys for the fields shared by the relations.

Examples

Example 1

You can join two relations with the **CROSS** clause.

Assume you must create a list of employees and their current salaries. The **EMPLOYEES** relation contains, among other things, a name and an employee identification number (**ID**) for each employee. The **SALARY_HISTORY** relation contains a **SALARY_AMOUNT** field for each salary level each employee has attained and the starting and ending date for each salary. For the current salary, the salary ending date is missing. These relations share the common field, **EMPLOYEE_ID**. Therefore, you can use the **CROSS** clause to join them over this field.

```
FOR E IN EMPLOYEES CROSS
  SH IN SALARY_HISTORY
  OVER EMPLOYEE_ID WITH
  SH.SALARY_END MISSING
PRINT E.LAST_NAME,
      SH.SALARY_AMOUNT
END_FOR
```

The result looks like a relation that lists the name of each employee and his or her current salary. Notice that the **LAST_NAME** field does not occur in the **SALARY_HISTORY** relation and the **SALARY_AMOUNT** field does not occur in the **EMPLOYEES** relation; the join relates the two fields.

The CROSS Clause: Joining Related Records

For more information about the concepts discussed in this section, see the:

- *VAX Rdb/VMS Guide to Using RDO, RDBPRE, and RDML* on using the CROSS expression
- *VAX Rdb/VMS Guide to Database Design and Definition* on the use of index keys

Views and the Record Selection Expression

4.8 Views and the Record Selection Expression

The record selection expression is an integral part of view definition. A **view** is a logical relation that lets you selectively use any field in any record from one or more existing relations. A view allows you to give a name to a selected set of fields from a relation or to a combination of fields from different relations. Thus, you can treat those combinations as you would a relation.

The following statement defines a view of data from the EMPLOYEES and JOB_HISTORY relations:

```
DEFINE VIEW CURRENT_JOB OF JH IN JOB_HISTORY
  CROSS E IN EMPLOYEES OVER EMPLOYEE_ID
  WITH JH.JOB_END MISSING.
  E.LAST_NAME.
  E.FIRST_NAME.
  E.EMPLOYEE_ID.
  JH.JOB_CODE.
  JH.DEPARTMENT_CODE.
  JH.SUPERVISOR_ID.
  JH.JOB_START.
  JH.JOB_END.
END VIEW.
```

You can use views without restriction as long as you are only reading the records in the relation. You can change or store data using a view as long as the view refers to only one relation. You cannot update data using a view that joins two or more relations.

See the *VAX Rdb/VMS Guide to Using RDO, RDBPRE, and RDML* for further information about the use of the record selection expression and views.

Field Attributes in Relations and Views

Field attributes define the characteristics of the data you store in relations. You can define the following field attributes:

- Name
- Description
- Data type
- Size
- Missing value
- Validity criteria for checking values on input
- Characteristics for DATATRIEVE access
 - DEFAULT VALUE
 - EDIT STRING
 - QUERY NAME
 - QUERY HEADER
- Collating sequence

There are two types of field attributes in an Rdb/VMS database:

- Global attributes

When you define a field globally, you supply a name and a data type. Optionally, you can supply a MISSING_VALUE clause, a VALID IF clause, DATATRIEVE support clauses, and a COLLATING_SEQUENCE clause. You can define global field attributes using a DEFINE FIELD statement or a DEFINE RELATION statement. Either way, Rdb/VMS enters the name and definition in the relation of global field definitions for the entire


```

DEFINE RELATION WEEKLY.
  LAST_NAME. <----- ❶
  EMPLOYEE_ID BASED ON ID_NUMBER <----- ❷
    QUERY_NAME FOR DATATRIEVE IS "ID".
  SALARY_AMOUNT BASED ON MONEY
    QUERY_NAME FOR DATATRIEVE IS "ANNUAL"
    QUERY_HEADER FOR DATATRIEVE
      IS "ANNUAL"/"SALARY". <----- ❸
  WEEKLY_SALARY
    COMPUTED BY ( SALARY_AMOUNT / 52 ) <----- ❹
    QUERY_NAME FOR DATATRIEVE IS "WEEKLY"
    QUERY_HEADER FOR DATATRIEVE
      IS "WEEKLY"/"SALARY".
  MONEY BASED ON BIGMONEY. <----- ❺
  COMMENT_FIELD DATATYPE IS TEXT SIZE 25. <----- ❻
END WEEKLY RELATION.

```

In the preceding example, the DEFINE FIELD statements define four fields with global attributes. You can now use any of these fields in relation definitions simply by naming them:

- ❶ The relation includes the field LAST_NAME by naming it.
- ❷ The relation includes the field with the local name EMPLOYEE_ID. This field derives its global attributes from the ID_NUMBER field. It also adds another local attribute, a DATATRIEVE query name.
- ❸ The relation includes the field with the local name SALARY_AMOUNT, which derives its global attributes from the MONEY field and adds two DATATRIEVE support clauses. These two clauses override the DATATRIEVE clauses defined for the field, MONEY.
- ❹ The WEEKLY_SALARY field is computed from another field in the relation. This definition includes two DATATRIEVE support clauses.
- ❺ Here, MONEY is a local field name. This MONEY field derives its definition not from the global MONEY field, but from BIGMONEY. You can have a locally defined name that is identical to a globally defined name.
- ❻ COMMENT_FIELD includes a data type specification, which is a global attribute. Therefore, COMMENT_FIELD is now entered in the global list of field definitions for the database, and you can use that definition by naming it in other relation definitions.

You can define a global field using the `DEFINE FIELD` statement or the `DEFINE RELATION` statement. In either case, you define the characteristics of the field with a set of *field attributes*. This chapter provides the syntax of field attributes.

Note *You cannot define a global field using the `DEFINE` clause of the `CHANGE RELATION` statement.*

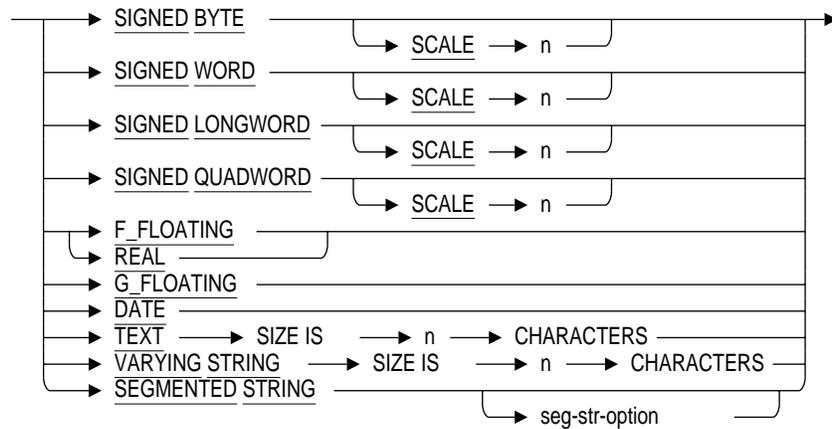
Only two parts are required in a field definition: a field name and a data type.

5.1 Data Type Clause

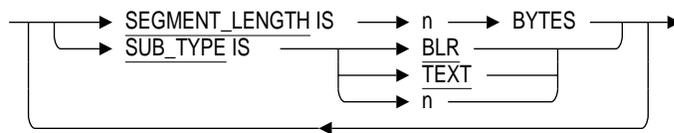
The data type clause describes the internal format of the data stored in the field. Rdb/VMS supports a subset of the standard VAX data types. It also supports an extra data type called *segmented string*. Table 5-1 summarizes the characteristics of the Rdb/VMS data types.

Format

data-type =



seg-str-option =



Data Type Clause

Arguments

SCALE *n*

A positive or negative integer that specifies the scaling factor for WORD, LONGWORD, and QUADWORD. For example, if you have a scale of -2 , the value in the affected field will have 2 places to the right of the decimal point.

REAL

Specifies that the field is a 32-bit floating-point number with precision to approximately seven decimal digits. REAL is used in VAX BASIC as an optional alternative to the floating-point data types.

DATE

Specifies that the field is a 64-bit VAX standard absolute date and time data type. For information on the compile-time translation of the YESTERDAY, TODAY, and TOMORROW character string literals, see Section 3.1.2.1. For information on the format of the DATE data type, see Section 3.1.2.3.

SIZE IS *n*

A positive integer that specifies:

- The number of characters for TEXT
- The maximum number of characters for VARYING STRING

SEGMENT_LENGTH

The length of each segment of a segmented string, in bytes. SEGMENT_LENGTH is a descriptive field that allows you to determine the proper buffer size when retrieving segmented strings within a program. You can store or retrieve segments that are greater than SEGMENT_LENGTH.

Data Type Clause

SUB_TYPE

The type of segmented string:

- BLR A segmented string that contains binary language representation (BLR). The database system uses BLR internally.
- TEXT A segmented string whose segments contain ASCII text. This is the default.
- n A user-defined subtype. You can assign a number to a subtype such as graphics or voice data. Your program can then read the subtype to determine how to process the segments of the segmented string field. If *n* is zero, the subtype is undefined.

Table 5-1 Rdb/VMS Data Types

VAX Rdb/VMS Data Type	Corresponding VAX Data Type	Size	Range/Precision	Other Parameters
SIGNED BYTE	Signed byte integer	8 bits	-128 to 127	n = scale factor
SIGNED WORD	Signed word integer	16 bits	-32768 to 32767	n = scale factor
SIGNED LONGWORD	Signed longword integer	32 bits	-2**31 to (2**31) -1	n = scale factor
SIGNED QUADWORD	Signed quadword integer	64 bits	-2**63 to (2**63) -1	n = scale factor
F_FLOATING	F_floating Single precision floating-point number	32 bits	0.29 x 10**(-38) to 1.7 x 10**38 Approximately 7 decimal digits	None
G_FLOATING	G_floating Extended precision floating-point number	64 bits	0.56 x 10**(-308) to 0.9 x 10**308 Approximately 15 decimal digits	None
DATE	Absolute date and time	64 bits	Not applicable	None

(continued on next page)

Data Type Clause

Table 5-1 (Cont.) Rdb/VMS Data Types

VAX Rdb/VMS Data Type	Corresponding VAX Data Type	Size	Range/Precision	Other Parameters
TEXT	ASCII text	n bytes	0 to 16,383 characters	n = number of characters (unsigned integer)
VARYING STRING	Varying length ASCII text	Varies	0 to 32,767 characters	n = maximum number of characters (unsigned integer)
SEGMENTED STRING	None	Varies	0 to 64K bytes per segment	None

The segmented string is a special Rdb/VMS data type designed to handle large pieces of data with a segmented internal structure. The maximum size of a string segment is 64K bytes. Except for the length of the string's segments, Rdb/VMS does not know anything about the type of data contained in a segmented string. In a segmented string, you can store large amounts of text, long strings of binary input from a data collecting device, or graphics data. A program can then extract the data from the database and handle it in the appropriate way.

Figure 5-1 shows a conceptual diagram of a record from a RESUMES relation that contains a segmented string. The physical record actually contains a pointer to the segments of the segmented string.

Data Type Clause

Figure 5-1 A Record with a Segmented String Field

EMPLOYEE ID	RESUME
00329	
	RICHARD D. BALLINGER
	92 Pistol Lane Born September 10, 1949
	Manchester, NH 03104 Health: Excellent
	Telephone: (603) 555-8899
	OBJECTIVES:
	I am seeking a position that combines
	technical and administrative duties,
	especially one that involves a long-term

ZK-7548-GE

Because Rdb/VMS does not know what kind of data is contained in a segmented string, you cannot perform many of the standard data manipulation functions on it. You cannot use logical operators, such as EQ and CONTAINING, to compare segmented strings. Rdb/VMS does not perform any data type conversion on data that is transferred into or out of a segmented string. You can use the MISSING operator on a segmented string field.

Rdb/VMS defines a special name to refer to the segments of a segmented string. This name is equivalent to a field name; it names the *fields* or segments of the string. Furthermore, because segments can vary in length, Rdb/VMS also defines a name for the length of a segment. You must use these names in the value expressions that you use to retrieve the length and value of a segment. These names are:

RDB\$VALUE The value stored in a segment of a segmented string
RDB\$LENGTH The length in bytes of a segment

Data Type Clause

Because a single segmented string field value is made up of multiple segments, you must manipulate the segments one at a time. Therefore, segmented string operations require an internal looping mechanism, much like the record stream set up by a FOR or START_STREAM statement.

For more information on segmented strings, see the descriptions of the following statements in Chapter 9:

- CREATE_SEGMENTED_STRING
- END_SEGMENTED_STRING
- START_SEGMENTED_STRING
- FOR with Segmented Strings
- STORE with Segmented Strings

5.2 Validity Clause

The validity clause specifies a validation check for a field. Rdb/VMS evaluates the conditional expression at run time when the field is stored or modified. Rdb/VMS does not allow you to store a value defined as invalid by the validity clause.

Format

validity-clause =
 → VALID IF → conditional-expr →

Argument

conditional-expr

A valid conditional expression. See Chapter 3 for a complete description of Rdb/VMS conditional expressions. The conditional expression used in a VALID IF clause must conform to the following rules:

- If the conditional expression compares the field value to another value, the other value must have the same data type as the field being defined.
- You can mention only the field name being defined in a VALID IF clause. This means that you cannot use value expressions that refer to other fields, such as statistical expressions, ANY, UNIQUE, or FIRST FROM.
- If you define a MISSING_VALUE for a field, you should *always* include MISSING in the VALID IF clause.

Examples

Example 1

You can use a VALID IF clause to establish a range of numeric values:

```
DEFINE FIELD SALARY_AMOUNT  
  DATATYPE IS SIGNED LONGWORD SCALE -2  
  VALID IF SALARY_AMOUNT BETWEEN 0 AND 200000.
```

Validity Clause

This statement ensures that no salary amount that is greater than \$200,000 or less than zero can be entered in the database.

Example 2

You can also use the `VALID IF` clause to establish a set of possible text values:

```
DEFINE FIELD SEX
  DATATYPE IS TEXT SIZE 1
  VALID IF SEX = "M" OR
          SEX = "F".
```

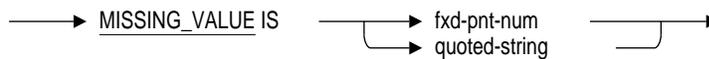
5.3 Missing Value Clause

The missing value clause specifies a numeric or character string literal denoting that no value is stored in the field. When you retrieve or display the field, Rdb/VMS returns the contents of the `MISSING_VALUE` string or number. When you store the value specified in the `MISSING_VALUE` clause, Rdb/VMS does not store a value in the field; rather, it marks the field as missing. Section 3.1.6 explains the concept of missing field values in detail.

When you specify a missing value for a field, the value should lie outside the set of possible values for the field. For example, zero is not a good missing value for fields that may actually have a value of zero, while `-1` may be a good missing value to use for a signed longword field that must be a positive integer. For text fields, a missing value such as “Not available” is often appropriate. The missing value you specify should be of the same data type as the field.

Format

missing-value-clause =



Arguments

fxd-pnt-num

A fixed-point number that Rdb/VMS uses to replace retrieved null values. A fixed-point number always includes a decimal point, even when there are no digits to the right of the decimal point.

For example, if a field contains dollar and cent values, you may want to use `0.00` or `.00` as the missing value to preserve the value context when Rdb/VMS retrieves and displays null values. In this case, if you do not allow null values and you do not specify a missing value, Rdb/VMS fills the retrieved field with zeros.

Specify a number that does not exceed the defined field length, and include a decimal point.

Missing Value Clause

quoted-string

A character string that Rdb/VMS uses to replace retrieved null values. Specify a character string of any length and enclose the string in double or single quotation marks. See Chapter 3 for a complete description of the use of literal expressions.

Examples

Example 1

The following example defines a missing value for a date field:

```
DEFINE FIELD STANDARD_DATE
    DESCRIPTION IS /* Universal date field */
    DATATYPE IS DATE
    MISSING_VALUE IS "17-NOV-1858 00:00:00.00"
    EDIT_STRING FOR DATATRIEVE IS 'DD-MMM-YYYY'.
```

This definition defines a missing value for date fields to be the same as the VMS base date and time.

Example 2

The following example defines a missing value for a numeric field:

```
DEFINE FIELD SALARY
    DATATYPE IS SIGNED LONGWORD SCALE -2
    MISSING_VALUE IS 0.00
    EDIT_STRING FOR DATATRIEVE IS '$$$$, $$9.99'.
```

This statement specifies 0.00 as the missing value for SALARY fields.

Example 3

The following example defines a missing value for a text field:

```
DEFINE FIELD ADDRESS_DATA
    DATATYPE IS TEXT SIZE 20
    MISSING_VALUE IS "Not available".
```

When you retrieve a field whose value is missing, Rdb/VMS displays the string "Not available".

Missing Value Clause

Example 4

The STORE statement lets you leave a field empty even if the VALID IF clause associated with the field requires that the value in the field fall within a range. This can result in an error at define time:

```
RDO> DEFINE FIELD ID DATATYPE TEXT SIZE 5.
%RDO-W-NOCDUPDAT, database invoked by filename, the CDD will not be updated
RDO>
RDO> DEFINE FIELD STAT DATATYPE SIGNED WORD
cont>     MISSING_VALUE IS -1
cont>     VALID IF STAT >= 0.
RDO>
RDO> DEFINE RELATION PEOPLE.
cont>   ID.
cont>   STAT.
cont> END.
RDO>
RDO> STORE P IN PEOPLE USING
cont>   P.ID = "00001";
cont> END_STORE
%RDB-E-NOT_VALID, validation on field STAT caused operation to fail
-RDMS-F-FIELD_MISSING, field STAT is missing from this statement
```

Because the STORE statement does not mention the STAT field, STAT is missing, and the conditional expression in the VALID IF clause (STAT>=0) is violated.

You can avoid this error by defining the field as follows:

```
DEFINE FIELD STAT DATATYPE SIGNED WORD
      MISSING_VALUE IS -1
      VALID IF STAT >= 0
           OR STAT MISSING.
```

DATATRIEVE Support Clauses

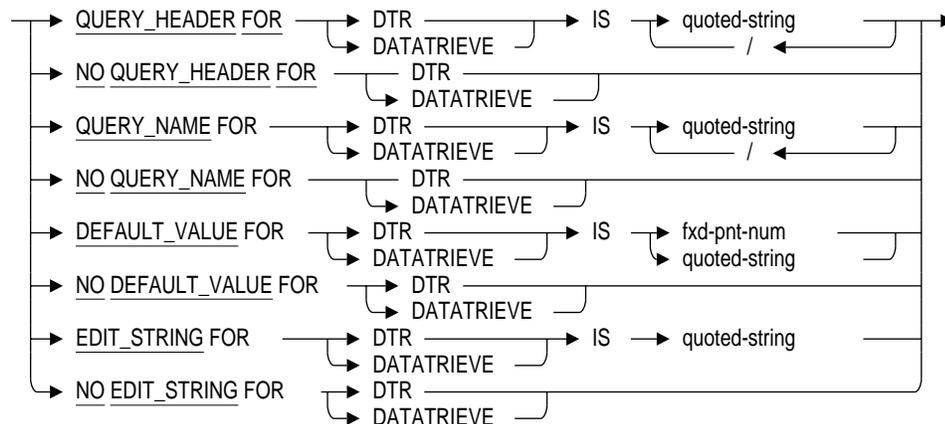
5.4 DATATRIEVE Support Clauses

Rdb/VMS provides support for DATATRIEVE, a query language that runs on the VMS operating system. In the field definition, you can specify a `DEFAULT_VALUE` clause and an `EDIT_STRING` clause for the field. These clauses are active only when DATATRIEVE accesses the field values.

The data type of a value specified in the `DEFAULT_VALUE` clause must be of the same data type as the field to which it is attached.

Format

dtr-clause =



Arguments

quoted-string

A literal expression that Rdb/VMS returns to DATATRIEVE.

Specify a character string of any length and enclose the string in double ("string") or single ('string') quotation marks. See the *VAX DATATRIEVE Reference Manual* for information about using edit strings, query names, and query headers with DATATRIEVE.

DATATRIEVE Support Clauses

Note *Rdb/VMS does not perform any validation testing on a DATATRIEVE edit string before storing the field definition. If you enter an invalid edit string, the DEFINE FIELD statement may execute successfully, but you may get an error when you try to access the field through DATATRIEVE.*

If the database is invoked by path name, CDD/Plus performs validation testing on the DATATRIEVE edit string before storing the field definition.

fxd-pnt-num

The default value returned to DATATRIEVE when Rdb/VMS retrieves null values. A fixed-point number is a number that always includes a decimal point, even when there are no digits to the right of the decimal point.

Specify a number that does not exceed the defined field length and include a decimal point.

If you specify a default value for a field and do not specify that field in a DATATRIEVE STORE or MODIFY statement, DATATRIEVE instructs Rdb/VMS to store the default value in that field. See the *VAX DATATRIEVE Reference Manual* for information about using DATATRIEVE default values.

NO EDIT_STRING

NO QUERY_NAME

NO QUERY_HEADER

NO DEFAULT_VALUE

These clauses are used in the CHANGE FIELD and CHANGE RELATION statements only. They cancel the effects of the other clauses.

Examples

Example 1

The following example defines a field and specifies a DATATRIEVE edit string for it:

```
DEFINE FIELD MIDDLE_INITIAL
  DATATYPE IS TEXT SIZE 1
  EDIT_STRING FOR DATATRIEVE IS 'X.'
```

The X represents any alphanumeric character. When you access this field through DATATRIEVE, the field value is followed by a period. The DATATRIEVE documentation set contains a complete list of editing characters.

DATATRIEVE Support Clauses

Example 2

The following example defines a DATATRIEVE query name for one field and a DATATRIEVE query header for another:

```
DEFINE FIELD STATE
    DATATYPE IS TEXT SIZE 2
    QUERY_NAME FOR DATATRIEVE IS "ST".
DEFINE FIELD SEX
    DATATYPE IS TEXT SIZE 1
    VALID IF SEX = 'M' OR SEX = 'F'
    QUERY_HEADER FOR DATATRIEVE IS "S"/"E"/"X".
```

These statements define DATATRIEVE query names and query headers for the STATE and SEX fields. The header for the SEX field will be one character wide, like the field itself.

Example 3

The following example shows how to use a local definition to override the global DATATRIEVE characteristics of a field:

```
CHANGE RELATION EMP_SUMMARY.
    CHANGE SEX NO QUERY_HEADER FOR DATATRIEVE.
END.
```

This statement instructs DATATRIEVE not to use the globally defined query header for the SEX field when it appears in the EMP_SUMMARY relation. DATATRIEVE uses the field name itself as the header when it displays fields from the EMP_SUMMARY relation.

5.5 Collating Sequence Clause

The `COLLATING_SEQUENCE` clause specifies a collating sequence for the named field.

The VMS NCS utility provides a set of predefined collating sequences and also lets you define collating sequences of your own. The RDO `COLLATING_SEQUENCE` clause accepts both predefined and user-defined NCS collating sequences.

Format

→ `COLLATING_SEQUENCE IS sequence-name` →
→ `NO COLLATING_SEQUENCE` →

Arguments

COLLATING_SEQUENCE IS sequence-name

Specifies the collating sequence to be used in the field you are defining. Before you use the `COLLATING_SEQUENCE` clause in a `DEFINE FIELD` statement, you must first specify the NCS collating sequence using the `DEFINE COLLATING_SEQUENCE` statement. The `sequence-name` argument in the `COLLATING_SEQUENCE` clause must be the same as the `sequence-name` in the `DEFINE COLLATING_SEQUENCE` statement.

NO COLLATING_SEQUENCE

The `NO COLLATING_SEQUENCE` clause specifies that the field will use the standard default collating sequence: that is, ASCII. Use the `NO COLLATING_SEQUENCE` clause to override the collating sequence defined for the database in the `DEFINE DATABASE` statement.

Collating Sequence Clause

Examples

Example 1

The following example defines a field with the predefined NCS collating sequence SPANISH. Note that you must first execute the DEFINE COLLATING_SEQUENCE statement.

```
RDO> INVOKE DATABASE FILENAME 'MF_PERSONNEL'
RDO> DEFINE COLLATING_SEQUENCE SPANISH SPANISH.
%RDO-W-NOCDDUPDAT, database invoked by filename, the CDD will not be updated
RDO> DEFINE FIELD LAST_NAME_SPANISH
cont> DATATYPE IS TEXT SIZE IS 12 CHARACTERS
cont> COLLATING_SEQUENCE IS SPANISH.
RDO> SHOW FIELD LAST_NAME_SPANISH
      LAST_NAME_SPANISH          text size is 12
                                   Collating Sequence SPANISH
```

6

RMU Command Syntax

The Rdb/VMS Management Utility (RMU) lets database administrators display useful information about Rdb/VMS databases. RMU commands are executed at the VMS system prompt. RMU command syntax follows the rules and conventions of the DIGITAL Command Language (DCL). This chapter provides reference information for the following RMU commands:

- RMU/ALTER
See also Chapter 7.
- RMU/ANALYZE
- RMU/ANALYZE/CARDINALITY
- RMU/ANALYZE/INDEXES
- RMU/ANALYZE/PLACEMENT
- RMU/BACKUP
- RMU/BACKUP/AFTER_JOURNAL
- RMU/CLOSE
- RMU/CONVERT
- RMU/COPY_DATABASE
- RMU/DUMP
- RMU/DUMP/AFTER_JOURNAL
- RMU/DUMP/BACKUP_FILE
- RMU/DUMP/RECOVERY_JOURNAL
- RMU/LOAD
- RMU/MONITOR REOPEN_LOG

- RMU/MONITOR START
- RMU/MONITOR STOP
- RMU/MOVE_AREA
- RMU/OPEN
- RMU/RECOVER
- RMU/RECOVER/RESOLVE
- RMU/RESOLVE
- RMU/RESTORE
- RMU/SET AUDIT
- RMU/SHOW
- RMU/UNLOAD
- RMU/VERIFY

RMU commands display the contents of database files, control the Rdb/VMS monitor process, verify data structures, and list information about current database users and database activity statistics.

RMU commands consist of words, generally verbs, that have parameters and qualifiers to define the action to be performed.

6.1 Command Parameters

One or more spaces separate command parameters and their qualifiers from the command keyword. Command parameters define the file, index, or entity on which the command will act. In most cases, you can omit the parameter from the command line and enter it in response to a prompt.

In the following sample command, RMU/DUMP is the command keyword and MF_PERSONNEL is the command parameter:

```
$ RMU/DUMP MF_PERSONNEL
```

When a storage area is a command parameter in an RMU command, use the *storage area name* instead of the *storage area file specification*.

6.2 Qualifiers

Qualifiers always begin with a slash (/) followed by a qualifier word. In some cases, an equals sign (=) and a qualifier argument follow the qualifier word. A qualifier argument can be simple (a value or a keyword) or compound (a string of values or keywords separated by commas, enclosed in parentheses).

Qualifiers

A default value for a qualifier indicates what qualifier will be used if you omit the qualifier completely. Omitting a qualifier is not the same thing as specifying a qualifier with a default *argument*.

Command qualifiers influence the overall action of a command. Command qualifiers must be placed following the command keyword but before any parameters.

In the following example, the command qualifier, /USERS, immediately follows the RMU/DUMP keyword and precedes the command parameter, MF_PERSONNEL:

```
$ RMU/DUMP/USERS MF_PERSONNEL
```

Note *RMU parameter qualifiers have positional semantics, and when placed randomly may be ignored or produce unexpected results.*

Parameter qualifiers (also referred to as file qualifiers or area qualifiers) affect the treatment of parameters in the command. If the command includes multiple instances of a given type of parameter, the placement of parameter qualifiers affects their scope of influence as follows:

- If you position the parameter qualifier after a particular parameter, the qualifier affects only that parameter. This is **local** use of a parameter qualifier.
- If you position the parameter qualifier before the first parameter, the qualifier applies to all instances of the parameter. This is **global** use of a parameter qualifier. Not all parameter qualifiers can be used globally. To identify such qualifiers, read the description of the qualifier.
- Local parameter qualifiers take precedence over global parameter qualifiers, in most cases.

The following example demonstrates the local use of the area qualifier, /THRESHOLDS, to change the threshold settings for the AREA1 area:

```
$ RMU/RESTORE MF_PERSONNEL AREA1/THRESHOLDS=(70,80,90)
```

Note *See the VAX Rdb/VMS Guide to Database Maintenance and Performance for tutorial information on how to use these RMU commands and interpret their output.*

Note that if you specify an RMU qualifier in both the negated and positive forms, the last occurrence of the qualifier is the one that takes effect. For example, the /NOLOG qualifier takes effect in this command:

```
RMU/BACKUP/LOG/NOLOG MF_PERSONNEL MF_PERS
```

This is consistent with DCL behavior for negated and positive qualifiers.

RMU/ALTER Command

6.3 RMU/ALTER Command

The RMU/ALTER command invokes the RdbALTER utility for VAX Rdb/VMS. The RdbALTER utility provides a low-level patch capability that allows you to repair corruption on VAX Rdb/VMS database pages. In addition, it allows you to relocate database root, area, and snapshot files to other disks or directories. This is helpful when you move a database from a single node to a VAXcluster environment.

To invoke the RdbALTER utility, type the following:

```
$ RMU/ALTER [root-file-spec]
```

The optional root file parameter identifies the database you wish to alter. If you specify this parameter, you automatically attach to the specified database. If you do not specify this parameter, you must use the RdbALTER ATTACH command. See Section 7.2 for more information on the ATTACH command.

The RMU/ALTER command responds with the following prompt:

```
RdbALTER>
```

This prompt means that the system expects RdbALTER command input. Complete information on the RdbALTER utility is found in Chapter 7, which provides a syntax diagram and description for each RdbALTER command.

You must have the VMS SYSPRV privilege to use RdbALTER.

6.4 RMU/ANALYZE Command

Gathers and displays statistics on how the database uses space.

Format

RMU/ANALYZE database-file-name

Command Qualifiers

/AREAS [= storage-area-list]
 /[NO]BINARY_OUTPUT [= file-option-list]
 /[NO]LAREAS [= logical-area-list]
 /OUTPUT=file-name
 /OPTION = {NORMAL | FULL | DEBUG}
 /START = integer
 /END = integer

Defaults

/AREAS
 /NOBINARY_OUTPUT
 /LAREAS
 /OUTPUT = SYSS\$OUTPUT
 /OPTION = NORMAL
 /START = first-page
 /END = last-page

Description

The RMU/ANALYZE command provides a maintenance tool for database administrators. It generates a formatted display of statistical information that describes storage utilization in the database. Information is displayed selectively for storage areas and logical areas, or for a range of pages in a storage area. You can use the RMU/ANALYZE command to analyze the following:

- Space utilization for database pages
- Space utilization for storage areas
- Space utilization for logical areas

Command Parameter

database-file-name

The name of the database to be analyzed. The default file type is RDB.

RMU/ANALYZE Command

Command Qualifiers

/AREAS [= storage-area-list]

The storage areas to be analyzed. The default, /AREAS, results in analysis of all storage areas. You can also specify /AREAS=* to analyze all storage areas. If you specify more than one storage area name, separate the storage area names in the storage-area-list with a comma and enclose the list in parentheses.

You can use the /START and /END qualifiers with the /AREAS qualifier to analyze specific pages. If you use the /START and /END qualifiers when you name more than one storage area in the storage-area-list, the same specified range of pages will be analyzed in each named storage area.

/BINARY_OUTPUT [= file-option-list]

/NOBINARY_OUTPUT

Specifying the /BINARY_OUTPUT qualifier allows you to output the summary results to a binary file, and to create a record definition file that is compatible with CDD/Plus for the binary output file. The binary output can be loaded into an Rdb/VMS database with the RMU/LOAD/RMS_RECORD_DEF command for use by a user-written management application or procedure. The binary output can also be used directly by the user-written application or procedure.

The valid file options are:

- **FILE=file-spec**

The FILE option causes the ANALYZE data to be output to an RMS file that contains a fixed-length binary record for each storage area and logical area analyzed. The default file type for the binary output file is UNL. The following command creates the binary output file ANALYZE_OUT.UNL:

```
$ RMU/ANALYZE/BINARY_OUTPUT=FILE=ANALYZE_OUT MF_PERSONNEL.RDB
```

- **RECORD_DEFINITION=file-spec**

The RECORD_DEFINITION option causes the ANALYZE data record definition to be output to an RMS file. The output file contains the definition in a subset of the CDD/Plus CDO command format, a format very similar to RDO field and relation definitions. The default file type for the record definition output file is RRD. The following command creates the output file ANALYZE_OUT.RRD:

```
$ RMU/ANALYZE/BINARY_OUTPUT=RECORD_DEFINITION=ANALYZE_OUT -  
_ $ MF_PERSONNEL.RDB
```

RMU/ANALYZE Command

You can specify both file options in one command by separating them with a comma and enclosing them within parentheses:

```
$ RMU/ANALYZE/BINARY_OUTPUT= -  
_$(FILE=ANALYZE_OUT,RECORD_DEFINITION=ANALYZE_OUT) -  
_$(MF_PERSONNEL.RDB
```

The default is the /NOBINARY_OUTPUT qualifier, which does not create an output file.

/LAREAS [= logical-area-list] ***/NOLAREAS***

The logical areas to be analyzed. Each table in the database is associated with a logical area name. The default, /LAREAS, results in analysis of all logical areas. You can also specify /LAREAS=* to analyze all logical areas. If you specify more than one logical area name, separate the logical area names in the logical-area-list with a comma and enclose the list in parentheses.

/OUTPUT=file-name

The name of the file where output will be sent. The default file type is LIS. If you do not specify the /OUTPUT qualifier, the default output is SYSS\$OUTPUT.

/OPTION=type

The type of information and level of detail the analysis will include. Three types of output are available:

- NORMAL
Output includes only summary information. NORMAL is the default.
- FULL
Output includes histograms and summary information.
- DEBUG
Output includes internal information about the data, as well as histograms and summary information. In general, use DEBUG for diagnostic support purposes. You can also use the DEBUG option to extract data and perform an independent analysis.

/START=integer

The starting page number for the analysis. The default is page number one.

/END=integer

The ending page number for the analysis. The default is the end of the storage area file.

RMU/ANALYZE Command

Usage Notes

You must have the Rdb/VMS ADMINISTRATOR (SQL DBADM) privilege to use the RMU/ANALYZE command.

Examples

Example 1

The following command analyzes the EMPIDS_LOW and EMP_INFO storage areas in the sample personnel database:

```
$ RMU/ANALYZE/AREAS=(EMPIDS_LOW,EMP_INFO)/OUTPUT=EMP.OUT MF_PERSONNEL.RDB
```

RMU/ANALYZE/CARDINALITY Command

6.5 RMU/ANALYZE/CARDINALITY Command

Generates a formatted display of the actual and stored cardinality values for tables and indexes. Also, if the stored cardinality values are different than the actual cardinality values, the RMU/ANALYZE/CARDINALITY command permits you the option of updating the stored cardinality values.

The actual cardinality values for tables and indexes can be different from the stored cardinality values in your database's RDB\$SYSTEM storage area if RDB\$SYSTEM has been set to read-only. When rows are added to or deleted from tables and indexes after the RDB\$SYSTEM storage area has been set to read-only, the cardinality values for these tables and indexes are not updated.

For indexes, the cardinality value is the number of unique entries for an index that allows duplicates. For tables, the cardinality value is the number of rows in the table. Rdb/VMS uses the cardinality values of indexes and tables to influence decisions made by the optimizer. If the actual cardinality values of tables and indexes are different than the stored cardinality values, this can adversely affect the optimizer's performance.

Format

```
RMU/ANALYZE/CARDINALITY database-file-name  
[table-or-index-name[,...]]
```

Command Qualifiers

```
/[NO]CONFIRM  
/[NO]UPDATE  
/OUTPUT = file-name
```

Defaults

```
/NOCONFIRM  
/NOUPDATE  
/OUTPUT = SYS$OUTPUT
```

Description

When you use the RDO CHANGE DATABASE statement to make the RDB\$SYSTEM storage area read-only for your database, the Rdb/VMS system

RMU/ANALYZE/CARDINALITY Command

tables in the RDB\$SYSTEM storage area become read-only. When the Rdb/VMS system tables are read-only:

- Automatic updates to table and index cardinality are disabled
- Manual changes made to the cardinalities to influence the optimizer are stabilized
- The I/O associated with the cardinality update is eliminated

With the RMU/ANALYZE/CARDINALITY command, you can:

- Display the stored and actual cardinality values for the specified tables and indexes.
- Update the stored cardinality value for a specified table or index with either the actual value or an alternate values of your own choosing. Digital Equipment Corporation recommends that you update the stored cardinality value with the actual cardinality value. Specifying a value other than the actual cardinality value can result in poor database performance.

Command Parameters

database-file-name

The name of the database for which you want information. The default file type is RDB. This parameter is required.

table-or-index-name

The name of the index or table whose cardinality you are interested in. The default is all defined indexes and tables. This parameter is optional.

Command Qualifiers

/CONFIRM

/NOCONFIRM

Specify the ***/CONFIRM*** qualifier when the ***/UPDATE*** qualifier is selected to gain more control over the update function. When you specify ***/CONFIRM***, you are asked whether the update should be performed for each selected table or index whose stored cardinality value is different than its actual cardinality value. You can respond with “YES”, “NO”, “QUIT”, or an alternate value for the stored cardinality. Specifying “YES” means that you want to update the stored cardinality with the actual cardinality value. Specifying “NO” means that you do not want to update the stored cardinality value. Specifying “QUIT”

RMU/ANALYZE/CARDINALITY Command

aborts the RMU/ANALYZE/CARDINALITY command, rolls back any changes you made to stored cardinalities, and returns you to the DCL prompt. Specifying an alternate value updates the stored cardinality value with the alternate value.

When you specify /NOCONFIRM, you are not given the option of updating stored cardinality values with an alternate value of your own choosing. Instead, the stored cardinality values that differ from the actual cardinality values are automatically updated with the actual cardinality values.

The /CONFIRM and /NOCONFIRM qualifiers are meaningless and are ignored if they are specified without the /UPDATE qualifier.

/UPDATE

/NOUPDATE

Specify the /UPDATE qualifier to update the stored cardinality values of tables and indexes. You can perform an update only when the stored cardinality values differ from the actual cardinality values. When updating cardinality values, Digital Equipment Corporation recommends that you update the stored cardinality values with the actual cardinality values, not with an alternate value of your own choosing. Specifying a value other than the actual cardinality value can result in poor database performance. Using the /UPDATE qualifier allows you to update the stored cardinality values of the specified tables and indexes even when the RDB\$SYSTEM storage area is read-only.

Specify the /NOUPDATE qualifier when you only want to display the stored and actual cardinality values for the specified tables and indexes.

/OUTPUT = file-name

The name of the file where output will be sent. SYSS\$OUTPUT is the default. The default output file type is LIS, if you specify a file name.

Usage Notes

You must have the Rdb/VMS ADMINISTRATOR (SQL DBADM) privilege to use the RMU/ANALYZE/CARDINALITY command.

You must have snapshot files enabled to use the RMU/ANALYZE /CARDINALITY command. When the command is issued, all logical areas are checked. If any read/write transactions are active when the cardinality values are being analyzed, the snapshot file will be accessed to provide the cardinality information.

RMU/ANALYZE/CARDINALITY Command

If you specify a name for the `table-or-index-name` parameter that is both an index name and a table name, the `RMU/ANALYZE/CARDINALITY` command performs the requested operation for both the table and index.

If you are updating the stored cardinality for a table or index, and a system failure occurs before the `RDB$SYSTEM` storage area is changed back to read-only, use the `RDO CHANGE DATABASE` statement to manually change the database back to read-only.

Examples

Example 1

The following command provides information on the cardinality for all indexes and tables in the sample personnel database:

```
$ RMU/ANALYZE/CARDINALITY/NOUPDATE MF_PERSONNEL
```

Example 2

The following command provides information on the cardinality for the `EMPLOYEES` table in the sample personnel database:

```
$ RMU/ANALYZE/CARDINALITY/NOUPDATE MF_PERSONNEL EMPLOYEES
```

RMU/ANALYZE/INDEXES Command

6.6 RMU/ANALYZE/INDEXES Command

Generates a formatted display of statistical information that describes the index structures for the database.

Format

```
RMU/ANALYZE/INDEXES database-file-name [index-name[,...]]
```

Command Qualifiers

```
/OUTPUT = file-name  
/[NO]BINARY_OUTPUT [ = file-option-list ]  
/OPTION = {NORMAL | FULL | DEBUG}
```

Defaults

```
/OUTPUT = SYS$OUTPUT  
/NOBINARY_OUTPUT  
/OPTION = NORMAL
```

Command Parameters

database-file-name

The name of the database for which you want information. The default file type is RDB. This parameter is required.

index-name

The name of the index for which you want information. The default is all defined indexes. This parameter is optional.

Command Qualifiers

/OUTPUT=file-name

The name of the file where output will be sent. SYS\$OUTPUT is the default. The default output file type is LIS, if you specify a file name.

/BINARY_OUTPUT [= file-option-list]

/NOBINARY_OUTPUT

Specifying the */BINARY_OUTPUT* qualifier allows you to output the summary results to a binary file, and to create a record definition file that is compatible with CDD/Plus for the binary output file. The binary output can be loaded into an Rdb/VMS database with the RMU/LOAD/RMS_RECORD_DEF command for use by a user-written management application or procedure. The binary output can also be used directly by the user-written application or procedure.

RMU/ANALYZE/INDEXES Command

The valid file options are:

- **FILE=file-spec**

The **FILE** option causes the **ANALYZE** data to be output to an RMS file that contains a fixed-length binary record for each index analyzed.

Specifying the **/OPTION=FULL** qualifier also generates an additional record for each level of each index analyzed. The default file type for the binary output file is **UNL**. The following command creates the binary output file **ANALYZE_OUT.UNL**:

```
$ RMU/ANALYZE/INDEXES -
_ $ /BINARY_OUTPUT=FILE=ANALYZE_OUT MF_PERSONNEL.RDB
```

- **RECORD_DEFINITION=file-spec**

The **RECORD_DEFINITION** option causes the **ANALYZE** data record definition to be output to an RMS file. The output file contains the definition in a subset of the **CDD/Plus CDO** command format, a format very similar to **RDO** field and relation definitions. The default file type for the record definition output file is **RRD**. The following command creates the output file **ANALYZE_OUT.RRD**:

```
$ RMU/ANALYZE/INDEXES -
_ $ /BINARY_OUTPUT=RECORD_DEFINITION=ANALYZE_OUT MF_PERSONNEL.RDB
```

You can specify both file options in one command by separating them with a comma and enclosing them within parentheses:

```
$ RMU/ANALYZE/INDEXES/BINARY_OUTPUT= -
_ $ (FILE=ANALYZE_OUT,RECORD_DEFINITION=ANALYZE_OUT) -
_ $ MF_PERSONNEL.RDB
```

The default is the **/NOBINARY_OUTPUT** qualifier, which does not create an output file.

/OPTION=type

The type of information and the level of detail the analysis will include. Three types of output are available:

- **NORMAL**

Output includes only summary information. **NORMAL** is the default.

- **FULL**

Output includes histograms and summary information. This option displays a summary line for each **SORTED** index level.

RMU/ANALYZE/INDEXES Command

- **DEBUG**

Output includes internal information about the data, as well as histograms and summary information. In general, use DEBUG for diagnostic support purposes. You can also use the DEBUG option to extract data and perform an independent analysis.

Usage Notes

You must have the Rdb/VMS ADMINISTRATOR (SQL DBADM) privilege to use the RMU/ANALYZE/INDEXES command.

Examples

Example 1

The following command analyzes the JH_EMPLOYEE_ID and SH_EMPLOYEE_ID indexes in the sample personnel database:

```
$ RMU/ANALYZE/INDEXES MF_PERSONNEL JH_EMPLOYEE_ID, SH_EMPLOYEE_ID
```

RMU/ANALYZE/PLACEMENT Command

6.7 RMU/ANALYZE/PLACEMENT Command

Generates a formatted display of statistical information describing the record placement relative to the index structures for the database.

Format

RMU/ANALYZE/PLACEMENT database-file-name [index-name[,...]]

Command Qualifiers

/AREAS [= storage-area-list]
/[NO]BINARY_OUTPUT [= file-option-list]
/OUTPUT=file-name
/OPTION = {NORMAL | FULL | DEBUG}

Defaults

/AREAS
/NOBINARY_OUTPUT
/OUTPUT = SYS\$OUTPUT
/OPTION = NORMAL

Description

You can use the RMU/ANALYZE/PLACEMENT command to determine:

- The maximum and average path length to a data record (the maximum and average number of index records touched to reach a data record).
- The maximum I/O path length or the total number of pages traversed to reach a data record.
- The minimum I/O path length, which determines whether the index and data records would both be in the buffer.
- The frequency distributions for the dbkey path lengths, maximum I/O path lengths, and minimum I/O path lengths for specified indexes.
- The distribution of data records on data pages in a storage area by logical area id and dbkey, the number of keys needed to reach each data record, the maximum and minimum I/O path lengths needed to reach the data record, the length of each key, and the specific key for the data record.

RMU/ANALYZE/PLACEMENT Command

Command Parameter

database-file-name

The name of the database to be analyzed. The default file type is RDB.

index-name

The name of the index for which you want information. The default is all defined indexes. This parameter is optional.

Command Qualifiers

/AREAS [= storage-area-list]

The storage areas to be analyzed. The default, /AREAS, results in analysis of all storage areas. If you omit /AREAS, information for all the storage areas will be displayed. If you specify more than one storage area name, separate the storage area names in the storage-area-list with a comma and enclose the list in parentheses.

If you specify more than one storage area with the /AREAS qualifier, the analysis RMU provides is a summary for all the specified areas. The analysis is not broken out into separate information for each specified storage area. To get index information for a specific storage area, issue the RMU/ANALYZE /PLACEMENT command specifying only that area with the /AREAS qualifier.

/BINARY_OUTPUT [= file-option-list]

/NOBINARY_OUTPUT

Specifying the /BINARY_OUTPUT qualifier allows you to output the summary results to a binary file, and to create a record definition file that is compatible with CDD/Plus for the binary output file. The binary output can be loaded into an Rdb/VMS database with the RMU/LOAD/RMS_RECORD_DEF command for use by a user-written management application or procedure. The binary output can also be used directly by the user-written application or procedure.

The valid file options are:

- **FILE=file-spec**

The FILE option causes the ANALYZE data to be output to an RMS file that contains a fixed-length binary record for each index analyzed. The default file type for the binary output file is UNL. The following command creates the binary output file ANALYZE_OUT.UNL:

```
$ RMU/ANALYZE/PLACEMENT -  
_ $ /BINARY_OUTPUT=FILE=ANALYZE_OUT MF_PERSONNEL.RDB
```

RMU/ANALYZE/PLACEMENT Command

- RECORD_DEFINITION=file-spec

The RECORD_DEFINITION option causes the ANALYZE data record definition to be output to an RMS file. The output file contains the definition in a subset of the CDD/Plus CDO command format, a format very similar to RDO field and relation definitions. The default file type for the record definition output file is RRD. The following command creates the output file ANALYZE_OUT.RRD:

```
$ RMU/ANALYZE/PLACEMENT -  
_$_ /BINARY_OUTPUT=RECORD_DEFINITION=ANALYZE_OUT MF_PERSONNEL.RDB
```

You can specify both file options in one command by separating them with a comma and enclosing them within parentheses:

```
$ RMU/ANALYZE/PLACEMENT/BINARY_OUTPUT= -  
_$_ (FILE=ANALYZE_OUT,RECORD_DEFINITION=ANALYZE_OUT) -  
_$_ MF_PERSONNEL.RDB
```

The default is the /NOBINARY_OUTPUT qualifier, which does not create an output file.

/OUTPUT=file-name

The name of the file where output will be sent. The default file type is LIS. If you do not specify the /OUTPUT qualifier, the default output is SYSS\$OUTPUT.

/OPTION=type

The type of information and level of detail the analysis will include. Three types of output are available:

- NORMAL
Output includes only summary information. NORMAL is the default.
- FULL
Output includes histograms and summary information.
- DEBUG
Output includes internal information about the data, as well as histograms and summary information. In general, use DEBUG for diagnostic support purposes. You can also use the DEBUG option to extract data and perform an independent analysis.

RMU/ANALYZE/PLACEMENT Command

Usage Notes

You must have the Rdb/VMS ADMINISTRATOR (SQL DBADM) privilege to use the RMU/ANALYZE/PLACEMENT command.

Examples

Example 1

The following command provides information on record storage relative to the DEPARTMENTS_INDEX index of the sample personnel database:

```
$ RMU/ANALYZE/PLACEMENT MF_PERSONNEL DEPARTMENTS_INDEX
```

RMU/BACKUP Command

6.8 RMU/BACKUP Command

Creates a backup copy of the database and places it in a file. If necessary, you can later use the RMU/RESTORE command to restore the database to its condition at the time of the backup.

Format

RMU/BACKUP root-file-spec backup-file-spec

Command Qualifiers

/[NO]INCREMENTAL[=COMPLETE|BY_AREA]
/EXCLUDE[=storage-area[...]]
/INCLUDE[=storage-area[...]]
/[NO]LOG
/[NO]ONLINE
/[NO]REWIND
/DENSITY=number-bpi
/LABEL[=(label-name-list)]
/BLOCK_SIZE=integer
/[NO]GROUP_SIZE=interval
/ACTIVE_IO=max-writes
/CRC[=AUTODIN_II]
/CRC=CHECKSUM
/NOCRC
/[NO]CHECKSUM_VERIFICATION
/OWNER=user-id
/TAPE_EXPIRATION=date-time
/PROTECTION[=vms-file-protection]
/LOCK_TIMEOUT=seconds

Defaults

See description
See description
See description
Current DCL verify value
/NOONLINE
/NOREWIND
System default for tape drive
See description
See description
See description
/ACTIVE_IO=3
See description
See description
See description
/NOCHECKSUM_VERIFICATION
See description
The current time
See description
See description

Description

The RMU/BACKUP command copies information contained in a database to a file. Rdb/VMS allows you to perform the following types of backup operations:

- Full backup

A full backup operation backs up all the database pages in a storage area.

RMU/BACKUP Command

- **Incremental backup**

An incremental backup operation backs up *only* those database pages in a storage area that have changed since the last full backup operation. An incremental backup always generates a backup file, even if no user data has changed since the last full backup.
- **Complete backup**

A complete backup operation backs up all the storage areas in a database. A complete backup can be either an incremental or full backup. If the complete backup is also an incremental backup, some storage areas may not have any database pages backed up (only the pages that have changed since the last full backup of the storage area are backed up).
- **By-area backup**

A by-area backup operation backs up only the storage areas you specify with the /INCLUDE or /EXCLUDE qualifiers. In a by-area backup operation, not all the storage areas in the database are backed up. A by-area backup operation can be either a full or incremental backup.
- **Full and complete backup**

A full and complete backup operation backs up all the database pages in all the storage areas in a database.

When a full backup operation is performed for one or more storage areas, the date and time for the last backup of those storage areas (as recorded in the backup (RBF) file) is updated. You can display the date and time of the last full backup operation for each of the storage areas in a database by executing an RMU/DUMP/BACKUP_FILE/OPTION=ROOT command on the latest backup (RBF) file for the database. The date and time that is displayed for the last backup of each storage area by the RMU/DUMP/BACKUP_FILE/OPTION=ROOT command is the date and time of the last *full* backup operation of the area. Note that an *incremental* backup operation for a storage area does *not* update the date and time for the last *full* backup of the storage area that is recorded in the backup file.

In the event of subsequent damage to the database, you can specify backup files in an RMU/RESTORE command to restore the database to the condition it was in when you backed it up.

The RMU/BACKUP command writes backup files in compressed format to save space. Free space in the root (RDB) file and on each database page in a storage area (RDA) file is not written to the backup file.

RMU/BACKUP Command

If you back up the database to tape, you must mount the backup media using the DCL MOUNT command before you issue the RMU/BACKUP command. The tape must be mounted as a FOREIGN volume. Like the VMS Backup utility, the RMU/BACKUP command performs its own tape label processing. This does not prohibit backing up an Rdb/VMS database to an RMS file on a Files-11 disk.

If you choose to perform a by-area backup, your database can be fully recovered after a system failure *only* if after-image journaling is enabled on the database. If your database has both read/write and read-only storage areas but does not have after-image journaling enabled, you should do complete backups (backups of all the storage areas in the database) at all times. Doing complete backups when after-image journaling is not enabled ensures that you can recover the entire database to its condition at the time of the previous backup.

Note *Use only the RMU/BACKUP command to back up Rdb/VMS multifile databases. Do not back up a multifile database using any other method (such as the VMS command BACKUP). The root file of a multifile database is updated only when RMU/BACKUP is used.*

Command Parameters

root-file-spec

The name of the root file. The root file name is also the name of the database. The default file type is RDB.

backup-file-spec

File specification for the backup file. The default file type is RBF.

When you back up your database to magnetic tape, Digital Equipment Corporation recommends that you supply a backup file name that is 17 or fewer characters in length. File names longer than 17 characters may be truncated. See the Usage Notes for more information on backup file names more than 17 characters in length.

If you use multiple tape drives, the backup-file-spec must include the tape device names. Separate the device names with commas.

```
$ RMU/BACKUP PERS.RDB /REWIND/DENSITY=6250 -  
_ $ $111$MUA0:PERS_FULL.RBF,$112$MUA1:
```

See the *VAX Rdb/VMS Guide to Database Maintenance and Performance* for more information on using multiple tape drives.

RMU/BACKUP Command

It is good practice to write backup files (as well as journal files) to a device other than the devices where the root, storage area, and snapshot files of the database are located. This way, if there is a problem with the database disks, you can still restore the database from backup. However, failure to place the backup copy on another disk (or on tape) does not constitute an error.

Command Qualifiers

/INCREMENTAL[=COMPLETE|BY_AREA]
/NOINCREMENTAL

Determines the extent of the backup to be performed. The four possible options are:

- ***/NOINCREMENTAL***

If you do not specify any of the possible */[NO]INCREMENTAL* options, the default is the */NOINCREMENTAL* option. With the */NOINCREMENTAL* option, a full and complete backup of the database is performed.

- ***/INCREMENTAL***

If you specify the */INCREMENTAL* option, an incremental backup of all the storage areas that have changed since the last full and complete backup of the database is performed.

- ***/INCREMENTAL=COMPLETE***

If you specify the */INCREMENTAL=COMPLETE* option, an incremental backup of all the storage areas that have changed since the last full and complete backup of the database is performed. Selecting the */INCREMENTAL=COMPLETE* option is the same as selecting the */INCREMENTAL* option.

- ***/INCREMENTAL=BY_AREA***

If you specify the */INCREMENTAL=BY_AREA* option, an incremental backup of the storage areas specified by the */INCLUDE* or */EXCLUDE* qualifier is performed. The */INCREMENTAL=BY_AREA* option backs up those database pages that have changed in each selected storage area since the last full backup of the area. The last full backup of the area is the later of the following:

- The last full and complete backup of the database
- The last full by-area backup of the area

RMU/BACKUP Command

With an INCREMENTAL by-area backup, each storage area may be backed up to a different time, which can make restoring multiple storage areas a more methodical process.

Following a full database backup, each subsequent incremental backup replaces all previous incremental backups.

/EXCLUDE[=*storage-area*[,...]]

Specifies storage areas that you want to exclude from the backup file. If neither the /EXCLUDE nor the /INCLUDE qualifier is specified with the RMU/BACKUP command, the default is for a full and complete backup of the database to be performed.

If you specify any one of the following options without specifying a list of storage area names, a full and complete backup of the database (the default) is performed:

- /EXCLUDE
- /INCLUDE
- /INCLUDE/EXCLUDE
- /EXCLUDE/INCLUDE

Specifying a list of storage area names with the /EXCLUDE qualifier tells RMU to exclude those storage areas from the backup operation but include all the other storage areas.

You receive an error message if you specify /EXCLUDE=*area-list* and /INCLUDE=*area-list* in one RMU/BACKUP command.

If you use the /EXCLUDE qualifier, your backup will be a by-area backup because the /EXCLUDE qualifier causes database storage areas to be excluded from the backup file. The following example shows the informational message you receive if you do not back up all the areas in the database:

```
$ RMU/BACKUP/EXCLUDE=(RDB$SYSTEM,MF_PERS_SEGSTR,EMPIDS_LOW,EMPIDS_MID) -
_ $ MF_PERSONNEL.RDB PARTIAL_MF_PERS.RBF
%RMU-I-NOTALLARE, Not all areas will be included in this backup file
```

By using RMU/BACKUP and RMU/RESTORE, you can back up and restore selected storage areas of your database. This RMU backup and restore by area feature is designed to:

- Speed recovery when corruption occurs in some (not all) of the storage areas of your database.

RMU/BACKUP Command

- Reduce the time needed to perform backups because some data (data in read-only storage areas, for example) does not need to be backed up with every backup of the database.

If you plan to use RMU/BACKUP and RMU/RESTORE to back up and restore only selected storage areas for a database, you *must* perform full and complete backups of the database at regular intervals. A full and complete backup is a full backup (*not* an incremental backup) of *all* the storage areas in the database. You must perform a full and complete backup because the database root file can only be restored from a full and complete backup file. Without a root file, restoring only the selected areas from by-area backup files does not result in a working database. Also, if the root file is corrupted, you can only recover storage areas up to (but not past) the date of the last full and complete backup. Therefore, Digital Equipment Corporation recommends that you perform full and complete backups regularly.

If you plan to back up and restore only selected storage areas for a database, Digital Equipment Corporation also strongly recommends that you enable after-image journaling for the database (in addition to performing the full and complete backup of the database described earlier). In other words, if you are not backing up and restoring *all* the storage areas in your database, you should have after-image journaling enabled. This ensures that you can recover all the storage areas in your database in the event of a system failure. If you *do not* have after-image journaling enabled and one or more of the areas restored with RMU/RESTORE are not current with the storage areas not restored, Rdb/VMS will not allow any transactions to use the storage areas that are not current in the restored database. In this situation, you can return to a working database by restoring the database using the backup file from the last full and complete backup of the database storage areas. However, any changes made to the database since the last full and complete backup are not recoverable. Or, if you used EXPORT instead of RMU/BACKUP to make a copy of your database, use IMPORT to restore your database to its previous state. Once again, any changes made to the database since the EXPORT operation are not recoverable. If you *do* have after-image journaling enabled, use RMU/RECOVER to apply transactions from the AIJ file to storage areas that are not current after the RMU/RESTORE completes. When the RMU/RECOVER command completes, your database will be consistent and usable.

If you specify more than one database storage area with /EXCLUDE, you must use a comma between each storage area name and enclose the list of storage area names within parentheses.

RMU/BACKUP Command

You receive an error message if you specify the `/EXCLUDE=*` option because the asterisk wild card causes all the storage areas to be excluded from the backup file.

`/INCLUDE[=storage-area[,...]]`

Specifies storage areas that you want to include in the backup file. If neither the `/INCLUDE` nor the `/EXCLUDE` qualifier is specified with the `RMU/BACKUP` command, the default is for a full and complete backup of the database to be performed.

If you specify any one of the following options without specifying a list of storage area names, a full and complete backup of the database (the default) is performed:

- `/INCLUDE`
- `/EXCLUDE`
- `/EXCLUDE/INCLUDE`
- `/INCLUDE/EXCLUDE`

Specifying a list of storage area names with the `/INCLUDE` qualifier tells `RMU` to include those storage areas in the backup operation but exclude all the other storage areas.

You receive an error message if you specify `/EXCLUDE=area-list` and `/INCLUDE=area-list` in one `RMU/BACKUP` command.

If you use the `/INCLUDE` qualifier, your backup will be a by-area backup, because the `/INCLUDE` qualifier causes database storage areas to be excluded from the backup file. The following example shows the informational message you receive if you do not back up all the areas in the database:

```
$ RMU/BACKUP/INCLUDE=(RDB$SYSTEM,MF_PERS_SEGSTR,EMPIDS_LOW,EMPIDS_MID) -
_ $ MF_PERSONNEL.RDB PARTIAL_MF_PERS2.RBF
%RMU-I-NOTALLARE, Not all areas will be included in this backup file
```

By using `RMU/BACKUP` and `RMU/RESTORE`, you can back up and restore selected storage areas of your database. This `RMU` backup and restore by area feature is designed to:

- Speed recovery when corruption occurs in some (not all) of the storage areas of your database.

RMU/BACKUP Command

- Reduce the time needed to perform backups because some data (data in read-only storage areas, for example) does not need to be backed up with every backup of the database.

If you plan to use RMU/BACKUP and RMU/RESTORE to back up and restore only selected storage areas for a database, you *must* perform full and complete backups of the database at regular intervals. A full and complete backup is a full backup (*not* an incremental backup) of *all* the storage areas in the database. You must perform a full and complete backup because the database root file can only be restored from a full and complete backup file. Without a root file, restoring only the selected areas from by-area backup files does not result in a working database. Also, if the root file is corrupted, you can only recover storage areas up to (but not past) the date of the last full and complete backup. Therefore, Digital Equipment Corporation recommends that you perform full and complete backups regularly.

If you plan to back up and restore only selected storage areas for a database, Digital Equipment Corporation also strongly recommends that you enable after-image journaling for the database (in addition to performing the full and complete backup of the database described earlier). In other words, if you are not backing up and restoring *all* the storage areas in your database, you should have after-image journaling enabled. This ensures that you can recover all the storage areas in your database in the event of a system failure. If you *do not* have after-image journaling enabled and one or more of the areas restored with RMU/RESTORE are not current with the storage areas not restored, Rdb/VMS will not allow any transactions to use the storage areas that are not current in the restored database. In this situation, you can return to a working database by restoring the database using the backup file from the last full and complete backup of the database storage areas. However, any changes made to the database since the last full and complete backup are not recoverable. Or, if you used EXPORT instead of RMU/BACKUP to make a copy of your database, use IMPORT to restore your database to its previous state. Once again, any changes made to the database since the EXPORT operation are not recoverable. If you *do* have after-image journaling enabled, use RMU/RECOVER to apply transactions from the AIJ file to storage areas that are not current after the RMU/RESTORE completes. When the RMU/RECOVER command completes, your database will be consistent and usable.

If you specify more than one database storage area with /INCLUDE, you must use a comma between each storage area name and enclose the list of storage area names within parentheses.

RMU/BACKUP Command

Using **/EXCLUDE** and **/INCLUDE** gives you greater backup flexibility, but with increased file management and recovery complexity. Users of large databases may find the greater backup flexibility to be worth the cost of increased file management and recovery complexity.

/LOG

/NOLOG

Specifies whether to report the processing of the command on SYSS\$OUTPUT. Specify the **/LOG** qualifier to request log output and the **/NOLOG** qualifier to prevent it. If you specify neither, the default is the current setting of the DCL verify switch. (The DCL SET VERIFY command controls the DCL verify switch.)

/ONLINE

/NOONLINE

When you specify the **/ONLINE** qualifier, users running active transactions at the time the command is entered can continue without interruption. The online backup operation will wait until all transactions other than read-only transactions complete (when the database reaches a quiet point) before proceeding. No new transactions can begin until the database reaches this quiet point and the online backup operation starts.

Any subsequent transactions that start during the online backup operation will work as long as the transactions do not require **EXCLUSIVE** access to the database, a table, or any index structure currently being backed up.

To perform an online database backup, snapshots (either **IMMEDIATE** or **DEFERRED**) must be enabled. You can use the **/ONLINE** qualifier with the **/INCREMENTAL** or **/NOINCREMENTAL** qualifiers.

If you use the default, **/NOONLINE**, users cannot be attached to the database. If a user has invoked the database and the **RMU/BACKUP** command is entered with **/NOONLINE** (or without **/ONLINE**), an RMU error results. For example:

```
$! another user has invoked the DB_DISK:PERSONNEL.RDB database
$ RMU/BACKUP DB_DISK:PERSONNEL DBS_BACKUPS:PERS_FULL.RBF
%RMU-I-FILACCERR, error opening database root file DB_DISK:PERSONNEL.RDB:1
-SYSTEM-W-ACCONFLICT, file access conflict
```

The **/NOONLINE** (offline) backup process has **EXCLUSIVE** access to the database and does not require snapshot files to work. Snapshots can be disabled when the **/NOONLINE** qualifier is used.

Digital Equipment Corporation recommends you close the database (with the **RMU/CLOSE** command) when you perform the offline backup operation of a large database.

RMU/BACKUP Command

/REWIND

/NOREWIND

When the */REWIND* qualifier is specified, the magnetic tape that contains the backup file will be rewound before processing begins. The tape will be initialized according to the */LABEL* and */DENSITY* qualifiers. The */NOREWIND* qualifier is the default and causes the backup file to be created starting at the current logical end-of-tape (EOT).

The */[NO]REWIND* qualifiers are applicable only to tape devices. RMU returns an error message if these qualifiers are used and the target device is not a tape device.

/DENSITY=number-bpi

The density in bits per inch (BPI) at which the output volume is to be written. The default value is the system default for the tape drive specified.

The */DENSITY* qualifier is applicable only to tape drives. RMU returns an error message if this qualifier is used and the target device is not a tape device.

The minimum allowable value that can be specified for */DENSITY* is 800. The maximum allowable value that can be specified for */DENSITY* is 39872.

/LABEL[=(label-name-list)]

The one- to six-character string with which the volumes of the backup file are to be labeled. The */LABEL* qualifier is applicable only to tape volumes. If it is not specified, the label will be derived from the first six characters of the backup file name.

You can specify a list of tape labels for multiple tapes. If you list multiple tape label names, separate the names with commas and enclose the list of names in parentheses.

In a normal restore operation, the */LABEL* qualifier you specify with the *RMU/RESTORE* command should be the same */LABEL* qualifier you specify with the *RMU/BACKUP* command to back up your database.

/BLOCK_SIZE=integer

The maximum record size for the backup file. The size can vary between 2048 and 65,024 bytes. The default value is device dependent. The appropriate block size is a compromise between tape capacity and error rate. The block size you specify must be larger than the largest page length in the database.

RMU/BACKUP Command

/GROUP_SIZE=interval
/NOGROUP_SIZE

The frequency at which XOR recovery blocks are written to tape. The group size can vary from 0 to 100. Specifying a group size of zero or specifying */NOGROUP_SIZE* results in no XOR recovery blocks being written. The */GROUP_SIZE* qualifier is only applicable to tape, and its default value is device dependent. RMU returns an error message if this qualifier is used and the target device is not a tape device.

/ACTIVE_IO=max-writes

The maximum number of write operations to a backup device that the RMU/BACKUP command will attempt simultaneously. This is not the maximum number of write operations in progress; that value is the product of active system I/O operations and the number of devices being written to simultaneously.

The value of the */ACTIVE_IO* qualifier may range from 1 to 5. The default value is 3. Values larger than 3 may improve performance with some tape drives.

/CRC[=AUTODIN_II]

Uses the AUTODIN-II polynomial for the 32-bit CRC calculation and provides the most reliable end-to-end error detection. This is the default for NRZ/PE (800/1600 bpi) tape drives.

If your CPU has the CRC instruction implemented in microcode, it will be used. If your CPU does not have the CRC instruction implemented in microcode, RMU will automatically use a high performance software procedure instead of the CRC instruction.

Typing */CRC* is sufficient to select the */CRC=AUTODIN_II* option. It is not necessary to type the entire qualifier.

/CRC=CHECKSUM

Uses ones complement addition, which is the same computation used to checksum the database pages on disk. This is the default for GCR (6250 bpi) tape drives and for TA78, TA79, and TA81 drives. These HSC drives have adequate error detection capability, but CI contention may cause data underruns and unrecoverable restore errors unless end-to-end error detection is employed.

The */CRC=CHECKSUM* option adequately detects data underrun errors and is about six times faster than the CRC microcode instruction.

RMU/BACKUP Command

/NOCRC

Disables end-to-end error detection. This is the default for TA90 (IBM 3480 class) drives.

Note *The overall effect of the /CRC=AUTODIN_II, /CRC=CHECKSUM, and /NOCRC defaults is to raise tape reliability to a par with disk reliability. If you retain your tapes longer than 1 year, the /NOCRC default may not be adequate. For tapes retained longer than 1 year, use the /CRC=CHECKSUM qualifier.*

If you retain your tapes longer than 3 years, you should always use the /CRC=AUTODIN_II qualifier.

Tapes retained longer than 5 years could be deteriorating and should be copied to fresh media.

/[NO]CHECKSUM_VERIFICATION

Specifying /CHECKSUM_VERIFICATION requests RMU/BACKUP to verify the checksum stored on each database page before it is backed up, thereby providing end-to-end error detection on the database I/O. You can save significant CPU resources by specifying the /CHECKSUM_VERIFICATION qualifier only when you are experiencing or have experienced disk, HSC, or CI port hardware problems.

The default is /NOCHECKSUM_VERIFICATION. When you specify this option, RMU/BACKUP does not verify the checksum on the database pages.

/OWNER_ID=user-id

Specifies the owner of the tape volume set. The owner is the user who will be permitted to restore the database. The user-id must be one of the following types of VMS identifier:

- A user identification code (UIC) in [group-name,member-name] alphanumeric format
- A user identification code (UIC) in [g,m] numeric format
- A general identifier, such as SECRETARIES
- A system-defined identifier, such as DIALUP

The /OWNER_ID qualifier cannot be used with a backup operation to disk. When used with tapes, the /OWNER_ID qualifier applies to all continuation volumes. The /OWNER_ID qualifier applies to the first volume only if the /REWIND qualifier is also specified. If the /REWIND qualifier is not specified,

RMU/BACKUP Command

the backup operation appends the file to a previously labeled tape, so the first volume can have a different protection than the continuation volumes.

See the Usage Notes for information on tape label processing.

/TAPE_EXPIRATION=date-time

Specifies the expiration date of the backup file. The default for this qualifier is "NOW" (the current time), meaning that the tape can be overwritten immediately. The */TAPE_EXPIRATION* qualifier cannot be used with a backup operation to disk.

See the Usage Notes for information on tape label processing.

/PROTECTION[=vms-file-protection]

Specifies the VMS file protection for the backup file that is being produced from the RMU/BACKUP command.

If you do not specify the */PROTECTION* qualifier, the backup file receives the default protection of S:RWED,O:RE,G,W.

The protection that can be assigned to tapes is different than VMS file protection. For example, tapes do not allow DELETE or EXECUTE access, and SYSTEM always receives READ and WRITE access. Also, a more restrictive class receives the access of less restrictive classes. For example, if you specify */PROTECTION=(S,O,G:W,W:R)*, that protection on tape becomes (S:RW,O:RW,G:RW,W:R).

/LOCK_TIMEOUT=seconds

Determines the maximum time the backup operation will wait for the quiet point lock during online backup operations. When you specify the */LOCK_TIMEOUT=seconds* qualifier, you must specify the number of seconds to wait for the quiet point lock. If the time limit expires, an error is signaled and the backup operation fails.

When the */LOCK_TIMEOUT=seconds* qualifier is not specified, the backup operation will wait indefinitely for the quiet point lock during an online backup operation.

The */LOCK_TIMEOUT=seconds* qualifier is ignored for offline backup operations.

RMU/BACKUP Command

Usage Notes

You must have either the Rdb/VMS ADMINISTRATOR or OPERATOR (SQL DBADM or OPERATOR) privilege to use the RMU/BACKUP command.

You may receive the following error messages if you try to back up your database soon after the last user has detached from the database or if the database was manually opened with RMU/OPEN and has not been manually closed with RMU/CLOSE:

```
$ RMU/BACKUP DATABASE1.RDB DB1_BU.RBF
%RMU-F-FILACCERR, error opening database root file DISK1:[CORP]DATABASE1.RDB
-SYSTEM-W-ACCONFLICT, file access conflict
```

To back up your database, you must have exclusive access to the database root file. These two error messages usually indicate that you do not have exclusive access to the database root file because VMS still has access to it. If your database was manually opened with RMU/OPEN, you should be able to gain exclusive access to the root file by manually closing the database with an RMU/CLOSE command. If your database does not need to be manually closed, wait a few minutes and try again. After waiting a few minutes, you will usually be able to back up your database successfully. You can also receive these two error messages when you attempt other operations for which you must have exclusive access to the database root file. The solution in all cases is to attempt the operation again later.

When you back up your database to magnetic tape, Digital Equipment Corporation recommends that you supply a name for the backup file that is 17 or fewer characters in length. File names longer than 17 characters may be truncated. The VMS operating system supports four file-header labels: HDR1, HDR2, HDR3, and HDR4. In HDR1 labels, the file identifier field contains the first 17 characters of the file name you supply. The remainder of the file name is written into the HDR4 label, provided that this label is allowed. If no HDR4 label is supported, a file name longer than 17 characters will be truncated. See the information on file-header labels in the *Guide to VMS Files and Devices*.

RMU/BACKUP Command

The following RMU commands are acceptable. The terminating period for the backup file name is not counted as a character, and the default file type of RBF is assumed. Therefore, VMS interprets the file name as WEDNESDAYS_BACKUP, which is 17 characters in length:

```
RMU/BACKUP/REWIND/LABEL=TAPE MF_PERSONNEL MUA0:WEDNESDAYS_BACKUP.  
RMU/RESTORE/REWIND/LABEL=TAPE MUA0:WEDNESDAYS_BACKUP.
```

The following RMU commands are not acceptable. Because no terminating period is supplied, VMS supplies a period and a file type of RBF, and interprets the backup file name as WEDNESDAYS_BACKUP.RBF, which is 20 characters in length:

```
RMU/BACKUP/REWIND/LABEL=TAPE MF_PERSONNEL MUA0:WEDNESDAYS_BACKUP  
RMU/RESTORE/REWIND/LABEL=TAPE MUA0:WEDNESDAYS_BACKUP
```

The steps in tape label checking are:

- 1 Use the DCL MOUNT/FOREIGN command to mount the tape you want to use.

You must mount the first tape volume; RMU mounts subsequent volumes automatically. If VMS encounters an error when mounting the tape, either the RMU function will abort or the problem will be reported and the operation retried. If a severe error occurs, RMU does not mount subsequent volumes.

- 2 The tape characteristics are checked.

For an RMU/BACKUP operation, the tape must be properly mounted and cannot be write protected. If these checks fail, you are asked to mount the correct relative volume without write protection and to indicate when you are finished.

- 3 The tape labels are checked.

If the volume label disagrees with the label specified with the /LABEL qualifier or by the default (the backup file name), you receive an informational message.

If the tape is not the first volume, or if you specified the /REWIND qualifier in the command, RMU checks the tape protection (the user-id specified with the /OWNER_ID=user-id qualifier of the RMU/BACKUP command) and the expiration date of the tape, and issues an informational message regarding the error. You can overwrite an expired tape. You can read a tape that has not expired.

RMU/BACKUP Command

If any of the checks fail, you are asked what you want to do with the tape volume. You must select one of the following options:

- **QUIT**
This option cancels the operation, and causes you to exit from RMU.
- **RETRY**
This option repeats the checks after dismounting and remounting the same tape. The **RETRY** option is useful if you left the drive off line, or if there was some other kind of correctable error.
- **UNLOAD**
This option dismounts and unloads the tape so you can load the correct tape on the drive.
- **OVERRIDE**
This option overrides the failed checks and uses the tape anyway. The **OVERRIDE** option can be used with the **RMU/RESTORE** and **RMU/DUMP/BACKUP_FILE** commands only.
- **INITIALIZE**
This option requests that the tape be rewound and relabeled. The **INITIALIZE** option can be used with the **RMU/BACKUP** command only.

4 The tape is remounted and reverified as requested.

Tape changes follow essentially the same procedure, with one difference. If the first volume is not at the beginning of the tape, complete label checking is not available.

The **RMU/BACKUP** command works correctly with unlabeled or nonstandard formatted tapes when the **/REWIND** qualifier is specified. However, tapes that have never been used or initialized and nonstandard tapes sometimes produce errors that make VMS mount attempts fail repeatedly. In this situation, RMU cannot continue until you use the VMS **INITIALIZE** command to correct the error.

RMU/BACKUP Command

Examples

Example 1

The following command performs a full backup of the PERSONNEL database and creates a log of the session:

```
$ RMU/BACKUP $111$DUA9:[DBS]PERSONNEL.RDB -  
_ $ $222$DUA20:[BACKUPS]PERS_FULL_BU.RBF /LOG
```

Example 2

To perform an incremental backup, include the /INCREMENTAL qualifier. Assume a full backup was done late Monday night. The following command performs an incremental backup of database updates just for the following day:

```
$ SET DEFAULT $111$DUA9:[DBS]  
$ RMU/BACKUP/INCREMENTAL PERSONNEL.RDB $222$DUA20:[BCK]TUESDAY_PERS_BKUP/LOG
```

Example 3

To back up the database while there are active users, specify the /ONLINE qualifier:

```
$ RMU/BACKUP/ONLINE PERSONNEL.RDB $222$DUA20:[BACKUPS]PERS_BU.RBF /LOG
```

Example 4

The following RMU/BACKUP command includes only the EMPIDS_LOW and EMPIDS_MID storage areas in the backup of the MF_PERSONNEL database. All the other storage areas in the MF_PERSONNEL database are excluded from the backup file:

```
$ SET DEFAULT $111$DUA9:[DBS]  
$ RMU/BACKUP/INCLUDE=(EMPIDS_LOW,EMPIDS_MID) -  
_ $ MF_PERSONNEL.RDB $222$DUA20:[BACKUPS]MF_PERS_BU.RBF
```

RMU/BACKUP/AFTER_JOURNAL Command

6.9 RMU/BACKUP/AFTER_JOURNAL Command

Creates a backup of the database after-image journal (AIJ) file. The RMU/BACKUP/AFTER_JOURNAL command copies completed transactions recorded in the primary AIJ file (always on a disk device) to the backup file (which may be on a tape or disk device). On completion, the primary AIJ file is truncated and reused. During periods of high update activity the truncation of the primary AIJ file may not be performed because of conflicting access to the AIJ file by other users. But the storage allocated to the primary AIJ file is still reused when the backup completes.

This command provides a two-stage journaling process that can save disk space and diminish the dependence on tape drives being available at all times.

The backup AIJ file is an actual, usable AIJ file that can be applied to the appropriate Rdb/VMS database in a roll forward operation.

The RMU/BACKUP/AFTER_JOURNAL command can be used while users are attached to the database.

Format

RMU/BACKUP/AFTER_JOURNAL root-file-spec backup-file-spec

Command Qualifiers

/[NO]CONTINUOUS
/[NO]INTERVAL=number-seconds
/[NO]LOG
/[NO]THRESHOLD=disk-blocks
/UNTIL=time

Defaults

/NOCONTINUOUS
/NOINTERVAL
Current DCL verify value
/NOTHRESHOLD
See description

Description

The RMU/BACKUP/AFTER_JOURNAL command saves disk space by spooling the AIJ file to tape. This RMU command replaces the RDO SPOOL statement used in previous versions of Rdb/VMS.

RMU/BACKUP/AFTER_JOURNAL Command

Specify the JOURNAL FILE IS clause in a CHANGE DATABASE statement in RDO to have Rdb/VMS record completed transactions in the AIJ file on disk. If you then use the RMU/BACKUP/AFTER_JOURNAL command, RMU will copy the AIJ file to tape (or a disk device). When this backup copy is complete, RMU truncates the primary (original) AIJ file. This technique allows you to refill the disk space used by the primary AIJ file. The backup AIJ file you create can be used with the RDO RECOVER statement to recover (roll forward) journaled transactions. In some cases, you may have to issue two RECOVER statements: one for the backup AIJ file and a second for the primary (and more recent) AIJ file.

If you back up the AIJ file to tape, you must mount the backup media using the DCL MOUNT command before you issue the RMU/BACKUP/AFTER_JOURNAL command. Because the RMU/BACKUP/AFTER_JOURNAL command will use RMS to write to the tape, the tape must be mounted as a VMS volume.

To create a true backup (duplicate) copy of the AIJ file, you must use the VMS Backup utility.

The RMU/BACKUP/AFTER_JOURNAL command can be used in a batch job to avoid tying up an interactive terminal for long periods of time. The /CONTINUOUS, /INTERVAL, /THRESHOLD, and /UNTIL qualifiers control the duration and frequency of the backup process. When you use the /CONTINUOUS qualifier, the command can occupy a terminal indefinitely. Therefore, it is good practice to issue the command through a batch process when executing a continuous after-image journal backup. However, remember that the portion of the command procedure that follows the RMU/BACKUP /AFTER_JOURNAL command will not be executed until after the time specified by the /UNTIL qualifier.

Command Parameters

root-file-spec

The name of the root file. The root file name is also the name of the database. An error results if you specify a database that does not have after-image journaling enabled. The default file type is RDB.

backup-file-spec

A file specification for the AIJ backup file. The default file type is AIJ. Normally, this file is on tape instead of disk.

RMU/BACKUP/AFTER_JOURNAL Command

Command Qualifiers

/CONTINUOUS

/NOCONTINUOUS

Specifies whether the AIJ process operates continuously, under control of the /INTERVAL and /UNTIL qualifiers. When you use the /CONTINUOUS qualifier, you must use /UNTIL to specify when the backup process should stop. You can also safely stop the backup process by using the DCL STOP command.

If you specify the /CONTINUOUS qualifier, Rdb/VMS does not terminate the backup process after truncating the primary AIJ file. Instead, the backup process waits for a period of time that you specify by using the /INTERVAL qualifier argument. After that time interval, the backup process tests to determine if the threshold has been reached. It then performs backup as needed and then waits again until the next interval break.

If you specify the /CONTINUOUS qualifier, the backup process occupies the terminal (that is, no DCL prompt occurs) until the process terminates. Therefore, you should generally enter the command through a batch process.

If you do not specify the /CONTINUOUS qualifier, the backup process determines whether the threshold has been reached, performs backup as needed, and stops. You must enter another RMU/BACKUP/AFTER_JOURNAL command to resume AIJ file backup.

If you specify the default, /NOCONTINUOUS, the backup process stops as soon as it completely backs up the AIJ file.

/INTERVAL=number-seconds

/NOINTERVAL

The number of seconds for which the backup process waits. Use this qualifier in conjunction with the /CONTINUOUS qualifier. The interval determines how often to test the primary AIJ file to determine if it contains more blocks than the value of the /THRESHOLD qualifier.

If you specify the default, /NOINTERVAL, this RMU/BACKUP/AFTER_JOURNAL job runs continuously, searching the primary AIJ file for new records to copy to the backup file.

Digital Equipment Corporation recommends using the /NOINTERVAL default initially because the interval that you choose can affect the performance of the database considerably. Too long an interval can result in a long primary AIJ file with a relatively long backup time, during which the database will be frozen against new transactions. Too short an interval can also hurt

RMU/BACKUP/AFTER_JOURNAL Command

performance, although the effect is likely to be less visible. In general, you can arrive at a good interval time on a given database only by judgment and experimentation.

If you specify both the /INTERVAL and /NOCONTINUOUS qualifiers, the /INTERVAL qualifier is ignored.

/LOG
/NOLOG

Specifies whether to report the processing of the command on SYSSOUTPUT. Specify the /LOG qualifier to request log output and the /NOLOG qualifier to prevent it. If you specify neither, the default is the current setting of the DCL verify switch. (The DCL SET VERIFY command controls the DCL verify switch.)

/THRESHOLD=disk-blocks
/NOTHRESHOLD

Sets an approximate limit on the size of the primary AIJ file. When the size of the primary AIJ file exceeds the threshold, you cannot initiate new transactions until the backup process finishes backing up and truncating (resetting) the primary AIJ file. During backup, existing transactions can continue to write to the AIJ file. Before new transactions can start, all activity issuing from existing transactions (including activity occurring after the threshold is exceeded) must be flushed from the primary AIJ disk file to the after-image backup file. At that time, the primary AIJ file will be completely truncated.

If you use the default, /NOTHRESHOLD, each backup cycle will completely back up the primary AIJ file. Digital Equipment Corporation recommends using the /NOTHRESHOLD default.

An appropriate value for the /THRESHOLD qualifier depends on the activity of your database, how much disk space you want to use, whether backup will be continuous, and how long you are willing to wait for backup to complete.

/UNTIL=time

The future time and date to stop the continuous backup process. There is no default. If you specify the /CONTINUOUS qualifier, you must specify /UNTIL.

Note The /UNTIL qualifier accepts all VMS Version 5.0 date and time strings, as well as international dates. See VMS RTL Library (LIB\$) Manual for more information. The date and time strings specified must be in quotes because they can contain spaces and other DCL syntax characters such as commas. A colon separator between the date and time is no longer valid.

RMU/BACKUP/AFTER_JOURNAL Command

Usage Notes

You must have either the Rdb/VMS ADMINISTRATOR or OPERATOR (SQL DBADM or OPERATOR) privilege to use the RMU/BACKUP/AFTER_JOURNAL command.

Examples

Example 1

Assuming that you have enabled after-image journaling for the PERSONNEL database, the following command will cause AIJ entries to be backed up continuously to the tape file MUA0:BU_PERSONNEL.AIJ:

```
$ DEFINE LIB$DT_INPUT_FORMAT "!DB-!MAAU-!Y4 !H04:!M0:!S0.!C2"
$ RMU/BACKUP/AFTER_JOURNAL/CONTINUOUS/THRESHOLD=500 -
_$/INTERVAL=300/UNTIL="15-JUN-1989 06:00:00.00" -
_$/PERSONNEL.RDB MUA0:BU_PERSONNEL.AIJ
```

Every 300 seconds, the backup process tests to determine if the primary AIJ file on disk has reached the threshold size of 500 blocks. If not, transaction processing continues normally for one or more 300-second intervals until the threshold test indicates that the primary AIJ file has reached a size of at least 500 blocks. When the AIJ file reaches that file size, RMU allows existing transactions to continue to write to the primary AIJ file but does not allow new transactions to start.

Assuming that the primary AIJ file contains 550 blocks, Rdb/VMS flushes those 550 blocks to the backup journal file and deletes them from the primary journal. Then, the backup process determines if the transactions already in progress have written more records to the primary journal during the backup operation. If so, RMU flushes those records to the backup file.

After Rdb/VMS completely flushes the primary journal, it truncates the journal to zero blocks. Rdb/VMS then allows new transactions to start and the backup process resumes threshold testing at 300-second intervals. The backup process continues until the time and date specified by the /UNTIL qualifier.

RMU/CLOSE Command

6.10 RMU/CLOSE Command

Closes an open database. See the OPEN IS {AUTOMATIC | MANUAL} clause of the RDO CHANGE DATABASE statement for more information on opening databases.

Format

RMU/CLOSE root-file-spec [...]

<u>Command Qualifiers</u>	<u>Defaults</u>
/[NO]ABORT=option	/ABORT=FORCEX
/[NO]CLUSTER	See description
/PATH	None
/[NO]WAIT	/NOWAIT

Description

RMU/CLOSE closes an open database. You can close the database immediately by specifying the /ABORT qualifier, or you can allow current users to finish their transactions by specifying /NOABORT.

You need to use RMU/CLOSE only if you have specified manual opening for your database. If you have specified automatic opening for your database, the RMU/CLOSE command affects current database users only; new users are allowed to attach to the database.

If you use an RMU/OPEN command to open a database, you must later use an RMU/CLOSE command to close it.

If you have specified manual opening for your database, you must use the RMU/OPEN command to manually open the database before any users can invoke it and the RMU/CLOSE command to manually close the database. You can also use the RMU/CLOSE command to close a database that is open because a process has invoked it. A root file is considered open if it has been specified in a previous RMU/OPEN command or has active users attached to it.

Use the RMU/SHOW USERS command to display information about databases in current use on your node.

RMU/CLOSE Command

Parameter

root-file-spec

Specifies an open root file. The default file type is RDB.

Command Qualifiers

/ABORT=option

/NOABORT

Specifies whether to close the database immediately or allow process rundown.

The */ABORT* qualifier has two options. Both refer to VMS system services. The options are as follows:

- **FORCEX**

When you use the FORCEX (forced exit) option, run units are recovered and no RUJ files are left in the directories. Therefore, RMU/BACKUP will work. The option cannot force an exit of a database process with a spawned subprocess or a suspended process. It aborts batch jobs that are using the database. FORCEX is the default.

- **DELPRC**

When you use the DELPRC (delete process) option, run units are not recovered. RUJ files are left in the directories to be recovered on the next invocation of the database. The processes and any subprocesses of all database users are deleted, thereby exiting the processes from the database. Therefore a RMU/BACKUP will not work.

The DELPRC and FORCEX options are based on VMS system services \$DELPRC and \$FORCEX. Refer to the *VMS System Services Reference Manual* for more information.

With the */NOABORT* option, users already attached to the database can continue, and the root file global sections remain mapped until all users exit the database. No new users will be allowed to attach to the database. When all current images terminate, RMU closes the database.

/CLUSTER

/NOCLUSTER

Specifying the */CLUSTER* qualifier closes a database on all nodes of a VAXcluster that currently have the database open. Specifying */CLUSTER* is equivalent to issuing an RMU/CLOSE command on every node in the cluster.

RMU/CLOSE Command

Specifying the `/NOCLUSTER` and the `/NOWAIT` qualifier works only from a node on which the database is open. If you enter this command on a node on which the database is not open, you receive an error message.

The default is `/CLUSTER` if the `/WAIT` qualifier is specified. The default is `/NOCLUSTER` if `/NOWAIT` is specified.

When the `/NOCLUSTER` qualifier is in effect, the database is closed only on the node from which you issue the command.

/PATH

The full or relative data dictionary path name in which the definitions reside for the database you want to close.

The `/PATH` qualifier is a positional qualifier. The path name cannot include wildcard characters.

/WAIT

/NOWAIT

If the `/WAIT` qualifier is specified, the `/CLUSTER` qualifier shuts down the database for the entire cluster, even if no other users are on the node from which the `RMU/CLOSE` command is issued.

If you specify the `/WAIT` qualifier, `RMU` closes and recovers the database before the `DCL` prompt is returned to you.

The default is `/NOWAIT`. With the `/NOWAIT` qualifier, the database may not be closed when the `DCL` prompt is returned to you. With `/NOWAIT`, you can receive `SYS-F-ACCONFLICT` errors when you attempt to open a database after issuing the `RMU/CLOSE` command.

Usage Notes

You must have the `VMS SYSPRV` privilege to use the `RMU/CLOSE` command.

Examples

Example 1

This command closes the `PERSONNEL` database on all nodes of the `VAXcluster`.

```
$ RMU/CLOSE/CLUSTER PERSONNEL
```

RMU/CLOSE Command

Example 2

When the following command is issued from a node in a cluster, the /CLUSTER qualifier shuts down the database for the entire cluster, even if no users are on the node from which the command is issued. The /WAIT qualifier causes RMU to close the database before the DCL prompt is returned.

```
$ RMU/CLOSE/CLUSTER/WAIT MF_PERSONNEL.RDB
```

Example 3

The following command closes the PERSONNEL database in the [WORK] directory, all the databases in the [TEST] directory, and the databases specified by the path names CDD\$TOP.FINANCE and SAMPLE_DB:

```
$ RMU/CLOSE DISK1:[WORK]PERSONNEL, CDD$TOP.FINANCE/PATH, -  
_ $ DISK1:[TEST]*, SAMPLE_DB/PATH
```

RMU/CONVERT Command

6.11 RMU/CONVERT Command

Converts any of the following versions of Rdb/VMS databases to a Version 4.0 database:

- Version 3.0
- Version 3.0A
- Version 3.0B
- Version 3.1
- Version 3.1A
- Version 3.1B

Format

RMU/CONVERT database-list

<u>Command Qualifiers</u>	<u>Defaults</u>
/[NO]COMMIT	/COMMIT
/[NO]CONFIRM	See description
/PATH	None
/[NO]ROLLBACK	/NOROLLBACK

Command Parameter

database-list

The database-list parameter is a list of databases to be converted. A list item can be either the file specification of a database root file or a data dictionary path name.

You can use wild cards in the file specification of a database root file.

You cannot use wild cards in a data dictionary path name.

RMU/CONVERT Command

Command Qualifiers

/COMMIT

/NOCOMMIT

Makes the database conversion permanent. When you specify the */COMMIT* option, the database is converted to a Version 4.0 database and cannot be returned to the previous version. The default is */COMMIT*.

When you specify the */NOCOMMIT* option, you can commit the database to Version 4.0 or roll it back to the previous version at a later time.

/CONFIRM

/NOCONFIRM

Requests user input during the conversion procedure. When you specify the */CONFIRM* option, RMU asks if you are satisfied with the backup of your database. If the database being converted has after-image journaling enabled, RMU asks if you want to disable after-image journaling. The default is */CONFIRM* when the conversion is executed interactively.

If you specify the */NOCONFIRM* option, you are not prompted during the conversion procedure. The default is */NOCONFIRM* if the conversion is executed from a batch job.

/PATH

Identifies that the database is being specified by its data dictionary path name instead of its file specification.

/ROLLBACK

/NOROLLBACK

Returns a database that has been converted to a Version 4.0 database (but not committed) to the previous version. You might decide to return to the previous version of the database for technical, performance, or business reasons.

The */NOROLLBACK* option prevents you from returning your converted database to the previous version. The default is */NOROLLBACK*.

If you specify both the */NOCOMMIT* option and the */ROLLBACK* option in the same RMU/CONVERT command, your database will be converted to Version 4.0 and immediately rolled back to the previous version when the RMU/CONVERT command is executed.

RMU/CONVERT Command

Usage Notes

To use the RMU/CONVERT command, you must have the VMS SYSPRV privilege or the RMU executable image must be installed with SYSPRV on your system.

If the database conversion does not complete (for example, because of a system failure or Rdb/VMS monitor shutdown), you can execute RMU/CONVERT again later. The ability to complete the conversion process later keeps you from having a half-converted database that is corrupted.

RMU/CONVERT operates by creating a converted copy of the system relations and indexes. This implies that the RDB\$SYSTEM storage area may grow during the conversion, but it is no longer likely that the system relations will be fragmented by the conversion process.

Because a copy of the system relations is made, the time taken by the conversion is proportional to the size of the system relations. This is typically a few minutes per database. However, if the database has very large system relations, the conversion can be costly. If the database has a large number of versions of some relations, it might be more efficient for you to use the EXPORT and IMPORT statements to convert the database.

After the conversion, both copies of the system relations are stored in the database. The /COMMIT option selects the converted copy and deletes the original copy. The /ROLLBACK option selects the original copy and deletes the converted copy. You can specify either the /COMMIT or /ROLLBACK option at a later time if you selected the /NOCOMMIT option when the database was converted.

While both copies of the system relations exist, the database is usable under Version 4.0, but not under the earlier version. Also, DDL (data definition language) operations to the database are prohibited to ensure that both copies of the system relations remain consistent. After you specify either the /COMMIT or /ROLLBACK option, you can again perform DDL operations on the database.

RMU/CONVERT Command

Examples

Example 1

The first command in the following example converts a Version 3.1 database with an AIJ file to a Version 4.0 database. Because the /NOCOMMIT qualifier was specified in the first command, you can roll back the converted database (the Version 4.0 database) to the original Version 3.1 database. In the second command, the converted database is rolled back to the original database.

```
$ RMU/CONVERT/CONFIRM/NOCOMMIT SAMPLE.RDB
Are you satisfied with your backup of DISK1:[RICK.RDB]SAMPLE.RDB;1 [N]? Y
After-image journaling will be disabled if the RMU/CONVERT of DISK1:[RICK.R
DB]SAMPLE.RDB;1 continues. Do you wish to proceed [N]? Y
%RMU-I-LOGCONVRT, database root converted to current structure level
%RMU-S-CVTDBSUC, database DVD21:[RICK.RDB]SAMPLE.RDB;1 successfully convert
ed from version V3.1 to V4.0
%RMU-I-LOGCREAIJ, created after-image journal file DISK2:[AIJS]SAMPLE_AIJ.A
IJ;2
$ RMU/CONVERT/ROLLBACK SAMPLE.RDB
Are you satisfied with your backup of DISK1:[RICK.RDB]SAMPLE.RDB;1 [N]? Y
After-image journaling will be disabled if the RMU/CONVERT of DISK1:[RICK.R
DB]SAMPLE.RDB;1 continues. Do you wish to proceed [N]? Y
%RMU-I-LOGCONVRT, database root converted to current structure level
%RMU-I-CVTROLSUC, CONVERT rolled-back for DISK1:[RICK.RDB]SAMPLE.RDB;1 to v
ersion 3.1
%RMU-I-LOGCREAIJ, created after-image journal file DISK2:[AIJS]SAMPLE_AIJ.A
IJ;3
```

Example 2

The following command converts all the databases in DISK1:[RICK] and its subdirectories and also the SPECIAL_DB database that is identified by its data dictionary path name. The /NOCONFIRM qualifier is specified, so RMU does not request user input. The /NOCOMMIT qualifier is not specified, so the converted databases cannot be rolled back.

```
$ RMU/CONVERT/NOCONFIRM DISK1:[RICK...]*.RDB,CDD$TOP.RICK.SPECIAL_DB/PATH
```

RMU/COPY_DATABASE Command

6.12 RMU/COPY_DATABASE Command

Permits you to create a duplicate database. Like the RMU/RESTORE command, the RMU/COPY_DATABASE command allows you to modify certain area parameters when the copy operation is performed. As in an RMU/BACKUP operation, all the files are processed simultaneously during an RMU/COPY_DATABASE operation. The RMU/COPY_DATABASE command's performance is similar to that of the RMU/BACKUP command. The RMU/COPY_DATABASE command eliminates the need for intermediate storage media.

Format

RMU/COPY_DATABASE root-file-spec storage-area-list

Command Qualifiers

/[NO]AFTER_JOURNAL[=file-spec]
/[NO]CHECKSUM_VERIFICATION
/DIRECTORY=directory-spec
/[NO]LOG
/NODES_MAX=n
/[NO]ONLINE
/OPTION=file-spec
/PAGE_BUFFERS=n
/ROOT=file-spec
/USERS_MAX=n

File or Area Qualifiers

/BLOCKS_PER_PAGE=n
/FILE=file-spec
/SNAPSHOTS=FILE=file-spec
/THRESHOLDS=(n,n,n)

Defaults

See description
/CHECKSUM_VERIFICATION
None
Current DCL verify value
Current value
/NOONLINE
None
n=3
None
Current value

Defaults

None
None
None
None

Command Parameters

root-file-spec

The name of the database root file for the database you wish to duplicate.

RMU/COPY_DATABASE Command

storage-area-list

The name of one or more storage areas whose parameters you are changing. The storage-area-list parameter is optional. Unless you are using the RMU /COPY_DATABASE command to modify the parameters of one or more storage areas, you should not specify any storage area names.

Command Qualifiers

/AFTER_JOURNAL=file-spec

/NOAFTER_JOURNAL

The /AFTER_JOURNAL qualifier creates a new after-image journal file and turns on journaling. The /NOAFTER_JOURNAL qualifier turns off journaling. The default is to retain the current journaling state and to create a new journal file. You facilitate recovery by creating a new journal file because a single journal file cannot be applied across a COPY_DATABASE operation that changes an area page size.

/CHECKSUM_VERIFICATION

/NOCHECKSUM_VERIFICATION

Requests that the page checksum be verified for each page copied. The default is to perform this verification.

/DIRECTORY=directory-spec

Specifies the default destination for the copied areas. There is no default for this qualifier, and it may be omitted.

/LOG

/NOLOG

Specifies whether to report the process of the command on SYSSOUTPUT. Specify the /LOG qualifier to request log output and the /NOLOG qualifier to prevent it. If you specify neither, the default is the current setting of the DCL verify switch. (The DCL SET VERIFY command controls the DCL verify switch.)

/NODES_MAX=n

Specifies a new value for the database MAXIMUM CLUSTER NODES parameter. The default is to leave the value unchanged.

RMU/COPY_DATABASE Command

/ONLINE

/NOONLINE

Specifies that the COPY_DATABASE operation be performed while other users are attached to the database. The areas to be moved are locked for read-only access, so the operation is compatible with all but EXCLUSIVE access.

The default is /NOONLINE.

/OPTION=file-spec

Specifies an option file with a format exactly like the RMU/RESTORE command's option file. There is no default for this qualifier. If this qualifier is specified, the storage-area-list parameter is ignored.

/PAGE_BUFFERS=n

Specifies the number of buffers to be allocated for each file to be moved. The number of buffers used is twice the number specified; half are used for reading the file and half for writing the copy. Values specified with the /PAGE_BUFFERS qualifier may range from 1 to 5. The default value is 3. Larger values may improve performance, but they increase memory usage.

/ROOT=file-spec

Requests that the root file be copied to the specified location.

/USERS_MAX=n

Specifies a new value for the database MAXIMUM CONCURRENT USERS parameter. The default is to leave the value unchanged.

/BLOCKS_PER_PAGE=n

Specifies a new page size for the storage area to which it is applied. You cannot decrease the page size of a storage area, and you cannot change the size of a uniform storage area. The /BLOCKS_PER_PAGE qualifier is a positional qualifier.

/FILE=file-spec

Specifies a new location for the storage area to which it is applied. The /FILE qualifier is a positional qualifier.

/SNAPSHOTS=FILE=file-spec

Specifies a new snapshot file location for the storage area to which it is applied. The /SNAPSHOTS qualifier is a positional qualifier.

RMU/COPY_DATABASE Command

/THRESHOLDS=(n,n,n)

Specifies new SPAM thresholds for the storage area to which it is applied. The thresholds of a uniform format storage area cannot be changed. The **/THRESHOLDS** qualifier is a positional qualifier.

Usage Notes

You must have the Rdb/VMS ADMINISTRATOR or OPERATOR (SQL DBADM or OPER) privilege, and access to the database files to use the RMU/COPY_DATABASE command. To have access to the database files, you must have default ACLs set up, or the VMS SYSPRV privilege, or RMU must be installed with SYSPRV on your system.

The parameter (file and area) qualifiers for the RMU/COPY_DATABASE command have positional semantics, and when placed randomly may be ignored or produce unexpected results. See Section 6.2 for more information on parameter qualifiers.

Examples

Example 1

The following command makes a duplicate copy of the MF_PERSONNEL database in the DDV12:[RICK.SQL] directory:

```
$ RMU/COPY_DATABASE MF_PERSONNEL /DIRECTORY=DDV12:[RICK.SQL]
```

RMU/DUMP Command

6.13 RMU/DUMP Command

Displays or writes to a specified output file the contents of database, storage area, and snapshot files, including root information.

Use this command to examine the contents of your database (RDB), storage area (RDA), and snapshot (SNP) files, to display current settings for database definition options, and to display a list of active database users. The list of database users is maintained clusterwide in a VAXcluster environment.

You can display the contents of all pages in the data storage area of the database or display the contents of just those pages in which records and indexes for a specific table are stored.

See the chapter that explains the internal database page format in the *VAX Rdb/VMS Guide to Database Maintenance and Performance* for tutorial information.

Format

RMU/DUMP database-file-name

File Qualifiers

/[NO]AREAS [= storage-area-list]
/OUTPUT = file-name
/[NO]HEADER
/OPTION = {NORMAL | FULL | DEBUG}
/[NO]SNAPSHOTS [= storage-area-list]
/[NO]LAREAS [= logical-area-list]
/START = integer
/END = integer
/[NO]USERS
/STATE=BLOCKED

Defaults

/NOAREAS
/OUTPUT=SYS\$OUTPUT
See description
/OPTION = NORMAL
/NOSNAPSHOTS
/NOLAREAS
/START = first-page
/END = last-page
/NOUSERS
See description

Note The */START* and */END* qualifiers apply only when the */AREAS*, */LAREAS*, or */SNAPSHOTS* qualifier is specified.

RMU/DUMP Command

Description

Depending on your selection of qualifiers, the RMU/DUMP command can list:

- A formatted display of any number of pages in the storage area of the database.
- A formatted display of any number of pages in a logical area of the database.
- A formatted display of any number of pages in the snapshot area of the database.
- Header information. This is listed by default if no qualifiers are specified.
- Current users of the database.

Command Parameter

database-file-name

The database whose root file header information, user information, storage area file pages, or snapshot area file pages you wish to display.

Command Qualifiers

/OUTPUT=file-name

The name of the file where output will be sent. SYSS\$OUTPUT is the default. The default output file type is LIS, if you specify a file name.

/HEADER

/NOHEADER

Indicates whether to include the database header in the output. Specify the /HEADER qualifier to include database header information in the output. Specify the /NOHEADER qualifier to suppress the database header listing.

The default is /HEADER if the RMU/DUMP command does not call for a storage display or a display of snapshot files. Otherwise, /NOHEADER is the default.

/OPTION=type

The type of information and level of detail the output will include. Three types of output are available:

- NORMAL

The output includes summary information. This is the default.

RMU/DUMP Command

- FULL

In addition to the NORMAL information, the output includes more detailed information.

- DEBUG

In addition to NORMAL and FULL information, the output includes internal information about the data. In general, use DEBUG for diagnostic support purposes. You can also use the DEBUG option to extract data and perform an independent analysis.

/SNAPSHOTS [= storage-area-list]

/NOSNAPSHOTS

Specifies a display that consists of snapshot file pages. The RMU/DUMP command does not display snapshot pages if you omit the /SNAPSHOTS qualifier or if you specify the /NOSNAPSHOTS qualifier.

In a single-file database, there is only one snapshot file. In a multifile database, each storage area has a corresponding snapshot file. Note that this parameter argument is the storage area name, not the snapshot file name. If you specify more than one storage area name, separate the storage area names with commas, and enclose the storage-area-list in parentheses. If you specify the /SNAPSHOTS qualifier without a storage area name, information is displayed for all snapshot file pages.

You can use the /START and /END qualifiers to display a range of snapshot file pages.

The default is /NOSNAPSHOTS.

/AREAS [= storage-area-list]

/NOAREAS

Specifies a display that consists of storage area pages. If you specify more than one storage area name, separate the storage area names in the storage area list with a comma and enclose the list in parentheses.

You can also specify /AREAS=* to display all storage areas. You can use the /START and /END qualifiers to display a range of storage area pages.

The default is /NOAREAS.

RMU/DUMP Command

/LAREAS [= logical-area-list]

/NOLAREAS

Specifies a display that consists of storage area pages allocated to a logical area or areas. In a single-file database, each table in the database is stored in its own logical area.

You cannot use the */LAREAS* qualifier with storage areas that have a mixed page format.

If you specify more than one logical area name, separate the storage area names in the logical area list with a comma and enclose the list in parentheses.

You can also specify */LAREAS=** to display all logical areas.

The default is */NOLAREAS*.

/START=integer

Specifies the lowest-numbered area or snapshot page to include in the display. The default is the first page; that is, */START=1*.

If you also use the */LAREAS* option, note that the */START* and */END* qualifiers now specify a page range relative to the logical area, not a specific storage area page number.

/END=integer

Specifies the highest-numbered area or snapshot page to include in the display. The default is the last page.

If you also use the */LAREAS* option, note that the */START* and */END* qualifiers now specify a page range relative to the logical area, not a specific storage area page number.

/USERS

/NOUSERS

The */USERS* option lists information about the current users of the database, including all users in a VAXcluster environment. */NOUSERS* is the default.

/STATE=BLOCKED

Specifies a list of all unresolved distributed transactions in the blocked database. A blocked database is a database that is not committed or rolled back and is involved in an unresolved distributed transaction. The

RMU/DUMP Command

`/STATE=BLOCKED` option displays the following information about each transaction:

- Process identification (PID)
- Stream identification
- Monitor identification
- Transaction identification
- Name of the recovery journal
- Transaction sequence number (TSN)
- Distributed TID
- Name of the node on which the failure occurred
- Name of the node initiating the transaction (parent node)

You can use the `/STATE=BLOCKED` qualifier only with the `/USERS` qualifier. For more information on resolving unresolved transactions with the `RMU/DUMP/USER` command, see the *VAX Rdb/VMS Guide to Distributed Transactions*.

Usage Notes

You must have the VMS SYSPRV privilege to use the `RMU/DUMP` command.

Examples

Example 1

This example displays to `SYSS$OUTPUT` the header information for the `PERSONNEL` database.

```
$ RMU/DUMP PERSONNEL
```

Example 2

This example generates a list of unresolved transactions for the `PERSONNEL` database.

```
$ RMU/DUMP/USERS/STATE=BLOCKED PERSONNEL
```

Note See the *VAX Rdb/VMS Guide to Database Maintenance and Performance* and the *VAX Rdb/VMS Guide to Distributed Transactions* for more examples showing the `RMU/DUMP` command.

RMU/DUMP/AFTER_JOURNAL Command

6.14 RMU/DUMP/AFTER_JOURNAL Command

Displays an after-image journal (AIJ) file in ASCII format. Use this command to examine the contents of your AIJ file.

An AIJ file contains header information and data blocks. Header information describes the data blocks, which contain copies of data stored in the database file.

Format

RMU/DUMP/AFTER_JOURNAL *ajj-file-name*

File Qualifiers

/OUTPUT = *file-name*
/NO]DATA
/START = *integer*
/END = *integer*
/STATE=PREPARED

Defaults

/OUTPUT=SYS\$OUTPUT
/DATA
/START = 1 <*data block*>
/END = <*end-of-file*>
None

Description

The RMU/DUMP/AFTER_JOURNAL command specifies an AIJ file, not a database file, as its parameter, and is a separate command from the RMU/DUMP command used to display database areas and header information.

The AIJ file is in binary format. This command translates the binary file into an ASCII display format.

The RMU/DUMP/AFTER_JOURNAL command always includes the header of the AIJ file in the display. You can use the /NODATA qualifier to exclude data blocks from the display entirely, or you can use the /START and /END qualifiers to restrict the data block display to a specific series of blocks. If you do not specify any of these qualifiers, RMU includes all data blocks.

RMU/DUMP/AFTER_JOURNAL Command

Command Parameter

aij-file-name

The AIJ file you want to display. The default file type is AIJ.

Command Qualifiers

/OUTPUT=file-name

The name of the file where output will be sent. SYSS\$OUTPUT is the default. The default output file type is LIS, if you specify a file name.

/DATA

/NODATA

Specifies whether you want to display data blocks of the AIJ file, or just the AIJ file header.

The */DATA* qualifier is the default. It causes the display of the AIJ file data blocks (in addition to the file header) in an ASCII display format.

The */NODATA* qualifier limits the display to the record headers of the AIJ file.

/START=integer

The number of the first data block that you want to display. If you do not use the */START* qualifier, the display begins with the first record in the AIJ file.

/END=integer

The number of the last data block that you want to display. The default integer is the number of the last data block in the file. If you do not use the */END* qualifier, RMU displays the entire AIJ file.

/STATE=PREPARED

Specifies a list of all records associated with unresolved transactions.

For more information on listing unresolved transactions with the RMU/DUMP /AFTER_JOURNAL/STATE=PREPARED command, see the *VAX Rdb/VMS Guide to Distributed Transactions*.

RMU/DUMP/AFTER_JOURNAL Command

Usage Notes

You must have the VMS SYSPRV privilege to use the RMU/DUMP/AFTER_JOURNAL command.

You receive a file access error message regarding the database's AIJ file if you issue the RMU/DUMP/AFTER_JOURNAL command when there are active processes updating the database. To avoid the file access error message, use the RMU/CLOSE command to close the database (which stops entries to the AIJ file), then issue the RMU/DUMP/AFTER_JOURNAL command.

Examples

Example 1

The following command generates a list of records associated with unresolved transactions in the AIJ file:

```
$ RMU/DUMP/AFTER_JOURNAL/STATE=PREPARED PERSONNEL.AIJ
```

See the *VAX Rdb/VMS Guide to Database Maintenance and Performance* and the *VAX Rdb/VMS Guide to Distributed Transactions* for more examples of the RMU/DUMP/AFTER_JOURNAL command.

RMU/DUMP/BACKUP_FILE Command

6.15 RMU/DUMP/BACKUP_FILE Command

Displays or writes to a specified output file the contents of a backup file. Use this command to examine the contents of a backup (RBF) file created by the RMU/BACKUP command.

Format

RMU/DUMP/BACKUP_FILE backup-file-spec

Command Qualifiers

/[NO]REWIND
/BLOCK_SIZE=integer
/ACTIVE_IO=max-reads
/OUTPUT=file-name
/OPTIONS=options-list
/SKIP=skip-list
/PROCESS=process-list
/LABEL[(label-name-list)]

Defaults

/NOREWIND
Existing value
/ACTIVE_IO=3
/OUTPUT=SYS\$OUTPUT
Dump nothing
Skip nothing
Process everything
See description

Description

If you do not specify the /OPTIONS qualifier or if you specify /OPTIONS=NORMAL, the backup file will be read, but dump output will not be generated. This is useful to verify the backup file integrity and to detect media errors.

The RMU/DUMP/BACKUP_FILE command displays contents of an RBF file. It uses an RBF file, not a database file, as its parameter, and is a separate command from the RMU/DUMP command.

Command Parameter

backup-file-spec

A file specification for the backup file. The default file type is RBF.

If you use multiple tape drives, the backup-file-spec must include the tape device specifications. Separate the device specifications with commas.

```
$ RMU/DUMP/BACKUP_FILE $111$MUA0:PERS_FULL.RBF,$112$MUA1:
```

RMU/DUMP/BACKUP_FILE Command

When multiple volume tape files are processed, RMU dismounts and unloads all but the last volume containing the file, which is the customary practice for multiple volume tape files. See the *VAX Rdb/VMS Guide to Database Maintenance and Performance* for more information on using multiple tape drives.

Command Qualifiers

/REWIND

/NOREWIND

The ***/REWIND*** qualifier specifies that the magnetic tape that contains the backup file will be rewound before processing begins. The ***/NOREWIND*** qualifier is the default.

The ***/REWIND*** and ***/NOREWIND*** qualifiers are applicable only to tape devices. You should use these qualifiers only when the target device is a tape device.

See the Usage Notes for information on tape label processing.

/BLOCK_SIZE=integer

The maximum record size for the backup file. The size can vary between 2048 and 65,024 bytes. The default value is device dependent. The ***/BLOCK_SIZE*** qualifier is meaningful only when you are writing the backup file to tape volumes. If you specify the ***/BLOCK_SIZE*** qualifier when writing to a disk device, the effect will be a change in the number of I/O operations needed to write the backup file and a degradation in the performance of the RMU/BACKUP command.

/ACTIVE_IO=max-reads

The maximum number of read operations to the backup file that the RMU/BACKUP command will attempt simultaneously. The value of the ***/ACTIVE_IO*** qualifier may range from one to five. The default value is three. Values larger than three may improve performance with multiple tape drives.

/OUTPUT=file-name

The name of the file where output will be sent. SYSSOUTPUT is the default. The default output file type is LIS, if you specify a file name.

/OPTIONS=options-list

The type of information and level of detail the output will include. If you do not specify the ***/OPTIONS*** qualifier or if you specify ***/OPTIONS=NORMAL***, the backup file will be read, but dump output will not be generated. This is useful to verify the backup file integrity and to detect media errors.

RMU/DUMP/BACKUP_FILE Command

If you specify more than one type of /OPTIONS output, you must separate the options with a comma and enclose the options-list in parentheses. Seven types of output are available:

- RECORDS
Dumps the backup file record structure.
- BLOCKS
Dumps the backup file block structure.
- DATA
The DATA option can be used with either the RECORDS option, the BLOCKS option, or both. When specified with the RECORDS and BLOCKS options, the DATA option dumps the contents of the backup file's records and blocks. When you do not specify the DATA option, the RECORDS and BLOCKS options dump the backup file's record structure and block structure only, not their contents.
- ROOT
Dumps the database root file contents as recorded in the backup file.
- NORMAL
The backup file will be read, but no dump output is generated. This is useful to verify the backup file integrity and to detect media errors.
- FULL
This is the same as specifying ROOT, RECORDS, and BLOCKS. The contents of the backup file's record structure and block structure are not dumped when FULL is specified.
- DEBUG
This is the same as specifying ROOT, RECORDS, BLOCKS, FULL, and DATA. The contents of the backup file's record structure and block structure are dumped when DEBUG is specified.

/SKIP=skip-list

A list of keywords that determines where the output display begins. The keywords indicate the position in the backup file from which to start the dump. If you specify more than one type of /SKIP position, separate the options with

RMU/DUMP/BACKUP_FILE Command

a comma and enclose the skip-list in parentheses. You can specify the following three items in the skip-list:

- **VOLUMES=integer**
The number of volumes to ignore before starting.
- **BLOCKS=integer**
The number of blocks to ignore before starting.
- **RECORDS=integer**
The number of records to ignore before starting.

/PROCESS=process-list

A list of keywords that determines how much of the backup file is to be dumped. If you specify more than one type of /PROCESS position, separate the options with a comma and enclose the process-list in parentheses. You can specify the following three items in the process-list:

- **VOLUMES=integer**
The number of volumes to dump, starting at the position specified in the /SKIP qualifier for volumes.
- **BLOCKS=integer**
The number of blocks to dump, starting at the position specified in the /SKIP qualifier for blocks.
- **RECORDS=integer**
The number of records to dump, starting at the position specified in the /SKIP qualifier for records.

/LABEL[=(label-name-list)]

The one- to six-character string with which the volumes of the backup file are to be labeled. The /LABEL qualifier is applicable only to tape volumes. If it is not specified, the label will be derived from the first six characters of the backup file name.

You can specify a list of tape labels for multiple tapes. If you list multiple tape label names, separate the names with commas and enclose the list of names in parentheses.

RMU/DUMP/BACKUP_FILE Command

In a normal restore operation, the /LABEL qualifier you specify with the RMU /RESTORE command should be the same /LABEL qualifier as you specified with the RMU/BACKUP command that backed up your database.

See the Usage Notes for information on tape label processing.

Usage Notes

You must have the VMS SYSPRV privilege to use the RMU/DUMP/BACKUP_FILE command.

You must also have VMS VOLPRO privilege on systems with VMS Version 5.0 or higher if you are using a multivolume tape set. For you to mount foreign volumes, your UIC must match the owner UIC on the tape.

The steps in tape label checking are:

- 1 Use the DCL MOUNT/FOREIGN command to mount the tape you want to use.

You must mount the first tape volume; RMU mounts subsequent volumes automatically. If VMS encounters an error when mounting the tape, either the RMU function will abort or the problem will be reported and the operation retried. If a severe error occurs, RMU does not mount subsequent volumes.

- 2 The tape characteristics are checked.

For an RMU/BACKUP operation, the tape must be properly mounted and cannot be write protected. If these checks fail, you are asked to mount the correct relative volume with write protection and to indicate when you are finished.

- 3 The tape labels are checked.

If the volume label disagrees with the label specified with the /LABEL qualifier or by the default (the backup file name), you receive an informational message.

If the tape is not the first volume, or if you specified the /REWIND qualifier in the command, RMU checks the tape protection (the user-id specified with the /OWNER_ID=user-id qualifier of the RMU/BACKUP command) and the expiration date of the tape, and issues an informational message regarding the error. You can read a tape that has not expired.

RMU/DUMP/BACKUP_FILE Command

If any of the checks fail, you are asked what you want to do with the tape volume. You must select one of the following options:

- **QUIT**
This option cancels the operation, and causes you to exit from RMU.
- **RETRY**
This option repeats the checks after dismounting and remounting the same tape. The **RETRY** option is useful if you left the drive off line, or if there was some other kind of correctable error.
- **UNLOAD**
This option dismounts and unloads the tape so you can load the correct tape on the drive.
- **OVERRIDE**
This option overrides the failed checks and uses the tape anyway. The **OVERRIDE** option can be used with the **RMU/RESTORE** and **RMU/DUMP/BACKUP_FILE** commands only.
- **INITIALIZE**
This option requests that the tape be rewound and relabeled. The **INITIALIZE** option can be used with the **RMU/BACKUP** command only.

4 The tape is remounted and reverified as requested.

Tape changes follow essentially the same procedure, with one difference. If the first volume is not at the beginning of the tape, complete label checking is not available.

Note See the VAX Rdb/VMS Guide to Database Maintenance and Performance for examples that show the **RMU/DUMP/BACKUP_FILE** command.

RMU/DUMP/RECOVERY_JOURNAL Command

6.16 RMU/DUMP/RECOVERY_JOURNAL Command

Displays a recovery-unit journal (RUJ) file in ASCII format. Use this command to examine the contents of an RUJ file. You may find RUJ files on your system following a system failure.

An RUJ file contains header information and data blocks. Header information describes the data blocks, which contain copies of data modified in the database file.

Format

RMU/DUMP/RECOVERY_JOURNAL *ruj-file-name*

Command Qualifiers

/OUTPUT = file-name
/[NO]DATA

Defaults

/OUTPUT=SYS\$OUTPUT
/DATA

Description

The RMU/DUMP/RECOVERY_JOURNAL command specifies an RUJ file, not a database file, as its parameter, and is a separate command from the RMU/DUMP command used to display database areas and header information.

The RUJ file is in binary format. This command translates the binary file into an ASCII display format.

Command Parameter

ruj-file-name

The RUJ file. The default file type is RUJ.

Command Qualifiers

/OUTPUT=file-name

The name of the file where output will be sent. SYS\$OUTPUT is the default. The default output file type is LIS, if you specify a file name.

RMU/DUMP/RECOVERY_JOURNAL Command

/DATA

/NODATA

Specifies whether you want to display data blocks of the RUJ file or just the RUJ file header.

The /DATA qualifier is the default. It causes the display of the RUJ file data blocks (in addition to the file header) in an ASCII display format.

The /NODATA qualifier limits the display to the file header of the RUJ file.

Usage Notes

You must have the VMS SYSPRV privilege to use the RMU/DUMP/RECOVERY_JOURNAL command.

Note *See the VAX Rdb/VMS Guide to Database Maintenance and Performance for examples showing the RMU/DUMP/RECOVERY_JOURNAL command.*

RMU/LOAD Command

6.17 RMU/LOAD Command

Loads tables of the database from any of the following:

- Sequential RMS files created by an application program or by the RMU/UNLOAD command
- Specially structured files (file type UNL) created using the RMU/UNLOAD command
- A VMS security audit journal

Format

RMU/LOAD database-file-name table-name input-file-name

Command Qualifiers

/AUDIT
/BUFFERS=n
/COMMIT_EVERY=n
/FIELDS=(column-name-list)
/[NO]PLACE
/RMS_RECORD_DEF=option
/[NO]SKIP=n
/TRANSACTION_TYPE=share-mode
/[NO]TRIGGER_RELATIONS[=(table-name-list)]

Defaults

No audit table loaded
See description
See description
See description
/NOPLACE
See description
/NOSKIP
PROTECTED
/TRIGGER_RELATIONS

Description

You can use the RMU/LOAD command to:

- Perform the initial load of an Rdb/VMS database
- Reload a table after performing a restructuring operation
- Load an archival database
- Move data from one database to another
- Load security audit records from a VMS security audit table into an Rdb/VMS table

RMU/LOAD Command

To load a database from sequential RMS files, use the `/RMS_RECORD_DEF` qualifier to specify the record structure of the RMS sequential file that contains the data. You can use `RMU/LOAD` to load data from an RMS file. The records must be fixed-length, and the data in each record must fill the entire record. If the last field is character data and the information is shorter than the field length, the remainder of the field must be filled with spaces. You cannot load a field that contains data stored in packed decimal format.

The specially structured files created by the `RMU/UNLOAD` command contain metadata for the table that was unloaded. The RMS files contain only data; the metadata can be found either in the data dictionary or in a file created using the `/RMS_RECORD_DEF` qualifier with the `RMU/UNLOAD` command. The file created using the `/RMS_RECORD_DEF` qualifier is file type `RRD` by default. The records in the file you load from must match the record definition in both size and data type. Note that `Rdb/VMS` does not issue an error message if the records and the record definitions do not match.

If you have enabled security auditing for your database using the `RMU/SET AUDIT` command, you can use the `RMU/LOAD/AUDIT` command to load audit records from the security audit journal into a table in your database. The `Rdb/VMS` table that you load the security audit journal records into should be defined with the columns shown in Table 6–1 so that the audit journal records can be loaded successfully into the table. If the table does not exist, the `RMU/LOAD/AUDIT` command will create it with the columns shown in Table 6–1. The `RDO TEXT` data type is equivalent to the `SQL CHAR` data type. You can give the table any valid name.

Table 6–1 Columns in a Database Table for Storing Security Audit Journal Records

Column Name	SQL Data Type and Length
<code>RDBVMS\$EVENT</code>	<code>CHAR 16</code>
<code>RDBVMS\$SYSTEM_NAME</code>	<code>CHAR 15</code>
<code>RDBVMS\$SYSTEM_ID</code>	<code>CHAR 12</code>
<code>RDBVMS\$TIME_STAMP</code>	<code>CHAR 48</code>
<code>RDBVMS\$PROCESS_ID</code>	<code>CHAR 12</code>
<code>RDBVMS\$USER_NAME</code>	<code>CHAR 12</code>

(continued on next page)

RMU/LOAD Command

Table 6-1 (Cont.) Columns in a Database Table for Storing Security Audit Journal Records

Column Name	SQL Data Type and Length
RDBVMS\$TSN	CHAR 12
RDBVMS\$OBJECT_NAME	CHAR 255
RDBVMS\$OBJECT_TYPE	CHAR 12
RDBVMS\$OPERATION	CHAR 32
RDBVMS\$DESIRED_ACCESS	CHAR 16
RDBVMS\$SUB_STATUS	CHAR 32
RDBVMS\$FINAL_STATUS	CHAR 32
RDBVMS\$RDB_PRIV	CHAR 16
RDBVMS\$VMS_PRIV	CHAR 16
RDBVMS\$GRANT_IDENT	CHAR 192
RDBVMS\$NEW_ACE	CHAR 192
RDBVMS\$OLD_ACE	CHAR 192
RDBVMS\$RMU_COMMAND	CHAR 512

Performance of the RMU/LOAD operation can be adversely affected by the presence of indexes, constraints, triggers, or COMPUTED-BY fields. If you use RMU/LOAD for the initial load operation, define only placement indexes before the load; define constraints, triggers, and other indexes after you load the data. During the RMU/LOAD operation, triggers are automatically disabled.

By default, the named table is reserved for PROTECTED WRITE.

Table 6-2 shows the data type conversions that can occur while you are performing a load/unload operation.

RMU/LOAD Command

Table 6-2 Data Type Conversions Performed by Rdb/VMS

Original Data Type	New Data Type
SIGNED BYTE	SIGNED LONGWORD or SIGNED QUADWORD
SIGNED WORD	SIGNED LONGWORD or SIGNED QUADWORD
SIGNED LONGWORD	SIGNED WORD or SIGNED QUADWORD
SIGNED QUADWORD	SIGNED WORD or SIGNED LONGWORD
F_FLOATING	G_FLOATING, TEXT, and VARYING STRING
G_FLOATING	F_FLOATING, TEXT, and VARYING STRING
DATE	TEXT or VARYING STRING
TEXT	F_FLOATING, G_FLOATING, DATE, and VARYING STRING

Command Parameters

database-file-name

The name of the database into which the table will be loaded. The default file type is RDB.

When the /AUDIT qualifier is specified, the Rdb/VMS audit records in the security audit journal for the specified database are searched to find those audit records generated for that database name.

table-name

The name of the table to be loaded.

When the /AUDIT qualifier is specified, the table-name is the name of the table in which you want the security audit journal records to be loaded. If the table does not exist, RMU/LOAD/AUDIT will create the table. If the table does exist, RMU/LOAD/AUDIT will load the table.

input-file-name

The name of the file containing the data to be loaded. The default file type is UNL.

When the /AUDIT qualifier is specified, the input-file-name is the name of the journal containing the audit record data to be loaded. The default file type is AUDIT\$JOURNAL. You can determine the name of the security audit journal by using the DCL command SHOW AUDIT/JOURNAL.

RMU/LOAD Command

Command Qualifiers

/AUDIT

Allows you to load a database's security audit records from a VMS security audit journal into an Rdb/VMS table.

If you specify the */AUDIT* qualifier, you cannot specify the */FIELDS*, */TRIGGER_RELATIONS*, or */RMS_RECORD_DEF* qualifiers.

/BUFFERS=n

Specifies the number of database buffers used for storing data during the RMU/LOAD operation. If no value is specified, the default value for the database is used. (You can determine the default value for the database using the RMU/DUMP/HEADER command.) Less I/O is required if you can store as much data as possible in memory when many indexes or constraints are defined on the target table. Therefore, specify more buffers than allowed by the default value to increase the speed of the RMU/LOAD operation.

/COMMIT_EVERY=n

Specifies the frequency with which Rdb/VMS commits the data being loaded. Rdb/VMS will commit the data after every *n* records that are stored. The default is to commit only after all records have been stored. To determine how frequently you should commit data, decide how many records you are willing to reload if the original RMU/LOAD operation fails. You will receive a message indicating the number of records loaded at each commit. If a failure occurs, you can determine where by specifying */COMMIT_EVERY=1*, and */SKIP=n*, where *n* equals the number of records already loaded. The load will then commit at every record until it again reaches the point of failure.

/FIELDS=(column-name-list)

Specifies the column or columns of the table to be loaded into the database. If you list multiple columns, separate the column names with a comma, and enclose the list of column names in parentheses. This qualifier also specifies the order of the columns to be loaded if that order differs from the order defined for the table. The number and data type of the columns specified must agree with the number and data type of the columns in the input file. The default is all columns defined for the table in the order defined.

/PLACE

/NOPLACE

Sorts records by target page number before they are stored.

RMU/LOAD Command

The `/PLACE` qualifier automatically builds an ordered set of dbkeys when loading data and automatically stores the records in dbkey order, sequentially, page by page.

A significant performance improvement occurs when the records are stored by means of a hashed index. By using the `/COMMIT` qualifier with the `/PLACE` qualifier, you can specify how many records to load between `COMMIT` statements.

The default is `/NOPLACE`.

/RMS_RECORD_DEF=option

Specifies the RMS record definition or the data dictionary record definition to be used when data is loaded into the database. The default file type is `RRD`. The syntax for the `RRD` file is similar to that used by the `CDO`. You must define columns before you can define rows. You can place only one column on a line. You can create a sample `RRD` file using the `RMU/UNLOAD` command with the `/RMS_RECORD_DEF` qualifier. You must ensure that the record definition in the `RRD` file and the actual data are consistent with each other. `Rdb/VMS` does not check to see that data types in the record definition and the data match. See the *VAX Rdb/VMS Guide to Database Maintenance and Performance* for more information about the format of the `RRD` file. The options available are:

- **FILE=file-name**
Specify `FILE=file-name` when the record definition is in an RMS file. The record definition uses the `CDO` record and field definition format.
- **PATH=pathname**
Specify `PATH=pathname` when the record definition is to be extracted from the data dictionary. If the record definition in the data dictionary contains variants, `RMU` will not be able to extract it.

/SKIP=n

/NOSKIP

Ignores the first `n` data records in the input file. Use this qualifier in conjunction with the `/COMMIT EVERY` qualifier when restarting an aborted load operation. An aborted load operation displays a message indicating how many records have been committed. Use this value for `n`. If you specify a negative number, you will get an error message. If you specify a number greater than the number of records in the file, you will get an error message saying that no records have been stored. The default is `/NOSKIP`.

RMU/LOAD Command

/TRANSACTION_TYPE=share-mode

Specifies the share mode for the RMU/LOAD operation. The following share modes are available:

- SHARED
- PROTECTED
- EXCLUSIVE
- BATCH_UPDATE

You must specify a value if you use the */TRANSACTION_TYPE* qualifier. If you do not specify the */TRANSACTION_TYPE* qualifier, the default share mode is PROTECTED.

/TRIGGER_RELATIONS[(table-name-list)]

/NOTRIGGER_RELATIONS

You can use the */TRIGGER_RELATIONS* qualifier in three ways:

- */TRIGGER_RELATIONS=(table-name-list)*

Specifies the tables to be reserved for WRITE transactions. Using this qualifier, you can explicitly lock tables that will be updated by triggers on STORE operations into the target table. If you list multiple tables, separate the table names with a comma and enclose the list of table names in parentheses.
- */TRIGGER_RELATIONS*

If you omit the list of table names, the tables updated by triggers will be locked automatically as required. This is the default.
- */NOTRIGGER_RELATIONS*

Disables triggers on the target table. This option requires DELETE access on the tables affected by the triggers, although the triggers are not actually deleted. You cannot specify a list of table names with this option.

Usage Notes

You must have either the Rdb/VMS ADMINISTRATOR or OPERATOR (SQL DBADM or OPERATOR) privilege and the Rdb/VMS WRITE (SQL INSERT) privilege to the table being loaded to use the RMU/LOAD command. You must also have VMS READ access to the file you are loading into the database.

RMU/LOAD Command

To use the RMU/LOAD/AUDIT command, you must have the Rdb/VMS or SQL SECURITY privilege or the VMS SECURITY privilege. You must also have VMS READ access to the security audit journal you are loading into the database.

Examples

Example 1

This command loads the data from the RMS file, NAMES.UNL, into the newly created RETIREES table of the PERSONNEL database. The record structure of RETIREES is in the file NAMES.RRD. The NAMES.UNL and NAMES.RRD files were created by a previous RMU/UNLOAD operation. The RMU/UNLOAD operation unloaded data from a view derived from a subset of columns in the EMPLOYEES table.

```
$ RMU/LOAD/RMS_RECORD_DEF=FILE=NAMES.RRD PERSONNEL RETIREES NAMES.UNL
```

Example 2

This command restarts an aborted RMU/LOAD operation that was loading the newly created RETIREES table of the PERSONNEL database from the NAMES.UNL file. The columns being loaded are EMPLOYEE_ID, LAST_NAME, FIRST_NAME. The original RMU/LOAD operation had committed 25 records. Beginning with the 26th record, the restarted RMU/LOAD operation will commit the transaction at every record until it reaches the original point of failure.

```
$ RMU/LOAD/FIELDS=(EMPLOYEE_ID, LAST_NAME, FIRST_NAME) -  
_$/COMMIT_EVERY=1/SKIP=25 PERSONNEL RETIREES NAMES.UNL
```

Example 3

This example loads a new table, PENSIONS, into the PERSONNEL database using record definitions located in the data dictionary.

This example assumes that you have first defined a temporary view, TEMP_PENSIONS, combining appropriate columns of the EMPLOYEES and SALARY_HISTORY tables. You must also create a permanent table, PENSIONS, into which you will load the data.

Unload the TEMP_PENSIONS view using the RMU/UNLOAD command with the /RMS_RECORD_DEF=FILE=filename qualifier to create both an RRD file containing the column definitions and a DATA.UNL file containing the data from the TEMP_PENSIONS view. Load the new record definitions from the PENSIONS.RRD file into the data dictionary by using the @ command at the

RMU/LOAD Command

CDO prompt. Then you can load the data into the PENSIONS table of the PERSONNEL database using the RMU/LOAD command.

```
$ RMU/UNLOAD/RMS_RECORD_DEF=FILE=PENSIONS.RRD PERSONNEL -
_ $ TEMP_PENSIONS DATA.UNL
$ DICTIONARY OPERATOR
Welcome to CDO V1.0
The CDD/Plus V4.1 User Interface
Type HELP for help
CDO> @PENSIONS.RRD
CDO> EXIT
$ RMU/LOAD/RMS_RECORD_DEF=PATH=PENSIONS PERSONNEL PENSIONS DATA.UNL
```

Example 4

The following command loads the audit records for the MF_PERSONNEL database from the security audit journal file into the AUDIT_TABLE table in the MF_PERSONNEL database. Note that if the AUDIT_TABLE table does not exist, the RMU/LOAD/AUDIT command creates it with the columns show in Table 6-1.

```
$ RMU/LOAD/AUDIT MF_PERSONNEL AUDIT_TABLE SYS$MANAGER:SECURITY_AUDIT
%RMU-I-DATRECSTO, 15 data records stored
```

RMU/MONITOR REOPEN_LOG Command

6.18 RMU/MONITOR REOPEN_LOG Command

Closes the current Rdb/VMS monitor log file and opens another one without stopping the monitor.

Format

```
RMU/MONITOR REOPEN_LOG
```

Description

The RMU/MONITOR REOPEN_LOG command closes the current Rdb/VMS monitor log file and opens another log file (SYSSYSTEM:RDMMON.LOG) without stopping the monitor. You or your database administrator should use the RMU/MONITOR REOPEN_LOG command if the monitor log file gets too large.

If the disk that contains the Rdb/VMS monitor log file becomes full, you, your database administrator, or your system manager must produce free space on the disk. Once there is sufficient space on this disk, use the RMU/MONITOR REOPEN_LOG command and consider backing up (using the VMS Copy or Backup utility) the old monitor log file on tape.

When the disk that contains the monitor log becomes full, Rdb/VMS stops writing to the log file, but the Rdb/VMS system does not stop operating.

Usage Notes

To use the RMU/MONITOR REOPEN_LOG command, you must have the VMS WORLD privilege.

Examples

Example 1

The existing monitor log file is closed and a new one created without stopping the Rdb/VMS monitor.

```
$ RMU/MONITOR REOPEN_LOG
```

RMU/MONITOR START Command

6.19 RMU/MONITOR START Command

Activates the Rdb/VMS monitor process.

Format

RMU/MONITOR START

Command Qualifiers

/OUTPUT = file-name

/PRIORITY = integer

/[NO]SWAP

Defaults

/OUTPUT=SYS\$SYSTEM:RDMMON.LOG

/PRIORITY = 15

/NOSWAP

Description

The RMU/MONITOR START command activates the Rdb/VMS monitor process (RDMS_MONITOR), sets the priority of this process, and specifies a device, directory, and file name in which to create the monitor log file. If the monitor process is active already, you will receive the following error message:

```
%RMU-F-MONMBXOPN, monitor is already running
```

An Rdb/VMS monitor process must be running on a VAX node for users logged into that node to use any Rdb/VMS database. In a VAXcluster environment, a monitor process must be running on each node in the VAXcluster from which databases will be accessed.

The Rdb/VMS monitor process controls all database access and initiates the automatic database recovery procedure following a system failure or abnormal database user process termination.

Command Qualifiers

/PRIORITY=integer

The base priority of the monitor process. This priority should always be higher than the highest database user process priority. This qualifier is valid only when used with the START option.

RMU/MONITOR START Command

By default, the monitor runs at the highest interactive priority possible, 15. You should not normally have to lower the monitor process priority. If you change this to a lower priority, an attach operation can deadlock. Deadlock occurs when multiple processes with higher priority than the monitor attempt to attach at the same time. Failure can occur because the monitor must contend for CPU time with multiple higher-priority processes and is perpetually locked out.

/SWAP
/NOSWAP

Enables or disables swapping of the monitor process. The default is */NOSWAP*. This qualifier is valid only when used with the *START* option. The */SWAP* qualifier is not recommended for time-critical applications.

/OUTPUT=file-name

The device, directory, and file name that will receive the monitor log. The default device and directory is the *SYSS\$SYSTEM* directory. The default log file name is *RDMMON.LOG*. You can use this qualifier to redirect the placement of your monitor log file.

Usage Notes

You must have the *VMS WORLD* privilege and the *VMS SETPRV* privilege to use the *RMU/MONITOR START* command.

If the monitor has not been previously started on the system, use the *RMONSTART.COM* command file in the *SYSS\$MANAGER* directory instead of the *RMU/MONITOR START* command.

Start the monitor from the *SYSTEM* account, which has the *SETPRV* privilege. The process starting the monitor attempts to give *RDMS_MONITOR* all privileges. In particular, the privileges required are *ALTPRI*, *CMKRNL*, *DETACH*, *PSWAPM*, *SETPRV*, *SYSGBL*, *SYSNAM*, and *WORLD*.

Examples

Example 1

The following command activates the *Rdb/VMS* monitor process:

```
$ RMU/MONITOR START
```

See the *VAX Rdb/VMS Guide to Database Maintenance and Performance* for more examples that show the *RMU/MONITOR* commands.

RMU/MONITOR STOP Command

6.20 RMU/MONITOR STOP Command

Stops the Rdb/VMS monitor process.

Format

RMU/MONITOR STOP

Command Qualifiers

/NOABORT
/ABORT [= { FORCEX | DELPRC }]
/[NO]WAIT

Defaults

/NOABORT
See description
/NOWAIT

Description

The RMU/MONITOR STOP command deactivates the Rdb/VMS monitor process (RDMS_MONITOR) normally, either with process rundown or an immediate abort. You can use the RMU/MONITOR STOP command to shut down all database activity on your VAX node, optionally aborting user processes by forcing an image exit or deleting their processes.

An Rdb/VMS monitor process must be running on a VAX node for users logged into that node to use any Rdb/VMS database. In a VAXcluster environment, a monitor process must be running on each node in the VAXcluster from which databases will be accessed.

The Rdb/VMS monitor process controls all database access and initiates the automatic database recovery procedure following a system failure or abnormal database user process termination.

Command Qualifiers

/NOABORT
/ABORT=FORCEX
/ABORT=DELPRC

If you enter the RMU/MONITOR STOP command with no qualifier, the default is **/NOABORT**. The **/NOABORT** qualifier allows current user processes to continue and run down before stopping. New users on the node will not be allowed to attach to any database, but existing database users will be able

RMU/MONITOR STOP Command

to complete their sessions normally. Once existing database user processes terminate, the database monitor shuts down.

The `/ABORT=FORCEX` qualifier stops the monitor immediately without allowing current Rdb/VMS users to complete active transactions or detach from their databases. However, the user processes are not deleted. Using the `/ABORT` qualifier with no option is equivalent to specifying the `/ABORT=FORCEX` qualifier.

The `/ABORT=DELPRC` qualifier stops the monitor immediately without allowing current Rdb/VMS users to complete active transactions or detach from their databases. Each user process that was attached to an Rdb/VMS database is deleted immediately.

`/WAIT`

`/NOWAIT`

Specifies whether the RMU operation completes when the monitor acknowledges the stop request (`/NOWAIT`), or whether RMU waits until the monitor finishes shutting down (`/WAIT`). The default is `/NOWAIT`.

Usage Notes

You must have VMS WORLD privilege to use the RMU/MONITOR STOP command.

Note If DECTrace for VMS software is installed on your system, you will hang the Rdb/VMS monitor process with the RMU/MONITOR STOP command unless you do one of the following:

- Shut down DECTrace, then shut down the Rdb/VMS monitor (in that order).
- Use the `RMU/MONITOR STOP/ABORT=DELPRC` command to shut down Rdb/VMS and force the monitor out of the DECTrace database.

RMU/MONITOR STOP Command

Examples

Example 1

The Rdb/VMS monitor process will shut down once existing database users end their access to the database. New users on this VAX node will be unable to attach to any Rdb/VMS database.

```
$ RMU/MONITOR STOP
```

Example 2

The Rdb/VMS monitor stops immediately without allowing current Rdb/VMS users to complete active transactions (they are rolled back) or detach (FINISH) from their databases. However, the user processes are not deleted. Because the monitor is shut down, all Rdb/VMS activity on this VAX node is terminated.

```
$ RMU/MONITOR STOP /ABORT=FORCEX
```

Example 3

The Rdb/VMS monitor stops immediately without allowing current Rdb/VMS users to complete active transactions (they are rolled back) or detach (FINISH) from their databases. Each user process that was attached to an Rdb/VMS database on this node is deleted immediately.

```
$ RMU/MONITOR STOP /ABORT=DELPRC
```

6.21 RMU/MOVE_AREA Command

Permits you to move one or more storage areas to different disks. You can also choose to move the root file to a different disk. Like the RMU/RESTORE command, the RMU/MOVE_AREA command lets you modify certain area parameters when the move operation is performed. As in an RMU/BACKUP operation, all the files are processed simultaneously during the move operation. The RMU/MOVE_AREA command's performance is similar to that of the RMU/BACKUP command, and it eliminates the need for intermediate storage media.

Format

RMU/MOVE_AREA root-file-spec storage-area-list

Command Qualifiers

/[NO]AFTER_JOURNAL[=file-spec]
 /[NO]AREA
 /[NO]CHECKSUM_VERIFICATION
 /DIRECTORY=directory-spec
 /[NO]LOG
 /NODES_MAX=n
 /OPTION=file-spec
 /PAGE_BUFFERS=n
 /ROOT=file-spec
 /USERS_MAX=n

File or Area Qualifiers

/BLOCKS_PER_PAGE=n
 /FILE=file-spec
 /SNAPSHOTS=FILE=file-spec
 /THRESHOLDS=(n,n,n)

Defaults

See description
 /AREA
 /CHECKSUM_VERIFICATION
 None
 Current DCL verify value
 Keep current value
 None
 n=3
 None
 Keep current value

Defaults

None
 None
 None
 None

Command Parameters

root-file-spec

The name of the database root file for the database whose storage areas you wish to move.

RMU/MOVE_AREA Command

storage-area-list

The name of one or more storage areas that you want to move.

Command Qualifiers

/AFTER_JOURNAL=file-spec

/NOAFTER_JOURNAL

The ***/AFTER_JOURNAL*** qualifier creates a new after-image journal file and turns on journaling. The ***/NOAFTER_JOURNAL*** qualifier turns off journaling. The default is to retain the current journaling state and to create a new journal file. You facilitate recovery by creating a new journal file because a single journal file cannot be applied across a MOVE_AREA operation that changes an area page size.

/AREA

/NOAREA

Controls whether or not specific storage areas are moved. If you specify the ***/AREA*** qualifier, only the storage areas specified in the option file or the storage-area-list are moved. If you specify ***/NOAREA***, all the storage areas in the database are moved. The default is ***/AREA***.

/CHECKSUM_VERIFICATION

/NOCHECKSUM_VERIFICATION

Requests that the page checksum be verified for each page moved. The default is to perform this verification.

/DIRECTORY=directory-spec

Specifies the default destination for the moved areas. There is no default for this qualifier, and it may be omitted.

/LOG

/NOLOG

Specifies whether to report the process of the command on SYSS\$OUTPUT. Specify the ***/LOG*** qualifier to request log output and the ***/NOLOG*** qualifier to prevent it. If you specify neither, the default is the current setting of the DCL verify switch. (The DCL SET VERIFY command controls the DCL verify switch.)

/NODES_MAX=n

Specifies a new value for the database MAXIMUM CLUSTER NODES parameter. The default is to leave the value unchanged.

Use the ***/NODES_MAX*** qualifier only if you move the database root file.

RMU/MOVE_AREA Command

/OPTION=file-spec

Specifies an option file with a format exactly like the RMU/RESTORE command's option file. There is no default for this qualifier. If the /OPTION qualifier is specified, the storage-area-list parameter is ignored.

/PAGE_BUFFERS=n

Specifies the number of buffers to be allocated for each file to be moved. The number of buffers used is twice the number specified; half are used for reading the file and half for writing the copy. Values specified with the /PAGE_BUFFERS qualifier may range from 1 to 5. The default value is 3. Larger values may improve performance, but they increase memory usage.

/ROOT=file-spec

Requests that the root file be moved to the specified location.

/USERS_MAX=n

Specifies a new value for the database MAXIMUM CONCURRENT USERS parameter. The default is to leave the value unchanged.

Use the /USERS_MAX qualifier only if you move the database root file.

/BLOCKS_PER_PAGE=n

Specifies a new page size for the storage area to which it is applied. You cannot decrease the page size of a storage area, and you cannot change the size of a uniform storage area. If you change page sizes, Digital Equipment Corporation recommends that you perform an RMU/BACKUP operation after the RMU/MOVE_AREA operation. Performing the backup operation simplifies recovery if that become necessary. The /BLOCKS_PER_PAGE qualifier is a positional qualifier.

/FILE=file-spec

Specifies a new location for the storage area to which it is applied. The /FILE qualifier is a positional qualifier.

/SNAPSHOTS=FILE=file-spec

Specifies a new snapshot file location for the storage area to which it is applied. The /SNAPSHOTS qualifier is a positional qualifier.

/THRESHOLDS=(n,n,n)

Specifies new SPAM thresholds for the storage area to which it is applied. The thresholds of a uniform format storage area cannot be changed. The /THRESHOLDS qualifier is a positional qualifier.

RMU/MOVE_AREA Command

Usage Notes

You must have the Rdb/VMS ADMINISTRATOR or OPERATOR (SQL DBADM or OPER) privilege, and access to the database files to use the RMU/MOVE_AREA command. To access the database files, you must have default ACLs set up, or the VMS SYSPRV privilege, or RMU must be installed with SYSPRV on your system.

The parameter (file and area) qualifiers for the RMU/MOVE_AREA command have positional semantics, and when placed randomly may be ignored or produce unexpected results. See Section 6.2 for more information on parameter qualifiers.

Examples

Example 1

If a storage area is on a disk that is logging error messages, you can move the storage area to another disk using the RMU/MOVE_AREA command. The following command moves the DEPARTMENTS storage area (DEPARTMENTS.RDA and DEPARTMENTS.SNP) of the MF_PERSONNEL database to the DDV12:[RICK.SQL] directory:

```
$ RMU/MOVE_AREA MF_PERSONNEL DEPARTMENTS /DIRECTORY=DDV12:[RICK.SQL]
```

6.22 RMU/OPEN Command

Opens a root file and maps its global section. Overhead normally charged to your process is absorbed by Rdb/VMS when you use the RMU/OPEN command. You can use the RMU/OPEN command in conjunction with the RDO CHANGE DATABASE command to control access to the database. See the description of the OPEN IS {AUTOMATIC | MANUAL} clause of the CHANGE DATABASE command for details.

Format

```
RMU/OPEN root-file-spec [...]
```

Command Qualifier

```
/PATH
```

Description

Once you have used RMU/OPEN to open a database, the database remains open and mapped until it is closed by a RMU/CLOSE command and all users have exited the database with the FINISH or EXIT statements. When you issue a RMU/OPEN command before invoking a database, you attach more efficiently to the database.

If you have specified automatic opening for your database, users can invoke the database at any time without first issuing a RMU/OPEN command. However, using the RMU/OPEN command saves time when users invoke the database. If you modify the database for manual opening, the RMU/OPEN command must be entered before users can invoke the database.

Parameter

root-file-spec

Specifies the database to open. If the root file is open, you get an informational message. An error returns if you specify a nonexistent root file. The default file type is RDB.

RMU/OPEN Command

Command Qualifier

/PATH

The full or relative data dictionary path name in which the definitions reside for the database you want to open.

The */PATH* qualifier is a positional qualifier. The path name cannot include wildcard characters.

Usage Notes

You must have the Rdb/VMS ADMINISTRATOR (SQL DBADM) privilege to use the RMU/OPEN command.

Examples

Example 1

This command opens the PERSONNEL database.

```
$ RMU/OPEN PERSONNEL
```

Example 2

The following command opens the PERSONNEL database in the [WORK] directory, all the databases in the [TEST] directory, and the databases specified by the path names CDD\$TOP.FINANCE and SAMPLE_DB:

```
$ RMU/OPEN DISK1:[WORK]PERSONNEL, CDD$TOP.FINANCE/PATH, -  
_ $ DISK1:[TEST]*, SAMPLE_DB/PATH
```

6.23 RMU/RECOVER Command

Completes a database reconstruction by processing past transactions from the after-image journal (AIJ) against a database restored from a backup file.

Format

RMU/RECOVER aij-file-name

Command Qualifiers

/AIJ_BUFFERS=integer
 /AREAS [= storage-area[...]]
 /[NO]LOG
 /[NO]TRACE
 /UNTIL=date-time
 /ROOT=root-file-name
 /RESOLVE

Defaults

/AIJ_BUFFERS=20
 All storage areas
 The setting of the DCL VERIFY flag
 The setting of the DCL VERIFY flag
 /UNTIL=now
 See the RMU/RECOVER/RESOLVE command

Description

You can use the RMU/RECOVER command to apply the contents of an after-image journal file to a restored copy of your database. RMU will roll forward the transactions in the after-image journal file into the restored copy of the database. You can use this feature to maintain an up-to-date copy of your database for fast recovery after a failure. To do this, use RMU/RECOVER to periodically apply your after-image journals to a separate copy of the database. You can also use RMU/BACKUP/AFTER_JOURNAL to copy your database's AIJ file to tape and truncate the original AIJ file without shutting your database down.

Although RMU/RECOVER accepts only one AIJ file at a time, you can issue this command once for each AIJ file until you have rolled forward your database to the desired state. However, you must be careful to apply these AIJ files to the database in the order in which they were created. RMU checks the validity of the journal entries against your database and applies only appropriate transactions. If none of the transactions apply, you will receive a warning message.

RMU/RECOVER Command

You can read your database between recovery steps, but you must not perform additional updates if you want to perform more recovery steps.

If a system failure causes a recovery step to abort, you can simply reissue the RMU/RECOVER command. RMU scans the AIJ file until it finds the first transaction that has not yet been applied to your restored database. RMU begins recovery at that point.

Parameter

aij-file-name

Specifies the file containing the after-image journal. The default file type is AIJ.

Command Qualifiers

/AIJ_BUFFERS=integer

Specifies the number of buffers to be used by the roll forward process. The default is 20 buffers. The valid range is 1 to 1024 buffers.

/AREAS [= storage-area[,...]]

Specifies the areas you want to roll forward. You should use the /AREAS qualifier only if you have inconsistent storage areas to recover. The default for /AREAS is all storage areas.

If the /UNTIL qualifier is not specified, a recover operation by area rolls the specified areas forward to the current time. If the /UNTIL qualifier is specified, a recover by area operation recovers until the storage areas being rolled forward are current with the other storage areas, then recovery stops, regardless of the time specified by the /UNTIL qualifier.

If you have restored the database by area, you do not need to specify the areas to be recovered on the RMU/RECOVER command line. (Such areas are marked inconsistent and cannot be readied until they are rolled forward to the current time of the database.) If you do not specify the areas to recover, RMU recovers the necessary areas up to the current time (the time of the last committed transaction in the database) and then stops applying transactions.

If you have restored the database by area and choose to specify the areas to be recovered on the RMU/RECOVER command line, you are responsible for correctly specifying the inconsistent storage areas that you want RMU to recover. If you specify one or more storage areas to be recovered, RMU/RECOVER will recover only the areas you specify. Since you could inadvertently specify a consistent area or omit an inconsistent area, you are

RMU/RECOVER Command

more likely to recover your database correctly if you do not specify any storage areas on the RMU/RECOVER command line after you have restored the database by area.

When you use the asterisk wild card character with /AREAS, RMU does not necessarily recover all the storage areas in the specified database. In other words, the following two commands may not be equivalent:

```
$ RMU/RECOVER MF_PERSONNEL
$ RMU/RECOVER/AREAS=* MF_PERSONNEL
```

When /AREAS is not specified, RMU recovers all the storage areas in the database to the time specified by the /UNTIL qualifier or to the time of the last committed transaction in the AIJ file. When /AREAS=* is specified, RMU recovers only those storage areas that are not current with the root file of the database to the earliest consistent state.

/LOG

/NOLOG

The /LOG qualifier specifies that the recovery activity be logged. The default is the setting of the DCL VERIFY flag, which is controlled by the DCL SET VERIFY command. When recovery activity is logged, the output from the /LOG qualifier provides the number of transactions committed, rolled back, and ignored during the recovery process. You can specify the /TRACE qualifier with the /LOG qualifier.

/TRACE

/NOTRACE

The /TRACE qualifier specifies that the recovery activity be logged. The default is the setting of the DCL VERIFY flag, which is controlled by the DCL SET VERIFY command. When recovery activity is logged, the output from the /TRACE qualifier identifies transactions in the AIJ file by TSN and describes what RMU did with each transaction during the recovery process. You can specify the /LOG qualifier with the /TRACE qualifier.

/UNTIL="date-time"

Defines a date and time in VMS Version 5.0 and higher date and time strings. Use /UNTIL to limit the recovery to those transactions in the journal file bearing a starting time and date stamp no later than the specified time. Thus, you can specify recovery up to the point when you think the error occurred.

For example:

```
$ DEFINE LIB$DT_INPUT_FORMAT "!DB-!MAAU-!Y4 !H04:!M0:!S0.!C2"
$ RMU/RECOVER/UNTIL="10-JUN-1989 12:56" USER3:[AIJ_FILES]MF_AIJ_FILE.AIJ;1
```

RMU/RECOVER Command

If you do not specify the `/UNTIL` qualifier, all committed transactions in the after-image journal file will be applied to your database.

If the `/UNTIL` qualifier is specified, a recover by area operation recovers until the storage areas being rolled forward are current with the other storage areas, then recovery stops, regardless of the time specified by the `/UNTIL` qualifier.

Note *The `/UNTIL` qualifier accepts VMS Version 5.0 or higher date and time strings, as well as international dates. See the VMS RTL Library (LIB\$) Manual for more information. The date and time strings specified must be in quotes because they can contain spaces and other DCL syntax characters such as commas. A colon separator between the date and time is no longer valid.*

`/ROOT=root-file-name`

Specifies the name of the database that the journal should be applied to. The `/ROOT` qualifier allows you to specify a copy of a database instead of the original whose file specification is in the AIJ file. Use `/ROOT` to specify the new location of your database root file.

This lets you roll forward a database copy (possibly residing on a different disk) by following these steps:

- 1 Use `RMU/BACKUP` to make a backup copy of the database:

```
$ RMU/BACKUP MF_PERSONNEL.RDB MF_PERS_FULL_BU.RBF
```

This writes a backup file of database `MF_PERSONNEL` to the file `MF_PERS_FULL_BU.RBF`.

- 2 Use `RMU/RESTORE` with the `/ROOT` and `/FILE` qualifiers, stating the file specifications of the root and area files in the database copy. The usual purpose of using `RMU/RESTORE/FILE` is to establish a copy of the database on a different disk.

```
$ RMU/RESTORE/ROOT=DB3:[USER]MF_PERSONNEL/FILE=DB3:[USER] MF_PERS_FULL_BU
```

This restores the database on disk `DB3`: in the directory `[USER]`. Default file names and file types are used.

- 3 If the database uses after-image journaling, you can use `RMU/RECOVER` to roll forward the copy.

```
$ RMU/RECOVER DBJNL.AIJ/ROOT=DB3:[USER]MF_PERSONNEL.RDB
```

Thus, transactions processed and journaled since the backup are recovered on the `DB3`: copy.

RMU/RECOVER Command

If you do not specify the `/ROOT` qualifier, RMU examines the after-image journal file to determine the exact name of the database root file to which the journaled transactions will be applied. This name, which was stored in the after-image journal file, is the fully qualified file name that your database root file had when after-image journaling was enabled.

The journal file for a single-file database does not include the filename for the database, so to recover a single-file database you must specify the location of the database to be recovered using the `/ROOT` qualifier.

/RESOLVE

Recovers a corrupted database and resolves an unresolved transaction by completing the transaction.

See the `RMU/RECOVER/RESOLVE` command for a description of the options available with the `/RESOLVE` qualifier.

Usage Notes

You must have the VMS `SYSPRV` privilege to use the `RMU/RECOVER` command.

You can use `RMU/RECOVER` to apply the contents of an after-image journal file to a restored copy of your database. RMU will roll forward the transactions in the after-image journal file into the restored copy of the database. You can use this feature to maintain an up-to-date copy of your database for fast recovery after a failure. To do this, use `RMU/RECOVER` to periodically apply your after-image journals to a separate copy of the database. You can also use `RMU/BACKUP/AFTER_JOURNAL` to copy your database's AIJ file to tape and truncate the original AIJ file without shutting your database down.

Although `RMU/RECOVER` accepts only one AIJ file at a time, you can issue this command once for each AIJ file until you have rolled forward your database to the desired state. However, you must be careful to apply these AIJ files to the database in the order in which they were created. RMU checks the validity of the journal entries against your database and applies only appropriate transactions. If none of the transactions apply, you will receive a warning message.

You can read your database between recovery steps, but you must not perform additional updates if you want to perform more recovery steps.

RMU/RECOVER Command

If a system failure causes a recovery step to abort, you can simply reissue the RMU/RECOVER command. RMU scans the AIJ file until it finds the first transaction that has not yet been applied to your restored database. RMU begins recovery at that point.

Examples

Example 1

The first command below specifies the input time format. The RMU/RECOVER command requests recovery from the after-image journal file PERSONNEL.AIJ located on PR\$DISK in the SMITH directory. It specifies that recovery should continue until 1:35 p.m. on June 30, 1989. Because the /TRACE qualifier is specified, the RMU/RECOVER command displays detailed information about the recovery operation to SYS\$OUTPUT.

```
$ DEFINE LIB$DT_INPUT_FORMAT "!DB-!MAAU-!Y4 !H02:!M0:!S0.!C2"
$ RMU/RECOVER/UNTIL="30-JUN-1989 13:35"/TRACE PR$DISK:[SMITH]PERSONNEL
```

Example 2

The following example shows how to recover a database using an AIJ file:

```
$ !
$ SET DEFAULT DDV21:[TEST]
$ !
$ RDO := $RDO
$ RDO
RDO> DEFINE DATABASE 'DDV21:[TEST]SAMPLE_DB'
cont>     DICTIONARY IS NOT USED
cont>     DICTIONARY IS NOT REQUIRED.
RDO> !
RDO> ! Use the FINISH statement to detach from the database, then
RDO> ! issue the CHANGE DATABASE statement that automatically invokes
RDO> ! the specified database
RDO> !
RDO> FINISH
RDO> !
RDO> ! Create after-image journaling. The AIJ file
RDO> ! is given the name SAMPLE.AIJ (and is placed on a disk other
RDO> ! than the disk holding the RDB and SNP files):
RDO> !
RDO> CHANGE DATABASE FILENAME 'DDV21:[TEST]SAMPLE_DB'
cont>     JOURNAL FILE IS USER$DISK:[CORP]SAMPLE.AIJ.
RDO> EXIT
```

RMU/RECOVER Command

```
$ !
$ ! Using RMU/BACKUP, make a backup copy of the database.
$ ! This command ensures that you have a copy of the
$ ! database at a known time, in a known state.
$ !
$ RMU/BACKUP/LOG DDV21:[TEST]SAMPLE_DB USER2:[BACKUPS]SAMPLE_BACKUP.RBF
%RMU-I-BCKTXT_01, Thread 1 uses devices USER2:
%RMU-I-BCKTXT_08, Thread 1 was assigned file DDV21:[TEST]SAMPLE_DB.RDB;1
%RMU-I-BCKTXT_00, Backed up root file
%RMU-I-BCKTXT_02, Full backup of storage area DDV21:[TEST]SAMPLE_DB.RDB;1
%RMU-I-BCKTXT_02, Full backup of storage area DDV21:[TEST]SAMPLE_DB.RDB;1
%RMU-I-BCKTXT_04,      ignored 1 space management page
%RMU-I-BCKTXT_05,      backed up 3 inventory pages
%RMU-I-BCKTXT_06,      backed up 114 logical area bitmap pages
%RMU-I-BCKTXT_07,      backed up 285 data pages
$ !
$ ! Now you can use RDO with after-image journaling enabled.
$ !
$ RDO
RDO> !
RDO> ! Invoke the database and perform some data definition and storage.
RDO> !
RDO> INVOKE DATABASE FILENAME 'DDV21:[TEST]SAMPLE_DB'
RDO> START_TRANSACTION READ_WRITE
RDO> DEFINE FIELD NEW_FIELD DATATYPE TEXT 10.
%RDO-W-NOCDUPDAT, database invoked by filename, the CDD will not be updated
RDO> !
RDO> DEFINE RELATION RELATION1.
cont>   NEW_FIELD.
cont>   END.
RDO> COMMIT
RDO> !
RDO> START_TRANSACTION READ_WRITE
RDO> STORE R IN RELATION1 USING
cont>   R.NEW_FIELD = "data"
cont>   END_STORE
RDO> COMMIT
RDO> !
RDO> ! Imagine that a disk crash occurred here. In such a situation,
RDO> ! the current database is inaccessible. You need a prior copy
RDO> ! of the database to roll forward all the transactions in the
RDO> ! AIJ file.
RDO> !
RDO> FINISH
RDO> EXIT
```

RMU/RECOVER Command

```
$ !
$ ! You know that the backup of the database is
$ ! uncorrupted. Use RMU/RESTORE to restore the database.
$ ! Because the after-image journal file is enabled, a
$ ! new after-image journal file is created as part of this
$ ! restore operation. (The new AIJ file is SAMPLE.AIJ;2.)
$ RMU/RESTORE/NOCD_INTEGRATE/NEW_VERSION/LOG/DIR=DDV21:[TEST] -
_$ USER2:[BACKUPS]SAMPLE_BACKUP.RBF
%RMU-I-REXTXT_04, Thread 1 uses devices USER2:
%RMU-I-REXTXT_00, Restored root file DDV21:[TEST]SAMPLE_DB.RDB;2
%RMU-I-LOGRESSST, restored storage area DDV21:[TEST]SAMPLE_DB.RDB;1
%RMU-I-LOGRESSST, restored storage area DDV21:[TEST]SAMPLE_DB.RDB;1
%RMU-I-REXTXT_05, rebuilt 1 space management page
%RMU-I-REXTXT_06, restored 3 inventory pages
%RMU-I-REXTXT_07, rebuilt 114 logical area bitmap pages
%RMU-I-REXTXT_08, restored 285 data pages
%RMU-I-REXTXT_01, Initialized snapshot file DDV21:[TEST]SAMPLE_DB.SNP;2
%RMU-I-LOGINIFIL, contains 100 pages, each page is 2 blocks long
%RMU-I-LOGCREAIJ, created after-image journal file USER$DISK:[CORP]SAMPLE.AIJ;2
$ !
$ ! LOOP:
$ !
$ !
$ ! Use the RMU/RECOVER command to roll forward the
$ ! database changes contained in the AIJ file.
$ ! The date used in the UNTIL clause in this example
$ ! represents some time in the future. This causes Rdb/VMS
$ ! to apply the entire AIJ file. Note that SAMPLE.AIJ;1 is
$ ! specified because SAMPLE.AIJ;1 contains the transactions
$ ! that occurred before the RMU/RESTORE command (which created
$ ! a new version of the AIJ file) was issued. Note that the AIJ
$ ! file is applied to the version of the database (SAMPLE_DB.RDB;2)
$ ! created by the RMU/RESTORE command. If the version of the database
$ ! is not specified in the RMU/RECOVER command, Rdb/VMS attempts to
$ ! recover the previous version of the database and the recovery
$ ! attempt fails because of a mismatch between the database file and
$ ! after-image journal file.
$ !
$ RMU/RECOVER/TRACE/UNTIL=01-JAN-1999/ROOT=DDV21:[TEST]SAMPLE_DB.RDB;2 -
_$ USER$DISK:[CORP]SAMPLE.AIJ;1
%RMU-I-LOGOPNAIJ, opened journal file USER$DISK:[CORP]SAMPLE.AIJ;1
%RMU-I-LOGRECSTAT, transaction with TSN 8 committed
%RMU-I-LOGRECSTAT, transaction with TSN 9 committed
$ !
$ ! If you want to apply more than one journal file to
$ ! the same database, repeat the steps from the
$ ! "LOOP:" label.
$ !
```

RMU/RECOVER Command

Example 3

The following example shows the output from the /LOG qualifier:

```
$ RMU/RECOVER/LOG/ROOT=DDV2:[RICK.RDB]MF_PERS.RDB;2 -
_ $ USER3:[AIJ_FILES]MF_AIJ_FILE.AIJ;1
%RMU-I-LOGRECDB, recovering database file DDV2:[RICK.RDB]MF_PERS.RDB;2
%RMU-I-LOGRECOVR, 2 transactions committed
%RMU-I-LOGRECOVR, 0 transactions rolled back
%RMU-I-LOGRECOVR, 2 transactions ignored
%RMU-I-AIJSUCCESS, database recovery completed successfully
```

RMU/RECOVER/RESOLVE Command

6.24 RMU/RECOVER/RESOLVE Command

Recovers a corrupted database and resolves an unresolved distributed transaction by completing the transaction.

For complete information on unresolved transactions, see the *VAX Rdb/VMS Guide to Distributed Transactions*.

Format

RMU/RECOVER/RESOLVE aij-file-name

<u>Command Qualifiers</u>	<u>Defaults</u>
/[NO]CONFIRM	See description
/STATE={COMMIT ABORT}	See description

Description

Use the RMU/RECOVER/RESOLVE command to commit or abort any unresolved distributed transactions in the after-image journal file. You must complete the unresolved transactions to the same state (COMMIT or ABORT) in every after-image journal file affected by the unresolved transactions.

The RMU/RECOVER/RESOLVE command performs the following tasks:

- Displays identification information for an unresolved transaction.
- Prompts you for the state (COMMIT, ABORT, or IGNORE) to which you want the unresolved transaction resolved (if you did not specify the /STATE qualifier on the command line).
- Prompts for confirmation of the state you specified.
- Commits, aborts, or ignores the unresolved transaction.
- Continues until it displays information for all unresolved transactions.

RMU/RECOVER/RESOLVE Command

Command Parameter

aij-file-name

The name of the file containing the after-image journal. The default file type is AIJ.

Command Qualifiers

/STATE=option

The /STATE qualifier is used to select the state to which all unresolved transactions will be resolved.

Options for the /STATE qualifier are:

- COMMIT—to commit all unresolved transactions.
- ABORT—to abort all unresolved transactions.

If you do not specify the /STATE qualifier, RMU prompts you to enter an action, either COMMIT, ABORT, or IGNORE, for each unresolved transaction in that after-image journal file. If you enter IGNORE, RMU attempts to contact the coordinator to resolve the transaction. The transaction remains unresolved until the coordinator becomes available again and instructs the transaction to complete or until you manually complete the transaction by using the RMU/RECOVER/RESOLVE command again.

/CONFIRM

/NOCONFIRM

Specify the /CONFIRM qualifier to have RMU prompt you for confirmation of each transaction state you alter. The default for interactive processing is /CONFIRM.

Specify the /NOCONFIRM qualifier to suppress this prompt. The default for batch processing is /NOCONFIRM.

Usage Notes

You must have the VMS SYSPRV privilege to use the RMU/RECOVER/RESOLVE command.

RMU/RECOVER/RESOLVE Command

Examples

Example 1

This command recovers the PERSONNEL database and rolls the database forward from the old AIJ file to resolve the unresolved distributed transactions. Because the /STATE qualifier is not specified, RMU will prompt the user for a state for each unresolved transaction.

```
$ RMU/RECOVER/RESOLVE PERSONNEL.AIJ;1
```

Example 2

This example specifies that all unresolved transaction records in the PERSONNEL.AIJ file be committed.

```
$ RMU/RECOVER/RESOLVE/STATE=COMMIT PERSONNEL.AIJ
```

For more examples of the RMU/RECOVER/RESOLVE command, see the *VAX Rdb/VMS Guide to Distributed Transactions*.

6.25 RMU/RESOLVE Command

Resolves all unresolved distributed transactions for the specified database. For more information on unresolved transactions, see the *VAX Rdb/VMS Guide to Distributed Transactions*.

Format

RMU/RESOLVE root-file-spec

Command Qualifiers

/[NO]CONFIRM
 /[NO]LOG
 /PARENT_NODE=node-name
 /PROCESS=process-id
 /STATE={COMMIT|ABORT}
 /TSN=tsn

Defaults

See description
 Setting of DCL VERIFY flag

Description

Use the RMU/RESOLVE command to commit or abort any unresolved distributed transactions in the database. You must resolve the unresolved transactions to the same state (COMMIT or ABORT) in every database affected by the unresolved transactions.

The RMU/RESOLVE command performs the following tasks:

- Displays identification information for an unresolved transaction.
- Prompts you for the state (COMMIT, ABORT, or IGNORE) to which you want the unresolved transaction resolved (if you did not specify the /STATE qualifier on the command line).
- Prompts you for confirmation of the state you chose.
- Commits, aborts, or ignores the unresolved transaction. If you commit or abort the unresolved transaction, it is resolved and cannot be resolved again.

RMU/RESOLVE Command

- Continues to display and prompt for states for subsequent unresolved transactions until it has displayed information for all unresolved transactions.

Use the `/PARENT_NODE`, `/PROCESS`, or `/TSN` qualifiers to limit the number of unresolved transactions that RMU displays. Use the `/USERS` `/STATE=BLOCKED` qualifiers of the `RMU/DUMP` command to determine values for the `/PARENT_NODE`, `/PROCESS`, and `/TSN` qualifiers.

Command Parameter

root-file-spec

Specifies the root file of the database for which you want to resolve unresolved transactions.

Command Qualifiers

/CONFIRM

/NOCONFIRM

Specify the `/CONFIRM` qualifier to have RMU prompt you for confirmation of each unresolved transaction. This is the default for interactive processing.

Specify the `/NOCONFIRM` qualifier to suppress this prompt. This is the default for batch processing.

/LOG

/NOLOG

Specifies whether or not to report the processing of the command to `SYSS$OUTPUT`. Specify the `/LOG` qualifier to request that summary information about the resolve operation be reported on `SYSS$OUTPUT`, and the `/NOLOG` qualifier to prevent this reporting. If you specify neither, the default is the current setting of the `DCL VERIFY` flag. (The `DCL SET VERIFY` command controls the setting of the `DCL VERIFY` flag.)

/PARENT_NODE=node-name

Specifies the node name to limit the selection of transactions to those originating from the specified node.

You cannot specify the `/TSN` or `/PROCESS` qualifier with the `/PARENT_NODE` qualifier.

RMU/RESOLVE Command

/PROCESS=process-id

Specifies the process identification to limit the selection of transactions to those associated with the specified process.

You cannot specify the `/PARENT_NODE` or `/TSN` qualifier with the `/PROCESS` qualifier.

/STATE=option

Specifies the state to which all unresolved transactions will be resolved.

Two states are available:

- `COMMIT`—to commit unresolved transactions.
- `ABORT`—to abort unresolved transactions.

If you do not specify the `/STATE` qualifier, RMU prompts you to enter an action, either `COMMIT`, `ABORT`, or `IGNORE`, for each unresolved transaction on that database. If you enter `IGNORE`, RMU attempts to contact the coordinator to resolve the transaction. The transaction remains unresolved until the coordinator becomes available again and instructs the transaction to complete, or until you manually complete the transaction by using the `RMU/RESOLVE` command again.

/TSN=tsn

Specifies the transaction sequence number of the unresolved transactions whose state you want to modify.

You cannot specify the `/PARENT_NODE` or `/PROCESS` qualifier with the `/TSN` qualifier.

Usage Notes

You must have the VMS `SYSPRV`, `BYPASS`, or `SETPRV` privilege to use the `RMU/RESOLVE` command.

Examples

Example 1

This example specifies that the first displayed unresolved transaction in the `PERSONNEL` database be changed to the `ABORT` state and rolled back.

```
$ RMU/RESOLVE/LOG/STATE=ABORT PERSONNEL
```

RMU/RESOLVE Command

Example 2

The following command will display a list of all transactions coordinated by node GREEN and might be useful if node GREEN failed while running an application that used the two-phase commit protocol.

```
$ RMU/RESOLVE/PARENT_NODE=GREEN PERSONNEL
```

Example 3

This example displays a list of all transactions initiated by process 41E0364A. The list might be useful for resolving transactions initiated by this process if the process were deleted.

```
$ RMU/RESOLVE/PROCESS=41E0364A PERSONNEL
```

Example 4

This example completes unresolved transactions for the PERSONNEL database, and confirms and logs the operation.

```
$ RMU/RESOLVE/LOG/CONFIRM PERSONNEL
```

For more examples of the RMU/RESOLVE command, see the *VAX Rdb/VMS Guide to Distributed Transactions*.

RMU/RESTORE Command

6.26 RMU/RESTORE Command

Restores a database to its condition at the time of a full or incremental backup that was performed with an RMU/BACKUP command. When you use the RMU/RESTORE command on a Version 3.0 or higher database, an RMU/CONVERT/NOCONFIRM/COMMIT command is automatically executed as part of the RMU/RESTORE command. Therefore, by executing the RMU/RESTORE command on a Version 3.0 or higher database, you convert that database to a Version 4.0 Rdb/VMS database.

Format

RMU/RESTORE backup-file-spec [storage-area-name[,...]]

Command Qualifiers

/INCREMENTAL
/AREA
/[NO]CONFIRM
/[NO]AFTER_JOURNAL=file-spec
/[NO]CDD_INTEGRATE
/PATH=cdd-path
/DIRECTORY=full-file-spec
/[NO]LOG
/NODES_MAX=number-VAX-nodes
/[NO]NEW_VERSION
/ROOT=root-file-spec
/USERS_MAX=number-users
/[NO]REWIND
/ACTIVE_IO=max-reads
/PAGE_BUFFERS=number-buffers
/OPTIONS=file-spec
/LABEL[(label-name-list)]

Defaults

Full restore
See description
See description
See description
/CDD_INTEGRATE
Existing value
See description
Current DCL verify value
Existing value
/NONEW_VERSION
Existing value
Existing value
/NOREWIND
/ACTIVE_IO=3
/PAGE_BUFFERS=3
None
See description

RMU/RESTORE Command

File or Area Qualifiers

/[NO]BLOCKS_PER_PAGE=integer
/FILE=file-spec
/SNAPSHOT=(FILE=file-spec)
/THRESHOLDS=(val1[,val2[,val3]])

Defaults

/NOBLOCKS_PER_PAGE
See description
See description
Existing area file value

Description

The RMU/RESTORE command rebuilds a database from a backup file, produced earlier by an RMU/BACKUP command, to the condition the database was in when the backup was performed.

You can specify only one backup file parameter in an RMU/RESTORE command. If this parameter is a full backup file, you cannot use the /INCREMENTAL qualifier. However, you must use the /INCREMENTAL qualifier if the parameter names an incremental backup file.

To restore a database from both a full backup file and an incremental backup file, first enter an RMU/RESTORE command that specifies a full backup file. Then enter an RMU/RESTORE/INCREMENTAL command that specifies the most recent incremental backup file.

Note *You can rename or move the files that comprise an Rdb/VMS database by using the RMU/BACKUP and RMU/RESTORE commands. To move a multifile Rdb/VMS database (a database having at least one storage area file (RDA)), you must use one of the following:*

- *The RMU/BACKUP and RMU/RESTORE commands*
- *The RDO EXPORT and IMPORT statements*
- *The RMU/MOVE_AREA command*
- *The RMU/COPY_DATABASE command*

Using the DCL COPY command with a multifile database (assuming the files are copied to a new location) will result in an unusable database. This happens because the DCL COPY command cannot update the full file specification

RMU/RESTORE Command

pointers (stored in the root file) to the other database files (RDA, SNP, optional AIJ).

By default, the RMU/RESTORE command integrates the metadata stored in the database root (RDB) file with the CDD/Plus copy of the metadata (assuming the data dictionary is installed on your system). However, you can prevent dictionary integration by specifying the /NO_CDD_INTEGRATE qualifier.

When you specify the /INCREMENTAL qualifier, do not specify the following additional qualifiers:

- /DIRECTORY
- /NODES_MAX
- /[NO]NEW_VERSION
- /USERS_MAX

The /CONFIRM qualifier is ignored if you omit the /INCREMENTAL qualifier. Also, you must specify the /ROOT qualifier when you restore an incremental backup file to a new version database, renamed database, or database restored in a new location.

When you use the RMU/RESTORE command on a Version 3.0 or higher Rdb/VMS database, an RMU/CONVERT/NOCONFIRM/COMMIT command is automatically executed as part of the RMU/RESTORE command. Therefore, by executing the RMU/RESTORE command on a Version 3.0 or higher Rdb/VMS database, you convert that database to a Version 4.0 Rdb/VMS database.

Command Parameters

backup-file-spec

A file specification for the backup file produced by a previous RMU/BACKUP command. The default file type is RBF.

If you use multiple tape drives, the backup-file-spec must include the tape device specifications. Separate the device specifications with commas.

```
$ RMU/RESTORE /REWIND $111$MUA0:PERS_FULL_NOV30.RBF,$112$MUA1:
```

See the *VAX Rdb/VMS Guide to Database Maintenance and Performance* for more information on using multiple tape drives.

RMU/RESTORE Command

***storage-area-name* [,...]**

A storage area name from the database. This parameter is optional. Use it in the following situations:

- When you want to change the values for thresholds or blocks per page
- When you want to change the names specified with the /SNAPSHOT or /FILE qualifiers for the restored database
- If you want to restore only selected storage areas from your backup file, you must use the /AREA qualifier and specify the names of the storage areas you want to restore in either the storage-area-name parameter on the RMU/RESTORE command line, or in the file specified with the /OPTIONS command qualifier.

To use this option, specify the storage area *name* rather than the VMS file specification for the storage area.

You must have the VMS access right SYSPRV to restore one or more storage areas.

By using RMU/BACKUP and RMU/RESTORE, you can back up and restore selected storage areas of your database. This RMU backup and restore by area feature is designed to:

- Speed recovery when corruption occurs in some (not all) of the storage areas of your database.
- Reduce the time needed to perform backups because some data (data in read-only storage areas, for example) does not need to be backed up with every backup of the database.

If you plan to use RMU/BACKUP and RMU/RESTORE to back up and restore only selected storage areas for a database, you *must* perform full and complete backups of the database at regular intervals. A full and complete backup is a full backup (*not* an incremental backup) of *all* the storage areas in the database. You must perform a full and complete backup because the database root file can only be restored from a full and complete backup file. Without a root file, restoring only the selected areas from by-area backup files does not result in a working database. Also, if the root file is corrupted, you can only recover storage areas up to (but not past) the date of the last full and complete backup. Therefore, Digital Equipment Corporation recommends that you perform full and complete backups regularly.

RMU/RESTORE Command

If you plan to back up and restore only selected storage areas for a database, Digital Equipment Corporation also strongly recommends that you enable after-image journaling for the database (in addition to performing the full and complete backup of the database described earlier). In other words, if you are not backing up and restoring *all* the storage areas in your database, you should have after-image journaling enabled. This ensures that you can recover all the storage areas in your database in the event of a system failure. If you *do not* have after-image journaling enabled and one or more of the areas restored with RMU/RESTORE are not current with the storage areas not restored, Rdb/VMS will not allow any transactions to use the storage areas that are not current in the restored database. In this situation, you can return to a working database by restoring the database using the backup file from the last full and complete backup of the database storage areas. However, any changes made to the database since the last full and complete backup are not recoverable. Or, if you used EXPORT instead of RMU/BACKUP to make a copy of your database, use IMPORT to restore your database to its previous state. Once again, any changes made to the database since the EXPORT operation are not recoverable. If you *do* have after-image journaling enabled, use RMU/RECOVER to apply transactions from the AIJ file to storage areas that are not current after the RMU/RESTORE operation completes. When the RMU/RECOVER command completes, your database will be consistent and usable.

Command Qualifiers

/INCREMENTAL

Restores a database from an incremental backup file.

Use the */INCREMENTAL* qualifier only when you have first issued an RMU/RESTORE command that names the full backup file, and when the time and date stamps indicate that the full backup file was produced before the incremental backup file.

After running both the full and the incremental backups, you would normally restore the database to the condition it was in when you performed the incremental database backup.

By default, the RMU/RESTORE command performs a full restoration on the backup file.

RMU/RESTORE Command

/AREA

Specifies that only the storage areas listed in the storage-area-name parameter on the command line or in the /OPTIONS file are to be restored. You can use this qualifier to simplify physical restructuring of a large database.

By default, the /AREA qualifier is not specified, and all the storage areas in the backup file are restored.

By using RMU/BACKUP and RMU/RESTORE, you can back up and restore selected storage areas of your database. This RMU backup and restore by area feature is designed to:

- Speed recovery when corruption occurs in some (not all) of the storage areas of your database.
- Reduce the time needed to perform backups because some data (data in read-only storage areas, for example) does not need to be backed up with every backup of the database.

If you plan to use RMU/BACKUP and RMU/RESTORE to back up and restore only selected storage areas for a database, you *must* perform full and complete backups of the database at regular intervals. A full and complete backup is a full backup (*not* an incremental backup) of *all* the storage areas in the database. You must perform a full and complete backup because the database root file can only be restored from a full and complete backup file. Without a root file, restoring only the selected areas from by-area backup files does not result in a working database. Also, if the root file is corrupted, you can only recover storage areas up to (but not past) the date of the last full and complete backup. Therefore, Digital Equipment Corporation recommends that you perform full and complete backups regularly.

If you plan to back up and restore only selected storage areas for a database, Digital Equipment Corporation also strongly recommends that you enable after-image journaling for the database (in addition to performing the full and complete backup of the database described earlier). In other words, if you are not backing up and restoring *all* the storage areas in your database, you should have after-image journaling enabled. This ensures that you can recover all the storage areas in your database in the event of a system failure. If you *do not* have after-image journaling enabled and one or more of the areas restored with RMU/RESTORE are not current with the storage areas not restored, Rdb/VMS will not allow any transactions to use the storage areas that are not current in the restored database. In this situation, you can return to a working database by restoring the database using the backup file from the last full and complete backup of the database storage areas. However, any

RMU/RESTORE Command

changes made to the database since the last full and complete backup are not recoverable. Or, if you used EXPORT instead of RMU/BACKUP to make a copy of your database, use IMPORT to restore your database to its previous state. Once again, any changes made to the database since the EXPORT operation are not recoverable. If you *do* have after-image journaling enabled, use RMU/RECOVER to apply transactions from the AIJ file to storage areas that are not current after the RMU/RESTORE operation completes. When the RMU/RECOVER command completes, your database will be consistent and usable.

If you attempt to restore a database area that is not in the backup file, you will receive an error message and the database will typically be inconsistent or unusable until the affected area is properly restored.

In the following example, the DEPARTMENTS storage area is excluded from the backup operation; therefore, an error message is displayed when the attempt is made to restore DEPARTMENTS, which is not in the backup file. Note that when this restore operation is attempted on a usable database, it completes, but the DEPARTMENTS storage area is now inconsistent.

```
$ RMU/BACKUP /EXCLUDE=DEPARTMENTS MF_PERSONNEL PERS_BACKUP5JAN88
$ RMU/RESTORE /NEW_VERSION /AREA PERS_BACKUP5JAN88.RBF DEPARTMENTS
%RMU-W-AREAEXCL, The backup does not contain the storage area - DEPARTMENTS
```

If you create a backup file using the RMU/BACKUP command and the /EXCLUDE qualifier, it is your responsibility to ensure that all areas of a database are restored and recovered when you use RMU/RESTORE and RMU/RECOVER to duplicate the database.

/CONFIRM ***/NOCONFIRM***

Specifies that RMU notify you of the name of the database on which you are performing the incremental restoration. Thus, you can be sure that you have specified the correct root file name to which the incremental backup file will be applied. This is the default for interactive processing.

Confirmation is especially important on an incremental restoration if you have changed the root file name or created a new version of the database during restoration from the full backup file. A new version always requires the /ROOT qualifier in the incremental restoration.

Specify the /NOCONFIRM qualifier to have the RMU/RESTORE command apply the incremental backup to the database without prompting for confirmation. This is the default for batch processing.

RMU/RESTORE Command

RMU ignores the /CONFIRM and /NOCONFIRM qualifiers unless you use the /INCREMENTAL qualifier.

/AFTER_JOURNAL=file-spec

/NOAFTER_JOURNAL

A new AIJ file for the restored version of the database. You can enable after-image journaling with this qualifier if it was not enabled when you backed up the database. You should specify a new device and directory for the AIJ file. If you do not specify a device and directory, you will receive a warning message. To protect yourself against media failures, put the AIJ file on a different device from that of your database files.

Specify the /NOAFTER_JOURNAL qualifier to disable after-image journaling if it is currently enabled.

If you omit both the /NOAFTER_JOURNAL and /AFTER_JOURNAL qualifiers, the RMU/RESTORE command uses information in the backup file to determine whether to enable after-image journaling. If after-image journaling was enabled at backup time, the RMU/RESTORE command creates a new version of the default AIJ file.

/CDD_INTEGRATE

/NOCDD_INTEGRATE

Integrates the metadata from the root file into the CDD/Plus data dictionary (assuming the data dictionary is installed on your system).

If you specify the /NOCDD_INTEGRATE qualifier, no integration will occur during the restoration, although you can request integration with a subsequent RDO INTEGRATE command.

You may want to delay integration of the database metadata with the data dictionary until after the restoration finishes successfully.

You can use the /NOCDD_INTEGRATE option even if the DICTIONARY IS REQUIRED clause was used when the database was defined.

Unlike the INTEGRATE statement in RDO, the /CDD_INTEGRATE qualifier integrates definitions *in one direction only*—from the database file to the dictionary. The /CDD_INTEGRATE qualifier does *not* integrate definitions from the dictionary to the database file.

/PATH=cdd-path

A data dictionary path into which the database definitions will be integrated. If you do not specify the /PATH qualifier, RMU uses the CDD\$DEFAULT logical name value of the user who entered the RMU/RESTORE command.

RMU/RESTORE Command

If you specify a relative path name, Rdb/VMS appends to the CDD\$DEFAULT value the *relative* path name you enter. If the cdd-path parameter contains nonalphanumeric characters, you must enclose it in double quotation marks ("").

Rdb/VMS ignores the /PATH qualifier if you use /NOCDD_INTEGRATE or if the data dictionary is not installed on your system.

/DIRECTORY=full-file-spec

A global device and directory file specification portion for the root, database storage, and snapshot files of the database.

For example:

```
$ RMU/RESTORE/DIRECTORY=DEB$:[SAMUEL] BCK$ : PERS_BU_FULL.RBF
```

You can override the /DIRECTORY file specification with a local /FILE qualifier. This way, the /DIRECTORY specification can cover most cases, and the /FILE specification can identify exceptions.

/LOG

/NOLOG

Specifies whether to report the processing of the command on SYSS\$OUTPUT. Specify the /LOG qualifier to request that the progress of the restore operation be written to SYSS\$OUTPUT and the /NOLOG qualifier to suppress this report. If you specify neither, the default is the current setting of the DCL verify switch. (The DCL SET VERIFY command controls the DCL verify switch.)

/NODES_MAX=number-VAX-nodes

A new upper limit on the number of VAXcluster nodes from which users may access the restored database. The /NODES_MAX qualifier will accept values between 1 and 64 VAXcluster nodes. The actual maximum is the highest number of VAXcluster nodes possible in the current version of VMS. The default value is the limit defined for the database before it was backed up.

You cannot specify the /NODES_MAX qualifier if you use the /INCREMENTAL qualifier.

/NEW_VERSION

/NONEW_VERSION

Specifies whether new versions of database files should be produced if the destination device and directory contain a previous version of the database files.

RMU/RESTORE Command

If you use the `/NEW_VERSION` qualifier, the new database file versions are produced.

If you use `/NONEW_VERSION`, the default, an error occurs if an old copy exists of any of the database files being restored.

/ROOT=root-file-spec

The root file specification of the restored database. This gives the database a new location or a new name, or both. The default file type is RDB.

Rdb/VMS derives the file specification for the restored root file by starting with the `/ROOT` qualifier, then adding missing file specification components from the `/DIRECTORY` qualifier and ultimately adding any other remaining file specification components from the original root file specification.

/USERS_MAX=number-users

A new upper limit on the number of users that may simultaneously access the restored database. The valid range is between 1 and 2032 users. The default value is the value defined for the database before it was backed up.

You cannot specify the `/USERS_MAX` qualifier if you use the `/INCREMENTAL` qualifier.

/REWIND

/NOREWIND

The `/REWIND` qualifier specifies that the tape that contains the backup file will be rewound before processing begins. The `/NOREWIND` qualifier, the default, causes the search for the backup file to begin at the current tape position.

The `/REWIND` and `/NOREWIND` qualifiers are applicable only to tape devices. RMU returns an error message if you use these qualifiers and the target device is not a tape device.

See the Usage Notes for information on tape label processing.

/ACTIVE_IO=max-reads

The maximum number of read operations to the backup file that RMU will attempt simultaneously. The value of the `/ACTIVE_IO` qualifier may range from one to five. The default value is three. Values larger than three may improve performance with multiple tape drives.

RMU/RESTORE Command

/PAGE_BUFFERS=number-buffers

The maximum number of buffers Rdb/VMS will use during the RMU/RESTORE operation while the database files are being created. The value of the /PAGE_BUFFERS qualifier may range from one to five. The default is three buffers. Values larger than three may improve performance, especially during incremental restore operations. When the RMU/RESTORE command enters the stage of reconstructing internal structures at the end of the restoration, a high /PAGE_BUFFERS value can be useful for very large databases.

However, the cost of using these extra buffers is that memory use is high. Thus, the tradeoff during a restoration is memory use against performance. For systems where physical memory is not a limitation, try setting the /PAGE_BUFFERS value higher during restorations.

/OPTIONS=file-spec

The options file contains storage area names, followed by the storage area qualifiers that you want applied to that storage area. *Do not* separate the storage area names with commas. Instead, put each storage area name on its own line in the file. The storage area qualifiers that you can include in the options file are /BLOCKS_PER_PAGE, /FILE, /SNAPSHOT, and /THRESHOLDS. You can use the DCL line continuation character, a hyphen (-), or the comment character (!) in the options file. The default file type is OPT.

/LABEL[=(label-name-list)]

The one- to six-character string with which the volumes of the backup file are to be labeled. The /LABEL qualifier is applicable only to tape volumes. If it is not specified, the label will be derived from the first six characters of the backup file name.

You can specify a list of tape labels for multiple tapes. If you list multiple tape label names, separate the names with commas and enclose the list of names in parentheses.

In a normal restore operation, the /LABEL qualifier you specify with the RMU/RESTORE command should be the same /LABEL qualifier you specified with the RMU/BACKUP command that backed up your database.

See the Usage Notes for information on tape label processing.

RMU/RESTORE Command

Command or Area Qualifiers

/BLOCKS_PER_PAGE=integer

/NOBLOCKS_PER_PAGE

Lets you restore the database with larger page sizes than existed in the original database. This creates new free space on each page in the storage area file and does not interfere with record clustering. RMU ignores this qualifier when it specifies an integer less than or equal to the current page size of the area.

If you use the default, */NOBLOCKS_PER_PAGE*, RMU takes the page size for the storage area from the page size specified for the database you backed up.

/FILE=file-spec

A new file specification for the storage-area-name parameter it qualifies. It can be used to restore all or selected database files in a different disk or directory. When you select the */FILE* qualifier, you must supply a file name.

/SNAPSHOT=(FILE=file-spec)

A file specification for a snapshot file associated with the storage-area-name parameter it qualifies. In general, use this qualifier to place snapshot (SNP) files on devices apart from the storage area (RDA) files. Doing so can reduce disk I/O contention in applications with a heavy mixture of concurrent read-only and read/write transactions.

The */SNAPSHOT* qualifier is positional. If you use it globally, RMU restores all snapshot area files on the device and directory named in the */FILE=file-spec* argument, except those storage areas specified with a local */SNAPSHOT* qualifier that name a different device and directory. A local */SNAPSHOT* qualifier overrides a global */SNAPSHOT* qualifier for a particular storage area.

When you do not specify the */SNAPSHOT* qualifier, RMU restores snapshot areas according to the information stored in the backup file.

/THRESHOLDS=(val1 [,val2 [,val3]])

A storage area's fullness percentage threshold. You can adjust space area management (SPAM) thresholds to improve future space utilization in the storage area. Each threshold value represents a percentage of fullness on a data page. When a data page reaches the percentage of fullness defined by a given threshold value, the space management entry for the data page is updated to contain that threshold value.

The *THRESHOLDS* qualifier applies only to storage areas with a mixed page format.

RMU/RESTORE Command

If a record would occupy 50 percent of free space of any database page, Rdb/VMS examines (during the record storage operation) the threshold values to determine which pages will have room for the new record. If you do not use the /THRESHOLDS qualifier with the RMU/RESTORE command, Rdb/VMS uses the storage area's original thresholds.

Usage Notes

To use the RMU/RESTORE command, you must have the VMS SYSPRV privilege or the RMU executable image must be installed with SYSPRV on your system.

A nonprivileged user does not have write access to the restored database file by default. To give a nonprivileged user VMS read and write access to the restored database file, the system manager can create the following access control list entry (ACE) on the directory or subdirectory to which the database file will be restored:

```
( IDENTIFIER=[ USER , UIC ] , OPTION=DEFAULT , ACCESS=READ+WRITE )
```

Each RMU/RESTORE operation requires exclusive access to the root file and the area files being restored.

When you back up your database to magnetic tape, Digital Equipment Corporation recommends that you supply a name for the backup file that is 17 or fewer characters in length. File names longer than 17 characters may be truncated. The VMS operating system supports four file-header labels: HDR1, HDR2, HDR3, and HDR4. In HDR1 labels, the file identifier field contains the first 17 characters of the file name you supply. The remainder of the file name is written into the HDR4 label if this label is allowed. If no HDR4 label is supported, a file name longer than 17 characters will be truncated. See the information on file-header labels in the *Guide to VMS Files and Devices*.

The following RMU commands are acceptable. The terminating period for the backup file name is not counted as a character, and the default file type of RBF is assumed. Therefore, VMS interprets the file name as WEDNESDAYS_BACKUP, which is 17 characters in length:

```
RMU/BACKUP/REWIND/LABEL=TAPE MF_PERSONNEL MUA0:WEDNESDAYS_BACKUP.  
RMU/RESTORE/REWIND/LABEL=TAPE MUA0:WEDNESDAYS_BACKUP.
```

RMU/RESTORE Command

The following RMU commands are not acceptable. Because no terminating period is supplied, VMS supplies a period and a file type of RBF, and interprets the backup file name as WEDNESDAYS_BACKUP.RBF, which is 20 characters in length:

```
RMU/BACKUP/REWIND/LABEL=TAPE MF_PERSONNEL MUA0:WEDNESDAYS_BACKUP
RMU/RESTORE/REWIND/LABEL=TAPE MUA0:WEDNESDAYS_BACKUP
```

The steps in tape label checking are:

- 1 Use the DCL MOUNT/FOREIGN command to mount the tape you want to use.

You must mount the first tape volume; RMU mounts subsequent volumes automatically. If VMS encounters an error when mounting the tape, either the RMU function will abort or the problem will be reported and the operation retried. If a severe error occurs, RMU does not mount subsequent volumes.

- 2 The tape characteristics are checked.

For an RMU/BACKUP operation, the tape must be properly mounted and cannot be write protected. If these checks fail, you are asked to mount the correct relative volume with write protection and to indicate when you are finished.

- 3 The tape labels are checked.

If the volume label disagrees with the label specified with the /LABEL qualifier or by the default (the backup file name), you receive an informational message.

If the tape is not the first volume, or if you specified the /REWIND qualifier in the command, RMU checks the tape protection (the user-id specified with the /OWNER_ID=user-id qualifier of the RMU/BACKUP command) and the expiration date of the tape, and issues an informational message regarding the error. You can overwrite an expired tape. You can read a tape that has not expired.

If any of the checks fail, you are asked what you want to do with the tape volume. You must select one of the following options:

- QUIT

This option cancels the operation, and causes you to exit from RMU.

RMU/RESTORE Command

- **RETRY**
This option repeats the checks after dismounting and remounting the same tape. The RETRY option is useful if you left the drive off line, or if there was some other kind of correctable error.
 - **UNLOAD**
This option dismounts and unloads the tape so you can load the correct tape on the drive.
 - **OVERRIDE**
This option overrides the failed checks and uses the tape anyway. The OVERRIDE option can be used with the RMU/RESTORE and RMU/DUMP/BACKUP_FILE commands only.
 - **INITIALIZE**
This option requests that the tape be rewound and relabeled. The INITIALIZE option can be used with the RMU/BACKUP command only.
- 4 The tape is remounted and reverified as requested.

Tape changes follow essentially the same procedure, with one difference. If the first volume is not at the beginning of the tape, complete label checking is not available.

Examples

Example 1

The following example restores the PERSONNEL database from the backup file PERS_BU.RBF and requests a new version of the database file:

```
$ RMU/RESTORE/NEW_VERSION/AFTER_JOURNAL=AIJ_DISK:PERSAIJ -  
_$_ /NOCDI_INTEGRATE/LOG PERS_BU -  
_$_ EMPIDS_INFO /THRESHOLDS=(65,75,80)/BLOCKS_PER_PAGE=3
```

The command changes the after-image journal file location and name to AIJ_DISK:PERSAIJ.AIJ, prevents integration with the data dictionary, and displays the progress of the restore operation. For the storage area, EMPIDS_INFO, the command changes the SPAM threshold values to 65, 75, and 80 percent, and increases the number of blocks per page to 3 blocks.

RMU/RESTORE Command

Assume that at 10 a.m., Wednesday, October 25, 1989, a disk device hardware failure corrupts all the files on the device, including the PERSONNEL.RDB file. The following command restores the full database backup (PERS_FULL_OCT22.RBF) performed on the previous Sunday and then restores the incremental backup files made on Tuesday. Note that an incremental database backup was performed on Monday, but each new incremental backup made since the latest full backup replaces previous incremental backups made since the last full backup.

```
$ RMU/RESTORE/LOG MUA1:PERS_FULL_OCT22.RBF
$ RMU/RESTORE/INCREMENTAL/CONFIRM/LOG PERS_INCR_OCT24.RBF
```

At this point, the database is current up until 11:30 p.m., Tuesday, when the last incremental backup was made of PERSONNEL. Because after-image journaling is enabled for this database, the AIJ file from 11:30 p.m., Tuesday, until just before the hardware failure can be applied to the restored PERSONNEL.RDB file and its storage area files. For example:

```
$ DEFINE LIB$DT_INPUT_FORMAT "!DB-!MAAU-!Y4 !H04:!M0:!S0.!C2"
$ RMU/RECOVER/UNTIL = "25-OCT-1989 09:55:00.00" AIJ_DISK:PERS.AIJ
```

Example 2

If a storage area is on a disk that crashes, you may want to move that storage area to another disk using RMU/RESTORE. The following RMU/RESTORE command restores only the EMPIDS_OVER storage area from the full backup of MF_PERSONNEL, and moves the EMPIDS_OVER storage area and snapshot file to a new location on the 333\$DUA11 disk.

```
$ SET DEFAULT 222$DUA20:[BACKUPS]
$ RMU/RESTORE/AREA 222$DUA20:[BACKUPS]MF_PERS_BU.RBF -
_$ EMPIDS_OVER /FILE=333$DUA11:[DBS]EMPIDS_OVER.RDA -
_$ /SNAPSHOT=(FILE=$333$DUA11:[DBS]EMPIDS_OVER.SNP)
```

RMU/RESTORE Command

Example 3

The following example demonstrates how you can use by-area backup and restore operations for a single storage area in the MF_PERSONNEL database:

```
$ ! Give yourself sufficient privilege for the operations to follow:
$ SET PROCESS/PRIVILEGE=SYSPRV
$ !
$ SET DEFAULT DISK1:[USER]
$ !
$ ! Create the AIJ file for the database. Digital Equipment Corporation
$ ! strongly recommends that you enable after-image journaling for any
$ ! database on which you will perform by-area backup and restore
$ ! operations.
$ RDO
RDO> CHANGE DATABASE FILENAME 'MF_PERSONNEL'
cont> JOURNAL FILE IS 'DISK2:[AIJ_FILES]MF_AIJ.AIJ'.
RDO> EXIT
$ !
$ RDO
RDO> INVOKE DATABASE FILENAME 'MF_PERSONNEL'
RDO> !
RDO> ! On Monday, define a new record in the DEPARTMENTS table. The
RDO> ! new record is stored in the DEPARTMENTS storage area.
RDO> START_TRANSACTION READ_WRITE
RDO> !
RDO> STORE D IN DEPARTMENTS USING
cont>   D.DEPARTMENT_CODE = "WLNS";
cont>   D.DEPARTMENT_NAME = "Wellness Center";
cont>   D.MANAGER_ID = "00188";
cont>   D.BUDGET_PROJECTED = 0;
cont>   D.BUDGET_ACTUAL = 0;
cont> END_STORE
RDO> COMMIT
RDO> FINISH
RDO> EXIT
$ !
$ ! Assume that you know that the only storage area ever updated in
$ ! the MF_PERSONNEL database on Tuesdays is the SALARY_HISTORY
$ ! storage area, and you decide that you will do an incremental
$ ! backup of just the SALARY_HISTORY storage area on Tuesday.
$ ! Before you perform the by-area backup of SALARY_HISTORY on
$ ! Tuesday, you must perform a full and complete backup of the
$ ! MF_PERSONNEL database when it is in a known and working state.
$ RMU/BACKUP/LOG MF_PERSONNEL.RDB DISK3:[BACKUPS]MF_MONDAY_FULLL.RBF
%RMU-I-BCKTXT_01, Thread 1 uses devices DISK3:
%RMU-I-BCKTXT_08, Thread 1 was assigned file DISK1:[USER]MF_PERS_DEFAULT.RDA;1
%RMU-I-BCKTXT_08, Thread 1 was assigned file DISK1:[USER]SALARY_HISTORY.RDA;1
%RMU-I-BCKTXT_08, Thread 1 was assigned file DISK1:[USER]EMPIDS_LOW.RDA;1
.
.
.
%RMU-I-BCKTXT_06,      backed up 0 logical area bitmap pages
%RMU-I-BCKTXT_07,      backed up 27 data pages
```

RMU/RESTORE Command

```
$ !
$ RDO
RDO> INVOKE DATABASE FILENAME 'MF_PERSONNEL'
RDO> !
RDO> ! On Tuesday, two new records are stored in the SALARY_HISTORY
RDO> ! storage area.
RDO> START_TRANSACTION READ_WRITE
RDO> !
RDO> FOR SH IN SALARY_HISTORY WITH
cont>   SH.SALARY_START = "14-JAN-1983 00:00:00.00" AND
cont>   SH.EMPLOYEE_ID = "00164"
cont>     MODIFY SH USING SH.SALARY_END = "20-JUL-1989 00:00:00.00"
cont>     END_MODIFY
cont> END_FOR
RDO> !
RDO> FOR SH IN SALARY_HISTORY WITH
cont>   SH.SALARY_START = "5-JUL-1980 00:00:00.00" AND
cont>   SH.EMPLOYEE_ID = "00164"
cont>     MODIFY SH USING SH.SALARY_START = "3-JUL-1980 00:00:00.00"
cont>     END_MODIFY
cont> END_FOR
RDO> COMMIT
RDO> FINISH
RDO> EXIT
$ !
$ ! On Tuesday, you do an incremental backup of the SALARY_HISTORY
$ ! storage area only. All database storage areas except the
$ ! SALARY_HISTORY storage area are explicitly excluded from the
$ ! by-area backup file. RMU provides an informational message telling
$ ! you that not all storage areas in the database are included in
$ ! the MF_TUESDAY_PARTIAL.RBF backup file.
$ RMU/BACKUP/EXCLUDE=(RDB$SYSTEM,MF_PERS_SEGSTR,EMPIDS_LOW,EMPIDS_MID, -
_$ EMPIDS_OVER,DEPARTMENTS,JOBS,EMP_INFO)/INCREMENTAL/LOG -
_$ DISK1:[USER]MF_PERSONNEL.RDB DISK3:[BACKUPS]MF_TUESDAY_PARTIAL.RBF
%RMU-I-NOTALLARE, Not all areas will be included in this backup file
%RMU-I-LOGLASCOM, Last full and complete backup was dated 7-AUG-1989 23:35:13.69
%RMU-I-BCKTXT_01, Thread 1 uses devices DISK3:
%RMU-I-BCKTXT_08, Thread 1 was assigned file DISK1:[USER]SALARY_HISTORY.RDA;1
%RMU-I-BCKTXT_00, Backed up root file DISK1:[USER]MF_PERSONNEL.RDB;1
%RMU-I-BCKTXT_03, Incremental backup of storage area DISK1:[USER]SALARY_HISTORY.RDA;1
%RMU-I-BCKTXT_03, Incremental backup of storage area DISK1:[USER]SALARY_HISTORY.RDA;1
%RMU-I-BCKTXT_04,      ignored 1 space management page
%RMU-I-BCKTXT_05,      backed up 0 inventory pages
%RMU-I-BCKTXT_06,      backed up 0 logical area bitmap pages
%RMU-I-BCKTXT_07,      backed up 1 data page
$ !
$ RDO
RDO> INVOKE DATABASE FILENAME 'MF_PERSONNEL'
RDO> !
RDO> START_TRANSACTION READ_WRITE
RDO> !
```

RMU/RESTORE Command

```
RDO> ! Add another new record to the SALARY_HISTORY table:
RDO> FOR SH IN SALARY_HISTORY WITH
cont>     SH.SALARY_START = "21-SEP-1981 00:00:00.00" AND
cont>     SH.EMPLOYEE_ID = "00164"
cont>     MODIFY SH USING SH.SALARY_START = "23-SEP-1981 00:00:00.00"
cont>     END_MODIFY
cont> END_FOR
RDO> COMMIT
RDO> EXIT

$ ! Assume that a disk device hardware error occurs here and only the
$ ! SALARY_HISTORY storage area is lost. Also assume that the root
$ ! file and other storage areas in the database are still fine and
$ ! do not need to be restored or recovered. Therefore, you do not
$ ! need to restore the root file or other storage areas from the
$ ! full and complete backup file. Because only the SALARY_HISTORY
$ ! storage area was lost, you can restore just the SALARY_HISTORY
$ ! storage area from the latest incremental backup file. In this
$ ! example, there is only one incremental backup file.
$ RMU/RESTORE/NOCCD_INTEGRATE/INCREMENTAL/LOG -
_$ /AREA DISK3:[BACKUPS]MF_TUESDAY_PARTIAL.RBF SALARY_HISTORY
%RMU-I-REXTXT_04, Thread 1 uses devices DISK3:
DISK1:[USER]MF_PERSONNEL.RDB;1, restore incrementally? [N]:Y
%RMU-I-LOGRESSST, restored storage area DISK1:[USER]SALARY_HISTORY.RDA;1
%RMU-I-LOGRESSST, restored storage area DISK1:[USER]SALARY_HISTORY.RDA;1
%RMU-I-REXTXT_09,      initialized 0 space management pages
%RMU-I-REXTXT_10,      restored 0 inventory pages
%RMU-I-REXTXT_11,      initialized 0 logical area bitmap pages
%RMU-I-REXTXT_12,      restored 1 data page
%RMU-I-REXTXT_13,      initialized 0 data pages
%RMU-I-REXTXT_01, Initialized snapshot file DISK1:[USER]SALARY_HISTORY.SNP;1
%RMU-I-LOGINIFIL,      contains 10 pages, each page is 2 blocks long
%RMU-I-LOGCREAIJ, created after-image journal file DISK2:[AIJ_FILES]MF_AIJ.AIJ;2
$ !
$ ! Now finish recovering the database by applying the AIJ file that
$ ! was in effect when the full and complete backup file and the
$ ! incremental by-area backup file were created. Note that because
$ ! the database was restored by area in the previous step, you do not
$ ! need to specify the areas to be recovered. RMU will recover the
$ ! necessary areas up to the current time and then stop applying
$ ! transactions.
$ RMU/RECOVER/TRACE DISK2:[AIJ_FILES]MF_AIJ.AIJ;1
%RMU-I-LOGOPNAIJ, opened journal file DISK2:[AIJ_FILES]MF_AIJ.AIJ;1
%RMU-I-LOGRECSTAT, transaction with TSN 80 ignored
%RMU-I-LOGRECSTAT, transaction with TSN 81 ignored
%RMU-I-LOGRECSTAT, transaction with TSN 88 ignored
%RMU-I-RESTART, restarted recovery after ignoring 3 committed transactions
%RMU-I-LOGRECSTAT, transaction with TSN 96 committed
$ !
$ ! The recovery of MF_PERSONNEL is now complete
```

RMU/RESTORE Command

Example 4

In the following example, the options file requests that the EMPIDS_LOW.RDA, EMPIDS_MID.RDA, and EMPIDS_OVER.RDA storage area files and the EMPIDS_LOW.SNP, EMPIDS_MID.SNP, and EMPIDS_OVER.SNP snapshot files be restored in the USER3:[CORPORATE] directory. Other database storage area and snapshot files not specified in the options file are restored to the disk and directory specified in the backup file, MF_PERS_BCK.RBF:

```
$ SET DEFAULT USER1:[SUPERS]
$ TYPE OPTIONS_FILE.OPT
EMPIDS_LOW /FILE=USER3:[CORPORATE]EMPIDS_LOW.RDA -
           /SNAPSHOT=(FILE=USER3:[CORPORATE]EMPIDS_LOW.SNP)
EMPIDS_MID /FILE=USER3:[CORPORATE]EMPIDS_MID.RDA -
           /SNAPSHOT=(FILE=USER3:[CORPORATE]EMPIDS_MID.SNP)
EMPIDS_OVER /FILE=USER3:[CORPORATE]EMPIDS_OVER.RDA -
           /SNAPSHOT=(FILE=USER3:[CORPORATE]EMPIDS_OVER.SNP)
$ !
$ RMU/RESTORE/NOCCD_INTEGRATE/OPTIONS=OPTIONS_FILE.OPT MF_PERS_BCK.RBF
```

6.27 RMU/SET AUDIT Command

Enables Rdb/VMS security auditing. When security auditing is enabled, Rdb/VMS sends security alarm messages to terminals that have been enabled as security operators and makes entries in the database's security audit journal whenever specified audit events are detected.

Format

RMU/SET AUDIT database-file-name

Command Qualifiers

/DISABLE=enable-disable-options
 /ENABLE=enable-disable-options
 /[NO]EVERY
 /FIRST
 /[NO]FLUSH
 /START
 /STOP
 /TYPE={ALARM|AUDIT}

Defaults

See description
 See description
 /EVERY
 Synonym for /NOEVERY
 /NOFLUSH
 ALARM and AUDIT

Description

The RMU/SET AUDIT command is the Rdb/VMS equivalent to the VMS DCL command, SET AUDIT. Because Rdb/VMS security auditing uses many VMS system-level auditing mechanisms, certain auditing characteristics (such as /FAILURE_MODE) can only be set and modified using the VMS SET AUDIT command, which requires the VMS SECURITY privilege.

Command Parameters

database-file-name

The name of the database for which auditing information will be modified.

RMU/SET AUDIT Command

Command Qualifiers

/DISABLE=enable-disable-options

Disables security auditing for the specified audit event classes. To disable alarms and audits for all classes, specify the ALL option. You can also selectively disable alarms and audits for one or more classes that are currently enabled. You must specify at least one class when you specify the /DISABLE qualifier. See the /ENABLE qualifier description for a list of the classes you can specify with the /DISABLE qualifier.

When you specify audit classes with the /DISABLE option, the events you specify are immediately disabled. For other audit events that have not been explicitly disabled with the /DISABLE option, records will continue to be recorded in the security audit journal and alarms will continue to be sent to security-enabled terminals, as specified.

In processing the RMU/SET AUDIT command, Rdb/VMS processes the /DISABLE qualifier last. If you accidentally specify both /ENABLE and /DISABLE for the same event type in the same command, the /DISABLE qualifier prevails.

/ENABLE=enable-disable-options

Enables security auditing for the specified audit event classes. To enable alarms and audits for all events, specify the ALL option. You can also selectively enable alarms and audits for one or more classes that are currently disabled. You must specify at least one class when you specify the /ENABLE qualifier.

When you specify audit classes with the /ENABLE option, the audit events you specify are immediately enabled, so that audit events of currently attached users are recorded in the security audit journal and alarms are sent to security-enabled terminals, as specified.

With the /ENABLE and /DISABLE qualifiers, you can specify one or more of the following six valid classes: ALL, DACCESS, DACCESS=object-type, IDENT=(identifier-list), PROTECTION, and RMU. If you specify more than one class, separate the classes with commas and enclose the list of classes within parentheses. The following list provides a description of each class:

- ALL
Enables or disables all possible audit event classes.
- DACCESS
Enables or disables DACCESS (discretionary access) audit events.

RMU/SET AUDIT Command

A DACCESS audit event occurs whenever a user issues a statement that causes a check to be made for the existence of the appropriate privilege in an access privilege set (APS). To monitor access to a particular database object or group of objects, use the DACCESS=object-type class to specify that a DACCESS audit record be produced whenever an attempt is made to access the object.

Specifying the general DACCESS class enables or disables the general DACCESS audit event type. If DACCESS event auditing is enabled and started for specific objects, auditing takes place immediately after you issue the RMU/SET AUDIT/ENABLE=DAACCESS command. Auditing starts for any users specified in the IDENT=(identifier-list) class who are attached to the database when the command is issued.

- DACCESS=object-type[(object-name-list)]/PRIV=(privilege-list)

Allows you to audit access to database objects by users in the IDENT=(identifier-list) class with the privileges you specify.

A DACCESS type event record indicates the statement issued, the privilege used by the process issuing the command, and whether or not the attempt to access the object was successful.

The object-type enables or disables DACCESS auditing for the specified object type. You can specify one or more object types in an RMU/SET AUDIT command. The three valid object types are:

- SCHEMA

When you specify the SCHEMA object type, you must use the /PRIV option to specify one or more privileges to be audited for the database. Do not specify an object-name with the SCHEMA object type.

- TABLE

Specify the TABLE option for both tables and views. When you specify the TABLE object type, you must specify one or more table names with the object-name-list parameter. You must also use the /PRIV option to specify one or more privileges to be audited for the specified tables.

- COLUMN

When you specify the COLUMN object type, you must specify one or more column names with the object-name-list parameter. Specify the table name that the column is part of by using the following syntax:

```
table-name.column-name
```

RMU/SET AUDIT Command

If you specify more than one column, separate the list of table-name.column-names with commas and enclose the list in parentheses. You must also use the /PRIV option to specify one or more privileges to be audited for the specified columns.

The object-name-list enables or disables DACCESS auditing for the specified object or objects. If you specify more than one object-name, separate the object names with commas and enclose the list of object names within parentheses.

If you specify one or more object-names, you must select one or more privileges to audit. Use the /PRIV=privilege-list option to select the privileges that are to be audited for each of the objects in the object-name list when the selected objects are accessed. The privileges that can be specified with the /PRIV option are listed in Table 6-3.

Privilege names SUCCESS and FAILURE can be used as a convenient way to specify that all successful or failed accesses to that object for all privileges should be audited. The privilege name ALL can be used with the /ENABLE or /DISABLE qualifier to turn on or turn off auditing for all privileges applicable to the object.

If you specify a privilege that does not apply to an object, Rdb/VMS allows it, but will not produce any auditing for that privilege. You can specify only SQL privileges with the PRIV=(privilege-list) qualifier. The privileges that can be specified for each Rdb/VMS object type are shown in Table 6-3. The RDO privileges that correspond to the SQL privileges are included in Table 6-3 to help RDO users select the appropriate SQL privileges for auditing.

Table 6-3 DACCESS Privileges for Database Objects

SQL PRIVILEGE	RDO PRIVILEGE	SCHEMA	TABLE/VIEW	COLUMN
ALTER	CHANGE	Y	Y	N
CREATETAB	DEFINE	Y	Y	N
DBADM	ADMINISTRATOR	Y	N	N
DBCTRL	CONTROL	Y	Y	N
DELETE	ERASE	N	Y	N

(continued on next page)

RMU/SET AUDIT Command

Table 6-3 (Cont.) DACCESS Privileges for Database Objects

SQL PRIVILEGE	RDO PRIVILEGE	SCHEMA	TABLE/VIEW	COLUMN
DISTRIBTRAN	DISTRIBTRAN	Y	N	N
DROP	DELETE	Y	Y	N
INSERT	WRITE	N	Y	N
OPERATOR	OPERATOR	Y	N	N
REFERENCES	REFERENCES	N	Y	Y
SECURITY	SECURITY	Y	N	N
SELECT	READ	Y	Y	N
UPDATE	MODIFY	N	Y	Y
SUCCESS	SUCCESS	Y	Y	Y
FAILURE	FAILURE	Y	Y	Y
ALL	ALL	Y	Y	Y

■ IDENT=(identifier-list)

Enables or disables auditing of user access to objects listed in the /ENABLE=DACCESS=object-type qualifier. If you do not specify this class, no users will be audited for the DACCESS event. Any user whose identifier you specify will be audited for accessing the database objects with the privileges specified. You can specify wildcard characters within the identifiers to identify groups of users. The [*,*] identifier indicates public, and causes all users to be audited. If you specify a non-existent identifier, you receive an error message.

The order of identifiers in the identifier list is not significant. A user will be audited if the user holds any of the identifiers specified in the identifier list.

You can specify UIC identifiers, general identifiers, and system-defined identifiers in the identifier list. For more information on identifiers, see Section 9.17.

If you specify more than one identifier, separate the identifiers with commas and enclose the identifier list within parentheses. UIC identifiers

RMU/SET AUDIT Command

with commas such as [RDB,JONES] must be enclosed within double quotes, as follows:

```
IDENT=( INTERACTIVE , "[ RDB , JONES ] " , SECRETARIES )
```

- **PROTECTION**

Allows you to audit changes made to access privilege sets (APSs) for database objects by means of the RDO DEFINE PROTECTION, CHANGE PROTECTION, and DELETE PROTECTION statements and the SQL GRANT and REVOKE statements.

- **RMU**

Audits the use of RMU commands by users with the privilege to use them.

/EVERY

/NOEVERY

Sets the granularity of DACCESS event auditing for the database. When you specify */EVERY*, every access check for the specified objects using the specified privilege or privileges during a database attach is audited. When you specify */NOEVERY*, each user's first access check for the specified audit objects using the specified privilege or privileges during a database attach is audited. The */FIRST* qualifier is a synonym for the */NOEVERY* qualifier; the two qualifiers can be used interchangeably.

The default is */EVERY*.

/FIRST

Specifies that when DACCESS event auditing is enabled, each user's first access check for the specified audit objects using the specified privilege or privileges during a database attach is audited. The */FIRST* qualifier is a synonym for the */NOEVERY* qualifier; the two qualifiers can be used interchangeably.

/FLUSH

/NOFLUSH

Indicates whether forced writes of audit journal records are currently enabled for the database. Forced writes will cause Rdb/VMS to flush the audit journal record immediately out to disk when the audit record is produced, rather than waiting for the AUDIT_SERVER to flush the audit records every INTERVAL seconds.

RMU/SET AUDIT Command

The default is /NOFLUSH, which flushes audit records every interval seconds. To specify the interval, use the DCL command SET AUDIT /INTERVAL=JOURNAL_FLUSH=time.

/START

Starts Rdb/VMS security auditing for the database. The /START qualifier by itself starts both security alarms and security audit journal records. You can also supply the /TYPE=ALARM or /TYPE=AUDIT qualifier to start only security alarms or security audit journaling.

When the /START qualifier is specified, auditing starts immediately for all audit event classes that are currently enabled. Any subsequent audit events of currently attached users are recorded in the security audit journal, or alarms are sent to security-enabled terminals, or both, depending on what you have specified for your database.

/STOP

Stops Rdb/VMS security auditing for the database. The /STOP qualifier by itself stops both security alarms and security audit journal records. You can also supply the /TYPE=ALARM or /TYPE=AUDIT qualifier to stop only security alarms or security audit journaling.

When the /STOP qualifier is specified, the alarms, or audits (or both) of all audit event classes are immediately stopped (depending on whether the /TYPE=ALARM, /TYPE=AUDIT option, or neither is specified). The audit event classes previously specified with the /ENABLE qualifier remain enabled, and you can start them again by using the /START qualifier.

/TYPE=option

The following options are available with the /TYPE qualifier:

- **ALARM**

Causes subsequent qualifiers in the command line (/START, /STOP, /ENABLE, /DISABLE) to generate or affect security alarm messages that are sent to all terminals enabled as security operator terminals.

- **AUDIT**

Causes subsequent qualifiers in the command line (/START, /STOP, /ENABLE, /DISABLE) to generate or affect security audit journal records that are recorded in the security audit journal.

RMU/SET AUDIT Command

If you do not specify the /TYPE qualifier on the RMU/SET AUDIT command line, RMU enables or disables both security alarms and security audit journal records.

Usage Notes

You must have the VMS SECURITY privilege or the Rdb/VMS or SQL database SECURITY privilege to use the RMU/SET AUDIT command.

You can use the DACCESS=object-type class to enable DACCESS checking for specific objects, but the general DACCESS class is not enabled until you explicitly enable it using the /ENABLE=DACCESS option of the RMU/SET AUDIT command. Also, you need to use the /START qualifier of the RMU/SET AUDIT command to start the auditing and alarms that have been enabled.

Alarms are useful for real-time tracking of auditing information. At the moment an alarm occurs, text messages regarding the alarm are displayed on security-enabled terminals.

To enable a terminal to receive Rdb/VMS security alarms, enter the VMS command REPLY/ENABLE=SECURITY. You must have the VMS SECURITY privilege to use the REPLY/ENABLE=SECURITY command.

Audit records are useful for periodic reviews of security events. Audit records are stored in a security audit journal file, and can be reviewed after they have been loaded into a database table with the RMU/LOAD/AUDIT command. Use the DCL command SHOW AUDIT/JOURNAL to determine the security audit journal being used by your database.

The AUDIT class is always enabled for both alarms and audit records, but will not produce any alarms or audit records until auditing is started. The AUDIT class cannot be disabled.

When you specify the DACCESS=object-type class and one or more other classes in a class list, the /PRIV=(privilege-list) parameter must begin after the close parenthesis for the class list.

To display the results of an RMU/SET AUDIT command, enter the RMU/SHOW AUDIT command.

RMU/SET AUDIT Command

Examples

Example 1

In the following example, the first command enables alarms for the RMU and PROTECTION classes. The second command shows that alarms for the RMU and PROTECTION classes are enabled but not yet started. The AUDIT class is always enabled and cannot be disabled. The third command starts alarms for the RMU and PROTECTION classes. The fourth command shows that alarms for the RMU and PROTECTION classes are enabled and started.

```
$ ! Enable alarms for RMU and PROTECTION classes:
$ RMU/SET AUDIT/TYPE=ALARM/ENABLE=(RMU,PROTECTION) MF_PERSONNEL.RDB
$ !
$ ! Show that alarms are enabled, but not yet started:
$ RMU/SHOW AUDIT/ALL MF_PERSONNEL.RDB
Security auditing STOPPED for:
  PROTECTION (disabled)
  RMU (disabled)
  AUDIT (enabled)
  DACCESS (disabled)

Security alarms STOPPED for:
  PROTECTION (enabled)
  RMU (enabled)
  AUDIT (enabled)
  DACCESS (disabled)

Audit flush is disabled

Audit every access

Enabled identifiers:
  None

$ ! Start alarms for the enabled RMU and PROTECTION classes:
$ RMU/SET AUDIT/START/TYPE=ALARM MF_PERSONNEL.RDB
$ !
$ ! Show that alarms are started for the RMU and PROTECTION classes:
$ RMU/SHOW AUDIT/ALL MF_PERSONNEL
Security auditing STOPPED for:
  PROTECTION (disabled)
  RMU (disabled)
  AUDIT (enabled)
  DACCESS (disabled)

Security alarms STARTED for:
  PROTECTION (enabled)
  RMU (enabled)
  AUDIT (enabled)
  DACCESS (disabled)

Audit flush is disabled

Audit every access
```

RMU/SET AUDIT Command

```
Enabled identifiers:  
  None
```

```
$
```

Example 2

In this example, the first command shows that alarms are started and enabled for the RMU class. The second command disables alarms for the RMU class. The third command shows that alarms for RMU class are disabled:

```
$ ! Show that alarms are enabled and started for the RMU class:  
$ RMU/SHOW AUDIT/ALL MF_PERSONNEL.RDB  
Security auditing STOPPED for:  
  PROTECTION (disabled)  
  RMU (disabled)  
  AUDIT (enabled)  
  DACCESS (disabled)  
  
Security alarms STARTED for:  
  PROTECTION (disabled)  
  RMU (enabled)  
  AUDIT (enabled)  
  DACCESS (disabled)  
  
Audit flush is disabled  
  
Audit every access  
  
Enabled identifiers:  
  None  
  
$ ! Disable alarms for the RMU class:  
$ RMU/SET AUDIT/TYPE=ALARM/DISABLE=RMU MF_PERSONNEL.RDB  
$ !  
$ ! Show that alarms are disabled for the RMU class:  
$ RMU/SHOW AUDIT/ALL MF_PERSONNEL.RDB  
Security auditing STOPPED for:  
  PROTECTION (disabled)  
  RMU (disabled)  
  AUDIT (enabled)  
  DACCESS (disabled)  
  
Security alarms STOPPED for:  
  PROTECTION (disabled)  
  RMU (disabled)  
  AUDIT (enabled)  
  DACCESS (disabled)  
  
Audit flush is disabled  
  
Audit every access  
  
Enabled identifiers:  
  None  
  
$
```

RMU/SET AUDIT Command

Example 3

In this example, the first command enables auditing for users with the [SQL,RICK] and [RDB,SCHMIDT] identifiers. The second command shows the enabled identifiers. The third command enables DACCESS checks requiring SELECT and INSERT privileges for the EMPLOYEES and COLLEGES tables. The fourth command displays the DACCESS checks that have been specified for the COLLEGES and EMPLOYEES tables. Note that because the general DACCESS type has not been enabled, DACCESS for the EMPLOYEES and COLLEGES tables is displayed as "disabled."

```
$ ! Enable auditing for users with the [SQL,RICK] and
$ ! [RDB,SCHMIDT] identifiers:
$ RMU/SET AUDIT/ENABLE=IDENT=("[SQL,RICK]", "[RDB,SCHMIDT]") -
_$ MF_PERSONNEL
$ !
$ ! Show that [SQL,RICK] and [RDB,SCHMIDT] are enabled identifiers:
$ RMU/SHOW AUDIT/ALL MF_PERSONNEL
Security auditing STOPPED for:
    PROTECTION (disabled)
    RMU (disabled)
    AUDIT (enabled)
    DACCESS (disabled)

Security alarms STOPPED for:
    PROTECTION (disabled)
    RMU (disabled)
    AUDIT (enabled)
    DACCESS (disabled)

Audit flush is disabled

Audit every access

Enabled identifiers:
    (IDENTIFIER=[SQL,RICK])
    (IDENTIFIER=[RDB,SCHMIDT])

$ ! Enable and start DACCESS checks for the SELECT and INSERT
$ ! privileges for the COLLEGES and EMPLOYEES tables:
$ RMU/SET AUDIT/ENABLE=DACCESS=TABLE=(COLLEGES,EMPLOYEES) -
_$ /PRIV=(SELECT,INSERT)/START MF_PERSONNEL.RDB
$ !
$ ! Display the DACCESS checks that are enabled and
$ ! started for the COLLEGES and EMPLOYEES tables:
$ RMU/SHOW AUDIT/DACCESS=TABLE MF_PERSONNEL
Security auditing STARTED for:
    DACCESS (disabled)
        TABLE : EMPLOYEES
            (SELECT,INSERT)
        TABLE : COLLEGES
            (SELECT,INSERT)
```

RMU/SET AUDIT Command

```
Security alarms STARTED for:
  DACCESS (disabled)
    TABLE : EMPLOYEES
      (SELECT,INSERT)
    TABLE : COLLEGES
      (SELECT,INSERT)
```

```
$
```

Example 4

In this example, the first command enables auditing of the JOBS and EMPLOYEES table for DACCESS checks for users with the [SQL,RICK] or BATCH identifier. The /PRIV=ALL specifies that auditing will be produced for every privilege. The second command shows that auditing is enabled for users with the [SQL,RICK] or BATCH identifier. The third command shows that DACCESS checking for the JOBS and EMPLOYEES tables for all privileges is specified. The fourth command enables the general DACCESS class. The fifth command's output shows that the general DACCESS class is now enabled. The sixth command starts the auditing that is enabled, and the seventh command shows that the enabled auditing is started:

```
$ ! Enable DACCESS checks for users with the [SQL,RICK] or
$ ! BATCH identifier for the JOBS and EMPLOYEES tables:
$ RMU/SET AUDIT/TYPE=AUDIT -
_$/ENABLE=(IDENT=("[SQL,RICK]",BATCH),DACCESS=TABLE=(JOBS,EMPLOYEES)) -
_$/PRIV=ALL MF_PERSONNEL.RDB
$ !
$ ! Show that auditing is enabled for users with the [SQL,RICK]
$ ! or BATCH identifiers:
$ RMU/SHOW AUDIT/ALL MF_PERSONNEL
Security auditing STOPPED for:
  PROTECTION (disabled)
  RMU (disabled)
  AUDIT (enabled)
  DACCESS (disabled)

Security alarms STOPPED for:
  PROTECTION (disabled)
  RMU (disabled)
  AUDIT (enabled)
  DACCESS (disabled)

Audit flush is disabled

Audit every access

Enabled identifiers:
  (IDENTIFIER=[SQL,RICK])
  (IDENTIFIER=BATCH)
```

RMU/SET AUDIT Command

```
$ ! Show that DACCESS checking for all privileges for the
$ ! JOBS and EMPLOYEES tables is enabled:
$ RMU/SHOW AUDIT/DACCESS=TABLE MF_PERSONNEL
Security auditing STOPPED for:
    DACCESS (disabled)
      TABLE : EMPLOYEES
        (ALL)
      TABLE : JOBS
        (ALL)

Security alarms STOPPED for:
    DACCESS (disabled)

$ ! Enable the general DACCESS class:
$ RMU/SET AUDIT/ENABLE=DACCESS MF_PERSONNEL
$ !
$ ! Show that the general DACCESS class is enabled:
$ RMU/SHOW AUDIT/DACCESS=TABLE MF_PERSONNEL
Security auditing STOPPED for:
    DACCESS (enabled)
      TABLE : EMPLOYEES
        (ALL)
      TABLE : JOBS
        (ALL)

Security alarms STOPPED for:
    DACCESS (enabled)

$ ! Start the auditing that is enabled:
$ RMU/SET AUDIT/START MF_PERSONNEL
$ !
$ ! Show that the enabled auditing is started:
$ RMU/SHOW AUDIT/ALL MF_PERSONNEL
Security auditing STARTED for:
    PROTECTION (disabled)
    RMU (disabled)
    AUDIT (enabled)
    DACCESS (enabled)

Security alarms STARTED for:
    PROTECTION (disabled)
    RMU (disabled)
    AUDIT (enabled)
    DACCESS (enabled)

Audit flush is disabled

Audit every access

Enabled identifiers:
    (IDENTIFIER=[SQL,RICK])
    (IDENTIFIER=BATCH)

$
```

RMU/SET AUDIT Command

Example 5

In this example, the first command enables DACCESS checks requiring the INSERT privilege for the MF_PERSONNEL database, for the EMPLOYEES table, and for the EMPLOYEE_ID column of the EMPLOYEES table. The second command shows that the DACCESS check for the INSERT privilege is enabled for the specified objects.

```
$ ! Enable a DACCESS check for the INSERT privilege for the
$ ! MF_PERSONNEL database, EMPLOYEES table, and EMPLOYEE_ID
$ ! column of the EMPLOYEES table:
$ RMU/SET AUDIT -
_ $ /ENABLE=DACCESS=(SCHEMA, TABLE=EMPLOYEES, COLUMN=EMPLOYEES.EMPLOYEE_ID) -
_ $ /PRIV=(INSERT) MF_PERSONNEL
$ !
$ ! Show that the DACCESS check for the INSERT privilege is
$ ! enabled for the specified objects:
$ RMU/SHOW AUDIT/DACCESS=(SCHEMA, TABLE, COLUMN) MF_PERSONNEL
Security auditing STOPPED for:
  DACCESS (disabled)
    SCHEMA
      (INSERT)
    TABLE : EMPLOYEES
      (INSERT)
    COLUMN : EMPLOYEES.EMPLOYEE_ID
      (INSERT)

Security alarms STOPPED for:
  DACCESS (disabled)
    SCHEMA
      (INSERT)
    TABLE : EMPLOYEES
      (INSERT)
    COLUMN : EMPLOYEES.EMPLOYEE_ID
      (INSERT)

$
```

Example 6

In this example, the first command enables a DACCESS check requiring the INSERT privilege for the EMPLOYEES and COLLEGES tables, as well as for the EMPLOYEE_ID and LAST_NAME columns of the EMPLOYEES table and the COLLEGE_CODE column of the COLLEGES table in the MF_PERSONNEL database. The second command shows that the DACCESS check for the INSERT privilege is enabled for the specified objects.

RMU/SET AUDIT Command

```
$ ! Enable a DACCESS check for the INSERT privilege for the
$ ! EMPLOYEES and COLLEGES table, the LAST_NAME and EMPLOYEE_ID
$ ! column of the EMPLOYEES table, and the COLLEGE_CODE column
$ ! of the COLLEGES table:
$ RMU/SET AUDIT -
_$/ENABLE=DACCESS=(TABLE=(EMPLOYEES,COLLEGES), -
_$/COLUMN=(EMPLOYEES.EMPLOYEE_ID, -
_$/EMPLOYEES.LAST_NAME, -
_$/COLLEGES.COLLEGE_CODE)) -
_$/PRIV=(INSERT) MF_PERSONNEL
$ !
$ ! Show that the DACCESS check for the INSERT privilege is
$ ! enabled for the specified objects:
$ RMU/SHOW AUDIT/DACCESS=(SCHEMA,TABLE,COLUMN) MF_PERSONNEL
Security auditing STOPPED for:
  DACCESS (disabled)
    SCHEMA
      (NONE)
    TABLE : EMPLOYEES
      (INSERT)
    TABLE : COLLEGES
      (INSERT)
    COLUMN : COLLEGES.COLLEGE_CODE
      (INSERT)
    COLUMN : EMPLOYEES.EMPLOYEE_ID
      (INSERT)
    COLUMN : EMPLOYEES.LAST_NAME
      (INSERT)
Security alarms STOPPED for:
  DACCESS (disabled)
    SCHEMA
      (NONE)
    TABLE : EMPLOYEES
      (INSERT)
    TABLE : COLLEGES
      (INSERT)
    COLUMN : COLLEGES.COLLEGE_CODE
      (INSERT)
    COLUMN : EMPLOYEES.EMPLOYEE_ID
      (INSERT)
    COLUMN : EMPLOYEES.LAST_NAME
      (INSERT)
$
```

RMU/SHOW Command

6.28 RMU/SHOW Command

Displays current information about security audit characteristics, version numbers, active databases, active users, active recovery-units, or database statistics related to database activity on your VAX node. Note that in a VAXcluster, the RMU/SHOW command displays information for your current node only.

You must choose one of five output options: AUDIT, STATISTICS, SYSTEM, USERS, or VERSION. The option determines which command qualifiers apply and whether the database file specification is necessary or meaningful.

Format

RMU/SHOW option [database-file-name]

Description

The AUDIT and STATISTICS options have syntax and output that are entirely different from those of SYSTEM, USERS, and VERSION. Consequently, the five variants of the RMU/SHOW command are described separately in the following sections.

Command Parameters

option

The type of information to display. It must be one of the following:

- AUDIT
- STATISTICS
- SYSTEM
- USERS
- VERSION

database-file-name

The name of the database on which you want information. This parameter is meaningful when you select STATISTICS or USERS as the option.

RMU/SHOW AUDIT Command

6.28.1 RMU/SHOW AUDIT Command

Displays the set of security auditing characteristics that have been established with the RMU/SET AUDIT command.

Format

RMU/SHOW AUDIT database-file-name

Command Qualifiers

/ALL
/DACCESS[=object-type[...]]
/EVERY
/FLUSH
/IDENT
/PROTECTION
/OUTPUT[=file-spec]
/RMU
/TYPE={ALARM|AUDIT}

Defaults

/OUTPUT=SYS\$OUTPUT
ALARM and AUDIT

Description

The RMU/SHOW AUDIT command is the Rdb/VMS equivalent to the VMS DCL command, SHOW AUDIT. Because Rdb/VMS security auditing uses many VMS system-level auditing mechanisms, certain auditing characteristics such as /FAILURE_MODE can only be displayed using the VMS SHOW AUDIT command, which requires the VMS SECURITY privilege.

Command Parameters

database-file-name

The name of the database for which auditing information will be displayed.

Command Qualifiers

/ALL

Displays all available auditing information for the database, including the following: whether security auditing and security alarms are started or

RMU/SHOW AUDIT Command

stopped, types of security events currently enabled for alarms and audits, identifiers currently enabled for auditing, and the forced audit write setting.

/DACCESS[=object-type[,...]]

Specifying */DACCESS* displays whether the general *DACCESS* audit event class is currently enabled. Specifying one or more object types with */DACCESS* displays the object types and their associated privileges that are currently enabled for *DACCESS* event auditing. If you specify more than one object-type, enclose the list of object-types within parentheses.

The valid object types are:

- SCHEMA
- TABLE
- COLUMN

/EVERY

Displays the current setting for first or every *DACCESS* event auditing for the database.

/FLUSH

Displays the current setting for forced writes of audit journal records for the database.

/IDENT

Displays the UICs of the users who are currently enabled for *DACCESS* event auditing of specified objects.

/PROTECTION

Displays whether auditing is currently enabled for the *PROTECTION* audit event class.

/OUTPUT[=file-spec]

Controls where the output of the command is sent. If you do not enter the qualifier, or if you enter */OUTPUT* without a file specification, the output is sent to the current process default output stream or device, identified by the logical name *SYSS\$OUTPUT*.

/RMU

Displays whether auditing is currently enabled for the *RMU* event class.

RMU/SHOW AUDIT Command

/TYPE={ALARM|AUDIT}

Displays information about security alarms or security auditing. If you do not specify the */TYPE* qualifier, RMU displays information about both security alarms and security auditing.

Usage Notes

You need the VMS SECURITY privilege or the Rdb/VMS or SQL SECURITY database privilege to use the RMU/SHOW AUDIT command.

If you do not specify any qualifiers with the RMU/SHOW AUDIT command, the currently enabled alarm and audit security events are displayed.

Use the RMU/SHOW AUDIT command to check which auditing features are enabled whenever you plan to enable or disable audit characteristics with a subsequent RMU/SET AUDIT command.

Examples

Example 1

The following command shows that alarms are enabled for the RMU and PROTECTION audit classes for the MF_PERSONNEL database. Note that the display shows that alarms are also enabled for the AUDIT audit class. The AUDIT class is always enabled and cannot be disabled.

```
$ RMU/SHOW AUDIT/ALL MF_PERSONNEL
Security auditing STOPPED for:
  PROTECTION (disabled)
  RMU (disabled)
  AUDIT (enabled)
  DACCESS (disabled)

Security alarms STOPPED for:
  PROTECTION (enabled)
  RMU (enabled)
  AUDIT (enabled)
  DACCESS (disabled)

Audit flush is disabled

Audit every access

Enabled identifiers:
  None
```

RMU/SHOW AUDIT Command

Example 2

In the following example, the first command enables and starts alarms for the RMU audit class for the MF_PERSONNEL database. Following the first command is the alarm that is displayed on a security terminal when the first command is executed. The second command displays the auditing characteristics that have been enabled and started. Because the RMU/SHOW AUDIT/ALL command is an RMU command, it causes the alarm at the end of the example to be displayed on the security terminal. Note that security-enabled terminals only receive alarms if alarms have been both enabled *and* started.

```
$ RMU/SET AUDIT/TYPE=ALARM/ENABLE=RMU/START MF_PERSONNEL
%%%%%%%%%% OPCOM 22-JUN-1990 09:41:01.19 %%%%%%%%%%%
Message from user RICK on MYNODE
Rdb/VMS Security alarm (SECURITY) on MYNODE, system id:      32327
Database name:          DDV21:[RICK.SQL]MF_PERSONNEL.RDB;1
Auditable event:        Auditing change
Event time:             22-JUN-1990 09:41:01.17
PID:                    44802B9E
User name:              RICK
RMU command:            RMU/SET AUDIT/TYPE=ALARM/ENABLE=RMU/START MF_PERSONNEL
Final status:          %SYSTEM-S-NORMAL

$ RMU/SHOW AUDIT/ALL MF_PERSONNEL
Security auditing STOPPED for:
  PROTECTION (disabled)
  RMU (disabled)
  AUDIT (enabled)
  DACCESS (disabled)

Security alarms STARTED for:
  PROTECTION (disabled)
  RMU (enabled)
  AUDIT (enabled)
  DACCESS (disabled)

Audit flush is disabled

Audit every access
```

RMU/SHOW AUDIT Command

Enabled identifiers:
None

```
%%%%%%%%% OPCOM 22-JUN-1990 09:43:07.94 %%%%%%%%%%  
Message from user RICK on MYNODE  
Rdb/VMS Security alarm (SECURITY) on MYNODE, system id:      32327  
Database name:      DDV21:[RICK.SQL]MF_PERSONNEL.RDB;1  
Auditable event:    Attempted RMU command  
Event time:        22-JUN-1990 09:43:07.92  
PID:               44802B9E  
User name:         RICK  
RMU command:       RMU/SHOW AUDIT/ALL MF_PERSONNEL  
Final status:      %SYSTEM-S-NORMAL
```

\$

RMU/SHOW Statistics Command

6.28.2 RMU/SHOW STATISTICS Command

Displays usage statistics for a database. See the *VAX Rdb/VMS Guide to Database Maintenance and Performance* for tutorial information on how to interpret the output of the RMU/SHOW commands.

Format

RMU/SHOW STATISTICS [database-file-name]

Command Qualifiers

/OUTPUT = file-name
/INPUT = file-name
/[NO]INTERACTIVE
/[NO]LOG
/UNTIL = "date-time"
/TIME = integer

Defaults

See description
See description
See description
Current DCL verify value
See description
/TIME = 3 seconds

Description

The RMU/SHOW STATISTICS command dynamically samples activity statistics on a database. You can display the statistics on your terminal and can also write them to a formatted binary file.

The statistics show activity only from the VAXcluster node on which the command is executed.

The RMU/SHOW STATISTICS command operates in one of two modes, online and replay. In online mode, you can display or record current activity on a database. In replay mode, you can examine a previously recorded binary statistics file.

If you use the /INPUT qualifier, the RMU/SHOW STATISTICS command will execute in replay mode. In replay mode, this command generates an interactive display from a previously recorded binary statistics file.

If you do not use the /INPUT qualifier, then you must specify a database file name. The RMU/SHOW STATISTICS command will then execute in online mode. In online mode, the command can generate an interactive display (/INTERACTIVE) and can also record statistics in a binary file.

RMU/SHOW Statistics Command

The interactive display is made up of output pages. These pages include:

- Summary I/O statistics
- Index statistics
- VM (virtual memory) usage statistics
- Summary locking statistics
- AIJ statistics
- PIO (physical I/O) statistics
- Record statistics
- Snapshot statistics
- I/O stall time statistics
- Transaction durations
- Stall messages
- I/O statistics (by file)
- Locking statistics (one lock type)
- Locking statistics (one statistical field)

You can control the interactive display by means of menus, using arrow keys to select options. You can display most statistics in either a histogram or a columnar chart, although several display pages have special formats. In addition, you can produce time plot graphics for individual statistical fields. An extensive online help facility is available by:

- 1 Typing H
- 2 Placing the cursor over the item for which you want information

Use the `/OUTPUT` qualifier to direct statistical output to a file. The output is a formatted binary file and does not produce a legible printed listing. To read the output, you must use the `RMU/SHOW STATISTICS` command with the `/INPUT` qualifier.

The `/NOINTERACTIVE` qualifier suppresses the interactive display. Use this when you want to generate binary statistics output but do not want an online display.

RMU/SHOW Statistics Command

Database statistics are maintained in a global section on each computer on which Rdb/VMS is running. Statistics are reset to zero when you close a database. Running a RMU/SHOW STATISTICS command keeps the database open even when there are no users accessing the database.

You can use the RMU/SHOW STATISTICS command and then select the Stall Messages option from the Select Display window to get information on VMS locks.

Command Parameter

database-file-name

The name of the database on which you want statistics. If you use the /INPUT qualifier to supply a prerecorded binary statistics file, then you cannot specify a database file name. If you do not use the /INPUT qualifier, then you must specify a database file name.

Command Qualifiers

/OUTPUT=file-name

A binary statistics file into which the statistics will be written.

/INPUT=file-name

The prerecorded binary file from which you can read the statistics. This file must have been created by an earlier RMU/SHOW STATISTICS session that specified the /OUTPUT qualifier.

You cannot specify a database file name with the /INPUT qualifier. Also, you must not use the /UNTIL, /OUTPUT, or /NOINTERACTIVE qualifiers with the /INPUT qualifier. However, you can use the /TIME qualifier to change the rate of the display. This will not change the computed times as recorded in the original session. For example, you can record a session at /TIME=60. This session will gather statistics once per minute.

You can replay statistics gathered in a file by using the /TIME qualifier. To replay a file:

- Use the /OUTPUT qualifier to create a file of database statistics.
- Use the /INPUT and /TIME qualifiers to view the statistics again at a rate that you determine. For example, the command RMU/SHOW STATISTICS/PERS.LOG/TIME=1, will replay the PERS.LOG file and change the display once per second, thus replaying 10 hours of statistics in 10 minutes.

RMU/SHOW Statistics Command

If you do not specify the /INPUT qualifier, you must specify the database-file-name parameter.

/INTERACTIVE **/NOINTERACTIVE**

Displays the statistics dynamically on your terminal. The /INTERACTIVE qualifier is the default when you execute the RMU/SHOW STATISTICS command from a terminal. You can use the /NOINTERACTIVE qualifier with the /OUTPUT qualifier to generate a binary statistics file without generating a terminal display. The /NOINTERACTIVE qualifier is the default when you execute the RMU/SHOW STATISTICS command from a batch job.

In an interactive session, you can use either the menu interface or the predefined control characters to select display options (see online help for further information about the predefined control characters).

Select menu options by using the up and down arrow keys followed by pressing RETURN or ENTER. Cancel the menu by pressing CTRL/Z.

/LOG **/NOLOG**

Logs the creation of a binary statistics file to your SYSSOUTPUT file. This binary statistics file is created only if you have used the /OUTPUT qualifier. If you use the /NOLOG qualifier, no operations will be logged to your SYSSOUTPUT file.

The default is the current setting of the DCL verify switch. See HELP SET VERIFY in the DCL documentation for more information on changing the DCL verify switch.

/UNTIL=" date-time"

The time the statistics collection ends. When this point is reached, the RMU/SHOW STATISTICS command terminates and control returns to the DCL command level. When the RMU/SHOW STATISTICS command is executed in a batch job, the batch job will terminate at the time specified.

Note The /UNTIL qualifier accepts VMS Version 5.0 and higher date and time strings, as well as international dates. See VMS RTL Library (LIB\$) Manual for more information. The date and time strings specified must be in quotes because they can contain spaces and other DCL syntax characters such as commas. A colon separator between the data and time field is no longer accepted.

RMU/SHOW Statistics Command

An example is:

```
$ DEFINE LIB$DT_INPUT_FORMAT "!MAU !DB, !Y4 !H04:!M0:!S0.!C2"  
$ RMU/SHOW STATISTICS /UNTIL="JUNE 16, 1989 17:00:00.00" -  
_ $ MF_PERSONNEL
```

This stops execution of the RMU/SHOW STATISTICS command at 5 p.m. on June 16, 1989. You can omit the date if you wish to use the default of today's date.

If you do not use the /UNTIL qualifier, the RMU/SHOW STATISTICS command continues until you terminate it manually. In an interactive session, terminate the command by pressing CTRL/Z or by selecting Exit from the menu. When running the RMU/SHOW STATISTICS command with the /NOINTERACTIVE qualifier from a terminal, terminate the RMU/SHOW STATISTICS operation by pressing CTRL/C or CTRL/Y and then selecting Exit. When running the RMU/SHOW STATISTICS command in a batch job, terminate the operation by deleting the batch job.

/TIME=integer

The statistics collection interval in seconds. If you omit this qualifier, a sample collection will be made every 3 seconds. The integer has a *normal* range of 1 to 180 (1 second to 3 minutes). However, if you specify a negative number for the /TIME qualifier, the RMU/SHOW STATISTICS command interprets the number as hundredths of a second. For example, /TIME=-20 specifies an interval of 20/100 or 1/5 of a second.

If you are running the RMU/SHOW STATISTICS command interactively, it updates the screen display at the specified interval.

If you are also using the /OUTPUT qualifier, a binary statistics record is written to the output file at the specified interval. A statistics record is not written to this file if no database activity has occurred since the last record was written.

Usage Notes

You must have the Rdb/VMS ADMINISTRATOR or OPERATOR (SQL DBADM or OPERATOR) privilege to use the RMU/SHOW STATISTICS command.

RMU/SHOW Statistics Command

Examples

Example 1

The following example directs the results of the RMU/SHOW STATISTICS command to an output file:

```
$ RMU/SHOW STATISTICS PERSONNEL/OUTPUT=PERS.LOG
```

Example 2

The following example formats the binary results created in the previous example and produces a readable display:

```
$ RMU/SHOW STATISTICS/INPUT=PERS.LOG
```

RMU/SHOW System Command

6.28.3 RMU/SHOW SYSTEM Command

Displays a summary of which databases are in use on a particular node. It is the same as the RMU/SHOW USERS command, except that it has no database-file-name parameter. You can use it only to see systemwide user information.

Format

RMU/SHOW SYSTEM

Command Qualifier

/OUTPUT = file-name

Default

/OUTPUT = SYS\$OUTPUT

Description

The RMU/SHOW SYSTEM command displays information about all active database users on a particular node.

Command Qualifier

/OUTPUT=file-name

The name of the file where output will be sent. SYS\$OUTPUT is the default. The default output file type is LIS, if you specify a file name.

Usage Notes

You must have the VMS WORLD privilege to use the RMU/SHOW SYSTEM command.

RMU/SHOW Users Command

6.28.4 RMU/SHOW USERS Command

Displays information about active database users. It allows you to see the user activity of specified databases on a specific VAX node.

Format

RMU/SHOW USERS [database-file-name]

Command Qualifier

/OUTPUT = file-name

Default

/OUTPUT = SYS\$OUTPUT

Description

The RMU/SHOW USERS command displays information about all active database users or users of a particular database.

Command Parameter

database-file-name

The name of the database for which you want information. This parameter is optional. If you specify it, only users of that database are shown. Otherwise, all users of all active databases are shown.

Command Qualifier

/OUTPUT=file-name

The name of the file where output will be sent. SYS\$OUTPUT is the default. The default output file type is LIS, if you specify a file name.

Usage Notes

You must have the VMS WORLD privilege to use the RMU/SHOW USERS command if you do not specify a database. You must have the Rdb/VMS ADMINISTRATOR or OPERATOR (SQL DBADM or OPERATOR) privilege to specify the name of a database with the RMU/SHOW USERS command.

RMU/SHOW Users Command

Examples

Example 1

This command lists current USERS information in the file DBUSE.LIS.

```
$ RMU/SHOW USERS/OUTPUT=DBUSE USERS
```

RMU/SHOW Version Command

6.28.5 RMU/SHOW VERSION Command

Displays the Rdb/VMS software version number.

Format

RMU/SHOW VERSION

Command Qualifier

/OUTPUT = file-name

Default

/OUTPUT = SYS\$OUTPUT

Command Qualifier

/OUTPUT=file-name

The name of the file where output will be sent. SYS\$OUTPUT is the default. The default output file type is LIS, if you specify a file name.

Usage Notes

You do not need any special privileges to use the RMU/SHOW VERSION command.

Examples

Example 1

The following command displays the current version of Rdb/VMS software:

```
$ RMU/SHOW VERSION
Executing RMU for Rdb/VMS V4.0
```

RMU/UNLOAD Command

6.29 RMU/UNLOAD Command

Copies the data from a specified table or view of the database into a:

- Specially formatted file
- Sequential RMS file

Data from the specially formatted file can be reloaded only by using the RMU/LOAD command. Data from the sequential RMS file could be reloaded using an alternative utility such as DATATRIEVE.

Format

RMU/UNLOAD database-file-name table-name output-file-name

Command Qualifiers

/BUFFERS=n
/FIELDS=(column-name-list)
/RMS_RECORD_DEF=FILE=file-name

Defaults

See description
See description
See description

Description

The RMU/UNLOAD command copies data from a specified table or view and places it in a specially structured file or in a sequential RMS file. Be aware that the RMU/UNLOAD command does not remove data from the specified table; it merely makes a copy of the data.

The RMU/UNLOAD can be used to:

- Extract data for an application that cannot access the Rdb/VMS database directly
- Create an archival copy of data
- Perform restructuring operations
- Sort data by defining a view with a sorted-by clause, then unloading that view

RMU/UNLOAD Command

The specially structured files created by the RMU/UNLOAD command contain metadata for the table that was unloaded. The RMS files created by RMU/UNLOAD contain only data; the metadata can be found either in the data dictionary or in the RRD file created using the /RMS_RECORD_DEF qualifier. Specify the /RMS_RECORD_DEF qualifier to exchange data with an application that uses sequential RMS files.

The segmented string data type cannot be unloaded into a sequential RMS file; however, it can be unloaded into the specially structured file type. Data type conversions are valid only if Rdb/VMS supports the conversion.

The RMU/UNLOAD command executes a read-only transaction to gather the metadata and user data to be unloaded. It is compatible with all operations that do not require exclusive access.

Command Parameter

database-file-name

The Rdb/VMS database from which tables or views will be unloaded. The default file type is RDB.

table-name

The table or view name to be unloaded.

output-file-name

The destination file name. The default file type is UNL.

Command Qualifiers

/BUFFERS=n

Specifies the number of database buffers used for the unload operation. If no value is specified, the default value for the database is used. Although this qualifier might affect the performance of the unload operation, the default number of buffers for the database usually allows adequate performance.

/FIELDS=(column-name-list)

Specifies the column or columns of the table or view to be unloaded from the database. If you list multiple columns, separate the column names with a comma, and enclose the list of column names in parentheses. This qualifier also specifies the order in which the columns should be unloaded if that order differs from what is defined for the table or view. Changing the structure of the table or view could be useful when restructuring a database or when migrating

RMU/UNLOAD Command

data between two databases with different metadata definitions. The default is all the columns defined for the table or view in the order defined.

/RMS_RECORD_DEF=FILE=file-name

Creates a sequential RMS file containing the record structure definition for the output file. The record description uses the CDO record and field definition format. The default file type is RRD.

Usage Notes

You must have either the Rdb/VMS ADMINISTRATOR or OPERATOR (SQL DBADM or OPERATOR) privilege and the Rdb/VMS READ (SQL SELECT) privilege to the table or view being unloaded to use the RMU/UNLOAD command.

Examples

Example 1

This command unloads the EMPLOYEE_ID and LAST_NAME values from the EMPLOYEES table of the PERSONNEL database. The data is stored in NAMES.UNL.

```
$ RMU/UNLOAD -  
_$/FIELDS=(EMPLOYEE_ID, LAST_NAME) -  
_$/PERSONNEL EMPLOYEES NAMES.UNL
```

Example 2

This command unloads the EMPLOYEES table from the PERSONNEL database and places the data in the RMS file, NAMES.UNL. The NAMES.RRD file contains the record structure definitions for the data in NAMES.UNL.

```
$ RMU/UNLOAD/RMS_RECORD_DEF=FILE=NAMES.RRD PERSONNEL EMPLOYEES NAMES.UNL
```

6.30 RMU/VERIFY Command

Checks the internal integrity of database data structures. The RMU/VERIFY command does not verify the data itself. You can verify specific portions of a database using qualifiers.

Format

RMU/VERIFY root-file-spec

Command Qualifiers

/CHECKSUM_ONLY
 /INCREMENTAL
 /[NO]ROOT
 /ALL
 /INDEXES [= index-list]
 /[NO]DATA
 /[NO]LOG
 /OUTPUT=file-spec
 /CONSTRAINTS
 /SNAPSHOTS
 /AREAS [= storage-area-list]
 /LAREAS [= logical-area-list]
 /START=page-number
 /END=page-number
 /TRANSACTION_TYPE=option

Defaults

Full page verification
 See description
 /ROOT
 See description
 No index checking performed
 /DATA when /INDEXES is used
 Current DCL verify value
 SYS\$OUTPUT
 No constraint evaluation
 No snapshot verification
 No AREA checking performed
 No LAREA checking performed
 /START=1
 /END=last-page
 /TRANSACTION_TYPE=PROTECTED

If you specify RMU/VERIFY without any qualifiers, a root file verification and full page verification of the AIP and ABM pages in the default RDB\$SYSTEM storage area are performed. Also, the snapshot files are validated. If AIJ journaling is enabled, the AIJ file is validated.

The RMU/VERIFY command checks SPAM pages for proper format. The contents of the individual entries are verified as the individual data pages are verified. The command does not attempt to determine whether data within records is reasonable or plausible.

RMU/VERIFY Command

Description

The RMU/VERIFY command checks the internal integrity of database data structures. Digital Equipment Corporation strongly recommends that you verify your database following any kind of serious system malfunction. You should also verify your database as part of routine maintenance, perhaps before performing backups. You can use the various qualifiers to perform the maximum verification in the time available.

The RMU/VERIFY command checks only for corrupt data structures. It does not check for invalid data.

Command Parameter

root-file-spec

The Rdb/VMS database to verify. The default file type is RDB.

Command Qualifiers

/CHECKSUM_ONLY

Specify the */CHECKSUM_ONLY* qualifier with the */AREAS* qualifier to perform only checksum verification of pages. This reduces the degree of verification done on a database page. While the RMU/VERIFY command executes faster with the */CHECKSUM_ONLY* qualifier than without it, it does not verify pages completely. This qualifier allows you to make trade-offs between speed of verification and thoroughness of verification. For more information on these trade-offs, see the *VAX Rdb/VMS Guide to Database Maintenance and Performance*.

You can also specify the */INCREMENTAL*, */START*, and */END* qualifiers with this qualifier.

If this command finds a problem with a certain page, then that page can be verified in depth by using other qualifiers, such as the */INDEX*, */AREA* or */LAREA* qualifiers.

You cannot specify the */ALL* qualifier with the */CHECKSUM_ONLY* qualifier.

The default is for full verification of pages.

/INCREMENTAL

Directs RMU to verify database pages that have changed since the last full or incremental verification. RMU stores time stamps in the root file for both full and incremental verifications. To determine which pages have changed since

RMU/VERIFY Command

the last verify operation, RMU compares these time stamps with the page time stamps. The page time stamps are updated whenever pages are updated. An incremental verification performs the same number of I/O operations as a full verification, but the incremental verification takes fewer CPU cycles than a full verification, allowing you to perform incremental verifications more frequently than you would perform full ones. The default is to perform a full verification.

Note *If you use the /INCREMENTAL qualifier with RMU/VERIFY, Digital Equipment Corporation recommends that you use it only with the /ALL qualifier and not with any other qualifiers.*

The date and time stamps in the database root file are updated during full and incremental verifications only when the /ALL qualifier is specified. Therefore, if you do not specify the /ALL qualifier, two successive incremental verifications of the same storage area of the database perform the same verifications. This means that the second incremental verification will not skip pages verified by the first incremental verification, contrary to what you might expect.

If the /INCREMENTAL qualifier is not specified, all requested pages are verified, regardless of the time stamp.

/ROOT /NOROOT

The default /ROOT qualifier specifies that, in a multifile database, only fields in the root (RDB) file and all the pointers to the database (RDA, SNP, AIJ) files are verified. The snapshot files are validated; that is, only the first page is checked to make sure that it is indeed a snapshot file and belongs to the database being verified. If AIJ journaling is enabled, the AIJ file is validated. AIP and ABM pages are verified when you specify the /ROOT qualifier.

If you specify the /NOROOT qualifier, and no other qualifiers, only the AIP pages are verified. If you specify the /NOROOT qualifier, and the /AREAS or /LAREAS qualifier, ABM and SPAM pages are verified as the other pages in the storage area or logical area are verified.

You cannot specify the /ALL qualifier with the /ROOT qualifier.

You can specify the /ROOT qualifier for a single-file database.

RMU/VERIFY Command

/ALL

When you specify the /ALL qualifier, the entire database is checked. Specifying the /ALL qualifier is equivalent to issuing the list of qualifiers in the following command:

```
$ RMU/VERIFY/ROOT/CONSTRAINTS/INDEXES/DATA/AREAS -  
_$/SNAPSHOTS/LAREAS MF_PERSONNEL.RDB
```

If you specify the /ALL qualifier, the only other qualifiers you can specify in the same RMU/VERIFY command are /LOG, /OUTPUT, and /TRANSACTION_TYPE.

If you do not specify the /ALL qualifier, then the verification requested by the other qualifiers you have specified is performed.

/INDEXES [= index-list]

Verifies the integrity of all indexes in the database (/INDEXES or /INDEXES=*), of a specific index, or of multiple indexes you list. If you list multiple indexes, separate the index names with a comma, and enclose the index-list in parentheses.

You cannot use the /ALL, /START, or /END qualifiers with the /INDEXES qualifier.

By default, RMU does not verify any index structures.

/DATA

/NODATA

Specifies whether the data records are fetched when indexes are verified.

When the /DATA qualifier is specified, the data records are fetched. When the /NODATA qualifier is specified, the data records are not fetched. The default is /DATA.

If the /DATA qualifier is specified, a much greater number of I/O operations are performed than if the /NODATA qualifier is specified. For the performance implications of using this qualifier, see the *VAX Rdb/VMS Guide to Database Maintenance and Performance*.

The /DATA qualifier is only valid when it is used with the /INDEXES qualifier.

/LOG

/NOLOG

Specifies whether to report the processing of the command on SYSS\$OUTPUT.

By default, SYSS\$OUTPUT is your terminal. Specify the /LOG qualifier to request that each verify operation be displayed on SYSS\$OUTPUT and the /NOLOG qualifier to prevent this display. If you specify neither, the default

RMU/VERIFY Command

is the current setting of the DCL verify switch. (The DCL SET VERIFY command controls the DCL verify switch.)

When you specify the /LOG qualifier, RMU displays the time taken to verify each database area specified and the total time taken for the complete verification operation. The display from /LOG is also useful for showing you how much of the verification operation is completed.

/OUTPUT=file-spec

The name of the file where output will be sent. SYSS\$OUTPUT is the default. When you specify a file name, the default output file type is LIS.

/CONSTRAINTS

All constraints are loaded and executed to check the integrity of data in the database. See the *VAX Rdb/VMS Guide to Database Maintenance and Performance* for more information on verifying constraints.

You cannot use the /ALL, /START, or /END qualifiers with the /CONSTRAINTS qualifier.

/SNAPSHOTS

When /SNAPSHOTS is specified, the snapshot area of the specified storage areas are verified up to the page header level. The /SNAPSHOTS qualifier only performs checksum verification of snapshot pages. The /SNAPSHOTS qualifier can be used only when the /AREAS qualifier is also specified.

/AREAS [= storage-area-list]

Specifies the storage areas of the database to verify. Each storage area name must be the name defined in the RDO DEFINE STORAGE AREA clause or the SQL CREATE STORAGE AREA clause for the storage area, not the storage area file name. If you list multiple storage areas, separate the storage area names with a comma and enclose the storage-area-list in parentheses. The /AREAS qualifier with no arguments (or /AREAS=*) directs RMU to verify all storage areas of the database. With a single-file database, if you do not specify a storage area name, the RDB\$SYSTEM storage area is verified.

You cannot use the /ALL qualifier with the /AREAS qualifier.

When /AREAS is not specified, RMU does not verify any storage areas.

/LAREAS [= logical-area-list]

Specifies the storage area pages allocated to a logical area or logical areas that you want verified. If you list multiple logical areas, separate the logical area names with a comma and enclose the logical-area-list in parentheses. The

RMU/VERIFY Command

/LAREAS qualifier with no arguments (or **/LAREAS=***) directs RMU to verify all logical areas of the database.

If an index name is specified with the **/LAREAS** qualifier, the index is skipped; it is not verified as a logical area.

Use this qualifier to verify one or more tables.

You cannot use the **/ALL** qualifier with the **/LAREAS** qualifier.

By default, RMU does not verify logical areas.

/START=page-number

The first page to be verified. This qualifier is used in conjunction with the **/AREAS** and **/LAREAS** qualifiers. If you do not use the **/START** qualifier, the verification begins with the first page of the storage area.

You cannot use the **/CONSTRAINTS**, **/INDEXES**, or **/ALL** qualifiers with the **/START** qualifier.

/END=page-number

The last page to be verified. This qualifier is used in conjunction with the **/AREAS** and **/LAREAS** qualifiers. If you do not use the **/END** qualifier, RMU verifies all pages between the first page (or the page specified in the **/START** qualifier) and the last page of the storage area.

You cannot use the **/CONSTRAINTS**, **/INDEXES**, or **/ALL** qualifiers with the **/END** qualifier.

/TRANSACTION_TYPE=option

Sets the retrieval lock for the storage areas being verified. Valid options are **EXCLUSIVE**, **PROTECTED**, and **READ_ONLY**. The default retrieval mode is **PROTECTED**.

You can select only one of the three options (**EXCLUSIVE**, **PROTECTED**, or **READ_ONLY**) in an **RMU/VERIFY** command.

When you select **/TRANSACTION_TYPE=READ_ONLY**, you perform an online verification. Other users can access the database while the online verification is proceeding.

RMU/VERIFY Command

Usage Notes

You must have the Rdb/VMS ADMINISTRATOR or OPERATOR (SQL DBADM or OPERATOR) privilege to use the RMU/VERIFY command.

You can significantly improve the performance of RMU/VERIFY for your database by employing the verification strategies described in the *VAX Rdb/VMS Guide to Database Maintenance and Performance*.

Examples

Example 1

The following command verifies the entire MF_PERSONNEL database because the /ALL qualifier was specified:

```
$ RMU/VERIFY/ALL/LOG MF_PERSONNEL.RDB
```

Example 2

The following command verifies the storage areas EMPIDS_LOW, EMPIDS_MID, and EMPIDS_OVER in the MF_PERSONNEL database.

```
$ RMU/VERIFY/AREAS=(EMPIDS_LOW,EMPIDS_MID,EMPIDS_OVER)/LOG -  
_ $ MF_PERSONNEL.RDB
```

Example 3

The following command performs only a checksum verification on all the storage areas in the database called LARGE_DATABASE. The /CHECKSUM_ONLY qualifier quickly detects obvious checksum problems with the database. If a checksum problem is found on a page, you can dump the page using the RMU/DUMP command and verify the appropriate logical areas and indexes:

```
$ RMU/VERIFY/AREAS=*/CHECKSUM_ONLY/LOG LARGE_DATABASE
```

Example 4

The following command verifies the CANDIDATES and COLLEGES tables:

```
$ RMU/VERIFY/LAREAS=(CANDIDATES,COLLEGES)/LOG MF_PERSONNEL.RDB
```

RdbALTER Utility Command Syntax

This chapter describes the RdbALTER utility for VAX Rdb/VMS. The RdbALTER utility provides a low-level patch capability that allows you to repair corruption on VAX Rdb/VMS database pages. In addition, it allows you to relocate database root, area, and snapshot files to other disks or directories. This is helpful when you move a database from a single node to a VAXcluster environment.

Issue the RMU/ALTER command as follows:

```
$ RMU/ALTER [root-file-spec]
```

The optional root file parameter identifies the database you wish to alter. If you specify this parameter, you automatically attach to the specified database. If you do not specify this parameter, you must use the RdbALTER ATTACH command. See Section 7.2 for more information on the ATTACH command.

The RMU/ALTER command responds with the following prompt:

```
RdbALTER>
```

This prompt means that the system expects RdbALTER command input. The RdbALTER commands are:

- AREA . . . PAGE
- ATTACH
- COMMIT
- DEPOSIT
- DEPOSIT FILE
- DEPOSIT ROOT
- DETACH
- DISPLAY

- DISPLAY FILE
- DISPLAY ROOT
- EXIT
- HELP
- LOG
- MAKE_CONSISTENT
- MOVE
- NOLOG
- PAGE
- RADIX
- ROLLBACK
- UNCORRUPT
- VERIFY

When you invoke RdbALTER and the database needs to be recovered, RdbALTER displays a message to that effect. If you receive this message, you should recover the database and verify it before making any changes in RdbALTER. If you make changes in RdbALTER before recovering the database, the recovery procedure may overwrite your changes.

RdbALTER can receive command lines through command files or directly from your terminal.

You can abbreviate any of the RdbALTER command keywords. The only restriction is that you specify enough characters to avoid ambiguity.

RdbALTER interprets the exclamation point (!) as a comment character. Any characters on a line following an exclamation point are disregarded by RdbALTER.

You must have the VMS SYSPRV privilege to use RdbALTER.

The remainder of this chapter describes the full syntax and usage of these commands. For detailed information on using the RdbALTER utility, see the *VAX Rdb/VMS Guide to Database Maintenance and Performance*.

7.1 AREA . . . PAGE Command

Specifies a storage area or a storage page of the current attached database that you intend to alter, or both.

Only one page of one area is accessible to RdbALTER at a time. This command switches you from one page of the currently attached database to another.

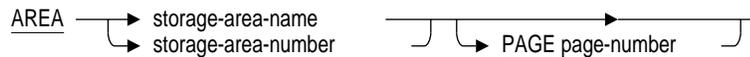
When you ATTACH to a database, RdbALTER automatically makes area 1 the current area and page 1 of that area the current page. To work on any other area and page, you must use the AREA . . . PAGE command.

If you specify AREA but not PAGE, RdbALTER makes the area you specify current and fetches page 1 of that area.

If you specify both AREA and PAGE, RdbALTER makes the area you specify current and fetches the page you specify from the new current area.

If you specify an area or page that does not exist, an error occurs.

Format



Command Parameters

storage-area-name

Specifies an area of the current database by the storage area name, which is the name given by the AREA clause in the schema.

storage-area-number

Specifies an area of the current database by the storage area number, which is assigned when the database is created and is given on the first line of a page display.

page-number

Identifies the area page to be altered. Express it as an integer from 1 to the number of pages in the area.

AREA . . . PAGE Command

Suppose you have been altering area 2, page 23 and now want to work on some other area or page of the database. The following examples show some possible commands.

Examples

Example 1

The following example specifies the area JOBS of the currently attached database. No page number has been specified, so RdbALTER fetches page 1 of the area.

```
RdbALTER> AREA JOBS
```

Example 2

The following example is similar to the previous example, except it specifies the area by its area number instead of its area name. If area 4 is the area named JOBS in the database, either command produces the same result.

```
RdbALTER> AREA 4
```

Example 3

The following example fetches area 3, page 100.

```
RdbALTER> AREA 3 PAGE 100
```

7.2 ATTACH Command

Attaches RdbALTER to a database, putting an exclusive update lock on the database. No other user can access the database while the RdbALTER ATTACH command is in effect.

The database specified by the *root-file-spec* parameter is attached to RdbALTER. Then you can alter the pages of its storage area files. Area 1, page 1 of the database is fetched automatically and remains the current page until you issue an `AREA . . . PAGE` command.

If the `RMU/ALTER` command includes a root file spec parameter, the database to which *root-file-spec* refers is attached to as part of the RdbALTER startup. In this one case, the ATTACH command is unnecessary. Otherwise, no commands changing database pages are allowed until an ATTACH command naming that database is issued.

You can use the ATTACH command to attach to only one database at a time. Before invoking another database for altering, you must use the DETACH command to detach from the current database. See Section 7.7 for more information on the DETACH command.

Format

`ATTACH` → *root-file-spec* →

Command Parameters

root-file-spec

Specifies the root file of the database whose pages you wish to alter. The default file type is RDB.

ATTACH Command

Examples

Example 1

The following example enters RdbALTER command level, then attaches to the PERSONNEL database.

```
$ RMU/ALTER
RdbALTER> ATTACH PERSONNEL
%RMU-I-ATTACH, now altering database "DISK:[USER]PERSONNEL.RDB;1"
```

Example 2

The following example enters RdbALTER command level and attaches to the PERSONNEL database in a single command.

```
$ RMU/ALTER PERSONNEL
%RMU-I-ATTACH, now altering database "DISK:[USER]PERSONNEL.RDB;1"
```

7.3 COMMIT Command

Writes all page changes back to the database since the last COMMIT or ROLLBACK command was entered.

The results of DEPOSIT and MOVE commands are kept in virtual memory until you issue a COMMIT or ROLLBACK command. When you issue a COMMIT command, all pages that you changed since the last COMMIT or ROLLBACK command are written to the database in their new form.

Changes are not permanent until you issue a COMMIT command. If you issue an EXIT command while altered but uncommitted pages exist, an error results. You cannot issue the EXIT command until you first issue either a COMMIT or a ROLLBACK command.

Format

COMMIT →

Examples

Example 1

The following example commits all page changes entered since the last COMMIT or ROLLBACK command.

```
RdbALTER> COMMIT
```

DEPOSIT Command

7.4 DEPOSIT Command

Alters specified data fields on the current database page.

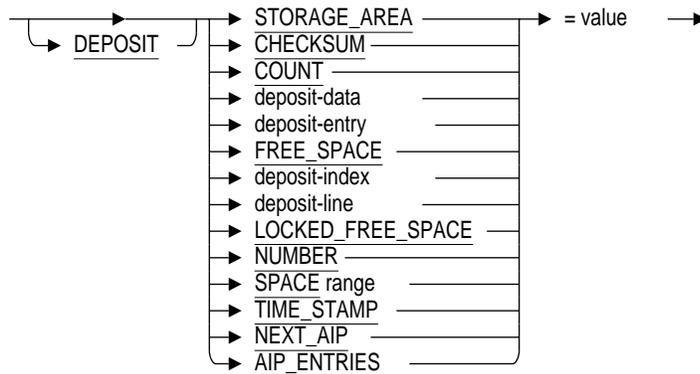
DEPOSIT is the default at RdbALTER command level. If no other command is present following the prompt, RdbALTER automatically parses the command line as a DEPOSIT command.

Specify the field to be altered just as you would specify a field to be displayed in the DISPLAY command.

The specification of the field to be altered must be followed by an equal sign (=) and a string of characters specifying the new value of the altered field.

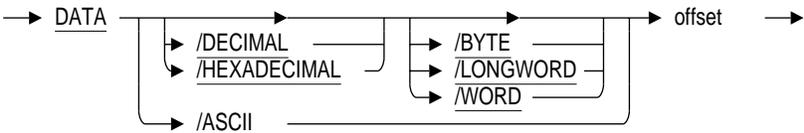
Do not use the DEPOSIT command immediately after a ROLLBACK command. The ROLLBACK command removes currency indicators. For this reason you must respecify your location by using either the DISPLAY or AREA . . . PAGE command immediately after a ROLLBACK command but before a DEPOSIT command.

Format

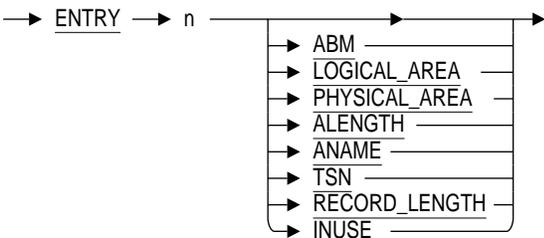


DEPOSIT Command

deposit-data =



deposit-entry =



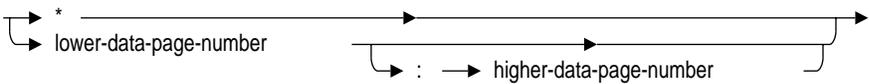
deposit-index =



deposit-line =



range =



DEPOSIT Command

Command Parameters

STORAGE_AREA

Deposits a value for the 2-byte storage area identification.

CHECKSUM

Deposits a value for the 4-byte page checksum field.

COUNT

Deposits a value for the 2-byte field showing the number of line index entries. If this number is 1, the page contains only the SYSTEM record.

DATA offset

Deposits the number of bytes specified. If you do not specify the /HEXADECIMAL or /DECIMAL parameter, the default radix is assumed. See Section 7.18 for information on how to set a default radix with the RADIX command.

The /BYTE, /LONGWORD, and /WORD qualifiers cannot be used with the /ASCII qualifier.

ENTRY

Refers to an area inventory page (AIP) entry on the database page. The value specified for n must be a number between 0 and the number of AIP_ENTRIES.

ABM

Deposits the new value on the first area bitmap page for the specified area inventory page (AIP) entry. The ABM value is contained in a longword.

LOGICAL_AREA

Deposits the new value for the number of the logical area. The LOGICAL_AREA value is contained in a word.

PHYSICAL_AREA

Deposits the new value for the number of the physical area. The PHYSICAL_AREA value is contained in a word.

ALENGTH

Deposits the new value for the length of the name of the logical area. The ALENGTH value is contained in a byte. The name of the logical area can be from 1 to 31 bytes in length.

DEPOSIT Command

ANAME

Deposits the new value for the name of the logical area. The ANAME value is contained in a 31-character text field.

TSN

Deposits a value for the last transaction sequence number (TSN) to enable snapshots for a logical area. The TSN value is contained in a longword.

RECORD_LENGTH

Deposits a value for the length in bytes of the record size. The RECORD_LENGTH value is contained in a word.

INUSE

Deposits the new value for the entry's in-use flag. The INUSE value is contained in a byte.

FREE_SPACE

Deposits a value for the 2-byte field indicating how much free space remains on the page.

Note In the next two arguments, the integers denoting *INDEX* and *LINE* are zero based. For example, *INDEX 0* refers to the first index, and *LINE 3* refers to the fourth line.

References to INDEX and LINE are invalid if the current page is a SPAM page.

INDEX n

Deposits a value for the offset field or the length field for the line index indicated by n. For example, if you enter DEPOSIT INDEX 3 OFFSET, the offset address field from the fourth line index is deposited.

LINE n

Deposits information for an individual storage segment. You can deposit a value for the record-type field.

LOCKED_FREE_SPACE

Deposits a value for the 2-byte field indicating how much free space is allocated for exclusive use by a run unit.

NUMBER

Deposits a value for the 4-byte page number field.

DEPOSIT Command

SPACE range

Deposits a value for a specified range of SPAM entries; it is valid only if the current page is a SPAM page. SPACE is the only option that you can use in DISPLAY and DEPOSIT commands that access a SPAM page. The range value can be an asterisk (*), referring to all entries, or a set of consecutive entries that you describe as follows:

`lower-data-page-number[:higher-data-page-number]`

Each entry on a SPAM page consists of 2 bits, containing a value 0 through 3 that represents a fullness threshold. For example, if the nth SPAM entry contains a 2, it means that the nth data page in the interval has reached a percentage of fullness greater than the second threshold for the area, but less than or equal to the third threshold.

TIME_STAMP

Deposits a value for the 8-byte time and date stamp field.

NEXT_AIP

Deposits a value for the page number of the next area inventory page (AIP).

AIP_ENTRIES

Deposits a value for the number of AIP_ENTRIES on the current area inventory page.

value

Specifies the new value of the field you are altering. The value is deposited in the default radix unless you specify otherwise in one of these ways:

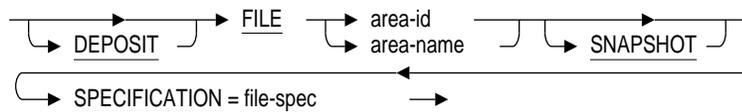
- With a prior RADIX command.
- By specifying HEXADECIMAL or DECIMAL in a DEPOSIT DATA command.
- By enclosing ASCII data in quotation marks (" "). Time stamps must always be enclosed in quotation marks because they include punctuation characters.

DEPOSIT FILE Command

7.5 DEPOSIT FILE Command

Puts a new file specification into the root file for an area file or snapshot file.

Format



Parameters

area-id

Specifies the number of the area or snapshot file whose file specification you are changing in the root file.

area-name

Specifies the area name of the area or snapshot file whose file specification you are changing in the root file.

SNAPSHOT

Specifies that the file whose root file fields you are changing is a snapshot file.

SPECIFICATION = file-spec

Specifies the file specification that the area or snapshot file will have. Use a full file specification (including version number) if the area or snapshot file is not in your default directory. The file must be in the location you specify, otherwise the DEPOSIT FILE command fails.

Examples

Example 1

This example deposits a new file specification for the JOBS storage area file. The word "(marked)" in the DEPOSIT FILE display indicates that the JOBS storage area file is marked for the specified location.

DEPOSIT FILE Command

```
RdbALTER> DISPLAY FILE JOBS
Area JOBS:
    File specification is: "DISK1:[RICK.RDB]JOBS.RDA;1"
    Corrupt flag is: OFF
    Inconsistent flag is: OFF

RdbALTER> DEPOSIT FILE JOBS SPECIFICATION=USER1:[RICK]JOBS.RDA;1
Area JOBS:
(marked) File specification is: "USER1:[RICK]JOBS.RDA;1"
```

See the *VAX Rdb/VMS Guide to Database Maintenance and Performance* for more examples of how to use the DEPOSIT FILE command.

DEPOSIT ROOT Command

7.6 DEPOSIT ROOT Command

Enters the new file specification of the root file. This will be the file specification after it is committed to the database. The change does not occur until you have committed all changes and ended the RdbALTER session.

Note *After you commit the changed root file name, the database cannot be accessed until you copy it to the location (using the exact fully qualified file specification) that you specify on the DEPOSIT ROOT command. After that, users attach to the database at the new location.*

Format



Command Parameters

root-file-spec

Specifies the full file specification (including version number) that the root file will have after it has been committed and the RdbALTER session is complete.

Examples

Example 1

The following example enters a new file specification for the root file. You must specify a version number in the new file specification for the root file. The word "(marked)" in the DEPOSIT ROOT display indicates that root file is marked for the specified location but has not been moved to that location.

```
RdbALTER> DISPLAY ROOT
      Root file specification is: "DISK1:[RICK.RDB]MF_PERSONNEL.RDB;1"
RdbALTER> DEPOSIT ROOT SPECIFICATION="USER1:[RICK]MF_PERSONNEL.RDB;1"
(marked) Root file specification is: "USER1:[RICK]MF_PERSONNEL.RDB;1"
```

See the *VAX Rdb/VMS Guide to Database Maintenance and Performance* for more examples of how to use the DEPOSIT ROOT command.

7.7 DETACH Command

Releases a previous database ATTACH command. The exclusive update lock is released, and other users can access the database again. The database to which RdbALTER is currently attached is released, with the exclusive update lock discontinued. No database page alterations are allowed until another ATTACH command is issued.

Format

DETACH →

Examples

Example 1

The following command detaches RdbALTER from the current database.

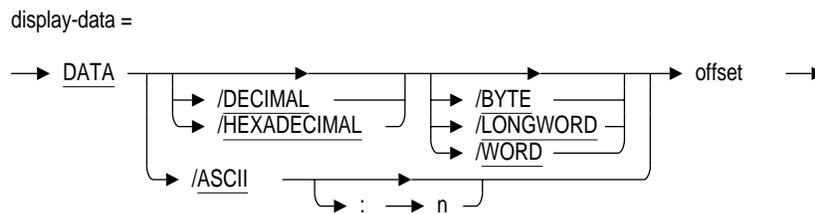
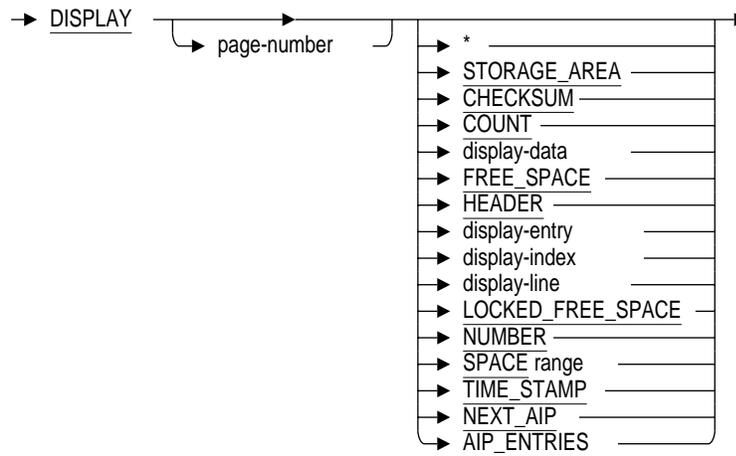
```
RdbALTER> DETACH
```

DISPLAY Command

7.8 DISPLAY Command

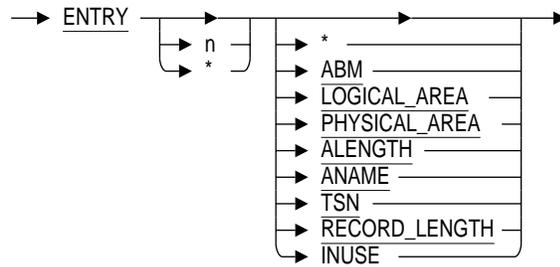
Requests display of data fields from a database page. An individual DISPLAY command can include only one of the display options shown.

Format

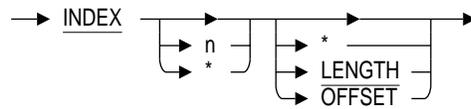


DISPLAY Command

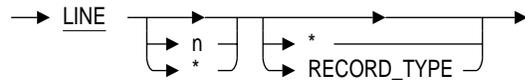
display-entry =



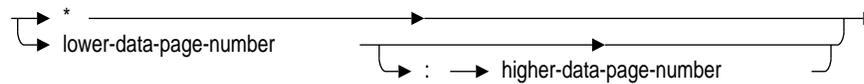
display-index =



display-line =



range =



Parameters

page-number

Identifies the page whose information you want to display. The current page is the default.

DISPLAY Command

DISPLAY *

Displays the entire page.

STORAGE_AREA

Displays the 2-byte storage area identification.

CHECKSUM

Displays the 4-byte page checksum field.

COUNT

Displays the 2-byte field showing the number of line index entries. If this number is 1, the page contains only the SYSTEM record.

DATA offset

Displays the number of bytes specified. If you do not specify the /HEXADECIMAL or /DECIMAL parameter, the default radix is assumed. See Section 7.18 for information on how to set a default radix using the RADIX command.

The value you specify for n is the number of bytes you want to display. The maximum value that can be specified for n is the size of the page minus the offset.

The /BYTE, /LONGWORD, and /WORD qualifiers cannot be used with the /ASCII qualifier.

FREE_SPACE

Displays the 2-byte field indicating how much free space remains on the page.

HEADER

Displays the entire page header.

ENTRY (n|*)

Refers to an area inventory page (AIP) entry on the database page. If you specify a value for n, it must be a number between 0 and the number of AIP_ENTRIES. If you specify the asterisk (*) parameter, the information you request with the other DISPLAY ENTRY parameters is displayed for each AIP entry on the database page.

asterisk (*)

Displays the same information that is displayed when you specify the ABM, LOGICAL_AREA, PHYSICAL_AREA, ALENGTH, ANAME, TSN, RECORD_LENGTH, and INUSE parameters.

DISPLAY Command

ABM

Displays the first area bitmap page for the specified area inventory page (AIP) entry. The value is contained in a longword.

LOGICAL_AREA

Displays the number of the logical area. The value is contained in a word.

PHYSICAL_AREA

Displays the number of the physical area. The value is contained in a word.

ALENGTH

Displays the length of the name of the logical area. The value is contained in a byte. The name of the logical area can be from 1 to 31 bytes in length.

ANAME

Displays the name of the logical area. The value is contained in a 31-character text field.

TSN

Displays the value of the last transaction sequence number (TSN) to enable snapshots for a logical area. The TSN value is contained in a longword.

RECORD_LENGTH

Displays the value for the length in bytes of the record size. The RECORD_LENGTH value is contained in a word.

INUSE

Displays the entry's in-use flag. The value is contained in a byte.

Note *In the next two arguments, the integers denoting INDEX and LINE are zero based. For example, INDEX 0 refers to the first index, and LINE 3 refers to the fourth line.*

The integer n is optional. The present value of the relevant pointer is the default.

References to INDEX and LINE are invalid if the current page is a SPAM page.

INDEX n

Displays the offset field, the length field, or both from the line index indicated by n. For example, if you enter DISPLAY INDEX 3 OFFSET, the offset address field from the fourth line index is displayed. If you enter DISPLAY INDEX 3 LENGTH, the length field from the fourth line index is displayed. If you enter

DISPLAY Command

either DISPLAY INDEX 3 or DISPLAY INDEX 3 *, both the offset and the length fields from the fourth line index are displayed.

LINE n

Displays information from an individual storage segment. You can display the record-type field or the entire content of the storage segment line indicated by n.

LOCKED_FREE_SPACE

Displays the 2-byte field indicating how much free space is allocated for exclusive use by a run unit.

NUMBER

Shows the 4-byte page number field.

SPACE range

Displays SPAM entries; it is valid only if the current page is a SPAM page. SPACE is the only option that you can use in DISPLAY and DEPOSIT commands that access a SPAM page. The optional range value can be an asterisk (*), referring to all entries, or a set of consecutive entries that you describe as follows:

```
lower-data-page-number[:higher-data-page-number]
```

When you specify a range, you reduce the output display. The specified range of SPAM entries is included in the display; other SPAM entries outside your specified range may also be included in the display.

Each entry on a SPAM page consists of 2 bits, containing a value 0 through 3 that represents a fullness threshold. For example, if the nth SPAM entry contains a 2, it means that the nth data page in the interval has reached a percentage of fullness greater than the second threshold for the area, but less than or equal to the third threshold.

TIME_STAMP

Displays the 8-byte time and date stamp field.

NEXT_AIP

Displays the page number of the next area inventory page (AIP).

AIP_ENTRIES

Displays a value for the number of AIP entries on the current area inventory page.

DISPLAY Command

Examples

Example 1

This example displays page 94 of area 1.

```
RdbALTER> AREA 1
RdbALTER> DISPLAY 94
      0001 0000005E 0000 page 94, physical area 1
          7B429FB0 0006 checksum = 7B429FB0
0093249F 9EBDB820 000A time stamp = 14-FEB-1990 10:42:35.81
      0000 0004 0012 4 free bytes, 0 locked

          FFFFFFFF 0016 next area bit map page -1
          00000000 001A max set bit index 0
          00000000 001E MBZ '....'
          00001E60 0022 bitvector count 7776

00000000000000000000000000000000 0026 bitvector '.....'
          :::: (59 duplicate lines)
      0000000000000000000000000000 03E6 bitvector '.....'

          00000000 03F2 MBZ '....'

          801F 03F6 bitmap page for logical area 31
      00000000000000000000 03F8 MBZ '.....'
```

Example 2

This example displays the number of the current storage area.

```
RdbALTER> DISPLAY STORAGE_AREA
      0001 0000 area 1
```

Example 3

This example displays the number of area inventory page entries on area 1, page 2 of the MF_PERSONNEL database, and the value of the highest snapshot-enabled TSN of area inventory page entry 14. This example also displays the number of the next area inventory page in the current storage area.

```
RdbALTER> AREA 1 PAGE 2
RdbALTER> DISPLAY AIP_ENTRIES
      0010 0022 16 logical area entries

RdbALTER> DISPLAY ENTRY 14 TSN
          entry #14
          00000001 0386 snaps enabled TSN 1

RdbALTER> DISPLAY NEXT_AIP
      00000003 0016 next area inventory page 3
```

DISPLAY Command

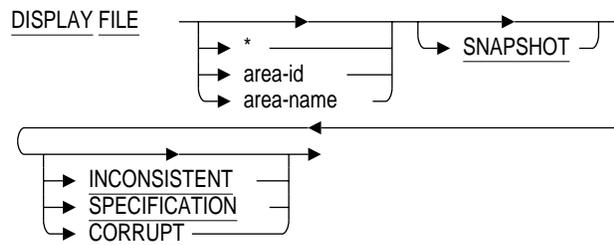
See the *VAX Rdb/VMS Guide to Database Maintenance and Performance* for more examples of how to use the DISPLAY command.

7.9 DISPLAY FILE Command

Displays the file specification in the root file for an area file or snapshot file. You can also use this command to display the current setting of the inconsistent flag or corrupt flag for an area file.

If you display a field that you have changed during the current RdbALTER session, the field is displayed as marked.

Format



Command Parameters

*** (asterisk)**

Displays all file characteristics.

area-id

Specifies the number of the area for which you want to display information from the root file.

area-name

Specifies the area name of the area for which you want to display information from the root file.

SNAPSHOT

Displays information about a snapshot file. If you select the SNAPSHOT parameter, you can specify the SPECIFICATION parameter; the INCONSISTENT and CORRUPT parameters are not valid options for snapshot files.

DISPLAY FILE Command

INCONSISTENT

Displays the current setting of the inconsistent flag. This parameter applies only to area files.

SPECIFICATION

Displays the full file specification (including version number) for the data storage or snapshot file of the area.

CORRUPT

Displays the current setting of the corrupt flag. This applies only to area files.

Examples

Example 1

The following example displays the file specification for the JOBS area file.

```
RdbALTER> DISPLAY FILE JOBS
Area JOBS:
    File specification is: "DISK1:[RICK.RDB]JOBS.RDA;1"
    Corrupt flag is: OFF
    Inconsistent flag is: OFF
```

DISPLAY ROOT Command

7.10 DISPLAY ROOT Command

Displays the current root file specification of the database. You can use this statement to be sure that you have assigned the file specification that you intended.

Format

DISPLAY ROOT →

Examples

Example 1

The following command displays the current root file specification of the database.

```
RdbALTER> DISPLAY ROOT
      Root file specification is: "DISK1:[RICK.RDB]MF_PERSONNEL.RDB;1"
```

EXIT Command

7.11 EXIT Command

The EXIT command terminates the RdbALTER session and returns you to DCL command level. You can also press CTRL/Z to end an RdbALTER session.

If EXIT is issued and altered but uncommitted pages exist, you are told to issue either a COMMIT or a ROLLBACK command. RdbALTER does not exit until a COMMIT or ROLLBACK operation has accounted for all altered pages.

The EXIT command performs an implicit NOLOG command if a LOG file is open.

Format

EXIT

Examples

Example 1

This example exits RdbALTER command level and returns you to DCL.

```
RdbALTER> EXIT
$
```

Example 2

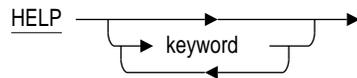
The following example shows that you cannot exit from the RdbALTER session if there are uncommitted changes in your database:

```
RdbALTER> DEPOSIT ROOT SPECIFICATION=DISK1:[RICK]MF_PERSONNEL.RDB;1
(marked) Root file specification is: "DISK1:[RICK]MF_PERSONNEL.RDB;1"
RdbALTER> EXIT
%RMU-F-COMMITROLL, currently modified ROOT fields must be committed or
rolled back
```

7.12 HELP Command

The HELP command provides information about RdbALTER commands, terminology, and concepts. If you type HELP at the RdbALTER prompt without specifying a topic, RdbALTER displays a list of topics on which help is available. If you type HELP followed by a topic, you get a brief description of that topic.

Format



Command Parameters

keyword

Identifies the help topic or subtopic you want explained.

Examples

Example 1

The following example requests two layers of help. First, the user types HELP without specifying a topic, and a list of topics for which help is available is displayed. Then the user selects the VERIFY topic to obtain information for the VERIFY command.

```
RdbALTER> HELP

Information available:

AREA-PAGE  ATTACH  COMMIT  DEPOSIT  DETACH  DISPLAY  ERRORS
EXIT       HELP    LOG     MAKE_CONSISTENT  MOVE    NOLOG
PAGE      RADIX  ROLLBACK  UNCORRUPT  VERIFY

Topic? VERIFY
```

LOG Command

7.13 LOG Command

Keeps an audit trail of all or part of an RdbALTER session. After you specify the LOG command, RdbALTER commands and their results are logged in the specified log file until you close the log file by entering a NOLOG command, EXIT command, or another LOG command.

Format

LOG file-spec →

Command Parameters

file-spec

Specifies a file to contain the audit trail log. The default file type is LIS.

Examples

Example 1

The following command creates the file AUDIT.LIS and begins audit trail logging.

```
RdbALTER> LOG AUDIT
```

Example 2

The following command creates the file AUDIT.TRL and begins audit trail logging.

```
RdbALTER> LOG AUDIT.TRL
```

MAKE_CONSISTENT Command

7.14 MAKE_CONSISTENT Command

Resets an area's inconsistent indication flag, thus allowing you to use the database.

When an area is restored from backup on a by-area basis, it does not reflect data that has been updated since the backup. The transaction level of the restored area reflects the transaction level of the backup file, not the transaction level of the database. Therefore, the transaction level of the restored area differs from that of the database. Rdb/VMS marks the area by setting a flag in the area file to inconsistent.

You can perform a recovery by area to upgrade the transaction level of the restored area to that of the database. (After-image journaling must be enabled in order to restore by area.) If you are certain that no updates have been made to the database since the backup, you can use the MAKE_CONSISTENT command in RdbALTER to change the setting of the flag from inconsistent to consistent.

Format



Parameters

storage-area-name

Specifies an area of the database by storage area name, which is the name given by the AREA clause in the schema.

storage-area-number

Specifies an area of the database by storage area number, which is assigned when the database is created and is given on the first line of a page display.

MAKE_CONSISTENT Command

Examples

Example 1

The following example resets the indication flag from inconsistent to consistent for the area JOBS:

```
RdbALTER> MAKE_CONSISTENT JOBS  
  
***** WARNING! *****  
  
BEWARE ATTEMPTING TO MAKE CONSISTENT A STORAGE  
AREA WITHOUT FIRST VERIFYING IT USING THE  
RMU/VERIFY COMMAND.  
  
AN RdbALTER ROLLBACK COMMAND WILL LEAVE THIS  
AREA MARKED INCONSISTENT.  
  
Area JOBS now marked consistent.
```

7.15 MOVE Command

Moves data (defined by beginning and ending offset addresses) from one page location to another location on the same page.

The number of bytes moved is:

$$(\text{old-offset-end}) - (\text{old-offset-start}) + 1$$

The sending field, defined by the old-offset arguments, remains unchanged.

The receiving field, defined by the new-offset argument and the length of the sending field, is replaced by the contents of the sending field.

Other information in the page is unchanged.

Format

MOVE old-offset-start:old-offset-end new-offset →

Command Parameters

old-offset-start

Specifies the hexadecimal offset address of the first byte in the data sequence to be moved.

old-offset-end

Specifies the hexadecimal offset address of the last byte in the data sequence to be moved.

new-offset

Specifies the hexadecimal offset address of the first byte in the sequence of bytes receiving the moved data.

MOVE Command

Examples

Example 1

This example moves data from offset location 34A through 34E to the starting offset location of 354.

```
RdbALTER> MOVE 34A:34E 354
```

See the *VAX Rdb/VMS Guide to Database Maintenance and Performance* for more examples of how to use the MOVE command.

7.16 NOLOG Command

Stops RdbALTER logging. The NOLOG command stops audit trail logging if a previous LOG command is still in effect. The EXIT command performs an implicit NOLOG if LOG is still active; thus, you need not enter a NOLOG command before exiting the RdbALTER session.

Format

NOLOG →

Examples

Example 1

The following example stops audit trail logging:

```
RdbALTER> NOLOG
```

PAGE Command

7.17 PAGE Command

Fetches a page from the current area. If you specify a valid page number, that page is fetched from the current area. If you enter the PAGE command without a page number, the next page in the current area is fetched. If you enter the PAGE command without a page number and you are already at the highest numbered page of the area, page 1 of the area is fetched.

Format

PAGE → *page-number* →

Command Parameters

page-number

Identifies a page to be fetched in the current area. If you do not specify the *page-number* parameter, the next page in the current area is fetched. If you do not specify a page number and you are at the highest numbered page in the area, page 1 of the current area is fetched.

Example

Example 1

The following example fetches page 14 of the current area:

```
RdbALTER> PAGE 14
```

Example 2

If the current page is page 15 of the JOBS area, the following command fetches page 16 of the JOBS area:

```
RdbALTER> PAGE
```

7.18 RADIX Command

Sets the default radix for entering numeric data. This command does not change the radix for specifying page offsets. Page offsets must always be specified in hexadecimal radix.

Format



Parameters

DECIMAL

Sets the default radix in decimal numbers.

HEXADECIMAL

Sets the default radix in hexadecimal numbers.

Examples

Example 1

The following example sets the default radix for entering numeric data to hexadecimal.

```
RdbALTER> RADIX HEXADECIMAL
```

ROLLBACK Command

7.19 ROLLBACK Command

Ignores all page changes since the last COMMIT or ROLLBACK command. Altered pages are stored in virtual memory until you issue a COMMIT or ROLLBACK command. ROLLBACK tells RdbALTER to ignore all changes since the last COMMIT or ROLLBACK command was issued.

Do not follow a ROLLBACK command with a DEPOSIT command without first respecifying your location with a DISPLAY or PAGE command.

RdbALTER does not allow you to issue an EXIT command until all altered pages are either committed or rolled back.

Format

ROLLBACK →

Examples

Example 1

The following example rolls back all page changes entered since the last COMMIT or ROLLBACK was issued.

```
RdbALTER> ROLLBACK
```

7.20 UNCORRUPT Command

Resets the area's corruption indication flag (FILID CORRUPT_FLG), thus allowing you to use the uncorrupted sections of a corrupted storage area. Storage areas are most often corrupted by attempting a VAX Rdb/VMS (not RdbALTER) rollback with one or more storage areas opened in batch-update transaction mode.

The UNCORRUPT command allows you to access a database that is in an uncertain condition. Accordingly, the following message is displayed when you enter it:

```
BEWARE ATTEMPTING TO UNCORRUPT A STORAGE AREA  
WITHOUT FIRST VERIFYING IT.
```

Format

UNCORRUPT → storage-area-name →
 → storage-area-number →

Command Parameters

storage-area-name

Specifies an area of the current database by storage area name, which is the name given by the AREA clause in the schema.

storage-area-number

Specifies an area of the current database by the storage area number, which is assigned when the database is created and is given on the first line of a page display.

Examples

Example 1

The following example resets the EMPIDS_LOW area's corruption indication flag.

UNCORRUPT Command

```
RdbALTER> UNCORRUPT EMPIDS_LOW  
        ***** WARNING! *****  
        BEWARE ATTEMPTING TO UNCORRUPT A STORAGE AREA  
        WITHOUT FIRST VERIFYING IT USING THE RMU/VERIFY  
        COMMAND.  
        AN RdbALTER ROLLBACK COMMAND WILL LEAVE THIS  
        AREA MARKED CORRUPT.  
Area EMPIDS_LOW now marked uncorrupt.
```

7.21 VERIFY Command

Verifies the current page statistically. When the VERIFY command is issued, the page header and page checksum are verified for the current page, and error messages are issued if header or checksum corruption is found.

Format

VERIFY →

Examples

Example 1

The following example verifies area 3 page 100.

```
RdbALTER> AREA 3 PAGE 100  
RdbALTER> VERIFY
```

Rdb/VMS System Relations

This chapter describes the Rdb/VMS system relations. Rdb/VMS stores information about the database as a set of relations called system relations. The system relations are the definitive source of Rdb/VMS metadata. Metadata defines the structure of the database; for example, metadata defines the fields that comprise a particular relation and the fields that can index that relation.

Although you can store your data definitions in the data dictionary, the database system (Database Control System, or DBCS) refers only to the system relations in the database file itself for these definitions. In a sense, the system relations are an internal data dictionary for the database. This method improves performance as the DBCS does not have to access the data dictionary at run time.

When you create a database, Rdb/VMS defines, populates, and manipulates the system relations. As the user performs data definition operations on the database, Rdb/VMS reads and modifies the system relations to reflect those operations. You *should not* modify any of the Rdb/VMS system relations using data manipulation language, nor should you define any fields based on system relation fields. However, you can use regular Rdb/VMS data manipulation statements to retrieve the contents of the system relations. This means that your program can determine the structure and characteristics of the database by retrieving the fields of the system relations.

The following BASIC program queries the system relations and determines whether a field can be updated.

```
3010 OPTION TYPE = EXPLICIT, SIZE = INTEGER LONG
      DECLARE STRING Field_name, &
                Rel_name, &
                More
      DECLARE INTEGER Update_yes
      DECLARE INTEGER CONSTANT Trim_blanks = 128%
```

```

&RDB& INVOKE DATABASE FILENAME 'disk1:[test]personnel'
3020 INPUT 'Name of relation'; Rel_name
      PRINT 'Starting query'
      PRINT 'In '; Rel_name; ' relation, fields:'

&RDB& FOR RF IN RDB$RELATION_FIELDS WITH
&RDB&     RF.RDB$RELATION_NAME = Rel_name
&RDB&     GET
&RDB&         Field_name = RF.RDB$FIELD_NAME;
&RDB&         Update_yes = RF.RDB$UPDATE_FLAG
&RDB&     END_GET

      IF (Update_yes = 1)
      THEN
        PRINT ' '; EDIT$(Field_name, Trim_blanks); ' can be updated'
      ELSE
        PRINT ' '; EDIT$(Field_name, Trim_blanks); ' cannot be updated'
      END IF

&RDB& END_FOR
3030 INPUT 'Another relation (Y/N)?'; More
      IF (More = 'Y')
      THEN
        GO TO 3020
      ELSE IF (More = 'N')
      THEN
        GO TO 3040
      ELSE
        PRINT 'Please enter Y or N'
        GO TO 3030
      END IF
3040 END

```

When you use the `START_TRANSACTION . . . RESERVING` statement to lock a set of relations for an Rdb/VMS operation, you normally exclude from the transaction all the relations not listed in the `RESERVING` clause. However, Rdb/VMS accesses and updates system relations as necessary, no matter which relations you have locked in the `START_TRANSACTION` statement.

When your transaction updates database metadata, Rdb/VMS reserves the system relations involved in the update in the `EXCLUSIVE` share mode. Other users are unable to perform data definition operations on these relations until you complete your transaction. For example:

- When you refer to a global field in an update transaction that changes data definitions, Rdb/VMS locks an index for the system relation, `RDB$RELATION_FIELDS`. No other users can refer to the same global field until you commit your transaction.
- When you change a relation or global field definition, Rdb/VMS locks an index in the system relation, `RDB$FIELD_VERSIONS`. No other users can change relation or global field definitions until you commit your transaction.

- When you change a relation definition, Rdb/VMS locks an index in the system relation, RDB\$RELATION_FIELDS. No other users can change relations in the same index node until you commit your transaction.

In this chapter, the fields that comprise each system relation are shown in a table. Each row includes the field's name, data type, and a summary of what information the field describes.

The data types have the following values:

text (n)	TEXT SIZE (n bytes)
seg-str	SEGMENTED STRING SUB_TYPE 0
word	SIGNED WORD SCALE 0
var-str (n)	VARYING STRING (n bytes)

The definitions of most system relations are standard, and are likely to remain stable in future versions. Furthermore, these definitions are standard for other Digital relational database products, such as VAX Rdb/ELN and VIDA.

However, some system relations and fields are specific to Rdb/VMS. Prior to each table in this chapter, a list identifies the fields that are Rdb/VMS extensions to the standard definitions. Definitions of these extensions are not guaranteed to remain stable in future versions or in other database products.

The Rdb/VMS system relations are as follows:

RDB\$CONSTRAINTS	Name and definition of each constraint.
RDB\$CONSTRAINT_RELATIONS	Name of each relation that participates in a given constraint.
RDB\$DATABASE	A pointer to the database file.
RDB\$FIELD_VERSIONS	One record for each version of each field definition in the database.
RDB\$FIELDS	Characteristics of each field in the database.
RDB\$INDEX_SEGMENTS	Fields that make up keys for relations.
RDB\$INDICES	Characteristics of the indexes for each relation.
RDB\$RELATION_FIELDS	Fields defined for each relation.
RDB\$RELATIONS	Relations in the database.

RDB\$VIEW_RELATIONS	Interdependencies of relations used in views.
RDBVM\$COLLATIONS	The collation sequence to be used for the database.
RDBVM\$INTERRELATIONS	Interdependencies of entities used in the database.
RDBVM\$PRIVILEGES	Protection for the database, relations, views, and fields.
RDBVM\$RELATION_CONSTRAINTS	Lists all relation-specific constraints.
RDBVM\$RELATION_CONSTRAINT_FLDS	Lists the fields that participate in unique, primary, or foreign key declarations for relation-specific constraints.
RDBVM\$STORAGE_MAPS	Characteristics of each storage map.
RDBVM\$STORAGE_MAP_AREAS	Characteristics of each storage area referred to by a storage map.
RDBVM\$TRIGGERS	Definition of a trigger.

Note In the following tables, BLR refers to binary language representation. This is low-level syntax used internally to represent Rdb/VMS data manipulation operations.

8.1 RDB\$CONSTRAINTS System Relation

The RDB\$CONSTRAINTS system relation contains the name and definition of each constraint. Table 8-1 provides information on the fields of the RDB\$CONSTRAINTS system relation.

Table 8-1 The RDB\$CONSTRAINTS System Relation

Field Name	Data Type	Summary Description
RDB\$CONSTRAINT_NAME	text 31	The systemwide unique name of the constraint.
RDB\$CONSTRAINT_BLR	seg-str	Low-level syntax that defines the constraint.
RDB\$CONSTRAINT_SOURCE	seg-str	The user's source for the constraint.

(continued on next page)

Table 8-1 (Cont.) The RDB\$CONSTRAINTS System Relation

Field Name	Data Type	Summary Description
RDB\$DESCRIPTION	seg-str	A user-supplied description of the contents of this record.
RDB\$EVALUATION_TIME	word	One of the following values: 0 Default (implementation-specific) 1 Commit time 2 Verb time
RDB\$EXTENSION_PARAMETERS	seg-str	Reserved for future use.

8.2 RDB\$CONSTRAINT_RELATIONS System Relation

The RDB\$CONSTRAINT_RELATIONS system relation lists all relations that participate in a given constraint. Table 8-2 provides information on the fields of the RDB\$CONSTRAINT_RELATIONS system relation.

Table 8-2 The RDB\$CONSTRAINT_RELATIONS System Relation

Field Name	Data Type	Summary Description
RDB\$CONSTRAINT_NAME	text 31	The systemwide unique name of the constraint.
RDB\$RELATION_NAME	text 31	The name of a relation involved in the constraint.

(continued on next page)

Table 8–2 (Cont.) The RDB\$CONSTRAINT_RELATIONS System Relation

Field Name	Data Type	Summary Description
RDB\$FLAGS	word	<p>The following flag bits are set:</p> <ul style="list-style-type: none"> ■ Bit offset 0 Reserved for future use. ■ Bit offset 1 If the constraint evaluates at update. ■ Bit offset 2 If the constraint evaluates at delete. ■ Bit offset 3 If the constraint evaluates with optimization by dbkey lookup. ■ Bit offset 4 If the constraint checks for existence. ■ Bit offset 5 If the constraint checks for uniqueness.
RDB\$CONSTRAINT_CONTEXT	word	The context variable of the relation involved in the constraint.

8.3 RDB\$DATABASE System Relation

The RDB\$DATABASE system relation contains a pointer to the database file. This relation can contain only one record. Table 8–3 provides information on the fields of the RDB\$DATABASE system relation. In the RDB\$DATABASE relation, all fields *except* the following are Rdb/VMS extensions:

- RDB\$DESCRIPTION
- RDB\$DATABASE_PARAMETERS
- RDB\$EXTENSION_PARAMETERS

RDB\$DATABASE_PARAMETERS and RDB\$EXTENSION_PARAMETERS are standard fields.

Table 8-3 The RDB\$DATABASE System Relation

Field Name	Data Type	Summary Description
RDB\$CDD_PATH	text 256	The dictionary path name for the database.
RDB\$FILE_NAME	text 255	The database file name.
RDB\$MAJ_VER	longword	Derived from the database major version.
RDB\$MIN_VER	longword	Derived from the database minor version.
RDB\$MAX_RELATION_ID	word	The largest relation identifier assigned. Rdb/VMS assigns the next relation an ID of MAX_RELATION_ID + 1.
RDB\$RELATION_ID	word	The unique identifier of the RDB\$RELATIONS relation. If you delete a relation, that identifier is not assigned to any other relation.
RDB\$RELATION_ID_ROOT_DBK	text 8	A pointer (database key, or dbkey) to the base of the RDB\$REL_REL_ID_NDX index on field RDB\$RELATION_ID.
RDB\$RELATION_NAME_ROOT_DBK	text 8	A pointer (dbkey) to the base of the RDB\$REL_REL_NAME_NDX index on field RDB\$RELATION_NAME.
RDB\$FIELD_ID	word	The identifier of the RDB\$FIELD_VERSIONS relation.
RDB\$FIELD_REL_FLD_ROOT_DBK	text 8	A pointer (dbkey) to the base of the RDB\$VER_REL_ID_VER_NDX index on fields RDB\$RELATION_ID and RDB\$VERSION.

(continued on next page)

Table 8-3 (Cont.) The RDB\$DATABASE System Relation

Field Name	Data Type	Summary Description
RDB\$INDEX_ID	word	The identifier of the RDB\$INDICES relation.
RDB\$INDEX_NDX_ROOT_DBK	text 8	A pointer (dbkey) to the base of the RDB\$NDX_NDX_NAME_NDX index on field RDB\$INDEX_NAME.
RDB\$INDEX_REL_ROOT_DBK	text 8	A pointer (dbkey) to the base of the RDB\$NDX_REL_NAM_NDX index on field RDB\$RELATION_ID.
RDB\$INDEX_SEG_ID	word	The identifier of the RDB\$INDEX_SEGMENTS relation.
RDB\$INDEX_SEG_FLD_ROOT_DBK	text 8	A pointer (dbkey) to the base of the RDB\$NDX_SEG_NAM_FLD_POS_NDX index on fields RDB\$INDEX_NAME and RDB\$FIELD_POSITION.
RDB\$SEGMENTED_STRING_ID	word	The logical area ID that contains the segmented strings.
RDB\$ACCESS_CONTROL	seg-str	The access control policy for the database.
RDB\$DESCRIPTION	seg-str	A user-supplied description of the contents of this record.

(continued on next page)

Table 8–3 (Cont.) The RDB\$DATABASE System Relation

Field Name	Data Type	Summary Description
RDB\$DATABASE_PARAMETERS	seg-str	Reserved for future use.
RDB\$EXTENSION_PARAMETERS	seg-str	Reserved for future use.
RDB\$FLAGS	word	The following flag bit offsets are set for the following conditions: 0 dictionary required 1 ANSI protection used 2 database file is a CDD\$DATABASE
RDBVMS\$SECURITY_AUDIT	longword	A bitmask that indicates the privileges that will be audited for the database, as specified in the RMU/SET AUDIT command.
RDBVMS\$SECURITY_ALARM	longword	A bitmask that indicates the privileges that will produce alarms for the database, as specified in the RMU/SET AUDIT command.
RDBVMS\$SECURITY_USERS	seg-str	An access control list that identifies users who will be audited or who will produce alarms for DACCESS events when DACCESS auditing is enabled for specific tables, columns, or the schema.

8.4 RDB\$FIELD_VERSIONS System Relation

The RDB\$FIELD_VERSIONS system relation is an Rdb/VMS extension. This relation contains one record for each version of each field definition in the database. This relation collects the information required to build relation definitions for any given versions of a relation. Table 8–4 provides information on the fields of the RDB\$FIELD_VERSIONS system relation.

Table 8-4 The RDB\$FIELD_VERSIONS System Relation

Field Name	Data Type	Summary Description
RDB\$RELATION_ID	word	The identifier for a relation within the database.
RDB\$FIELD_ID	word	An identifier used internally to name the field represented by this record.
RDB\$FIELD_NAME	text	The name of the field.
RDB\$VERSION	word	The version number for the relation definition to which this field belongs.
RDB\$FIELD_TYPE	word	The data type of the field represented by this record. This data type must be interpreted according to the rules for interpreting the DSC\$B_DTYPE field of class S descriptors (as defined in the VAX Procedure Calling and Condition Handling Standard). Segmented strings require a unique field type identifier. This identifier is currently 261.
RDB\$FIELD_LENGTH	word	The length of the field represented by this record. This length must be interpreted according to the rules for interpreting the DSC\$W_LENGTH field within class S and SD descriptors (as defined in the VAX Procedure Calling and Condition Handling Standard).
RDB\$OFFSET	word	The byte offset of the field from the beginning of the record.
RDB\$FIELD_SCALE	word	For numeric data types, the scale factor to be applied when interpreting the contents of the field represented by this record. This scale factor must be interpreted according to the rules for interpreting the DSC\$B_SCALE field of class SD descriptors (as defined in the VAX Procedure Calling and Condition Handling Standard). For nonnumeric data types, RDB\$FIELD_SCALE is 0.

(continued on next page)

Table 8-4 (Cont.) The RDB\$FIELD_VERSIONS System Relation

Field Name	Data Type	Summary Description
RDB\$FLAGS	word	Reserved for future use.
RDB\$VALIDATION_BLR	seg-str	The BLR that represents the VALID_IF clause defined in this version of the field.
RDB\$COMPUTED_BLR	seg-str	The BLR that represents the COMPUTED BY clause defined in this version of the field.
RDB\$MISSING_VALUE	seg-str	The BLR that represents the MISSING_VALUE clause defined in this version of the field.
RDB\$SEGMENT_LENGTH	word	The length of a segmented string segment as defined in the DEFINE FIELD statement. This field is informational only.
RDBVM\$COLLATION_NAME	text	The name of the collating sequence for the field.
RDB\$ACCESS_CONTROL	seg-str	The access control list for the field.
RDB\$DEFAULT_VALUE2	seg-str	The BLR for SQL default value.
RDBVM\$SECURITY_AUDIT	longword	Bit mask.
RDBVM\$SECURITY_ALARM	longword	Bit mask.

8.5 RDB\$FIELDS System Relation

The RDB\$FIELDS system relation describes the global (generic) characteristics of each field in the database. There is one record for each global field in the database. RDB\$EXTENSION_PARAMETERS field is a standard field. Table 8-5 provides information on the fields of the RDB\$FIELDS system relation.

Table 8-5 The RDB\$FIELDS System Relation

Field Name	Data Type	Summary Description
RDB\$FIELD_NAME	text	The name of the field represented by this record. Each record within RDB\$FIELDS must have a unique RDB\$FIELD_NAME field.
RDB\$FIELD_TYPE	word	The data type of the field represented by this record. This data type must be interpreted according to the rules for interpreting the DSC\$B_DTYPE field of class S descriptors (as defined in the VAX Procedure Calling and Condition Handling Standard). Segmented strings require a unique field type identifier. This identifier is 261.
RDB\$FIELD_LENGTH	word	The length of the field represented by this record. This length must be interpreted according to the rules for interpreting the DSC\$W_LENGTH field within class S and SD descriptors (as defined in the VAX Procedure Calling and Condition Handling Standard).

(continued on next page)

Table 8-5 (Cont.) The RDB\$FIELDS System Relation

Field Name	Data Type	Summary Description
RDB\$FIELD_SCALE	word	For numeric data types, the scale factor to be applied when interpreting the contents of the field represented by this record. This scale factor must be interpreted according to the rules for interpreting the DSC\$B_SCALE field of class SD descriptors (as defined in the VAX Procedure Calling and Condition Handling Standard). For nonnumeric data types, this field is always zero.
RDB\$SYSTEM_FLAG	word	A flag that indicates whether the field is a system field or a user field. The values for this field are: 0 User field 1 System relation field
RDB\$VALIDATION_BLR	seg-str	The BLR that represents the validation expression to be checked each time the field's contents are modified.
RDB\$COMPUTED_BLR	seg-str	The BLR that represents the expression used to calculate a value for this field at execution time.
RDB\$EDIT_STRING	var-str 255	The edit string used by VAX DATATRIEVE when printing the field.
RDB\$MISSING_VALUE	seg-str	The value used when the field's missing value is retrieved or displayed. RDB\$MISSING_VALUE does not store any value in a field; instead, it flags the field to indicate that the value is missing.

(continued on next page)

Table 8-5 (Cont.) The RDB\$FIELDS System Relation

Field Name	Data Type	Summary Description
RDB\$FIELD_SUB_TYPE	word	The data subtype of a field whose data type is a segmented string, as follows: 0 Unspecified 1 Text 2 BLR
RDB\$DESCRIPTION	seg-str	A user-supplied description of the contents of this record.
RDB\$VALIDATION_SOURCE	seg-str	The user's source text for the validation criteria.
RDB\$COMPUTED_SOURCE	seg-str	The user's source for the BLR used to calculate a value for this field at execution time.
RDB\$QUERY_NAME	text 31	The query name of this field for use by VAX DATATRIEVE. DATATRIEVE field attributes in RDB\$RELATION_FIELDS take precedence over attributes in RDB\$FIELDS. If the attribute value is missing in RDB\$RELATION_FIELDS, DATATRIEVE uses the value from RDB\$FIELDS.
RDB\$QUERY_HEADER	seg-str	The query header of this field for use by VAX DATATRIEVE. DATATRIEVE field attributes in RDB\$RELATION_FIELDS take precedence over attributes in RDB\$FIELDS. If the attribute value is missing in RDB\$RELATION_FIELDS, DATATRIEVE uses the value from RDB\$FIELDS.

(continued on next page)

Table 8–5 (Cont.) The RDB\$FIELDS System Relation

Field Name	Data Type	Summary Description
RDB\$DEFAULT_VALUE	seg-str	The default value used by VAX DATATRIEVE when no value is specified for a field during a STORE statement. It differs from RDB\$MISSING_VALUE in that it holds an actual field value. DATATRIEVE field attributes in RDB\$RELATION_FIELDS take precedence over attributes in RDB\$FIELDS. If the attribute value is missing in RDB\$RELATION_FIELDS, DATATRIEVE uses the value from RDB\$FIELDS.
RDB\$SEGMENT_LENGTH	word	The length of a segmented string segment as defined in the DEFINE FIELD statement. This field is informational only.
RDB\$EXTENSION_PARAMETERS	seg-str	Reserved for future use.
RDB\$CDD_NAME	seg-str	The fully qualified name of the dictionary entity upon which the field definition is based, as specified in the FROM PATHNAME clause.
RDBVMS\$COLLATION_NAME	text	The name of the collating sequence for the field.
RDB\$DEFAULT_VALUE2	seg-str	The BLR for SQL default value.

8.6 RDB\$INDEX_SEGMENTS System Relation

The RDB\$INDEX_SEGMENTS system relation describes the fields that make up an index's key. Each index must have at least one field within the key. Table 8–6 provides information on the fields of the RDB\$INDEX_SEGMENTS system relation.

Table 8-6 The RDB\$INDEX_SEGMENTS System Relation

Field Name	Data Type	Summary Description
RDB\$INDEX_NAME	text	The name of the index of which this record is a segment.
RDB\$FIELD_NAME	text	The name of a field that participates in the index key. This field name matches the name in the FIELD_NAME field of the RDB\$RELATION_FIELDS relation.
RDB\$FIELD_POSITION	word	The ordinal position of this key segment within the total index key. No two segments in the key may have the same RDB\$FIELD_POSITION.
RDB\$FLAGS	word	The following flag bits are set: 0 For ascending segments 1 For descending segments
RDB\$FIELD_LENGTH	word	Shortened length of text for compressed indexes.
RDBVMSS\$FIELD_MAPPING_LOW	longword	Shows the lower limit of the mapping range.
RDBVMSS\$FIELD_MAPPING_HIGH	longword	Shows the higher limit of the mapping range.

8.7 RDB\$INDICES System Relation

The RDB\$INDICES system relation contains information about indexes in the database. The RDB\$EXTENSION_PARAMETERS field is a standard field. Table 8-7 provides information on the fields of the RDB\$INDICES system relation.

Table 8-7 The RDB\$INDICES System Relation

Field Name	Data Type	Summary Description
RDB\$INDEX_NAME	text	A unique index name.
RDB\$RELATION_NAME	text	The name of the relation in which the index is used.

(continued on next page)

Table 8-7 (Cont.) The RDB\$INDICES System Relation

Field Name	Data Type	Summary Description
RDB\$UNIQUE_FLAG	word	A flag that determines whether duplicate values can be entered in the indexed field, as follows: 0 Duplicate values allowed 1 No duplicate values
RDB\$ROOT_DBK	text	A pointer to the base of the index.
RDB\$INDEX_ID	word	The identifier of the index.
RDB\$FLAGS	word	The following flag bit offsets indicate the type of index: 0 hashed index 1 compression-enabled index
RDB\$SEGMENT_COUNT	word	The number of segments in the key.
RDB\$DESCRIPTION	seg-str	A user-supplied description of the contents of this record.
RDB\$EXTENSION_PARAMETERS	seg-str	Stores NODE SIZE value and PERCENT FILL value for this index. Also reserved for other future use.
RDB\$CARDINALITY	word	The number of unique entries for a nonunique index.

8.8 RDB\$RELATION_FIELDS System Relation

The RDB\$RELATION_FIELDS system relation contains one record for each field in each relation. The RDB\$EXTENSION_PARAMETERS field is a standard field. Table 8-8 provides information on the fields of the RDB\$RELATION_FIELDS system relation.

Table 8-8 The RDB\$RELATION_FIELDS System Relation

Field Name	Data Type	Summary Description
RDB\$RELATION_NAME	text	The name of the relation that contains the field represented by this record.
RDB\$FIELD_NAME	text	The name of the field represented by this record within the relation. Each RDB\$RELATION_FIELDS record that has the same RDB\$RELATION_NAME must have a unique RDB\$FIELD_NAME.
RDB\$FIELD_SOURCE	text	The name of the generic field (from the RDB\$FIELD_NAME field within the RDB\$FIELDS relation) that supplies the definition for this field.
RDB\$FIELD_ID	word	An identifier that can be used within the BLR to name the field represented by this record. Rdb/VMS assigns each field an ID that is permanent for as long as the field exists within the relation.
RDB\$FIELD_POSITION	word	The ordinal position of the field represented by this record, relative to the other fields in the same relation. This is of primary interest to VAX DATATRIEVE.
RDB\$QUERY_NAME	text	The query name of this field for use by VAX DATATRIEVE.
RDB\$UPDATE_FLAG	word	A flag that indicates whether the field may be updated, as follows: 1 May be updated 0 May not be updated (continued on next page)

Table 8-8 (Cont.) The RDB\$RELATION_FIELDS System Relation

Field Name	Data Type	Summary Description
RDB\$QUERY_HEADER	seg-str	The query header of this field for use by VAX DATATRIEVE. DATATRIEVE field attributes in RDB\$RELATION_FIELDS take precedence over RDB\$FIELDS. If the attribute value is missing in RDB\$RELATION_FIELDS, DATATRIEVE uses the value from RDB\$FIELDS.
RDB\$DESCRIPTION	seg-str	A user-supplied description of the contents of this record.
RDB\$VIEW_CONTEXT	word	For view relations, this field identifies the context variable used to qualify the view field. This context variable must be defined within the record selection expression that defines the view. The context variable appears in the BLR represented by the field RDB\$VIEW_BLR in RDB\$RELATIONS.
RDB\$BASE_FIELD	text	The local name of the field used as a component of a view. The name is qualified by the context variable identified in RDB\$VIEW_CONTEXT.

(continued on next page)

Table 8–8 (Cont.) The RDB\$RELATION_FIELDS System Relation

Field Name	Data Type	Summary Description
RDB\$DEFAULT_VALUE	seg-str	The default value used by VAX DATATRIEVE when no value is specified for a field during a STORE statement. It differs from RDB\$MISSING_VALUE in that it holds an actual field value. DATATRIEVE field attributes in RDB\$RELATION_FIELDS take precedence over attributes in RDB\$FIELDS. If the attribute value is missing in RDB\$RELATION_FIELDS, DATATRIEVE uses the value from RDB\$FIELDS.
RDB\$EDIT_STRING	var-str 255	The edit string to be used by VAX DATATRIEVE when printing the field.
RDB\$EXTENSION_PARAMETERS	seg-str	Reserved for future use.
RDB\$ACCESS_CONTROL	seg-str	The access control list for the field.
RDB\$DEFAULT_VALUE2	seg-str	The BLR for SQL default value.
RDBVMS\$SECURITY_AUDIT	longword	Bit mask.
RDBVMS\$SECURITY_ALARM	longword	Bit mask.

8.9 RDB\$RELATIONS System Relation

Names all the relations and views within the database. There is one record for each relation or view. The RDB\$CARDINALITY field is an Rdb/VMS extension. The RDB\$EXTENSION_PARAMETERS field is a standard field. Table 8–9 provides information on the fields of the RDB\$RELATIONS system relation.

Table 8-9 The RDB\$RELATIONS System Relation

Field Name	Data Type	Summary Description
RDB\$RELATION_NAME	text	The name of a relation within the database. Each record within RDB\$RELATIONS must have a unique RDB\$RELATION_NAME.
RDB\$RELATION_ID	word	An identification number used within the BLR to identify a relation.
RDB\$STORAGE_ID	word	A pointer to the database logical area where the data for this relation is stored.
RDB\$SYSTEM_FLAG	word	A flag that indicates whether the relation is a system relation or a user relation. The values for this field are: 0 User relation 1 System relation
RDB\$DBKEY_LENGTH	word	The length in bytes of the database key. A database key for a record in a relation is 8 bytes, and "n times 8" for a view record, where "n" is the number of relations referred to in the view.
RDB\$MAX_VERSION	word	The number of the current version of the relation definition. This value is matched with the RDB\$VERSION field in RDB\$FIELD_VERSIONS to determine the current record format for the relation.
RDB\$CARDINALITY	longword	The number of records in the relation (cardinality).

(continued on next page)

Table 8-9 (Cont.) The RDB\$RELATIONS System Relation

Field Name	Data Type	Summary Description
RDB\$FLAGS	word	<p>The following flag bits are set:</p> <ul style="list-style-type: none"> ■ Bit offset 0 If this relation is a view. ■ Bit offset 1 If compression is disabled. When compression is disabled, the relation is not compressed. ■ Bit offset 2 Indicates that SQL's WITH CHECK OPTION clause was used in the view definition.
RDB\$VIEW_BLR	seg-str	The BLR that describes the record selection expression used to select the records for the view. If the relation is not a view, RDB\$VIEW_BLR is missing.
RDB\$DESCRIPTION	seg-str	A user-supplied description of the contents of this record.
RDB\$VIEW_SOURCE	seg-str	The user's source text for the view definition.

(continued on next page)

Table 8–9 (Cont.) The RDB\$RELATIONS System Relation

Field Name	Data Type	Summary Description
RDB\$ACCESS_CONTROL	seg-str	The access control policy for the relation.
RDB\$EXTENSION_PARAMETERS	seg-str	Reserved for future use.
RDB\$CDD_NAME	seg-str	The fully qualified name of the dictionary entity upon which the relation definition is based, as specified in the FROM PATHNAME clause.
RDBVMS\$SECURITY_AUDIT	longword	Bit mask.
RDBVMS\$SECURITY_ALARM	longword	Bit mask.

8.10 RDB\$VIEW_RELATIONS System Relation

The RDB\$VIEW_RELATIONS system relation lists all the relations that participate in a given view. There is one record for each relation or view in a view definition. Table 8–10 provides information on the fields of the RDB\$VIEW_RELATIONS system relation.

Table 8–10 The RDB\$VIEW_RELATIONS System Relation

Field Name	Data Type	Summary Description
RDB\$VIEW_NAME	text	Names a view or relation that uses another relation. The value of RDB\$VIEW_NAME is normally a view name, but might also be the name of a relation that includes a field computed using a statistical expression.
RDB\$RELATION_NAME	text	The name of a relation used to form the view.
RDB\$VIEW_CONTEXT	word	An identifier for the context variable used to identify a relation in the view. The context variable appears in the BLR represented by the field RDB\$VIEW_BLR in RDB\$RELATIONS.

8.11 RDBVMS\$COLLATIONS System Relation

The RDBVMS\$COLLATIONS system relation describes the collating sequence to be used in the database. Table 8–11 provides information on the fields of the RDBVMS\$COLLATIONS system relation.

Table 8–11 The RDBVMS\$COLLATIONS System Relation

Field Name	Data Type	Summary Description
RDBVMS\$COLLATION_NAME	text 31	Supplies the name by which the database's collating sequence is known within the database. This value is from the item RDB\$K_DPB2_CLTN_NAME specified in the RDB\$K_DPB2_COLLATION_SEQUENCE clause of the database parameter block (DPB).
RDBVMS\$COLLATION_SEQUENCE	seg-str	The BLR that identifies a table of bytes used internally by Rdb/VMS to perform operations that require use of the specified collating sequence. This value is from the item RDB\$K_DPB2_CLTN_SEQUENCE_TABLE specified in the RDB\$K_DPB2_COLLATION_SEQUENCE clause of the DPB.
RDB\$DESCRIPTION	seg-str	A user-supplied description of the collating sequence.
RDB\$FLAGS	word	Indicates whether the field is interpreted as ASCII or MCS format. Bit 0 indicates ASCII format. Bit 1 indicates MCS format.

8.12 RDBVMSS\$INTERRELATIONS System Relation

Contains information that indicates the interdependencies of objects in the database. The RDBVMSS\$INTERRELATIONS relation can be used to determine if an object can be deleted or if some other object depends upon its existence in the database. Table 8–12 provides information on the fields of the RDBVMSS\$INTERRELATIONS system relation.

Table 8–12 The RDBVMSS\$INTERRELATIONS System Relation

Field Name	Data Type	Summary Description
RDB\$RELATION_NAME	text	The name of the relation that cannot be deleted because it is used by some other entity in the database.
RDB\$FIELD_NAME	text	The name of the field that cannot be deleted because the field is used by another entity in the database.
RDBVMSS\$ENTITY_NAME1	text	The name of the entity that depends on the existence of the field identified by the RDB\$FIELD_NAME and RDB\$RELATION_NAME.
RDBVMSS\$ENTITY_NAME2	text	If used, the name of the entity, together with RDBVMSS\$ENTITY_NAME1, that depends on the existence of the field specified in RDB\$FIELD_NAME.
RDBVMSS\$USAGE	text	The relationship among RDB\$RELATION_NAME, RDB\$FIELD_NAME, RDBVMSS\$ENTITY_NAME1, and RDBVMSS\$ENTITY_NAME2. RDBVMSS\$USAGE can have the following values: constraint, computed field, storage map, view, or view field.
RDB\$FLAGS	word	Reserved for future use.
RDB\$CONSTRAINT_NAME	text 31	This field is the name of a constraint that is referred to from another system relation. The value in this field equates to a value for the same field in the RDB\$CONSTRAINTS system relation.

8.13 RDBVMS\$PRIVILEGES System Relation

Describes the protection for the database, relations, views, and fields. There is one record per grantor, grantee, and privileges combination per entity in the database.

A record is stored in the RDBVMS\$PRIVILEGES relation for each user who grants another user privileges for a database, relation, view, or field.

If the privilege for a database, relation, view, or field was granted without the SQL GRANT option, the record of the grantor and grantee is modified.

The privilege change takes effect at commit time of the command. Table 8–13 provides information on the fields of the RDBVMS\$PRIVILEGES system relation.

Note The RDBVMS\$PRIVILEGES system relation is used only in ANSI databases.

Table 8–13 The RDBVMS\$PRIVILEGES System Relation

Field Name	Data Type	Summary Description
RDB\$FIELD_ID	word	The field ID of the field for which protection is defined. If protection is on a database, relation, or view, this field is null. The value stored in this field must be unique within the database.
RDB\$RELATION_ID	word	The relation ID of the relation or view for which protection is defined. The field is null if the protection is defined for the database. The value stored in this field must be unique within the database.
RDBVMS\$GRANTOR	longword	The binary format UIC of the person who defined or changed the privileges. This is usually the UIC of the person who executed the protection command. For an IMPORT statement, the UIC is that of the person who originally defined the protection for the user, not necessarily the person who performed the IMPORT statement.

(continued on next page)

Table 8-13 (Cont.) The RDBVMS\$PRIVILEGES System Relation

Field Name	Data Type	Summary Description
RDBVMS\$GRANTEE	seg-str	The binary format of the UICs of the persons who hold privileges on the database relation or field.
RDBVMS\$PRIV_GRANT	longword scale 0	Specifies the access mask of privileges that the grantee has that he can grant to other users.
RDBVMS\$PRIV_NOGRANT	longword scale 0	Specifies the access mask of privileges that the grantee has that he can use himself but cannot give to other users.
RDB\$FLAGS	word	Reserved for future use.

8.14 RDBVMS\$RELATION_CONSTRAINTS System Relation

The RDBVMS\$RELATION_CONSTRAINTS system relation lists all relation-specific constraints. Table 8-14 provides information on the fields of the RDBVMS\$RELATION_CONSTRAINTS system relation.

Table 8-14 The RDBVMS\$RELATION_CONSTRAINTS System Relation

Field Name	Data Type	Summary Description
RDBVMS\$CONSTRAINT_MATCH_TYPE	word	The match type associated with a referential integrity relation-specific constraint. This field is not used by the database system. The value is always zero.

(continued on next page)

Table 8-14 (Cont.) The RDBVMS\$RELATION_CONSTRAINTS System Relation

Field Name	Data Type	Summary Description
RDB\$CONSTRAINT_NAME	text 31	The name of the constraint defined by the relation specified by RDB\$RELATION_NAME. The value in this field equates to a value for the same field in the RDB\$CONSTRAINTS system relation.
RDB\$CONSTRAINT_SOURCE	seg-str	This text string contains the DDL source of the constraint from the relation definition. This field is not used by the database system; it reflects the relation context expression of the constraint.
RDBVMS\$CONSTRAINT_TYPE	word	The type of relation-specific constraint defined. The values are shown in Table 8-15.
RDBVMS\$ERASE_ACTION	word	The type of referential integrity erase action specified. This field is not used by the database system. The value is always zero.
RDB\$FIELD_NAME	text 31	The name of the field for which a field level relation-specific constraint is defined. The field is null for a relation level constraint. This field is not used by the database system; it identifies field level constraints for the interfaces.

(continued on next page)

Table 8-14 (Cont.) The RDBVMS\$RELATION_CONSTRAINTS System Relation

Field Name	Data Type	Summary Description
RDB\$FLAGS	word	The relation constraint flag. This value is not used by the database system.
RDBVMS\$MODIFY_ACTION	word	The type of referential integrity modify action specified. This field is not used by Rdb/VMS. The value is always zero.
RDBVMS\$REFD_CONSTRAINT_NAME	text 31	The name of the unique or primary key constraint referred to by a referential integrity foreign key constraint. If the constraint is not a referential integrity constraint or no referential integrity constraint was specified, this field will be null. Otherwise, the value in this field will equate to a value for the same fields in the RDB\$CONSTRAINTS and RDBVMS\$RELATION_CONSTRAINTS_FLDS system relations. This field is used to determine the foreign key referenced relation name and referenced field names. (continued on next page)

Table 8-14 (Cont.) The RDBVMS\$RELATION_CONSTRAINTS System Relation

Field Name	Data Type	Summary Description
RDB\$RELATION_NAME	text 31	The name of the relation on which the specified constraint is defined. The value in this field equates to a value for the same field in the RDB\$RELATIONS system relation.

Table 8-15 Values for RDBVMS\$CONSTRAINT_TYPE

Value	Symbol	Meaning
1	RDB\$K_CON_CONDITION	Requires conditional expression constraint
2	RDB\$K_CON_PRIMARY_KEY	Primary key constraint
3	RDB\$K_CON_REFERENTIAL	Referential (foreign key) constraint
4	RDB\$K_CON_UNIQUE	Unique constraint
5	RDB\$K_CON_VIEW_CHECK ¹	View update check constraint
6	RDB\$K_CON_NOT_NULL	Not null (missing) constraint

¹Not used by the database system or by RDO

8.15 RDBVMS\$RELATION_CONSTRAINT_FLDS System Relation

RDBVMS\$RELATION_CONSTRAINT_FLDS lists the fields that participate in unique, primary, or foreign key declarations for relation-specific constraints.

There is one record for each field that represents all or part of a unique, primary, or foreign key field.

The records in this relation should not be modified. To change a record, delete it and store it again. Table 8–16 provides information on the fields of the RDBVMS\$RELATION_CONSTRAINT_FLDS system relation.

Table 8–16 The RDBVMS\$RELATION_CONSTRAINT_FLDS System Relation

Field Name	Data Type	Summary Description
RDB\$CONSTRAINT_NAME	text 31	The name of a constraint for which the specified field participates in the unique field or key declaration.
RDB\$FIELD_NAME	text 31	The name of the field that is all or part of a unique field or key for the specified constraint. The value in this field is the same as that stored in the RDB\$RELATION_FIELDS system relation.
RDB\$FIELD_POSITION	word	The ordinal position of the specified field within the field list that declares the unique or key field. For field level constraints, there will always be only one field in the list. The first field in the list has position value 1, the second has position value 2, and so on.
RDB\$FLAGS	word	The relation constraint field flags. This field is reserved for future use.

8.16 RDBVMS\$STORAGE_MAPS System Relation

The RDBVMS\$STORAGE_MAPS system relation contains information about each storage map. Table 8–17 provides information on the fields of the RDBVMS\$STORAGE_MAPS system relation.

Table 8–17 The RDBVMS\$STORAGE_MAPS System Relation

Field Name	Data Type	Summary Description
RDBVMS\$MAP_NAME	text	The name of the storage map.
RDB\$RELATION_NAME	text	The name of the relation to which the storage map refers.
RDB\$INDEX_NAME	text	The name of the index specified in the PLACEMENT VIA INDEX clause of the storage map.
RDB\$FLAGS	word	The following flag bit offsets indicate what the map is for: 0 mixed format area 14 index
RDBVMS\$MAP_SOURCE	seg-str	The user's source text for the storage map definition.
RDB\$DESCRIPTION	seg-str	A user-supplied description of the contents of this record.
RDB\$EXTENSION_PARAMETERS	seg-str	Reserved for future use.
RDBVMS\$VERTICAL_PARTITION_INDEX	word	Reserved for future use.

8.17 RDBVMS\$STORAGE_MAP_AREAS System Relation

The RDBVMS\$STORAGE_MAP_AREAS system relation contains information about each storage area to which a storage map refers. Table 8–18 provides information on the fields of the RDBVMS\$STORAGE_MAP_AREAS system relation.

Table 8–18 The RDBVMS\$STORAGE_MAP_AREAS System Relation

Field Name	Data Type	Summary Description
RDBVMS\$MAP_NAME	text	The name of the storage map.
RDBVMS\$AREA_NAME	text	The name of the storage area referred to by the storage map.

(continued on next page)

Table 8-18 (Cont.) The RDBVMS\$STORAGE_MAP_AREAS System Relation

Field Name	Data Type	Summary Description
RDB\$ROOT_DBK	text	A pointer to the base of the SORTED index, if it is a SORTED index.
RDBVMS\$ORDINAL_POSITION	word	The order of the storage area represented by this record in the map.
RDB\$STORAGE_ID	word	For a relation, a pointer to the database logical area. For a hashed index, a pointer to the system record.
RDB\$INDEX_ID	word	A pointer to the index logical area.
RDBVMS\$STORAGE_BLR	seg-str	The BLR that represents the WITH LIMIT OF clause in the storage map definition.
RDB\$DESCRIPTION	seg-str	A user-supplied description of the contents of this record.
RDB\$EXTENSION_PARAMETERS	seg-str	Reserved for future use.
RDBVMS\$VERTICAL_PARTITION_INDEX	word	Reserved for future use.

8.18 RDBVMS\$TRIGGERS System Relation

The RDBVMS\$TRIGGERS system relation describes the definition of a trigger. Triggers cannot be modified. To change a trigger, delete it and redefine it. Table 8-19 provides information on the fields of the RDBVMS\$TRIGGERS system relation.

Table 8-19 The RDBVMS\$TRIGGERS System Relation

Field Name	Data Type	Summary Description
RDB\$DESCRIPTION	seg-str	A user-supplied text string describing the trigger. Not used by the database system, but by the interfaces when displaying the trigger definition.
RDB\$FLAGS	word	This field is not used.
RDB\$RELATION_NAME	text	The name of the relation this trigger is defined for. The trigger may be selected on an update to the named relation (qualified by the fields described in the field name list). This relation is used as a subject relation for all contexts that refer to it.
RDBVMS\$TRIGGER_ACTIONS	seg-str	A text string containing all the sets of triggered actions defined for this trigger. The string consists of one or more sets of clumplets, one set for each triggered action. See Section 8.18.1 for a description of the clumplets that can appear within this segmented string.
RDBVMS\$TRIGGER_CONTEXTS	word	The context number used within the triggered action BLR to map the triggered action BLR to the current context of the triggering update statement.

(continued on next page)

Table 8-19 (Cont.) The RDBVMS\$TRIGGERS System Relation

Field Name	Data Type	Summary Description
RDBVMS\$TRIGGER_FIELD_NAME_LIST	seg-str	A text string composed of a count field and one or more counted strings. The count is an unsigned word that represents the number of strings in the list. The counted strings are ASCII names that represent field names. If the trigger is of event type MODIFY, it will be evaluated if one or more of the specified fields has been modified.
RDBVMS\$TRIGGER_NAME	text 31	The name of a trigger. This name must be a unique trigger name within the database.
RDBVMS\$TRIGGER_NEW_CONTEXT	word	A context number used within the triggered action's BLR to refer to the new row values for the subject relation for a MODIFY event.
RDBVMS\$TRIGGER_OLD_CONTEXT	word	A context number used within the triggered action's BLR to refer to the old row values of the subject relation that existed before a MODIFY event.
RDBVMS\$TRIGGER_SOURCE	seg-str	An optional text string for the trigger definition. The string is not used by the database system. It should reflect the full definition of the trigger. This field is used by the interfaces to display the trigger definition.

(continued on next page)

Table 8-19 (Cont.) The RDBVMS\$TRIGGERS System Relation

Field Name	Data Type	Summary Description
RDBVMS\$TRIGGER_TYPE	word	The type of trigger, as defined by the combination of the trigger action time and the trigger event. Action times are BEFORE and AFTER, and events are STORE, ERASE, and MODIFY. The values that represent the type of trigger are shown in Table 8-20.

Table 8-20 Trigger Type Values

Numeric Value	Symbolic Value	Description
1	RDB\$K_BEFORE_STORE	Trigger will be evaluated before a STORE.
2	RDB\$K_BEFORE_ERASE	Trigger will be evaluated before an ERASE.
3	RDB\$K_BEFORE_MODIFY	Trigger will be evaluated before a MODIFY.
4	RDB\$K_AFTER_STORE	Trigger will be evaluated after a STORE.
5	RDB\$K_AFTER_ERASE	Trigger will be evaluated after an ERASE.
6	RDB\$K_AFTER_MODIFY	Trigger will be evaluated after a MODIFY.

8.18.1 Clumplets That Can Be Used in RDBVMS\$TRIGGER_ACTIONS

RDBVMS\$TRIGGER_ACTIONS is a segmented string that contains all the sets of triggered actions defined for a specified trigger. The string consists of one or more sets of **clumplets**, one set for each triggered action. There can be multiple instances of triggered statements for each triggered action. The clumplets that can appear within the string are:

- RDB\$K_TRIGGERED_ACTION

This clumplet identifies a set of clumplets that represent a single triggered action and specifies the number of statements in the action.

This clumplet has the format:

```
RDB$K_TRIGGERED_ACTION <number>
    <number> ::= word
```

Argument

number

A word that specifies the number of statements in the action. The word must contain an integer value.

- RDB\$K_TRIGGERED_FREQUENCY

This clumplet specifies whether an action will be evaluated once per triggering statement, or for each row updated by the triggering statement. The default is RDB\$K_TRIGGER_ONCE.

This clumplet has the format:

```
RDB$K_TRIGGERED_FREQUENCY <flag>
    <flag> ::= byte
```

Argument

flag

A byte that determines how often the triggered action is executed. If the byte value is RDB\$K_TRIGGER_ONCE (0), then the triggered action is executed once for the triggering statement. If the byte value is RDB\$K_TRIGGER_FOR_EACH_ROW (1), then the triggered action is executed for each row updated by the triggering statement.

- RDB\$K_TRIGGERED_CONDITION

This clumplet represents the Boolean expression that must evaluate to true for the triggered statement to execute.

This clumplet has the format:

```
RDB$K_TRIGGERED_CONDITION <length> <blr_string>
    <length>          ::= word
    <blr_string>      ::= <blr-version>
                        <blr-expression>
    <blr-version>    ::= unsigned byte version number
    <blr-expression> ::= BLR expression
```

Arguments

length

A word that specifies the number of bytes in the BLR string.

blr_string

A string that specifies the Boolean expression BLR.

- RDB\$K_TRIGGERED_STATEMENT

This clumplet represents the triggered statement that will be executed.

This clumplet has the format:

```
RDB$K_TRIGGERED_STATEMENT <length> <blr_string>
    <length>           ::= word
    <blr_string>       ::= <blr-version>
                        <blr-expression>
    <blr-version>     ::= unsigned byte version number
    <blr-expression> ::= BLR expression
```

Arguments

length

A word that specifies the number of bytes in the BLR string.

blr_string

A string that specifies the triggered statement BLR.

Part 2

VAX Rdb/VMS Statements

This chapter describes the syntax and semantics of all statements in the VAX Rdb/VMS language. These include data definition statements, data manipulation statements, database maintenance and analysis statements, and statements that control the RDO interactive environment and give information. See Chapter 2 for a set of tables that summarizes the functions of these statements.

This chapter also documents the VAX Data Distributor statements, which are run in the RDO environment. The VAX Data Distributor statements are:

- DEFINE SCHEDULE
- DEFINE TRANSFER
- DELETE SCHEDULE
- DELETE TRANSFER
- REINITIALIZE TRANSFER
- SHOW TRANSFER
- START TRANSFER
- STOP TRANSFER

Reference descriptions of the VAX Data Distributor statements also appear in the *VAX Data Distributor Handbook*; the statements are documented here for the convenience of VAX Data Distributor users.

Note *Not all the Rdb/VMS statements described in this chapter are available with the run-time only (RTO) version of Rdb/VMS. Appendix C contains a table that identifies whether an Rdb/VMS statement can be executed using the RTO license.*

ANALYZE Statement

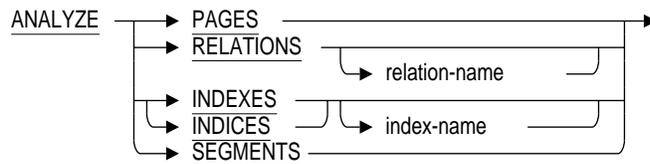
9.1 ANALYZE Statement

Displays statistics on how the database uses space. The ANALYZE statement works on the database most recently invoked, and uses a read/write transaction.

Note The RDO ANALYZE statement is available for single-file databases only. Use the RMU/ANALYZE command for multifile databases.

The ANALYZE statement displays different charts, depending on the option you choose (RELATIONS, INDEXES, PAGES, or SEGMENTS).

Format



Arguments

PAGES

Displays a bar chart that shows page usage for the database. This chart indicates how many pages the database uses and the percentage to which each page is filled.

RELATIONS

Displays the following information about the named relation:

- Occurrences
The total number of records.
- Bytes used
The total number of bytes.
- Average record in bytes
Average number of bytes occupied by each record.
- Percent fragmented

ANALYZE Statement

The percentage of records that are fragmented (split over more than one database page).

- Percent of total space

The percentage of space *available* for a relation that is used to store user data.

- Percent of used space

The percentage of total space in a relation *used* to store user data.

INDEXES

INDICES

Displays the following information about the named index:

- Index levels

The number of levels in the balanced-tree (B-tree) needed to store all the index nodes.

- Index nodes

The total number of B-tree nodes in the index.

- Length used

Total length in bytes of all the nodes in this index.

- Duplicate nodes

The number of duplicate values in the index. This value is always zero in an index defined with the DUPLICATES ARE NOT ALLOWED clause.

- Duplicate length used

Total length in bytes of all the duplicate nodes in this index.

SEGMENTS

Displays the following information about the segmented strings stored in the database:

- The total number of segmented string records
- The total number of bytes occupied by segmented strings
- Average number of bytes occupied by each segmented string record
- The percentage of segmented string records that are fragmented (split over more than one database page)

ANALYZE Statement

- The percentage of space *available* for segmented strings that is used to store user data
- The percentage of space *used* for segmented strings that is used to store user data

Usage Notes

You must have the Rdb/VMS ADMINISTRATOR privilege to use the ANALYZE statement.

Examples

Example 1

The following display indicates that 504 database pages are occupied by user data, index data, or space area management pages. Of these, 132 pages are between 80 and 90 percent full, while only 4 pages are between 10 and 20 percent full.

```
RDO> INVOKE DATABASE PATHNAME 'PERSONNEL'  
RDO> ANALYZE PAGES
```

```
Space utilization analysis - completed at 4-SEP-1985 09:31:57.73  
504 data pages, each page is 2 blocks long  
Available data storage area is 75 percent utilized
```

```
-- %used --- #data pages -----  
90 -100%      159 =====  
80 - 90%      132 =====  
70 - 80%       52 =====  
60 - 70%        3 =  
50 - 60%       38 =====  
40 - 50%       23 =====  
30 - 40%       31 =====  
20 - 30%        1 =  
10 - 20%        4 =  
 0 - 10%       61 =====
```

Example 2

The following command analyzes all the relations in a database.

ANALYZE Statement

RDO> ANALYZE RELATIONS

Record Name	Occurrences	Bytes Used	Avg Rec in Bytes	% Frag	% Total Space	% Used Space
COLLEGES	16	785	49	0	13	20
DEGREES	166	5628	34	0	48	56
DEPARTMENTS	26	1168	45	0	20	27
EMPLOYEES	100	8158	82	0	55	68
JOBS	15	584	39	0	10	16
JOB_HISTORY	274	11182	41	0	63	67
RDB\$CONSTRAINTS	0	0	0	0	0	0
RDB\$CONSTRAINT_RELATIONS	0	0	0	0	0	0
RDB\$DATABASE	1	597	597	0	10	17
RDB\$FIELDS	86	34658	403	0	74	90
RDB\$FIELD_VERSIONS	162	13284	82	0	64	76
RDB\$INDEX_SEGMENTS	15	1080	72	0	18	26
RDB\$INDICES	14	1330	95	0	23	30
RDB\$RELATIONS	22	1936	88	0	33	38
RDB\$RELATION_FIELDS	162	73062	451	0	89	94
RDB\$VIEW_RELATIONS	8	576	72	0	10	16
RDBVMS\$COLLATIONS	0	0	0	0	0	0
RDBVMS\$INTERRELATIONS	0	0	0	0	0	0
RDBVMS\$PRIVILEGES	0	0	0	0	0	0
RDBVMS\$RELATION_CONSTRAINTS	18	2718	151	0	46	47
RDBVMS\$RELATION_CONSTRAINT_FLDS	11	814	74	0	14	21
RDBVMS\$STORAGE_MAPS	0	0	0	0	0	0
RDBVMS\$STORAGE_MAP_AREAS	0	0	0	0	0	0
RDBVMS\$TRIGGERS	3	339	113	0	6	10
RESUMES	0	0	0	0	0	0
SALARY_HISTORY	729	24185	33	0	68	73
WORK_STATUS	3	92	31	0	2	3
Segmented string area	138	9162	66	0	52	69
	1937	187467				

Consider this portion of the preceding example:

Record Name	Occurrences	Bytes Used	Avg Rec in Bytes	% Frag	% Total Space	% Used Space
EMPLOYEES	100	8158	82	0	55	68

Here, the EMPLOYEES relation has 100 records, which occupy a total of 8158 bytes, for an average of 82 bytes per record. The user records in the EMPLOYEES relation occupy 55 percent of the space allocated for the relation in the database and 68 percent of the space actually in use. No records in the database are fragmented, which indicates that the page size is adequate to hold the longest records.

ANALYZE Statement

Example 3

The following command analyzes all the indexes in a database.

```
RDO> ANALYZE INDEXES
```

Index Name	Index levels	Index nodes	Length used	Duplicate nodes	Duplicate length used
COLL_COLLEGE_CODE	1	1	150	0	0
DEG_COLLEGE_CODE	1	1	99	12	3132
DEG_EMP_ID	2	3	731	67	6164
EMP_EMPLOYEE_ID	2	3	681	0	0
JH_EMPLOYEE_ID	2	3	740	80	7360
RDB\$CON_CONSTRAINT_NAME_X	1	1	0	0	0
RDB\$CR_CONSTRAINT_NAME_NDX	1	1	0	0	0
RDB\$CR_REL_NAME_NDX	1	1	0	0	0
RDB\$FIELDS_NAME_NDX	2	12	2726	0	0
RDB\$VER_REL_ID_VER_NDX	1	1	156	28	4604
RDB\$NDX_SEG_NAM_FLD_POS_NDX	2	4	797	0	0
RDB\$NDX_NDX_NAME_NDX	2	3	670	0	0
RDB\$NDX_REL_NAME_NDX	2	3	476	6	552
RDB\$REL_REL_ID_NDX	1	1	134	0	0
RDB\$REL_REL_NAME_NDX	2	4	743	0	0
RDB\$RFR_REL_FLD_NAMES_NDX	2	5	773	28	4604
RDB\$VIEW_REL_NAME_NDX	1	1	244	1	92
RDB\$VIEW_VIEW_NAME_NDX	1	1	244	1	92
RDBVMS\$COLLATIONS_NDX	1	1	0	0	0
RDBVMS\$INTER_CON_NAM_NDX	2	3	632	7	928
RDBVMS\$INTER_ENTITY_NDX	1	1	0	0	0
RDBVMS\$INTER_REL_FLD_NDX	1	1	0	0	0
RDBVMS\$STO_MAP_REL_NAM_NDX	1	1	0	0	0
RDBVMS\$PRV_FLD_ID_NDX	1	1	0	0	0
RDBVMS\$PRV_REL_ID_NDX	1	1	0	0	0
RDBVMS\$RLC_CONSTRAINT_NAME_NDX	2	3	623	0	0
RDBVMS\$RLC_FIELD_NAME_NDX	1	1	258	5	128
RDBVMS\$RLC_RELATION_NAME_NDX	1	1	324	5	112
RDBVMS\$RCF_CONSTRAINT_NAME_NDX	1	1	343	0	0
RDBVMS\$STO_MAP_MAP_NAM_NDX	1	1	0	0	0
RDBVMS\$STO_REL_NAM_NDX	1	1	0	0	0
RDBVMS\$STO_MAP_AREA_AREA_NDX	1	1	0	0	0
RDBVMS\$STO_MAP_AREA_MAP_NDX	1	1	0	0	0
RDBVMS\$TRG_RELATION_NAME_NDX	1	1	109	0	0
RDBVMS\$TRG_TRIGGER_NAME_NDX	1	1	109	0	0
SH_EMPLOYEE_ID	2	3	747	103	12180

ANALYZE Statement

Consider the portion of the previous display for the DEG_COLLEGE_CODE index. This index has been defined for the DEGREES relation with a DUPPLICATES ARE ALLOWED clause:

Index Name	Index levels	Index nodes	Length used	Duplicate nodes	Duplicate length used
DEG_COLLEGE_CODE	1	1	99	12	3132

This index includes one level of B-tree index nodes, one node on the B-tree, which occupies 99 bytes, and 12 duplicate nodes, which occupy 3132 bytes.

Example 4

The following example uses the SEGMENTS option in the ANALYZE statement to analyze only the segmented string portion of an Rdb/VMS database. Some information in Rdb/VMS system relations is stored in segmented strings. For this reason, the ANALYZE SEGMENTS statement will display information about segmented strings even if your database does not contain user-defined records with segmented string fields.

```
RDO> INVOKE DATABASE FILENAME PERSONNEL
RDO> SHOW FIELDS FOR RESUMES
Fields for relation RESUMES
EMPLOYEE_ID          text size is 5
    based on global field ID_NUMBER
RESUME                segmented string
    segment_length 512
RDO> FOR R IN RDB$RELATIONS
cont> WITH R.RDB$RELATION_NAME="RESUMES"
cont> PRINT R.RDB$RELATION_NAME, R.RDB$CARDINALITY
cont> END_FOR
RDB$RELATION_NAME      RDB$CARDINALITY
RESUMES                0
RDO> ! No user-defined segmented strings in this database
RDO> ANALYZE SEGMENTS
```

Record Name	Occurrences	Bytes Used	Avg Rec in Bytes	% Frag	% Total Space	% Used
Segmented string area	156	10722	69	0	61	71

```
RDO> ! Data reports on system relation information
RDO> ! stored in segmented strings
RDO> FINISH
RDO> EXIT

$ ! After storing six resumes in the database:
$ RDO
RDO> INVOKE DATABASE FILENAME PERSONNEL
RDO> FOR R IN RDB$RELATIONS
cont> WITH R.RDB$RELATION_NAME = "RESUMES"
```

ANALYZE Statement

```
cont> PRINT R.RDB$RELATION_NAME, R.RDB$CARDINALITY
cont> END_FOR
```

```
      RDB$RELATION_NAME          RDB$CARDINALITY
RESUMES                          6
```

```
RDO> ANALYZE SEGMENTS
```

Record Name	Occurrences	Bytes Used	Avg Rec in Bytes	% Frag	% Total Space	% Used Space
-----	-----	-----	-----	-----	-----	-----
Segmented string area	545	29231	54	0	66	77

```
RDO> EXIT
```

CHANGE DATABASE Statement

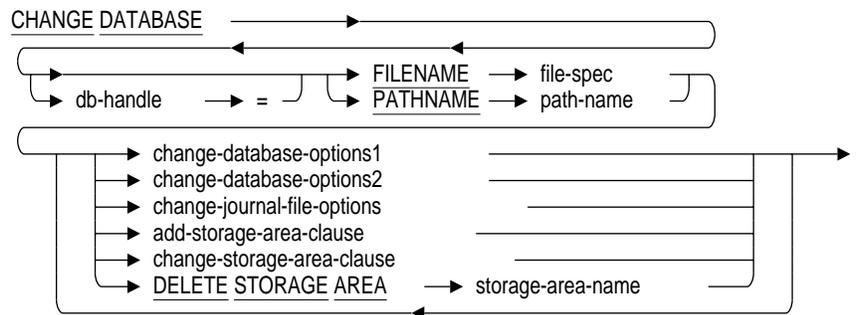
9.2 CHANGE DATABASE Statement

Changes characteristics of the database root file and storage area files. The CHANGE DATABASE statement lets you override certain characteristics. It also lets you control characteristics that you cannot specify in the DEFINE DATABASE statement. When the CHANGE DATABASE statement executes, Rdb/VMS updates the characteristics named in the statement. All other characteristics remain the same.

Use the CHANGE DATABASE statement to:

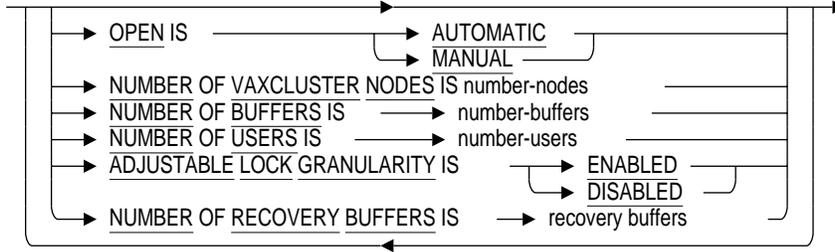
- Define, change, and delete storage areas in multifile databases
- Enable or disable after-image journaling and change journal file characteristics
- Enable or disable snapshot transactions and change snapshot file characteristics
- Change a read/write storage area to read-only, or a read-only storage area to read/write
- Change the number of recovery buffers
- Change physical parameters associated with the database
- Specify whether the database can be opened automatically or manually
- Require the use of the data dictionary

Format

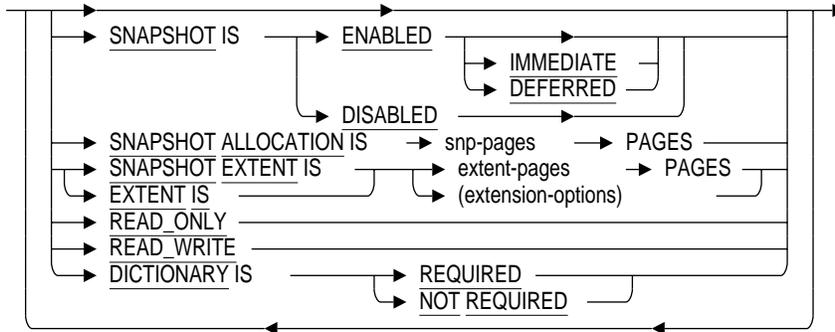


CHANGE DATABASE Statement

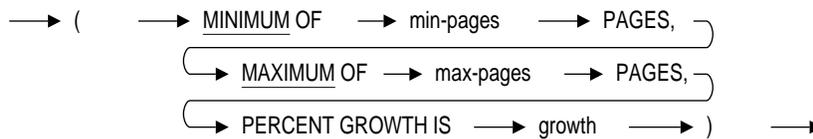
change-database-options1 =



change-database-options2 =

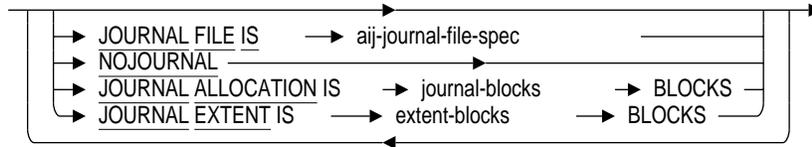


extension-options =

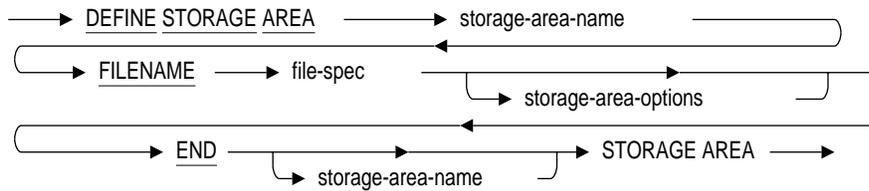


CHANGE DATABASE Statement

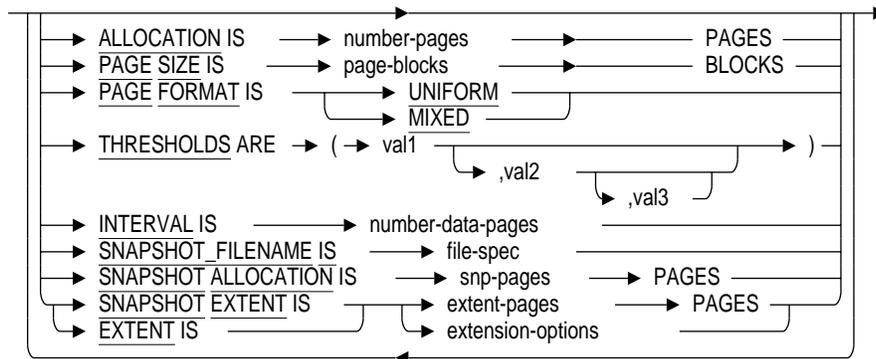
change-journal-file-options =



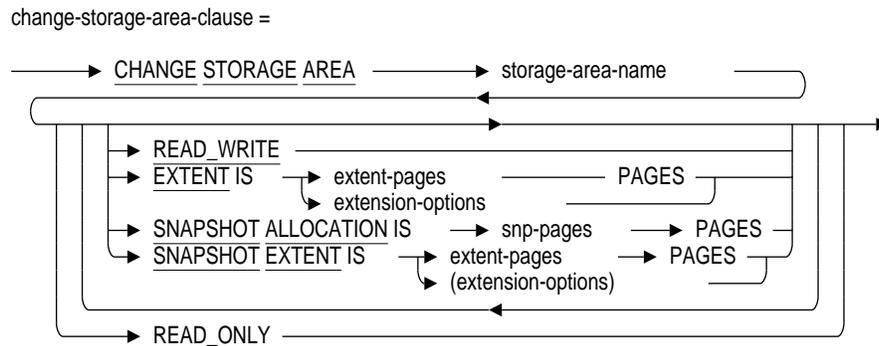
add-storage-area-clause =



storage-area-options =



CHANGE DATABASE Statement



Arguments

db-handle

A host language variable or name that you associate with the database. Use a database handle when you access more than one database at a time.

FILENAME file-spec

The name of the database file. Put the name in quotation marks.

PATHNAME path-name

The path name that refers to the data dictionary entity for the database. Put the name in quotation marks. When you specify the PATHNAME qualifier instead of the FILENAME qualifier, the path name indirectly specifies the database root file. Specify either:

- A full data dictionary path name, such as 'DISK1:[CDD]CORP.PERS'
- A relative data dictionary path name such as 'PERS'

If you use a relative path name, the current default directory must include all the path name segments preceding the relative path name.

OPEN IS AUTOMATIC

Specifies that any user can open a closed or previously unopened database by simply invoking it and executing a data manipulation language statement. The default is OPEN IS AUTOMATIC.

CHANGE DATABASE Statement

OPEN IS MANUAL

Specifies that only users with sufficient Rdb/VMS privilege (ADMINISTRATOR) for the database can enter an explicit OPEN statement to open the database.

NUMBER VAXCLUSTER NODES IS number-nodes

Sets the upper limit on the maximum number of VAXcluster nodes from which users can access the shared database. The default is 16 nodes. The range is 1 node to 64 nodes. The actual maximum limit is the current VMS VAXcluster limit.

You can change this characteristic only for multifile databases. Use the EXPORT and IMPORT statements to change this characteristic for single-file databases.

NUMBER BUFFERS IS number-buffers

The number of buffers Rdb/VMS allocates per process using this database. Specify an unsigned integer between 2 and 32768. The default is 20 buffers.

NUMBER USERS IS number-users

The maximum number of users allowed to access the database at one time. The default is 50 users. After the maximum number is reached, the next user who tries to invoke the database receives an error message and must wait. The largest number of users you can specify is 2032 and the fewest number of users is 1. With VAX ACMS, each attached server can support multiple users. Thus, the actual number of terminal users can be greater than 2032 if you are using VAX ACMS.

Note that the number of users is defined as the number of active attaches to the database. Thus, if a single process is running one program, but that program performs 12 INVOKE . . . FINISH operations, to Rdb/VMS there are 12 active users.

You can change this characteristic only for multifile databases. Use the EXPORT and IMPORT statements to change this characteristic for single-file databases.

ADJUSTABLE LOCK GRANULARITY IS ENABLED [Default]

ADJUSTABLE LOCK GRANULARITY IS DISABLED

Enables or disables adjustable locking granularity. The default is ENABLED. Generally, enabling adjustable locking granularity results in fewer locks being used. However, when contention for database pages is high, performance may be impaired as locking granularity is adjusted to a lower level. If your application is query intensive, enable adjustable locking granularity. If your

CHANGE DATABASE Statement

application processes specific records and performs a substantial number of update operations, you might want to disable adjustable locking granularity.

Disabling adjustable locking granularity may require that the VMS SYSGEN parameters for locks be increased.

For more information, see the *VAX Rdb/VMS Guide to Database Maintenance and Performance*.

NUMBER RECOVERY BUFFERS IS *recovery-buffers*

The number of database buffers used during the automatic recovery process that is initiated after a system or process failure. This recovery process uses the recovery-unit journal file. Specify an unsigned integer between 2 and 32768. The default is 20 buffers.

SNAPSHOT IS ENABLED IMMEDIATE [Default]

SNAPSHOT IS ENABLED DEFERRED

SNAPSHOT IS ENABLED IMMEDIATE specifies that read/write transactions write copies of records to the snapshot file before those records are modified, regardless of whether a read-only transaction is active.

SNAPSHOT IS ENABLED DEFERRED specifies that read/write transactions *not* write copies of records they modify to the snapshot file unless a read-only transaction is active. Read-only transactions that attempt to start after an active read/write transaction begins must wait for all active read/write users to complete their transactions.

You enable snapshot writing to all snapshot files for all storage areas when you specify the SNAPSHOT IS ENABLED clause.

The default is SNAPSHOT IS ENABLED IMMEDIATE. Although IMMEDIATE is the default, if you have set snapshots ENABLED DEFERRED, you must specify both ENABLED and IMMEDIATE to return the database to the default setting.

SNAPSHOT IS DISABLED

Disables snapshot writing. You disable snapshot writing to all snapshot files for all storage areas when you specify the SNAPSHOT IS DISABLED clause.

Do not delete snapshot files unless you also delete the database itself. If you do not want snapshots enabled, disable them with the SNAPSHOT IS DISABLED clause.

CHANGE DATABASE Statement

SNAPSHOT ALLOCATION IS snp-pages

The number of pages allocated for the snapshot file. The default is 100 pages. You can set the snapshot allocation to 0 pages. In the following cases, you may want to set the snapshot allocation to 0 pages:

- If you have disabled snapshots. By setting the snapshot allocation to 0 you may save space.
- If you have changed a read/write storage area to read-only. The snapshot file is not used, and you can save space by setting the snapshot allocation to 0 pages.

Rdb/VMS allocates pages in groups of three; therefore, if you specify that two pages be allocated for a storage area, Rdb/VMS allocates three pages instead. Rdb/VMS also allocates a space area management (SPAM) entry page initially for the storage area. Because Rdb/VMS allocates pages in groups of three and also allocates a SPAM page, a total of four pages are allocated for the storage area even though you requested an allocation of only two pages.

SNAPSHOT EXTENT IS extent-pages

EXTENT IS extent-pages

The number of pages of each extent. Use this parameter for simple control over the page extents. For greater control, and particularly for multivolume databases, use the MIN, MAX, and PERCENT parameters instead. The default is 100 pages. You can set the extent and the snapshot extent to 0 if you have disabled snapshots, or if you have changed a read/write storage area to read-only.

min-pages

The minimum number of pages of each extent. The default is 99 pages.

max-pages

The maximum number of pages of each extent. The default is 9,999 pages.

growth

The percent growth of each extent. The default is 20 percent growth.

READ_ONLY

READ_WRITE

In the change-database-options2 clause, the READ_ONLY and READ_WRITE options are used to change the RDB\$SYSTEM storage area (and the Rdb/VMS system relations stored in the area) to the selected option. Select the READ_WRITE option to change a read-only RDB\$SYSTEM storage area to read/write.

CHANGE DATABASE Statement

Select the `READ_ONLY` option to change a read/write `RDB$SYSTEM` storage area to read-only. You might choose the `READ_ONLY` option if your database is never or rarely updated. When the `RDB$SYSTEM` storage area is changed to read-only, locking conflicts occur less frequently, and the automatic updating of index and relation cardinality is inhibited.

No write operation can be done in a read-only storage area except a cardinality update. See Section 6.5 for more information and restrictions on updating cardinalities.

You must use the `change-storage-area-clause` to change a storage area other than the `RDB$SYSTEM` storage area to read-only or read/write, or to change a database to read-only. For information on how to make these changes, see the other `READ_ONLY` and `READ_WRITE` argument description later in this arguments list.

DICTIONARY IS REQUIRED

DICTIONARY IS NOT REQUIRED [Default]

Determines whether the database must be invoked by path name for data definition changes to occur. If you specify the `DICTIONARY IS REQUIRED` option, the database must be invoked by path name to change metadata and the data dictionary will be maintained. If you specify the `DICTIONARY IS NOT REQUIRED` option, the database can be invoked by either file name or path name to change metadata. The default is `DICTIONARY IS NOT REQUIRED`.

If you specify the `DICTIONARY` option, you cannot specify any other options in the same `CHANGE DATABASE` statement.

JOURNAL FILE IS `aij-journal-file-spec`

Directs Rdb/VMS to enable after-image journaling. When you define the database, by default, after-image journaling is disabled. The `aij-journal-file-spec` provides the file specification for the after-image journal (AIJ) file. It is a good idea to place the AIJ file on a disk other than the disk on which the database root (RDB) file, storage area (RDA) files, and snapshot (SNP) files reside.

NOJOURNAL

Directs Rdb/VMS to discontinue after-image journaling.

JOURNAL ALLOCATION IS `journal-blocks`

The number of blocks allocated to the after-image journal file. The default is zero blocks.

CHANGE DATABASE Statement

JOURNAL EXTENT IS extent-blocks

The size of each extent of the after-image journal file. The default is 512 blocks.

add-storage-area-clause

Allows you to give a file specification for a storage area you want to add to the database. This clause also allows you to specify storage area options for the new storage area. You cannot add a storage area to a single-file database. To restructure a single-file database into a multifile database, use the RDO EXPORT and IMPORT statements.

storage-area-name

The name of the storage area that you want to define, change, or delete. When you define a storage area, follow these rules when choosing a name:

- Use a name that is unique among all storage area names in the database.
- Use any valid VMS name. However, the name cannot end in a dollar sign (\$) or underscore (_).
- Do not use any Rdb/VMS reserved words (see Appendix A).
- Do not choose RDB\$SYSTEM as the storage area name. Use the change-database-options clause to change the RDB\$SYSTEM storage area to a read-only or a read/write area.

file-spec

The storage area file that is associated with the named storage area. The default file type is RDA. Put this file specification in quotation marks. Use either a full or partial file specification, or a logical name. If you use a simple file name, Rdb/VMS creates the storage area file in the current default directory.

Use a name that is unique among all storage area file names defined for the database.

ALLOCATION IS number-pages

The number of pages allocated to the storage area initially. Rdb/VMS automatically extends the allocation to handle the loading of data and subsequent expansion. The default is 400 pages. If you are loading a large database, a large allocation prevents the file from having to be extended many times.

CHANGE DATABASE Statement

PAGE SIZE IS page-blocks

The size in blocks of each storage area page. Page size is allocated in 512-byte blocks. The default is 2 blocks (1024 bytes). If your largest record is larger than approximately 950 bytes, allocate more blocks per page to prevent records from being fragmented.

The page size you specify must be smaller than the buffer size specified for the database or you will receive an error message.

PAGE FORMAT IS UNIFORM [Default]

PAGE FORMAT IS MIXED

Specifies whether a storage area contains UNIFORM or MIXED pages. You can use the PAGE FORMAT option with multifile databases only. In storage areas with UNIFORM page format, all pages in a specific logical area contain records from the same relation. In storage areas with MIXED page format, pages can hold records from different relations. The default is UNIFORM.

The default storage area, RDB\$SYSTEM, must have UNIFORM pages.

THRESHOLDS ARE (val1, val2, val3)

Specifies one, two, or three threshold values. The threshold values represent a fullness percentage on a data page and establish four possible ranges of guaranteed free space on the data pages. When a data page reaches the percentage defined by a given threshold value, the space area management (SPAM) entry for the data page is updated to reflect the new fullness percentage and its remaining free space.

The default thresholds are 70, 85, and 95 percent. If you specify only one or two values, unspecified values default to 100 percent. You can specify the THRESHOLDS option only for a storage area for a multifile database. The storage area page format must be MIXED.

INTERVAL IS number-data-pages

The number of data pages between SPAM pages in the physical storage area file, and thus the maximum number of data pages each SPAM page will manage. The default, and also the minimum interval, is 256 data pages. The first page of each storage area is a SPAM page. The interval you specify determines where subsequent SPAM pages are to be inserted, provided there are enough data pages in the storage file to require more SPAM pages.

The maximum SPAM interval you can specify is 4008 data pages for a database with a 2-block page size. Specifying a value greater than 4008 data pages for the SPAM interval causes data corruption.

CHANGE DATABASE Statement

You can specify the INTERVAL option only for a storage area for a multifile database. The storage area page format must be MIXED.

See the *VAX Rdb/VMS Guide to Database Maintenance and Performance* for information on formulas for calculating maximum and minimum SPAM intervals.

SNAPSHOT_FILENAME IS file-spec

Provides a separate file specification for the snapshot file. Do not specify a file type other than SNP. Rdb/VMS will assign the extension SNP to the file specification, even if you specify an alternate extension. If you do not specify this clause for a storage area, the SNP file is automatically placed in the same directory as the storage area file and has the same name as the storage area file. You cannot specify a global default for the SNAPSHOT_FILENAME option. Thus, in a multifile database, the SNAPSHOT_FILENAME option must be within a DEFINE STORAGE AREA clause.

READ_ONLY

READ_WRITE

Using the change-storage-area-clause, you can change any read/write storage area except the RDB\$SYSTEM storage area to read-only, or change any read-only storage area except the RDB\$SYSTEM storage area to read/write. (To change the RDB\$SYSTEM storage area, use the READ_ONLY or READ_WRITE keywords of the change-database-options clause described earlier in this arguments list.)

Select the READ_ONLY option to change a read/write storage area to read-only. Select the READ_WRITE option to change a read-only storage area to read/write.

Rdb/VMS provides support for both read-only databases and databases with one or more read-only storage areas. You can take advantage of this read-only support if you have a stable body of data that is never (or rarely) updated.

Read-only databases consist of:

- A read/write root file
- One or more read-only storage areas and no read/write storage areas

Read-only databases can be published and distributed on CDROM media.

CHANGE DATABASE Statement

Read-only storage areas offer the following benefits:

- Minimize locking problems in the read-only storage areas
- Are not backed up by RMU/BACKUP unless explicitly instructed (thus decreasing backup time for large areas of data that do not change)
- Are not restored by RMU/RESTORE unless explicitly instructed
- Are not recovered by RMU/RECOVER
- Eliminate page and record locking

You normally change a storage area from read/write to read-only. You might, however, change a read-only storage area back to read/write to facilitate batch updates to infrequently changed data.

A database with both read/write and read-only storage areas can be fully recovered after a system failure *only* if after-image journaling is enabled on the database. If your database has both read/write and read-only storage areas but does not have after-image journaling enabled, you should do full backups (including read-only areas) at all times. Doing full backups enables you to recover the entire database to its condition at the time of the previous backup.

Note Read-only storage areas on CDROM must be remastered with each upgrade of Rdb/VMS. The remastering is done with software provided by the CDROM publisher.

You cannot change the parameters of a read-only storage area. You must first change the storage area to a read/write area, then change the parameters.

Also, no write operation except a cardinality update can be done in a read-only storage area. See Section 6.5 for more information and restrictions on updating cardinalities.

For a complete description of read-only databases and read-only storage areas, see the *VAX Rdb/VMS Guide to Database Maintenance and Performance*.

DELETE STORAGE AREA *storage-area-name*

Deletes a named storage area and its associated snapshot file. You cannot delete the RDB\$SYSTEM storage area. You cannot delete a storage area if a storage map refers to it, or if data is in it.

CHANGE DATABASE Statement

Usage Notes

You need the Rdb/VMS ADMINISTRATOR privilege to use the CHANGE DATABASE statement.

No users can be attached to the database when you issue a CHANGE DATABASE statement. Do not invoke the database before you issue a CHANGE DATABASE statement. If you invoke the database before you issue a CHANGE DATABASE statement, you will get a file access conflict error. The exception is when *only* the OPEN IS MANUAL or OPEN IS AUTOMATIC option is specified. You can specify either OPEN IS option when you have invoked the database, or when other users are attached to the database.

A table indicating default, minimum, and maximum values for database-wide parameters can be found in Appendix D. The same values for storage area parameters can be found in Appendix E.

The CHANGE DATABASE statement does not leave the database invoked.

Note that in the CHANGE DATABASE statement, you cannot specify global defaults for storage areas by placing the storage area options outside the DEFINE STORAGE AREA clause or the CHANGE STORAGE AREA clause. The exception to this is if you place the SNAPSHOT IS ENABLED/DISABLED option outside the DEFINE STORAGE AREA or CHANGE STORAGE AREA clause. When you enable or disable snapshots, you do so for *all* storage areas. You cannot enable or disable snapshots for individual storage areas.

The order of precedence for storage area options for new storage areas defined with the DEFINE STORAGE AREA clause is:

- 1 Local use of the option, within a DEFINE STORAGE AREA clause
- 2 The Rdb/VMS default for that option

You cannot specify the RDB\$SYSTEM storage area in the CHANGE STORAGE AREA clause. To change the EXTENT, SNAPSHOT ALLOCATION, or SNAPSHOT EXTENT options for RDB\$SYSTEM, specify these qualifiers outside the CHANGE STORAGE AREA clause. When you specify these qualifiers outside the CHANGE STORAGE AREA clause, as a part of the change-database-options, these parameters will not be global defaults for any other storage area. However, they will change the RDB\$SYSTEM storage area.

CHANGE DATABASE Statement

You can add a storage area to a multifile database only. You cannot define a storage area for a single-file database with the CHANGE DATABASE statement. To restructure a single-file database into a multifile database, use the RDO EXPORT and IMPORT statements.

All options of the CHANGE DATABASE statement require ADMINISTRATOR and CHANGE privileges at the database level. However, any user with VMS SYSPRV or BYPASS privilege bypasses the database access control lists (ACLs) and can perform any task that otherwise can be controlled by the ACL.

If you use the SNAPSHOT IS ENABLED clause to enable snapshots on a multifile database, writing to all snapshot files for all storage areas (including RDB\$SYSTEM) is enabled. If you use the SNAPSHOT IS DISABLED clause, writing to all snapshot files is disabled.

Most changes that you can make with the CHANGE DATABASE statement are recoverable, because most metadata changes are journaled in the RUJ file and (if present) the AIJ file. However, certain changes are not journaled, and therefore you should back up the database before doing any of the following operations:

- Changing the number of users
- Changing the number of nodes
- Changing the AIJ file
- Adding or deleting a storage area

If an error occurs during any of these operations, the database may be corrupt; backing up the database before these operations enables you to restore it if it becomes corrupt.

The SNAPSHOT ALLOCATION clause of the CHANGE DATABASE statement does not allow you to increase the size of a snapshot file beyond the file's current size. If you specify a size greater than the current size with the SNAPSHOT ALLOCATION clause, the command executes without an error or warning message, but the size of the snapshot file is not increased.

You cannot delete a storage area that is referred to in a storage map.

See the *VAX Rdb/VMS Guide to Database Maintenance and Performance* for a complete discussion of when to use the IMPORT, EXPORT, and CHANGE DATABASE statements.

CHANGE DATABASE Statement

Examples

Example 1

The following example uses the CHANGE DATABASE statement to enable snapshots with the default IMMEDIATE option for a database that previously had snapshots disabled:

```
$ RDO
RDO> CHANGE DATABASE FILENAME 'DISK2:[USER.TEST]PERSONNEL'
cont>     SNAPSHOT IS ENABLED IMMEDIATE.
```

Example 2

Assume the snapshot file has grown in size. This example uses the SNAPSHOT ALLOCATION option to truncate the current snapshot file.

```
$ RDO
RDO> CHANGE DATABASE FILENAME 'DISK2:[USER.TEST]PERSONNEL'
cont>     SNAPSHOT ALLOCATION IS 200.
```

Example 3

Use the CHANGE DATABASE statement to enable after-image journaling and include a journal file specification; the AIJ file is placed on a different disk from the RDB file.

```
$ RDO
RDO> CHANGE DATABASE FILENAME '$222$DUA12:[TOP]PERSONNEL'
cont>     JOURNAL FILE IS '$111$DUA3:[RDB]PERSONNEL.AIJ'.
```

Example 4

You can also use the CHANGE DATABASE statement to discontinue after-image journaling.

```
$ RDO
RDO> CHANGE DATABASE FILENAME 'PERSONNEL'
cont>     NOJOURNAL.
```

This statement instructs Rdb/VMS to turn off after-image journaling.

Example 5

The database shutdown feature is designed to ensure that active users on a single node or across a VAXcluster cannot access the database while you perform administrative and maintenance tasks. For instance, use this feature when you need the database in a stable condition while performing backup and restore operations, or when you are tuning the database.

CHANGE DATABASE Statement

In the following example, a complete database shutdown is performed for all users of a database that resides on a common disk in a VAXcluster. The CHANGE DATABASE . . . OPEN IS MANUAL statement is used to prevent new users from automatically opening that database when the first data manipulation language statement is executed on the database.

```
$ RDO = "$RDO"
$ RDO CHANGE DATABASE FILENAME "$222$DUA17:[DBS]PERS" OPEN IS MANUAL.
$ REPLY/ALL "PERS.RDB database will be shut down in 15 minutes"
$ REPLY/ALL "Please complete any work with PERS before then"
$ WAIT 00:15:00.00
$ RMU/CLOSE/ABORT=FORCEX/CLUSTER $222$DUA17:[DBS]PERS
```

At this point, you are assured that no unprivileged users can invoke this database and you can perform maintenance tasks (like tuning). For example, you can add or delete indexes and compare the performance results in a stable testing environment.

When you are ready to make the database available again across the cluster, enter RDO and type:

```
RDO> CHANGE DATABASE FILENAME "$222$DUA17:[DBS]PERS"
cont> OPEN IS AUTOMATIC.
```

Example 6

The following example shuts down access to a clusterwide database and then reopens the database for a specific node:

```
$ RDO = "$RDO"
$ RDO CHANGE DATABASE FILENAME "$222$DUA17:[DBS]PERS" OPEN IS MANUAL.
$ REPLY/ALL "PERS.RDB database will be shut down in 15 minutes"
$ REPLY/ALL "Please complete any work with PERS before then"
$ WAIT 00:15:00.00
$ RMU/CLOSE/ABORT=FORCEX/CLUSTER $222$DUA17:[DBS]PERS
$ !
$ SET HOST MADABT
.
.
.
$ ! On node MADABT now
$ RMU/OPEN $222$DUA17:[DBS]PERS
```

The RMU/OPEN command opens the PERS database on MADABT only. Users logged in to other nodes in the VAXcluster who normally have access to the clusterwide database cannot access it until:

- The RMU/OPEN command is executed by a user with sufficient privilege on his or her node.

CHANGE DATABASE Statement

- Or, the CHANGE DATABASE . . . OPEN IS AUTOMATIC statement is executed by a user with sufficient privilege from any node in the VAXcluster.

Example 7

The following example demonstrates that the CHANGE DATABASE statement cannot be used when the database file is open or invoked:

```
RDO> ! Invoke the database:
RDO> INVOKE DATABASE FILENAME "WORK$DISK:PERSONNEL"
RDO> !
RDO> ! Attempt to change the database. This will fail:
RDO> !
RDO> CHANGE DATABASE FILENAME "WORK$DISK:PERSONNEL"
cont> JOURNAL FILE IS "AIJ$DISK:PERS.AIJ".
%RDO-W-NOCDDUPDAT, database invoked by filename, the CDD will not be updated
%RDB-F-SYS_REQUEST, error from system services request
-RDMS-F-FILACCERR, error opening database root file US:[PER]PERSONNEL.RDB;1
-SYSTEM-W-ACCONFLICT, file access conflict
RDO> !
RDO> ! Detach from the database:
RDO> FINISH
RDO> !
RDO> ! Now perform the change:
RDO> CHANGE DATABASE FILENAME "WORK$DISK:PERSONNEL"
cont> JOURNAL FILE IS "AIJ$DISK:PERS.AIJ".
RDO> !
RDO> ! Explicitly open the database:
RDO> OPEN FILENAME "WORK$DISK:PERSONNEL"
RDO> !
RDO> ! Try to change the database (this will fail):
RDO> CHANGE DATABASE FILENAME "WORK$DISK:PERSONNEL" NOJOURNAL.
%RDB-F-SYS_REQUEST, error from system services request
-RDMS-F-FILACCERR, error opening database root file US:[PER]PERSONNEL.RDB;1
-SYSTEM-W-ACCONFLICT, file access conflict
RDO> !
RDO> ! Using FINISH and reentering the CHANGE statement would work.
RDO> ! But use CLOSE, which will shut down the database:
RDO> CLOSE FILENAME "WORK$DISK:PERSONNEL" WITH EXIT FOR NODE
RDO> !
RDO> CHANGE DATABASE FILENAME "WORK$DISK:PERSONNEL" NOJOURNAL.
RDO>
```

CHANGE DATABASE Statement

Example 8

The following example uses the CHANGE DATABASE statement to define a new storage area for a multifile database:

```
RDO> CHANGE DATABASE FILENAME 'MF_PERSONNEL'  
cont> DEFINE STORAGE AREA ARCHIVED_EMPS  
cont>     FILENAME DISK3:ARCHIVED_EMPS  
cont>     ALLOCATION IS 50 PAGES  
cont>     PAGE FORMAT IS MIXED  
cont>     SNAPSHOT_FILENAME IS DISK4:ARCHIVED_EMPS  
cont> END ARCHIVED_EMPS STORAGE AREA.
```

Example 9

This example uses the **DICTIONARY IS REQUIRED** option to enforce use of the data dictionary if metadata updates occur. Users must invoke the database with the **PATHNAME** qualifier to perform any metadata changes.

Note that when you specify the **DICTIONARY** option, that is the only option you can specify in a **CHANGE DATABASE** statement. To specify other options, you must issue another **CHANGE DATABASE** statement.

```
RDO> CHANGE DATABASE FILENAME 'PERSONNEL'  
cont>     DICTIONARY IS REQUIRED.
```

Example 10

This example uses the **READ_ONLY** clause to change the **ARCHIVED_EMPLOYEES** storage area to a read-only storage area:

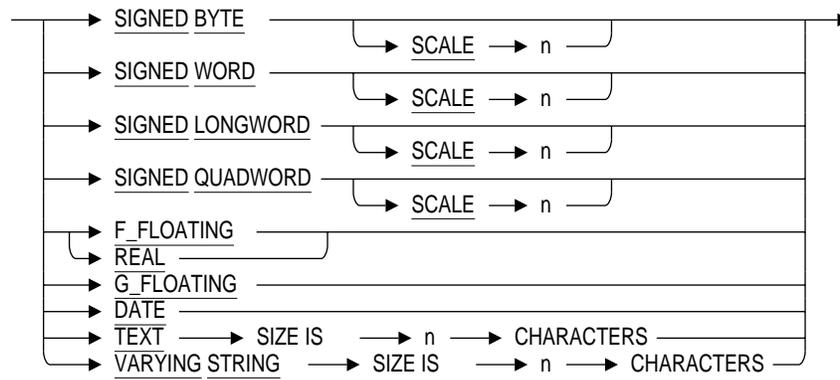
```
RDO> CHANGE DATABASE FILENAME 'MF_PERSONNEL'  
cont> CHANGE STORAGE AREA ARCHIVED_EMPLOYEES  
cont>     READ_ONLY.
```


CHANGE FIELD Statement

validity-clause =

→ VALID IF → conditional-expr →

data-type =

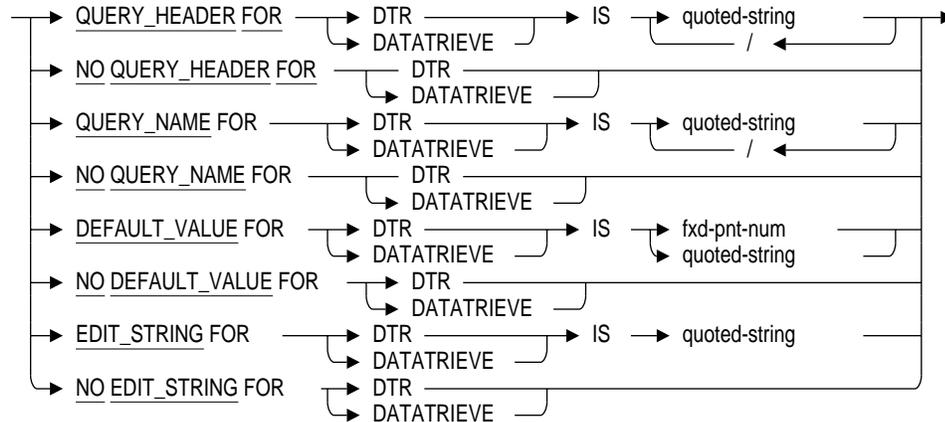


missing-value-clause =

→ MISSING_VALUE IS → fxd-pnt-num
 → NO MISSING_VALUE → quoted-string

CHANGE FIELD Statement

dtr-clause =



Arguments

name

The name of the field you want to change.

text

A text string that adds a comment to the field definition being changed.

field-attributes

A list of definitions that indicate what type of data you can store in the field and how Rdb/VMS uses that data. See Chapter 5 for a complete description of field attributes. If you do not change a field attribute in the `CHANGE FIELD` statement, the attribute remains the same.

Usage Notes

To change a field using the `CHANGE FIELD` statement, you need the Rdb/VMS `CHANGE` privilege for the field.

CHANGE FIELD Statement

You can change the following field attributes:

- Data type

When you change a data type specification, Rdb/VMS automatically converts all the data in all the fields that use the global field definition the first time it is accessed. Normally, Rdb/VMS converts any data type to any other data type. However, some precision may be lost in the process of conversion. For example, if you change from G-floating to F-floating data, Rdb/VMS rounds values. If you change back to G-floating, the rounded data is converted. The new values, therefore, differ from the original values.

If you change the data type of a field, pay attention to the data stored in the database. If you convert a data type to one with a smaller capacity (LONGWORD to WORD), first check for any values in the database that will overflow the new data type. If there are values larger than the new data type can hold, you will not be able to access the data.

You can change fields with QUADWORD data types to those with LONGWORD data types.

You can only store date fields into a DATE data type or a TEXT data type. Rdb/VMS restricts conversion of a field with a DATE data type. You can only change the DATE data type to the TEXT data type or VARYING STRING data type.

In addition, in order to change a DATE data type to a VARYING STRING data type and back to DATE, you must first change it to a TEXT data type, as in the following example:

```
RDO> INVOKE DATABASE FILENAME MF_PERSONNEL
RDO> CHANGE FIELD DATE_DOM DATATYPE IS TEXT SIZE IS 23.
RDO> CHANGE FIELD DATE_DOM DATATYPE IS VARYING STRING SIZE IS 23.
RDO> CHANGE FIELD DATE_DOM DATATYPE IS TEXT SIZE IS 23.
RDO> CHANGE FIELD DATE_DOM DATATYPE IS DATE.
```

You cannot issue a CHANGE FIELD statement to change the data type for a field used in an index or view definition. Rdb/VMS returns an error message in these instances and does not change the field. To change the data type for a field used in an index or view, first delete the index or view definition, then change the field, and finally, redefine the index or view.

CHANGE FIELD Statement

- MISSING_VALUE

Be careful about using the CHANGE FIELD statement to change the missing value for a global field. If you use the CHANGE FIELD statement to change the missing value for a global field, do the following afterwards:

- Delete any triggers that set as "missing" any field based on that global field; then define these triggers again.
- Recompile and relink any programs that set as "missing" any field based on that global field.

Rdb/VMS must re-evaluate the RDB\$MISSING() function so that the triggers and programs use the newly defined missing value. Otherwise, the old missing value will be assigned and the MISSING operator will not select those records.

In summary, do not change the missing value for a field *unless you fully understand what you are doing*.

- DATATRIEVE options

- COLLATING_SEQUENCE and NO COLLATING_SEQUENCE

You cannot issue a CHANGE FIELD statement to change the collating sequence for a field used in an index or view definition. Rdb/VMS returns an error message in these instances and does not change the field. To change the collating sequence for a field used in an index or view, first delete the index or view definition, then change the field, and finally, redefine the index or view.

- VALID IF

You can add or change a VALID IF clause with the CHANGE FIELD statement. However, if the database already contains data that violates the VALID IF clause, the VALID IF clause is rejected. Example 3 provides an illustration.

If the database is created with the DICTONARY IS REQUIRED option, you must invoke the database by path name, rather than file name, before you issue this statement.

You cannot change the attributes of a segmented string field.

If you change a field definition and terminate your transaction, the change will be visible to other users in transactions that follow.

CHANGE FIELD Statement

Digital Equipment Corporation recommends that you invoke the database by specifying the dictionary path name rather than the file name when you make changes to any data definitions. In this way, you can ensure that all programs and procedures refer to the correct and current database definitions.

Depending on the type of change you make, you may have to reprocess and relink application programs that refer to the changed fields.

You can change a field definition only if you have invoked the database that includes the field definition. You must execute the CHANGE FIELD statement in a read/write transaction. If there is no active transaction and you issue this statement, Rdb/VMS starts a read/write transaction implicitly.

Other users are allowed to be attached to the database when you issue the CHANGE FIELD statement.

You cannot execute the CHANGE FIELD statement when the RDB\$SYSTEM storage area is set to read-only. You must first set RDB\$SYSTEM to read/write. Use the READ_WRITE option of the CHANGE DATABASE statement's change-database-options2 clause to set RDB\$SYSTEM to read/write. See the *VAX Rdb/VMS Guide to Database Maintenance and Performance* for more information on the RDB\$SYSTEM storage area.

Examples

Example 1

The following example expands the POSTAL_CODE field to nine characters and adds DATATRIEVE support characteristics:

```
RDO> INVOKE DATABASE PATHNAME 'MF_PERSONNEL'
RDO> SHOW FIELDS POSTAL_CODE
      POSTAL_CODE                text size is 5
Description:      Postal code (in US = ZIP)
Missing value:
RDO> CHANGE FIELD POSTAL_CODE
cont>   DATATYPE IS TEXT SIZE IS 9 CHARACTERS
cont>   DEFAULT_VALUE FOR DTR IS "000000000"
cont>   EDIT_STRING FOR DTR IS "XXXXX-XXXX".
RDO> SHOW FIELDS POSTAL_CODE
      POSTAL_CODE                text size is 9
Description:      Postal code (in US = ZIP)
Edit string:      xxxxx-xxxx
Default value:    000000000
Missing value:
```

CHANGE FIELD Statement

```
RDO> FOR E IN EMPLOYEES
cont>   PRINT E.POSTAL_CODE
cont>   END_FOR
03817
03817
03301
03456
.
.
03301
03809
RDO> STORE E IN EMPLOYEES USING
cont> E.LAST_NAME = "Forester";
cont> E.EMPLOYEE_ID = "00876";
cont> E.POSTAL_CODE = "039875573"
cont> END_STORE
RDO> FOR E IN EMPLOYEES
cont>   WITH E.LAST_NAME = "Forester" OR
cont>   E.LAST_NAME = "Toliver"
cont> PRINT
cont>   E.EMPLOYEE_ID,
cont>   E.POSTAL_CODE
cont> END_FOR
00164  03817
00876  039875573
```

The field remains a text field, but the length is increased to nine characters, and an edit string and default value are specified for DATATRIEVE. When you display the data, existing field values are padded on the right with spaces. Thus an existing postal code would appear as "03104 ". Newly stored values can have nine characters.

Example 2

The accounting department has decided that the BUDGET field will now include pennies:

```
RDO> SHOW FIELDS BUDGET
      BUDGET                                signed longword scale  0
      Description:          Generic budget data
      Edit string:          $$$,$$$,$$$
RDO> FOR D IN DEPARTMENTS
cont> WITH D.DEPARTMENT_NAME = 'Manufacturing'
cont> PRINT D.BUDGET_ACTUAL
cont> END_FOR
      BUDGET_ACTUAL
              0
```

CHANGE FIELD Statement

```
RDO> CHANGE FIELD BUDGET
cont> DATATYPE IS SIGNED LONGWORD SCALE -2
cont> EDIT_STRING FOR DTR IS "$$$$, $$9.99".
RDO> SHOW FIELDS BUDGET
      BUDGET                                signed longword scale -2
      Description:          Generic budget data
      Edit string:          $$$$, $$9.99
RDO> FOR D IN DEPARTMENTS
cont> WITH D.DEPARTMENT_NAME = 'Manufacturing'
cont> PRINT D.BUDGET_ACTUAL
cont> END_FOR
      BUDGET_ACTUAL
      0.00
```

Example 3

You can add or change a VALID IF clause for a field. However, if the database contains any data that violates the specification of the VALID IF clause, the clause is rejected. The following example shows two CHANGE FIELD . . . VALID IF statements. In the first instance, the specification is rejected because existing records contain EMPLOYEE_ID values less than "98765"; in the second instance, the specification is accepted.

```
RDO> CHANGE FIELD ID_NUMBER VALID IF ID_NUMBER > "98765".
%RDB-E-NO_META_UPDATE, metadata update failed
-RDMS-E-NOT_VALID_FR, field ID in relation CURRENT_INFO fails validation
RDO> CHANGE FIELD ID_NUMBER VALID IF ID_NUMBER > "00050".
RDO> SHOW FIELD ID_NUMBER
      ID_NUMBER                                text size is 5
      Description:          Generic employee ID
      Missing value:
      Valid:                IF ID_NUMBER > "00050"
```

Example 4

The following example defines a field with the predefined NCS collating sequence FRENCH. Note that you must first execute the DEFINE COLLATING_SEQUENCE statement.

The example then changes the collating sequence to Finnish.

```
RDO> INVOKE DATABASE FILENAME 'MF_PERSONNEL'
RDO> DEFINE COLLATING_SEQUENCE FRENCH FRENCH.
%RDO-W-NO_CDDUPDAT, database invoked by filename, the CDD will not be
updated
RDO> DEFINE FIELD LAST_NAME_CHANGE_TEST
cont> DATATYPE IS TEXT SIZE IS 12 CHARACTERS
cont> COLLATING_SEQUENCE IS FRENCH.
RDO> SHOW FIELD LAST_NAME_CHANGE_TEST
      LAST_NAME_CHANGE_TEST                    text size is 12
                                              Collating Sequence FRENCH
```

CHANGE FIELD Statement

```
RDO> !
RDO> ! Now, change the collating sequece to Finnish. You must first
RDO> ! execute the DEFINE COLLATING_SEQUENCE statement.
RDO> !
RDO> DEFINE COLLATING_SEQUENCE FINNISH FINNISH.
RDO> CHANGE FIELD LAST_NAME_CHANGE_TEST
cont> DATATYPE IS TEXT SIZE IS 12 CHARACTERS
cont> COLLATING_SEQUENCE IS FINNISH.
RDO> !
RDO> SHOW FIELD LAST_NAME_CHANGE_TEST
      LAST_NAME_CHANGE_TEST          text size is 12
                                      Collating Sequence FINNISH
```

CHANGE INDEX Statement

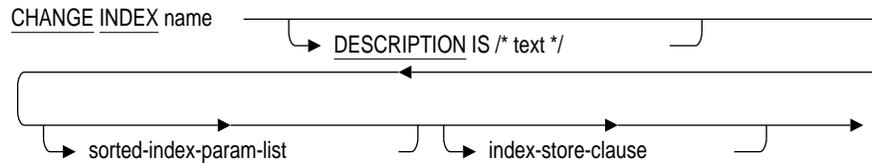
9.4 CHANGE INDEX Statement

Changes an index. The CHANGE INDEX statement allows you to change the following:

- The description text
- The physical characteristics of sorted index nodes:
 - Index node size
 - Initial fullness percentage
 - Index usage mode
- The storage map associated with the index

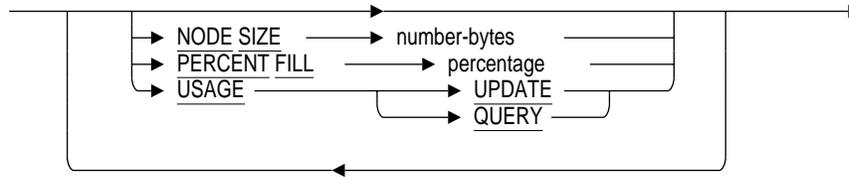
The CHANGE INDEX statement reads the data in the current index structure, creates a new index structure using the parameters you specify, and then deletes the old index structure. This is faster than using the DELETE INDEX statement to delete the index, and then the DEFINE INDEX statement to define a new index, because no sort operation is needed with the CHANGE INDEX statement.

Format

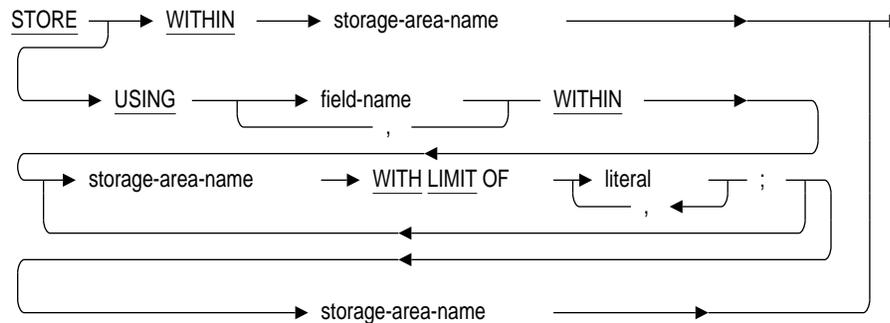


CHANGE INDEX Statement

sorted-index-param-list =



index-store-clause



Arguments

name

The name of the index you want to change.

text

A text string that adds a comment to the index definition.

NODE SIZE number-bytes

The size of each index node. The number and level of the resulting index nodes depend on this value, the number and size of the index keys, and the value of the PERCENT FILL clause. If you omit this clause in a CHANGE INDEX statement, the existing value is used.

CHANGE INDEX Statement

See the *VAX Rdb/VMS Guide to Database Maintenance and Performance* for information on estimating the valid range for a user-specified index node size.

PERCENT FILL percentage

Sets the *initial* fullness percentage for each node in the index structure being changed. The valid range is 1 percent to 100 percent. If you omit this clause in a CHANGE INDEX statement, the existing value is used. If the PERCENT FILL and USAGE clauses are both specified in the same CHANGE INDEX statement, the USAGE value is used.

USAGE UPDATE

Sets the initial fullness percentage of each index node at 70 percent. Supplying PERCENT FILL and USAGE UPDATE clauses is allowed in the syntax; however, the USAGE option takes precedence over an explicit PERCENT FILL value.

USAGE QUERY

Sets the initial fullness percentage value at 100 percent. Supplying PERCENT FILL and USAGE QUERY clauses is allowed in the syntax; however, the USAGE option takes precedence over an explicit PERCENT FILL value.

index-store-clause

A storage map definition for the index. You can specify a STORE clause for indexes in a multifile database only. The STORE clause in a CHANGE INDEX statement allows you to specify which storage area files will be used to store the index entries:

- All index entries can be associated with a single storage area.
- Index entries can be systematically distributed, or **partitioned**, among several storage areas by specifying upper limits on the values for a key in a particular storage area.

If you omit the storage map definition, the default is to store all the entries for an index in the main RDBSYSTEM storage area.

You should define a storage area for an index that matches the storage map for the relation with which it is associated.

USING field-name

The name of the field whose value will be used as a limit for partitioning the index across multiple storage areas. You must respecify this same field later in the index definition as one of the fields that makes up the index.

CHANGE INDEX Statement

If the index key is multisegmented, you can include some or all of the fields that are joined to form the index key. If you include only a subset of the fields from the multisegmented index, you must specify the fields in the order in which they were specified when the index key was defined. For example, the multisegmented index EMP_FULL_NAME consists of the fields LAST_NAME, FIRST_NAME, and MIDDLE_INITIAL. In the USING clause, you can specify the following combinations of fields:

- LAST_NAME
- LAST_NAME and FIRST_NAME
- LAST_NAME, FIRST_NAME, and MIDDLE_INITIAL

You cannot specify only the FIRST_NAME field, or the FIRST_NAME and MIDDLE_INITIAL fields.

Separate multiple field names with commas.

WITHIN storage-area-name

The name of the storage area in which you want the index stored. You must define this storage area with the DEFINE DATABASE statement before you refer to it in the index-store-clause.

If the index is a hashed index, the storage area must have a MIXED page format.

WITH LIMIT OF literal

The maximum value for the index key that will reside in the specified storage area. For multisegmented index keys, specify a literal value for each field.

The number of literals in this clause must be less than or equal to the number of fields in the USING clause. Repeat this clause to partition the index entries among multiple storage areas.

When you are modifying a multisegmented index using multisegmented keys and use the STORE USING . . . WITH LIMITS clause, if the values for the first key are all the same, then set the limit for the first key at that value. By doing this, you ensure that the value of the second key determines the storage area in which each record will be stored.

Note that the last storage area you specify *cannot* have a WITH LIMIT OF clause associated with it.

CHANGE INDEX Statement

Usage Notes

To change an index with the CHANGE INDEX statement, you need the Rdb/VMS CHANGE privilege for the relation.

When the CHANGE INDEX statement executes, Rdb/VMS changes the index definition in the physical database. If you have invoked the database by path name, the definition is also changed in the data dictionary.

If the database is created with the DICTONARY IS REQUIRED option, you must invoke the database by path name, rather than file name, before you issue this statement.

You *cannot* do the following three things with the CHANGE INDEX statement:

- Modify the field or fields used as keys in the index
- Change a SORTED index to a HASHED index, or a HASHED index to a SORTED index
- Change the duplicates clause for an index

You must execute the CHANGE INDEX statement in a read/write transaction. If you issue this statement when there is no active transaction, Rdb/VMS starts a read/write transaction implicitly.

You can change an index when there are other active users of the database. However, attempts to change an index will fail if that index is involved in a query at the same time. Users will have to detach from the database with a FINISH statement before you can change the index. When Rdb/VMS first accesses an object, such as the table, a lock is taken out on that object and not released until the user exits the database. Attempts to update this object will get a LOCK CONFLICT ON CLIENT message due to the other user's optimized access to the object.

Similarly, while you change an index, users cannot execute queries involving that index until you have completed the transaction with a COMMIT or ROLLBACK statement for the CHANGE statement. The user will receive a LOCK CONFLICT ON CLIENT error message. While DDL operations are performed, normal data locking mechanisms are used against system relations. (System relations hold information on objects in the database.) Therefore, attempts to update an object will lock out attempts to query that object. These locks are held until the COMMIT or ROLLBACK of the DDL operation.

CHANGE INDEX Statement

The WAIT/NOWAIT clause of the START_TRANSACTION statement does not affect attempts to update metadata with simultaneous queries. Even if you specify START_TRANSACTION WAIT for the metadata update transaction, you will get the following error message if a lock conflict exists:

```
%RDB-E-LOCK_CONFLICT, request failed due to locked resource; no-wait
parameter specified for transaction
-RDB-E-NO_META_UPDATE, metadata update failed
-RDMS-F-LCKCNFLCT, lock conflict on client
RDO>
```

However, a user's query will wait for a metadata update to complete with a ROLLBACK or COMMIT, even if the user specified NOWAIT on the START_TRANSACTION statement.

You cannot execute the CHANGE INDEX statement when the RDB\$SYSTEM storage area is set to read-only. You must first set RDB\$SYSTEM to read/write. See the *VAX Rdb/VMS Guide to Database Maintenance and Performance* for more information on the RDB\$SYSTEM storage area.

If you omit the STORE clause in the CHANGE INDEX statement, you can create by mistake a second index that stores data in the default area RDB\$SYSTEM.

Examples

Example 1

The following example changes the index node size to 100 bytes, and changes the initial fullness percentage from 70 percent to 95 percent:

```
RDO> CHANGE INDEX JH_EMPLOYEE_ID
cont>  NODE SIZE 100
cont>  PERCENT FILL 95.
```

Note that JH_EMPLOYEE_ID is a sorted index. You cannot change node size, percent fill, or the USAGE clause for a hashed index.

CHANGE INDEX Statement

Example 2

The following example changes how the index is stored by specifying a new index-store-clause for the index:

```
RDO>      CHANGE INDEX EMPLOYEES_HASH
cont>      DESCRIPTION IS /* Hashed index for employees */
cont>      STORE USING EMPLOYEE_ID
cont>      WITHIN
cont>      EMPIDS_LOW WITH LIMIT OF "00400";
cont>      EMPIDS_MID WITH LIMIT OF "00800";
cont>      EMPIDS_OVER.
```

Example 3

The following example changes the size of each index node and sets the initial fullness percentage for each node in the index structure being changed. It also specifies a storage map definition for the index.

```
RDO>      CHANGE INDEX COLL_COLLEGE_CODE
cont>      NODE SIZE 1250 PERCENT FILL 100
cont>      STORE WITHIN EMP_INFO.
```

CHANGE PROTECTION Statement

9.5 CHANGE PROTECTION Statement

Changes protection (access rights) for a single entry within the specified access control list.

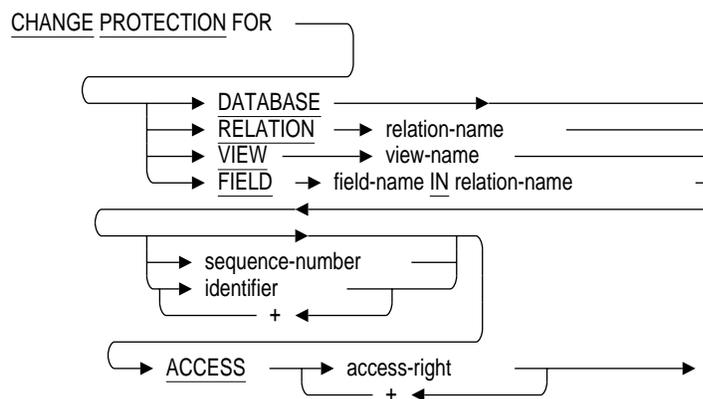
An access control list (ACL) is attached to each database and relation. Each list consists of entries that specify two items of information:

- An identifier that specifies a user or set of users.
- A set of access rights. These rights specify what operations the user or set of users can perform on the database or relation.

For a particular user, Rdb/VMS grants an access right to a relation only if that right is granted in the ACL for *both* the database and the relation. For a more complete explanation of access control lists, see Section 9.17 and the chapter on defining database protection in the *VAX Rdb/VMS Guide to Database Design and Definition*.

The new version of the ACL entry you create with the CHANGE PROTECTION statement inherits the characteristics of the old version. When you change protection on a database element, you must specify only the access rights you want to grant or deny.

Format



CHANGE PROTECTION Statement

Defaults

sequence-number

Rdb/VMS changes the first access control list entry in the specified list, if you do not specify an identifier.

ACCESS access-right

If you do not mention a particular access right, Rdb/VMS uses the previous version of the ACL entry to grant or deny that right. That is, the new ACL entry inherits the access rights that you do not deny explicitly.

Arguments

RELATION relation-name

Name of the relation for which you want to change the access control list.

VIEW view-name

Name of the view for which you want to change the access control list.

FIELD field-name IN relation-name

The name of the local field in a specified relation for which you want to change an ACL entry.

sequence-number

A number that identifies the entry within the specified access control list whose protection you want to change. The default is one.

If you specify a sequence number larger than the largest existing sequence number, Rdb/VMS returns an error message.

identifier

Any valid VMS identifier in the identifier clause:

- UIC identifiers

A VMS user identification code (UIC) that identifies an entry within the ACL. A UIC has the form:

[g,m]

where *g* refers to a group number and *m* is the number of a member of that group. You can replace either or both with the wildcard character (*) to specify a particular member in all groups, all members in a particular group, or all members in all groups.

CHANGE PROTECTION Statement

You cannot specify more than one UIC identifier in a CHANGE PROTECTION statement.

- General identifiers

General identifiers are defined by the VMS system manager in the system rights database to identify groups of users on the system.

- System-defined identifiers

System-defined identifiers are automatically defined by the system when the rights database is created at system installation time.

Identifiers are assigned depending on the type of login you execute.

For more information about these types of identifiers, see Section 9.17.

ACCESS access-right

An access right to be granted or denied to the user identified by a UIC. The new version of the ACL entry you create with the CHANGE PROTECTION statement does not inherit any characteristics from the old version. When you change protection on a database element, you need to specify the entire entry, including all the access rights you want to deny.

If you specify two or more access rights, separate each with a plus sign (+), but do not include any spaces.

If the list of access rights exceeds one line in length, place the list in quotation marks and use the continuation character (hyphen). Otherwise, Rdb/VMS reads the carriage return as the end of the list and an error results:

```
cont> ACCESS "DEFINE+CHANGE+DELETE -  
cont> +CONTROL+OPERATOR+ADMINISTRATOR".
```

See Table 9-3, Table 9-4, Table 9-5, and Table 9-6 in the DEFINE PROTECTION section for a complete list of Rdb/VMS access rights.

Usage Notes

You must have the CONTROL privilege to modify the access rights of other users with the CHANGE PROTECTION statement.

You must execute the CHANGE PROTECTION statement in a read/write transaction. If you issue this statement when there is no active transaction, Rdb/VMS starts a read/write transaction implicitly.

Rights on a field are determined by the rights defined on the field combined with those specified for the specific relation ACL.

CHANGE PROTECTION Statement

A user with MODIFY rights on the relation automatically gets the same rights on all fields in the relation. However, you can restrict MODIFY rights by defining them only on specific fields you want users to be able to modify and thus remove the right from the relation entry.

You can modify the data in a field with only MODIFY on the field and READ on the database.

Other users are allowed to be attached to the database when you issue the CHANGE PROTECTION statement.

Granting or revoking a privilege takes effect after you exit from the database.

You cannot execute the CHANGE PROTECTION statement when the RDB\$SYSTEM storage area is set to read-only. You must first set RDB\$SYSTEM to read/write. See the *VAX Rdb/VMS Guide to Database Maintenance and Performance* for more information on the RDB\$SYSTEM storage area.

Examples

Example 1

You can change the protection in an access control list entry identified by a UIC.

```
RDO> CHANGE PROTECTION FOR DATABASE
cont> [STAFF,SAMSON]
cont> ACCESS NOCONTROL+NOOPERATOR+NOADMINISTRATOR.
```

This statement performs the following actions:

- Identifies the user whose protection you want to change. Here, the user is identified by the UIC [STAFF,SAMSON].
- Changes the access rights to deny the user CONTROL, OPERATOR, and ADMINISTRATOR privileges. All other privileges remain as you defined them in the previous version of this ACL entry.

Example 2

You can identify the ACL entry by a sequence number.

```
RDO> CHANGE PROTECTION FOR DATABASE
cont> 4
cont> ACCESS WRITE+MODIFY+ERASE.
```

CHANGE PROTECTION Statement

This statement performs the following actions:

- Identifies the entry by specifying a sequence number 4. This means that this statement will change the fourth entry in the access control list for the PERSONNEL database.
- Changes the access rights by granting WRITE, MODIFY, and ERASE access. All other rights remain as they were before.

Example 3

If the CHANGE PROTECTION statement identifies the access control entry (ACE) by its identifiers, make sure the identifiers are in the same order.

```
RDO> CHANGE PROTECTION FOR DATABASE  
cont> [ADMIN,*]+MANAGER+INTERACTIVE  
cont> ACCESS NOCHANGE+NODELETE+NODEFINE.
```

If you issue a CHANGE PROTECTION statement, the new ACE takes effect after you close the database.

CHANGE RELATION Statement

9.6 CHANGE RELATION Statement

Changes the definition of the fields that make up a relation.

The CHANGE RELATION statement can add, modify, or delete the fields that make up an existing relation. If the database has been invoked by path name, these changes are also stored in the dictionary, ensuring consistency between the definitions in the database and the dictionary.

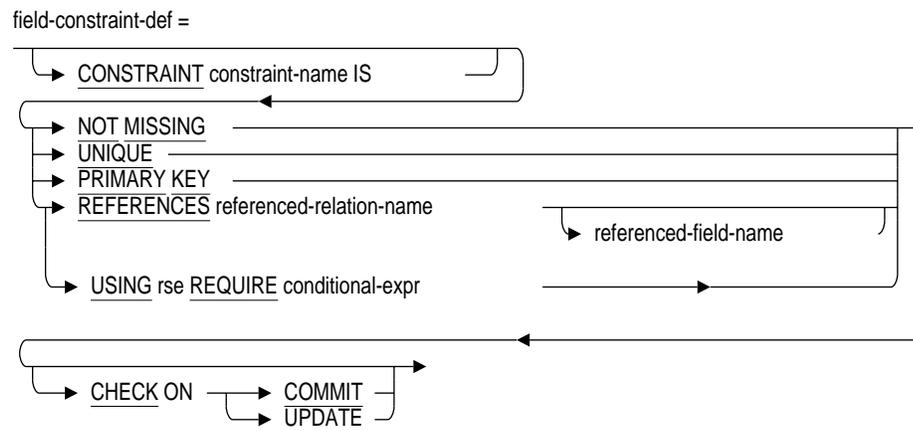
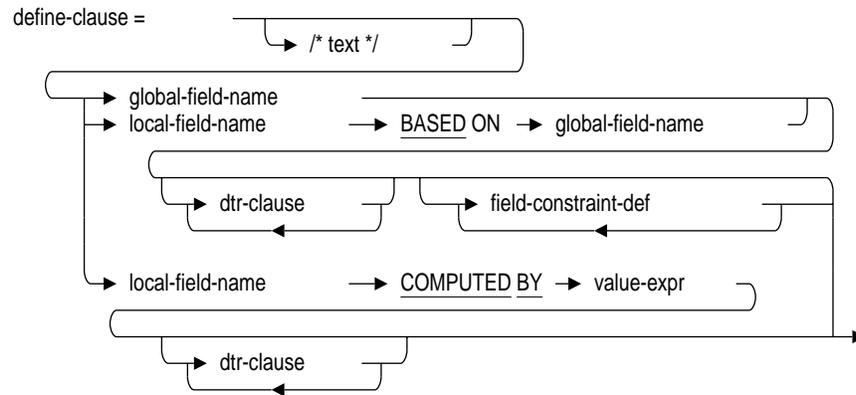
The CHANGE RELATION statement can also add or delete relation-specific constraints, updating the physical database appropriately. These constraints can be deleted or declared or both. Any constraint to be changed must be first specifically deleted by name and then declared again. The names for all constraints currently associated with a relation can be displayed using the SHOW RELATIONS and SHOW FIELD FOR RELATION statements. Any number of constraints can be deleted and declared at both the relation and field levels.

When you execute this statement, Rdb/VMS modifies the named field definition in the relation definition. All the fields that you do not mention remain the same. Rdb/VMS defines new versions of fields *before* defining constraints.

Then Rdb/VMS defines constraints and evaluates them before storing them. Therefore, if fields and constraints are defined within the same relation definition, constraints will always apply to the latest version of a field.

When you change a relation definition, other users see the revised definition only when they invoke the database after you commit the changes.

CHANGE RELATION Statement



CHANGE RELATION Statement

text

A text string that adds a comment to the field definition. You can apply the description to the entire relation definition using the DESCRIPTION keyword. You can also apply a separate description to each field using the text markers before the DEFINE, CHANGE, or DELETE clause.

relation-constraint-def

The clause with which you can name and specify the type of relation-level constraints to be defined within a specific relation definition.

See Section 9.18 for more information.

field-constraint-def

The clause with which you can name or specify the type of field-level constraints to be defined within a specific relation definition.

constraint-name

The name to be associated with the relation- or field-specific constraint. This name must be unique within the database. The constraint name can be referred to in other statements such as DEFINE RELATION, SHOW CONSTRAINT, and START_TRANSACTION.

The phrase “CONSTRAINT constraint-name is” is optional. If you specify the keyword CONSTRAINT, you must also provide a name for the constraint.

DEFINE define-clause

The action of the CHANGE RELATION statement with the DEFINE clause depends on the define-clause as follows:

- **DEFINE global-field-name**
Includes an existing global field definition in the relation.
If you include only a name with the DEFINE option, Rdb/VMS searches for a field with that name in the list of global field definitions for the database. If such a field definition exists, Rdb/VMS adds that field definition to the relation definition. If no field has that name, Rdb/VMS returns an error message.
- **DEFINE local-field-name BASED ON global-field-name**
Includes an existing field definition in the relation, but gives it a local name.

CHANGE RELATION Statement

If you include the **BASED ON** qualifier, Rdb/VMS uses the definition specified by *global-field-name*. However, the name of the new field in the relation is *local-field-name*. This local name does not become part of the global list of field names for the database.

You can specify local DATATRIEVE support clauses for this *local-field-name*. If you do, this local specification overrides the DATATRIEVE clauses attached to *global-field-name*.

- **DEFINE local-field-name COMPUTED BY expression**

Adds a new virtual field.

The **COMPUTED BY** qualifier causes Rdb/VMS to calculate the field's value at run time, based on the specified expression.

You can specify local DATATRIEVE support clauses for this field.

- If you include the **DEFINE CONSTRAINT** clause, Rdb/VMS defines the specified relation-specific constraint.

CHANGE change-clause

The **CHANGE RELATION** statement with the **CHANGE** option modifies the local attributes of an existing field. Only the attributes you specify in the statement change; all others stay as they are. You cannot use the **CHANGE** option of the **CHANGE RELATION** statement to modify the **COMPUTED BY** expression of a field. If you need to change a **COMPUTED BY** field, delete the field and then redefine a new **COMPUTED BY** field.

The effects of this option are summarized as follows:

- **CHANGE local-field-name BASED ON global-field-name**

Gives the specified field the attributes of another field.

This option changes the local definition by pointing to a different global definition. If *field-name* is a local field name, then Rdb/VMS changes the local definition to point to a new global field. If *field-name* is global, then this statement changes it to a local field name. That is, the name assigned to the field is now valid only for the relation in which it is assigned. The global field name still exists in the database, but this relation no longer uses it.

- **CHANGE local-field-name dtr-clause**
CHANGE global-field-name dtr-clause

Changes DATATRIEVE-support characteristics.

CHANGE RELATION Statement

This option changes one or both of the DATATRIEVE options, QUERY_HEADER and QUERY_NAME. You can use this form of the statement with both local and global field names. All other attributes remain the same.

- CHANGE local-field-name DELETE CONSTRAINT constraint-name
CHANGE global-field-name DELETE CONSTRAINT constraint-name

If you include the DELETE CONSTRAINT clause, Rdb/VMS deletes the relation-specific constraint that had been associated with the field.

- CHANGE local-field-name field-constraint-def
CHANGE global-field-name field-constraint-def

If you include the field-constraint-def clause, Rdb/VMS defines a relation-specific constraint on the local or global field.

Do not use the COMPUTED BY expression in the CHANGE change-clause. Instead, delete the existing field with the DELETE change-clause. Then redefine the field using the DEFINE change-clause with the COMPUTED BY expression.

DELETE field-name

Deletes the field from the relation. This option deletes the field and any associated field-level relation-specific constraints from the relation definition. The global field definition by this name is still defined for the database as a whole, and other relations can still refer to it.

You cannot use the DELETE option on a field if that field is referred to by a view definition, a record selection expression (RSE) of a view, a constraint, or a COMPUTED BY field.

Usage Notes

To change a relation with the CHANGE RELATION statement, you need the Rdb/VMS CHANGE privilege for the relation.

To add a constraint with the CHANGE RELATION statement, you need the CHANGE and DEFINE relation privileges to the relation being changed, and you need the READ and DEFINE relation privileges to the relation referenced by the constraint.

To add a field to a relation with the CHANGE RELATION statement, you need the CHANGE and DEFINE relation privileges to the relation being changed.

To change a field with the CHANGE RELATION statement, you need the CHANGE relation privilege.

CHANGE RELATION Statement

To delete a field or a constraint with the CHANGE RELATION statement, you need the CHANGE and DELETE relation privilege.

When the CHANGE RELATION statement executes, Rdb/VMS changes the relation definition in the physical database. If you have invoked the database by path name, the definition is also changed in the data dictionary.

If the database is created with the `DICTIONARY IS REQUIRED` option, you must invoke the database by path name, rather than file name, before you issue this statement.

You must execute this statement in a read/write transaction. If you issue this statement when there is no active transaction, Rdb/VMS starts a read/write transaction implicitly.

Attempts to change a relation will fail if that relation is involved in a query at the same time. Users will have to detach from the database with a `FINISH` statement before you can change the relation. When Rdb/VMS first accesses an object, such as the relation, a lock is taken out on that object and not released until the user exits the database. Attempts to update this object will get a `LOCK CONFLICT ON CLIENT` message due to the other user's optimized access to the object.

Similarly, while you are changing a relation, users cannot execute queries involving that relation until you have completed the transaction with a `COMMIT` or `ROLLBACK` statement for the `CHANGE` statement. The user will receive a `LOCK CONFLICT ON CLIENT` error message. While DDL operations are performed, normal data locking mechanisms are used against system relations. (System relations hold information on objects in the database.) Therefore, attempts to update an object will lock out attempts to query that object. These locks are held until the `COMMIT` or `ROLLBACK` of the DDL operation.

The `WAIT/NOWAIT` clause of the `START_TRANSACTION` statement does not affect attempts to update metadata with simultaneous queries. Even if you specify `START_TRANSACTION WAIT` for the metadata update transaction, you will get the following error message if a lock conflict exists:

```
%RDB-E-LOCK_CONFLICT, request failed due to locked resource; no-wait
parameter specified for transaction
-RDB-E-NO_META_UPDATE, metadata update failed
-RDMS-F-LCKCNFLCT, lock conflict on client
RDO>
```

CHANGE RELATION Statement

However, a user's query will wait for a metadata update to complete with a ROLLBACK or COMMIT, even if the user specified NOWAIT on the START_TRANSACTION statement.

Other users are allowed to be attached to the database when you issue the CHANGE RELATION statement.

When you use the DEFINE option of the CHANGE RELATION statement, you can add a globally defined field to a relation. However, you cannot use the CHANGE RELATION statement to change the attributes of a globally defined field. You can change only the locally defined attributes of a field within a relation. The results of the CHANGE RELATION statement depend on which option you choose and on the source of the specified field definition.

Declaring a constraint with a REFERENCES clause requires read access to the relation named as the referenced-relation.

It is possible to define multiple constraints that have equivalent or subset validation criteria. When defining constraints for a relation, it is advisable to display all the constraints for the affected relation with the SHOW CONSTRAINTS statement to make sure no redundant constraints have been defined.

You cannot execute the CHANGE RELATION statement when the RDB\$SYSTEM storage area is set to read-only. You must first set RDB\$SYSTEM to read/write. See the *VAX Rdb/VMS Guide to Database Maintenance and Performance* for more information on the RDB\$SYSTEM storage area.

Table 9-1 shows the relation privileges you must have to make certain types of changes using the CHANGE RELATION statement. To add a constraint, you need CHANGE and DEFINE privileges to the relation being changed plus READ and DEFINE privileges to the referenced relation.

CHANGE RELATION Statement

Table 9-1 CHANGE RELATION Options and Privileges

Option	Relation Privilege
CHANGE RELATION	CHANGE
CHANGE RELATION/ADD CONSTRAINT	CHANGE+DEFINE+ (READ+DEFINE) on referenced relation
CHANGE RELATION/ADD FIELD	CHANGE+DEFINE
CHANGE RELATION/CHANGE FIELD	CHANGE
CHANGE RELATION/DELETE CONSTRAINT	CHANGE+DELETE
CHANGE RELATION/DELETE FIELD	CHANGE+DELETE

Examples

Example 1

The following example adds an existing field definition to a relation:

```
CHANGE RELATION EMPLOYEES.  
    DEFINE SALARY.  
END EMPLOYEES RELATION.
```

This example simply names an existing global field whose definition becomes part of the definition for the relation.

Example 2

The BASED ON clause adds to a local field name.

```
CHANGE RELATION EMPLOYEES.  
    DEFINE CURRENT_SALARY BASED ON SALARY.  
END EMPLOYEES RELATION.
```

This statement performs the same function, but uses the BASED ON clause to give a local name to the field. The statement assumes that SALARY is defined globally in the database.

Example 3

You can change the local attributes for a field definition inside the CHANGE RELATION statement without changing the global attributes of the field for other relations that refer to it. The DATATRIEVE support clauses defined locally override those defined globally.

CHANGE RELATION Statement

```
CHANGE RELATION DEPARTMENTS.  
  CHANGE DEPARTMENT_NAME  
    QUERY_NAME FOR DATATRIEVE IS "DEPT".  
END DEPARTMENTS RELATION.
```

This example changes the QUERY_NAME field for the DEPARTMENT_NAME field, but only for the DEPARTMENTS relation. The definition of DEPARTMENT_NAME remains the same for any other relations that use it.

Example 4

You can change a local field name.

```
DEFINE FIELD SALARY  
  DATATYPE SIGNED LONGWORD SCALE -2  
  VALID IF SALARY > 8000  
  MISSING_VALUE IS 0  
  EDIT_STRING FOR DATATRIEVE "$$$$$$9.99".  
  
DEFINE FIELD MONEY  
  DATATYPE TEXT SIZE 8  
  VALID IF SALARY > 8000  
  MISSING_VALUE IS 0  
  EDIT_STRING FOR DATATRIEVE "$$$$$$9.99".  
  
CHANGE RELATION EMPLOYEES.  
  DEFINE SALARY.  
END.  
CHANGE RELATION EMPLOYEES.  
  CHANGE SALARY BASED ON MONEY.  
END.
```

This example assumes two fields, SALARY and MONEY, defined globally. They have different data types.

- The first CHANGE RELATION statement adds a field to the EMPLOYEES relation using the global SALARY field definition.
- The second CHANGE RELATION statement uses the BASED ON clause to substitute the MONEY definition for the global SALARY. The local name remains the same, but that name now points to a different global definition. There are now two fields named SALARY in the database, one local and one global.

CHANGE RELATION Statement

Example 5

A COMPUTED BY field is calculated from another field in the relation.

```
CHANGE RELATION SALARY_HISTORY.  
  DEFINE SS_DEDUCTION  
    COMPUTED BY (SALARY_AMOUNT * 0.0625).  
END SALARY_HISTORY RELATION.
```

This statement adds a *virtual* field whose value is computed from other fields.

Example 6

The following example deletes a field:

```
CHANGE RELATION COLLEGES.  
  DELETE CONTACT_NAME.  
END COLLEGES RELATION.
```

This example changes the COLLEGES relation by removing the CONTACT_NAME field from it. A global field is still defined for the database as a whole, and other relations can still refer to it. The global field may have some other name if CONTACT_NAME was defined with the BASED ON qualifier. This statement also makes the data associated with that field invisible.

Example 7

This example changes the field-level primary key constraint for the field DEPT_CODE to a field-level unique constraint.

```
RDO> CHANGE RELATION DEPARTMENTS  
cont> DELETE CONSTRAINT DEPARTMENTS_PRIMARY1  
cont> CONSTRAINT DEPARTMENTS_UNIQUE UNIQUE DEPARTMENT_CODE.  
cont> END.  
%RDB-E-NO_META_UPDATE, metadata update failed  
-RDMS-F-RLCREFDBYRLC, relation constraint DEPARTMENTS_PRIMARY1 is  
referenced by relation JOB_HISTORY constraint JOB_HISTORY_FOREIGN3  
RDO> SHOW CONSTRAINTS FOR DEPARTMENTS  
Constraints for relation DEPARTMENTS  
  DEPARTMENTS_PRIMARY1  
  defined by relation DEPARTMENTS  
    DEPARTMENTS.DEPARTMENT_CODE PRIMARY KEY  
  JOB_HISTORY_FOREIGN3  
  defined by relation JOB_HISTORY  
    JOB_HISTORY.DEPARTMENT_CODE REFERENCES DEPARTMENTS(DEPARTMENT_CODE)  
RDO> CHANGE RELATION JOB_HISTORY  
cont> DELETE CONSTRAINT JOB_HISTORY_FOREIGN3.  
cont> END.  
RDO> CHANGE RELATION DEPARTMENTS  
cont> DELETE CONSTRAINT DEPARTMENTS_PRIMARY1  
cont> CONSTRAINT DEPARTMENTS_UNIQUE UNIQUE DEPARTMENT_CODE.  
cont> END.
```

CHANGE RELATION Statement

```
RDO> SHOW CONSTRAINTS FOR DEPARTMENTS
Constraints for relation DEPARTMENTS
DEPARTMENTS_UNIQUE
defined by relation DEPARTMENTS
    for R in DEPARTMENTS require
    unique Q in DEPARTMENTS with
        R.DEPARTMENT_CODE = Q.DEPARTMENT_CODE
```

This example illustrates how constraints can refer to each other. Before you can delete the primary key constraint DEPARTMENTS_PRIMARY1, you must delete the foreign key constraint JOB_HISTORY_FOREIGN3.

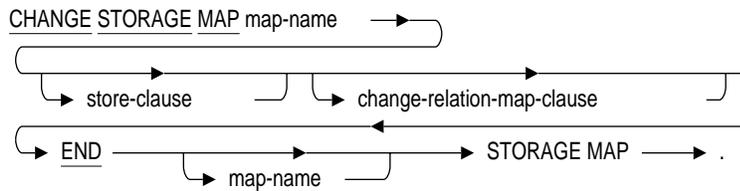
CHANGE STORAGE MAP Statement

9.7 CHANGE STORAGE MAP Statement

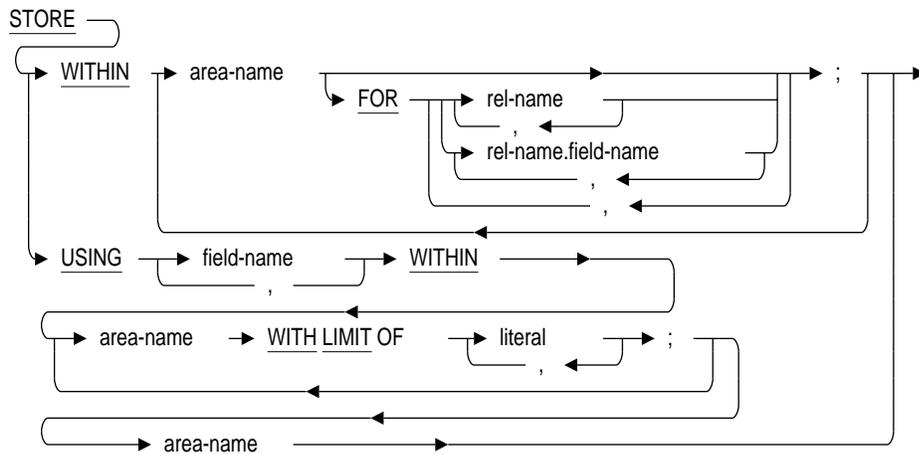
Changes the storage map definition for a relation. You can also change:

- Which index Rdb/VMS uses when it stores new records
- Whether records will be stored in a compressed format
- Whether data will be reorganized

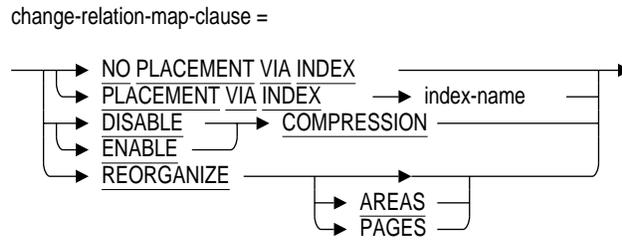
Format



store-clause =



CHANGE STORAGE MAP Statement



Arguments

map-name

The name of the storage map you want to modify.

store-clause

The criteria for determining the location to store a record. This clause contains a number of subclauses. The store-clause can be used to spread data over multiple disks and to cause related data to be stored together in the database. The WITH LIMIT OF clause allows you to partition data over storage areas according to specific values. If you specify storage areas without including the WITH LIMIT OF clause, Rdb/VMS randomly stores records within the specified storage areas. However, if you also specify a PLACEMENT VIA INDEX clause, records will be stored according to the indicated path in the PLACEMENT VIA INDEX clause.

If you do not specify the store-clause, the relation associated with the storage map is stored in the default storage area.

USING field-name

The names of the fields whose values will be used as limits for partitioning the relation across multiple storage areas. Rdb/VMS compares values in the fields to values in the WITH LIMIT OF clause to determine where to store the record initially.

WITHIN area-name

The name of the storage area in which you want records stored. You must define this storage area with the DEFINE DATABASE statement before you refer to it in the store-clause.

CHANGE STORAGE MAP Statement

FOR rel-name

The name of the relation whose segmented strings you want to store in the specified storage area. If you want to store the segmented strings of more than one relation in the storage area, separate the names of the relations with commas.

FOR rel-name.field-name

The name of the relation and segmented string field that you want to store in the specified storage area. If you want to store more than one segmented string field in the storage area, separate the list items with commas.

USING field-name

The names of the fields whose values will be used as limits for partitioning the relation across multiple storage areas. Rdb/VMS compares values in the fields to the values in the WITH LIMIT OF clause to determine where to store the record initially.

WITH LIMIT OF literal

The maximum value a field specified in the USING clause can have when it is initially stored in the specified storage area.

The number of literals specified in each WITH LIMIT OF clause must not exceed the number of fields specified in the USING clause.

Note that the last storage area you specify *cannot* have a WITH LIMIT OF clause associated with it.

When you are modify a multisegmented index using multiple keys and use the STORE USING . . . WITH LIMITS clause, if the values for the first key are all the same, then set the limit for the first key at that value. By doing this, you ensure that the value of the second key determines the storage area in which each record will be stored.

PLACEMENT VIA INDEX index-name

Directs Rdb/VMS to store a record in a way that optimizes access to that record using the indicated path.

If the index named is a hashed index, the storage area named must have a MIXED page format. If the hashed index definition and the storage map for the relation designate the same storage area, then the record is stored on the same page as the hashed index node. Otherwise, Rdb/VMS uses the same *relative* page within the data storage area as the target page.

CHANGE STORAGE MAP Statement

If the index named is a sorted index, Rdb/VMS finds the database key (dbkey) of the next lowest record to the one being stored and uses the page number in the dbkey as the target page.

The index named in the PLACEMENT VIA INDEX clause must be defined for the same relation that the storage map is being defined for.

NO PLACEMENT VIA INDEX

Negates the PLACEMENT VIA INDEX clause, so that subsequent records stored are not stored by means of the index named in the PLACEMENT VIA INDEX clause. This option is available only on the CHANGE STORAGE MAP statement. If you specify the CHANGE STORAGE MAP statement without the PLACEMENT VIA INDEX option or the NO PLACEMENT VIA INDEX option, the statement executes as if the clause specified on the DEFINE STORAGE MAP statement or previous CHANGE STORAGE MAP statement had been used.

DISABLE COMPRESSION

ENABLE COMPRESSION [Default]

Specifies whether to enable or disable data compression for the records when they are stored.

REORGANIZE

Causes records previously stored in specified relations to be moved according to the partitions specified in the CHANGE STORAGE MAP statement.

For details of how records are moved or not moved among storage areas depending whether the REORGANIZE argument is specified, see the *VAX Rdb/VMS Guide to Database Maintenance and Performance*.

Usage Notes

To change a storage map for a relation with the CHANGE STORAGE MAP statement, you need the Rdb/VMS CHANGE privilege for the relation.

If the database is created with the DICTIONARY IS REQUIRED option, you must invoke the database by path name, rather than file name, before you issue this statement.

You must specify either a store-clause, a [NO]PLACEMENT VIA INDEX clause, a REORGANIZE clause, or a COMPRESSION clause in a CHANGE STORAGE MAP statement.

CHANGE STORAGE MAP Statement

In the change-relation-map-clause, you can select one or more of the three clauses ([NO]PLACEMENT VIA INDEX clause, REORGANIZE clause, or COMPRESSION clause) in any order, but you cannot repeat a clause. Note that when the REORGANIZE clause is used, records are moved and assigned to new dbkeys.

If you omit the store-clause in the CHANGE STORAGE MAP statement, you can create a second index by mistake.

If you do not specify the store-clause but do specify the REORGANIZE clause, Rdb/VMS uses the partition and PLACEMENT VIA INDEX clause defined in the DEFINE STORAGE MAP statement.

You must execute this statement in a read/write transaction. If there is no active transaction and you issue the CHANGE STORAGE MAP statement, Rdb/VMS starts a read/write transaction implicitly.

Other users are allowed to be attached to the database when you issue the CHANGE STORAGE MAP statement. However, they are not allowed to be using the relation whose map is being changed.

Attempts to change a storage map will fail if that storage map is involved in a query at the same time. Users will have to detach from the database with a FINISH statement before you can change the storage map. When Rdb/VMS first accesses an object, such as the table, a lock is taken out on that object and not released until the user exits the database. Attempts to update this object will get a LOCK CONFLICT ON CLIENT message due to the other user's optimized access to the object.

Similarly, while you change a storage map, users cannot execute queries involving that table until you have completed the transaction with a COMMIT or ROLLBACK statement for the CHANGE statement. The user will receive a LOCK CONFLICT ON CLIENT error message. While DDL operations are performed, normal data locking mechanisms are used against system relations. (System relations hold information on objects in the database.) Therefore, attempts to update an object will lock out attempts to query that object. These locks are held until the COMMIT or ROLLBACK of the DDL operation.

The WAIT/NOWAIT clause of the START_TRANSACTION statement does not affect attempts to update metadata with simultaneous queries. Even if you specify START_TRANSACTION WAIT for the metadata update transaction,

CHANGE STORAGE MAP Statement

you will get the following error message if a lock conflict exists:

```
%RDB-E-LOCK_CONFLICT, request failed due to locked resource; no-wait  
parameter specified for transaction  
-RDB-E-NO_META_UPDATE, metadata update failed  
-RDMS-F-LCKCNFLCT, lock conflict on client  
RDO>
```

However, a user's query will wait for a metadata update to complete with a ROLLBACK or COMMIT, even if the user specified NOWAIT on the START_TRANSACTION statement.

You cannot delete a storage map that refers to a storage area that has data in it. See Section 9.33 for more information on deleting storage maps.

If you specify the REORGANIZE clause as part of the CHANGE STORAGE MAP statement, Rdb/VMS automatically reorganizes the database records in the following ways:

- If areas were named in the original map that are not named in the new map, records in those areas deleted from the map will be moved to the areas specified by the new map.
- If the new storage map definition includes the REORGANIZE AREAS clause, all other records will then be checked to determine whether they are in the correct area. If they are not in the correct area, they will be stored in the correct area and deleted from their current area.
- If the new storage map definition includes the REORGANIZE PAGES clause, the database software will check to see which records can be moved to the pages where they would be placed if they were being stored as new records. If the records will fit on those preferred pages or pages closer to the page than they are currently stored on, they are moved.
- If the new storage map definition includes the WITH LIMIT OF clause when the REORGANIZE clause is specified, all records are read and restored, whether or not new values are given.
- If the new storage map definition includes only the COMPRESSION clause, all records are read, the compression characteristics are changed, and all rows are restored, whether or not the REORGANIZE clause is specified.
- If the new storage map definition includes the PLACEMENT VIA INDEX clause when the REORGANIZE clause is specified, only the new records based on the new index name are stored.

CHANGE STORAGE MAP Statement

- If the new storage map definition includes the USING field-name clause when the REORGANIZE clause is specified, only the new records based on the new field name are stored.

If you do not specify the REORGANIZE clause as part of the CHANGE STORAGE MAP statement, Rdb/VMS treats the database records in the following ways:

- If the new storage map definition omits the name of a storage area that was in the original storage map definition:
 - 1 The records are unloaded from the omitted storage area.
 - 2 The records are stored into the named storage areas, according to the specified WITH LIMIT OF clause.
 - 3 The records are compressed according to the characteristics specified in the COMPRESSION clause.

If the new storage map definition does not omit any previously named storage areas, but does change the limits for a storage area, records that were already stored in the area are *not* moved according to the new WITH LIMIT OF clause. However, new records will be stored according to the new limits you specify.

The CHANGE STORAGE MAP statement can delete an area only if there are no segmented strings in the area. You cannot move a segment from one area to another. Changing the area or areas assigned to a relation will have no effect on current segmented strings, only on future storage.

You cannot execute the CHANGE STORAGE MAP statement when the RDB\$SYSTEM storage area is set to read-only. You must first set RDB\$SYSTEM to read/write. See the *VAX Rdb/VMS Guide to Database Maintenance and Performance* for more information on the RDB\$SYSTEM storage area.

Examples

Example 1

The following example disables compression for the CANDIDATES_MAP storage map:

```
RDO> CHANGE STORAGE MAP CANDIDATES_MAP
cont> DISABLE COMPRESSION
cont> END CANDIDATES_MAP STORAGE MAP.
```

CHANGE STORAGE MAP Statement

Example 2

The following example assigns new limits for storage areas:

```
RDO> CHANGE STORAGE MAP EMPLOYEES_MAP
cont> STORE USING EMPLOYEE_ID
cont>     WITHIN EMPIDS_LOW WITH LIMIT OF "00300";
cont>     EMPIDS_MID WITH LIMIT OF "00600";
cont>     EMPIDS_OVER
cont> END EMPLOYEES_MAP STORAGE MAP.
```

Current data will *not* be moved according to the new limits. However, when new data is stored, it will be stored according to the new limits in the storage map.

Example 3

The following example defines a new storage area EMPIDS_MID2 to handle the employee IDs 600–900, and to reorganize the data from one existing storage area, EMPIDS_OVER. The current data stored within the limits of employee IDs 601–900 will be moved according to the new limits. When the new data is stored, it will be stored according to the new limits set in the storage map definition.

```
RDO> CHANGE STORAGE MAP EMPLOYEES_MAP
cont> STORE USING EMPLOYEE_ID
cont>     WITHIN EMPIDS_LOW WITH LIMIT of "00300";
cont>     EMPIDS_MID WITH LIMIT OF "00600";
cont>     EMPIDS_MID2 WITH LIMIT OF "00900";
cont>     EMPIDS_OVER
cont>     REORGANIZE
cont> END EMPLOYEES_MAP STORAGE MAP.
```

9.8 COMMIT Statement

Ends a transaction and makes permanent any changes you made during that transaction. The COMMIT statement also performs these functions:

- Releases all locks
- Closes open streams

The COMMIT statement affects:

- All databases participating in the currently open transaction
- All changes to data made with Rdb/VMS data manipulation statements (ERASE, MODIFY, and STORE)
- All changes to data definitions made with Rdb/VMS data definition statements (CHANGE, DEFINE, and DELETE)

Format

```
COMMIT (TRANSACTION_HANDLE → var) on-error
```

Arguments

TRANSACTION_HANDLE var

A keyword followed by a host language variable. A transaction handle identifies each instance of a database attach. If you do not declare the transaction handle explicitly, Rdb/VMS attaches an internal identifier to the transaction.

In Callable RDO, use !VAL as a substitution marker for host language variables. See the *VAX Rdb/VMS Guide to Using RDO, RDBPRE, and RDML* for information about the use of !VAL.

You can put parentheses around the host language variable name.

Note Normally, you do not need to use this argument. The ability to declare a transaction handle is provided for compatibility with other database products and future releases of Rdb/VMS.

COMMIT Statement

Do not use the TRANSACTION_HANDLE argument in interactive RDO.

on-error

The ON ERROR clause. This clause specifies a host language statement to be performed if an Rdb/VMS error occurs.

Usage Notes

If you have invoked a database, you have the necessary privileges to use the COMMIT statement.

Because the COMMIT statement closes open streams, you should not use an explicit END_STREAM statement after a COMMIT statement. If you do, Rdb/VMS returns an error message.

Examples

Example 1

This example makes a change to a relation and writes that change to the database.

```
    DISPLAY "Enter employee's ID number:  "  -- ❶
      WITH NO ADVANCING.
    ACCEPT ID.
    DISPLAY "Percentage increase:  "  ----- ❷
      WITH NO ADVANCING.
    ACCEPT PERC.
    CALL "SYS$GETTIM" USING TODAY
      GIVING RETURN_VALUE.

&RDB& START_TRANSACTION READ_WRITE  ----- ❸
&RDB&   RESERVING SALARY_HISTORY FOR
&RDB&   PROTECTED WRITE

&RDB& FOR S IN SALARY_HISTORY
&RDB&   WITH S.EMPLOYEE_ID = ID  AND  ----- ❹
&RDB&   S.SALARY_END MISSING
&RDB&   MODIFY S USING  ----- ❺
&RDB&     ON ERROR
&RDB&       GO TO ERROR-PAR
&RDB&     END_ERROR
&RDB&   S.SALARY_END = TODAY
&RDB&   END_MODIFY
```

COMMIT Statement

```
&RDB&   STORE NEW IN SALARY_HISTORY USING   ----- ⑥
&RDB&   ON ERROR
&RDB&   GO TO ERROR-PAR
&RDB&   END_ERROR
&RDB&   NEW.EMPLOYEE_ID = S.EMPLOYEE_ID;
&RDB&   NEW.SALARY_AMOUNT =
&RDB&   ( S.SALARY_AMOUNT *
&RDB&   ( 1 + ( PERC / 100 ) ) );
&RDB&   NEW.SALARY_START = TODAY
&RDB&   END_STORE

&RDB& END_FOR

&RDB& COMMIT   ----- ⑦
```

This program fragment gives a raise to an employee. In order to maintain a consistent database, it performs three operations within one transaction. It performs the following operations:

- ① Prompts for an employee identification number (ID).
- ② Prompts for a percentage increase, which is used to calculate the increase.
- ③ Starts a read/write transaction. This statement uses the **RESERVING** clause to protect the **SALARY_HISTORY** record against conflicting updates, while allowing users access to all the other relations in the database.
- ④ Establishes a record stream that consists of the current **SALARY_HISTORY** record for the specified employee.
- ⑤ Uses the **MODIFY** statement to change the current salary record, by changing its date from missing to the current date (**TODAY**).
- ⑥ Uses the **STORE** statement to create a new **SALARY_HISTORY** record. Although this statement executes within the record stream defined by the **FOR** loop, it must declare a new context variable to identify the new record. All the fields of the new record can be derived from fields of the old one, except the start date, which must be calculated from the current date. Rdb/VMS calculates **NEW.SALARY_AMOUNT** from the **S.SALARY_AMOUNT**, using the specified percentage increase (**PERC**).
- ⑦ Uses the **COMMIT** statement to make the changes to the database permanent.

Note that each data manipulation statement provides an **ON ERROR** clause.

COMMIT Statement

Example 2

This example shows how to use the COMMIT statement to make a data definition permanent.

```
RDO> DEFINE DATABASE 'INVENTORY'.
RDO> SHOW DATABASES
Database with db_handle INVENTORY in path INVENTORY
  The CDD is being maintained
RDO> DEFINE FIELD PART DATATYPE TEXT SIZE 10.
RDO> DEFINE RELATION TEST.
cont> PART.
cont> END TEST RELATION.
RDO>
RDO> COMMIT
RDO> SHOW RELATIONS
User Relations in Database with pathname DISK1:[DICTIONARY]CORP.MIS.PERSONNEL;1
  TEST
RDO> SHOW FIELDS
User Fields in Database with pathname DISK1:[DICTIONARY]CORP.MIS.PERSONNEL;1
  PART          text size is 10
RDO> EXIT
```

This example shows how to define a database, a field, and a relation. These statements would normally be in an indirect command file. After you define the database and its components, the COMMIT statement makes the changes to the database permanent.

Errors

Your program can check for and handle the following errors, using the name listed here in the form RDB\$_error-name. The following list includes two types of errors:

- E Expected errors. These are run-time errors. Your program should check for these errors and provide error handlers.
- U Unexpected errors. Preprocessors detect these errors when the program is preprocessed, so your program does not need to check for them. Callable RDO, however, detects these errors at run time. A Callable RDO program should check for them.

BAD_TRANS_HANDLE (U)
invalid handle for transaction

DEADLOCK (E)
request failed due to resource deadlock

COMMIT Statement

LOCK_CONFLICT (E)

NO WAIT request failed because resource was locked

NO_PRIV (U)

privilege denied by database protection

REQ_NO_TRANS (U)

attempt to continue request after COMMIT/ROLLBACK

CREATE_SEGMENTED_STRING Statement

9.9 CREATE_SEGMENTED_STRING Statement

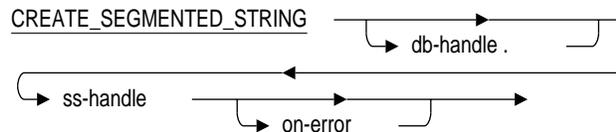
Allows you to store segments in a segmented string.

Because a single segmented string field value is made up of multiple segments, you must manipulate the segments one at a time. Therefore, segmented string operations require an internal looping mechanism, much like the record stream set up by a FOR or START_STREAM statement. For this reason, storing into a segmented string field requires two steps:

- 1 Storing the segments into the string
- 2 Storing the string field into the relation

In order to store the segments into the segmented string, the string must have a name. The CREATE_SEGMENTED_STRING statement initializes a segmented string and gives it a handle so you can store segments into the field. After you are finished storing segments, you store the entire field into the relation, using the handle to relate the newly built string to the field in which it belongs. Finally, you use the END_SEGMENTED_STRING statement to close the block. See the Examples section.

Format



Arguments

db-handle

A host language variable or name used to refer to a specific database you have invoked. Specifying a database handle allows you to have more than one database invoked when you create and store a segmented string. If you invoke more than one database, you must specify the database handle of the database you intend to access.

In Callable RDO, use the substitution marker !VAL to specify the db-handle.

CREATE_SEGMENTED_STRING Statement

ss-handle

A host language variable or name used to refer to the segmented string.

on-error

The ON ERROR clause. This clause specifies host language statements or Rdb/VMS data manipulation statements to be performed if an Rdb/VMS error occurs.

Usage Notes

If you have invoked a database, you have the necessary privileges to use the CREATE_SEGMENTED_STRING statement.

You cannot modify a segmented string.

When using the RDML and RDBPRE precompilers, be sure to define a sufficiently large value for the RDMS\$BIND_SEGMENTED_STRING_BUFFER logical name. An adequate buffer size is needed to store large segmented strings (using segmented string storage maps) in storage areas other than the default RDB\$SYSTEM storage area. The minimum acceptable value for the RDMS\$BIND_SEGMENTED_STRING_BUFFER logical name must be equal to the sum of the length of the segments of the segmented string. For example, if you know that the sum of the length of the segments is one megabyte, then 1,048,576 bytes is an acceptable value for this logical name.

You must specify the logical name value because when RDML and RDBPRE precompilers store segmented strings, Rdb/VMS does not know which table contains the string until after the entire string is stored. Rdb/VMS buffers the entire segmented string, if possible, and does not store it until the STORE statement executes.

If the segmented string remains buffered, it is stored in the appropriate storage area. If the string is not buffered (because it is larger than the defined value for the logical name or the default value of 10,000 bytes), it is not stored in the default storage area and the following exception message is displayed:

```
%RDB-F-IMP_EXC, facility-specific limit exceeded  
-RDMS-E-SEGSTR_AREA_INC, segmented string was stored incorrectly
```

To avoid this error, set the value of the RDMS\$BIND_SEGMENTED_STRING_BUFFER logical name to a sufficiently large value. Note that a value of up to 500 MB can be specified for this logical name. See Section 9.14 for more information on defining storage areas.

CREATE_SEGMENTED_STRING Statement

You must execute this statement in a read/write transaction. If there is no active transaction and you issue this statement, Rdb/VMS starts a read/write transaction implicitly.

Examples

Example 1

In the following example, PERS1 is the database handle and RESUME_HANDLE is the segmented string handle:

```
CREATE_SEGMENTED_STRING PERS1.RESUME_HANDLE
```

In host language programs, you can also use the ON ERROR clause of the CREATE_SEGMENTED_STRING statement.

Example 2

Store a segmented string.

```
START_TRANSACTION READ_WRITE RESERVING
  RESUMES FOR EXCLUSIVE WRITE
!
! Start a stream of segments. Give the stream a name.
!
CREATE_SEGMENTED_STRING RESUME_HANDLE
!
! Store the segments in the field.
!
STORE SEG IN RESUME_HANDLE USING
  SEG.RDB$VALUE =
    "This is the first line of an employee's resume."
END_STORE
!
STORE SEG IN RESUME_HANDLE USING
  SEG.RDB$VALUE =
    "This is the second line of an employee's resume."
END_STORE
!
STORE SEG IN RESUME_HANDLE USING
  SEG.RDB$VALUE =
    "This is the third line of an employee's resume."
END_STORE
!
STORE SEG IN RESUME_HANDLE USING
  SEG.RDB$VALUE =
    "This is the fourth line of an employee's resume."
END_STORE
```

CREATE_SEGMENTED_STRING Statement

```
!  
! Store the segmented string field in the relation.  
!  
STORE R IN RESUMES USING  
    R.EMPLOYEE_ID = "00164";  
    R.RESUME = RESUME_HANDLE  
END_STORE  
!  
END_SEGMENTED_STRING RESUME_HANDLE  
COMMIT
```

This sequence of statements demonstrates the steps required to store a segmented string:

- The `CREATE_SEGMENTED_STRING` statement starts a stream so that you can store the segments into the field. The segmented string handle gives the stream a name.
- Each segmented string `STORE` statement stores a text string in a segment. The context variable `SEG` relates the values being stored to the stream.
- The final `STORE` statement stores the segmented string field into the relation, along with the other field, `EMPLOYEE_ID`. Note that this `STORE` statement uses the segmented string handle as the value expression in the `STORE` assignment.

In most cases, this set of statements would be part of a program that reads lines from a text file and stores each line in a segment of the segmented string field.

Example 3

The following program reads a file and loads it into the specified employee's `RESUMES` record in the `PERSONNEL` database.

```
program STORE_RESUME
```

CREATE_SEGMENTED_STRING Statement

```
!
! STORE RESUME
! This program reads a file and loads it into the specified
! employee's RESUMES record in the PERSONNEL database
!
      option type = EXPLICIT
      declare long constant TRUE = -1%, FALSE = 0%
      declare
        string
          employee_id, resume_file, text_line,
          last_name, first_name,
        long
          found, line_count
&RDB& INVOKE DATABASE FILENAME "DB$:PERSONNEL31"
      print "*** Personnel RESUME Load ***"
      when error in
        input "Enter EMPLOYEE_ID"; employee_id
      use
        print "Program terminated"
        continue END_PROGRAM
      end when
&RDB& START_TRANSACTION READ_WRITE
&RDB& FOR E IN EMPLOYEES WITH E.EMPLOYEE_ID = employee_id
&RDB&   GET
&RDB&     last_name = E.LAST_NAME;
&RDB&     first_name = E.FIRST_NAME;
&RDB&   END_GET
&RDB&     found = TRUE
&RDB& END_FOR

      if not found then
        print "Error - employee " + employee_id + " not found"
        exit program
      else
        !
        ! Display the employee's name
        !
        print "Loading RESUME for employee " +
          TRM$(first_name) + ", " + TRM$(last_name)
        !
        ! Read the name of the resume source file
        !
```

CREATE_SEGMENTED_STRING Statement

```
GET_NAME:
  when error in
    input "Enter the resume file name"; resume_file
    open resume_file for input as file #1,      &
      organization sequential, recordtype ANY
  use
    if err = 11% then
      print "Program terminated"
      continue END_PROGRAM
    else
      print "Error - " + RIGHT(ERT$(err),2%)
      continue GET_NAME
    end if
  end when

&RDB&  CREATE_SEGMENTED_STRING RES
      !
      ! Loop and read each line from the resume, and store it
      ! in the segmented string
      !
      line_count = 0%
      while TRUE ! indefinite loop
        when error in
          linput #1, text_line
        use
          continue EOF
        end when
        text_line = TRM$(text_line)
        line_count = line_count + 1%
&RDB&  STORE R IN RES USING
&RDB&    R.RDB$VALUE = text_line;
&RDB&    R.RDB$LENGTH = LEN(text_line)
&RDB&  END_STORE
      next
  EOF:
    close #1
    print line_count; "lines stored in resume."
&RDB&  STORE RS IN RESUMES USING
&RDB&    RS.EMPLOYEE_ID = employee_id;
&RDB&    RS.RESUME = RES
&RDB&  END_STORE
&RDB&  END_SEGMENTED_STRING RES
  end if

&RDB&  commit
&RDB&  finish

  END_PROGRAM:
end program
```

CREATE_SEGMENTED_STRING Statement

Errors

Your program can check for and handle the following errors, using the name listed here in the form RDB\$_error-name. The following list includes two types of errors:

- E Expected errors. These are run-time errors. Your program should check for these errors and provide error handlers.
- U Unexpected errors. Preprocessors detect these errors when the program is preprocessed, so your program does not need to check for them. Callable RDO, however, detects these errors at run time. A Callable RDO program should check for them.

BAD_SEGSTR_HANDLE (U)

invalid handle for segmented string

DEADLOCK (E)

request failed due to resource deadlock

LOCK_CONFLICT (E)

NO WAIT request failed because resource was locked

NO_PRIV (U)

privilege denied by database protection

REQ_NO_TRANS (U)

attempt to continue request after COMMIT/ROLLBACK

SEGSTR_NO_TRANS (U)

attempt to use a segmented string after COMMIT/ROLLBACK

9.10 DCL Invoke (\$) Statement

Provides access to the DIGITAL Command Language (DCL) environment from inside RDO.

The dollar sign (\$) instructs RDO to spawn a subprocess and pass the next keyword to DCL for processing. You must follow the dollar sign with a DCL command. After DCL processes the command, it logs out of the subprocess and returns control to RDO.

Format

```
$ dcl-command
```

Argument

dcl-command
A valid DCL command.

Usage Note

You need no special Rdb/VMS privileges to use this statement.

Examples

Example 1

This example shows how to use the DCL DIRECTORY command from RDO:

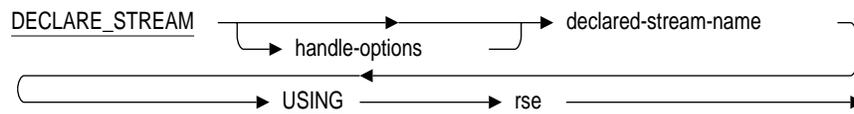
```
RDO> $ DIRECTORY *.RDO
Directory DISK2:[DEPT3.ACCT]
DEFPRO.RDO;6    NOTEQUAL.RDO;1    QUERY.RDO;1    REFEXAM.RDO;12
STORE.RDO;1    UPDATE.RDO;2
Total of 6 files.
RDO>
```

DECLARE_STREAM Statement

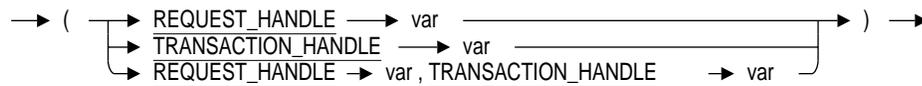
9.11 DECLARE_STREAM Statement

Declares the context of a record stream and is thereby able to associate a stream name with a record selection expression (RSE). This provides Rdb/VMS with the context needed to allow you to place the elements of the `START_STREAM . . . FETCH . . . END_STREAM` construct in any order within a program, and within separate procedures in the same program module.

Format



handle-options =



Arguments

REQUEST_HANDLE var

A keyword followed by a host language variable. A request handle points to the location of a compiled Rdb/VMS request. If you do not supply a request handle explicitly, Rdb/VMS associates a request handle with the compiled request.

Use `!VAL` in Callable RDO as a marker for host language variables.

See the *VAX Rdb/VMS Guide to Using RDO, RDBPRE, and RDML* for information on using request handles.

DECLARE_STREAM Statement

TRANSACTION_HANDLE var

A keyword followed by a host language variable. A transaction handle identifies each instance of a database attach. If you do not declare the transaction handle explicitly, Rdb/VMS attaches an internal identifier to the transaction.

Use !VAL in Callable RDO as a substitution marker for host language variables.

Note *Normally, you do not need to use this argument. The ability to declare a transaction handle is provided for compatibility with other database products and future releases of Rdb/VMS.*

Do not use this argument in interactive RDO.

declared-stream-name

A name you give to the stream you declare.

rse

A record selection expression. A phrase that defines the specific conditions that individual records must meet before Rdb/VMS includes them in a record stream.

Usage Notes

If you have invoked a database, you have the necessary privileges to use the DECLARE_STREAM statement.

Use the DECLARE_STREAM statement in conjunction with the declared START_STREAM statement. The DECLARE_STREAM statement will not work in conjunction with the undeclared START_STREAM statement.

Digital Equipment Corporation recommends that all programs use the DECLARE_STREAM statement (with the declared START_STREAM statement) instead of the undeclared START_STREAM statement. The declared START_STREAM statement provides all the functions of the undeclared START_STREAM statement and provides more flexibility in programming than the undeclared START_STREAM statement.

Put the DECLARE_STREAM statement before the associated declared START_STREAM, FETCH, and END_STREAM statements.

DECLARE_STREAM Statement

The `DECLARE_STREAM` statement allows the use of fewer or more `END_STREAM` statements than `START_STREAM` statements within the same module, as long as at execution time exactly one `END_STREAM` statement is executed for each `START_STREAM` statement.

Examples

Example 1

The following example declares a record stream named `EMP_STREAM` in `RDO`:

```
RDO> DECLARE_STREAM EMP_STREAM USING  
cont>     E IN EMPLOYEES SORTED BY E.LAST_NAME, E.FIRST_NAME
```

DEFINE COLLATING_SEQUENCE Statement

9.12 DEFINE COLLATING_SEQUENCE Statement

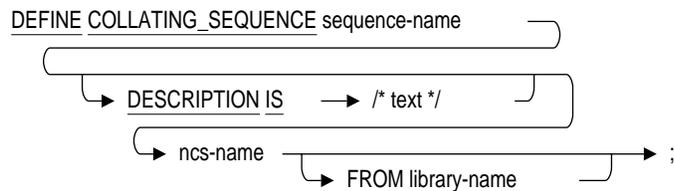
Identifies a collating sequence that has been defined using the VMS National Character Set (NCS) Utility. Use the DEFINE COLLATING_SEQUENCE statement to identify collating sequences other than the database default collating sequence that you plan to use with certain global fields. (The default collating sequence for a database is established by the COLLATING_SEQUENCE IS clause in the DEFINE DATABASE statement; if you omit that clause at database definition time, the default sequence is ASCII.)

You must enter a DEFINE COLLATING_SEQUENCE statement specifying a collating sequence before you enter the name of that sequence in any of the following statements:

- DEFINE FIELD . . . COLLATING_SEQUENCE
- CHANGE FIELD . . . COLLATING_SEQUENCE

Format

```
DEFINE COLLATING_SEQUENCE sequence-name  
    DESCRIPTION IS /* text */  
    ncs-name FROM library-name ;
```



Arguments

sequence-name

Specifies the name by which the collating sequence named in the ncs-name argument will be known to the database. The sequence-name and ncs-name arguments can be the same.

DESCRIPTION IS /* text */

Adds a comment about the collating sequence. RDO displays the text when it executes a SHOW COLLATING_SEQUENCE statement.

DEFINE COLLATING_SEQUENCE Statement

ncs-name

Specifies the name of a collating sequence in the default NCS library, SYSSLIBRARY:NCS\$LIBRARY, or in the NCS library specified by the argument library-name. You can view the collating sequence names by using the command NCS/LIST at DCL level.

The collating sequence can be either one of the predefined NCS collating sequences or one that you have defined yourself using NCS.

FROM library-name

Specifies the name of an NCS library other than the default. The default NCS library is SYSSLIBRARY:NCS\$LIBRARY.

Usage Notes

To define a collating sequence using the DEFINE COLLATING_SEQUENCE statement, you must have the Rdb/VMS DEFINE privilege for the database.

The DEFINE COLLATING_SEQUENCE statement is the first step in specifying an alternate collating sequence. After you define the collating sequence, you must specify that it will apply to a particular field or database. The following list shows abbreviated forms of the statements that will associate the collating sequence with a particular field or database:

- DEFINE FIELD . . . COLLATING_SEQUENCE sequence-name;
- DEFINE DATABASE . . . COLLATING_SEQUENCE sequence-name;
- IMPORT . . . COLLATING_SEQUENCE sequence-name;

Use the complete form of the following statement to alter a collating sequence after you have associated it with a field:

- CHANGE FIELD . . . COLLATING_SEQUENCE sequence-name;

You must execute the DEFINE COLLATING_SEQUENCE statement in a read/write transaction. If you issue this statement when there is no active transaction, Rdb/VMS starts a read/write transaction implicitly.

You can issue the DEFINE COLLATING_SEQUENCE statement when other users are attached to the database.

See Section 9.3 for information on changes that can be made to a collating sequence.

DEFINE COLLATING_SEQUENCE Statement

You cannot execute the DEFINE COLLATING_SEQUENCE statement when the RDB\$SYSTEM storage area is set to read-only. You must first set RDB\$SYSTEM to read/write. See the *VAX Rdb/VMS Guide to Database Maintenance and Performance* for more information on the RDB\$SYSTEM storage area.

Examples

Example 1:

The following example creates a collating sequence using the predefined collating sequence FRENCH. It then shows the defined collating sequence by using the SHOW COLLATING_SEQUENCE statement:

```
RDO> INVOKE DATABASE FILENAME 'MF_PERSONNEL'  
RDO> DEFINE COLLATING_SEQUENCE FRENCH FRENCH.  
%RDO-W-NOCDUPDAT, database invoked by filename, the CDD will not be updated  
RDO> !  
RDO> SHOW COLLATING_SEQUENCE  
User Collating Sequences in Database with filename mf_personnel  
FRENCH
```

DEFINE CONSTRAINT Statement

9.13 DEFINE CONSTRAINT Statement

Creates a constraint for a relation or relations.

A **constraint** defines a set of conditions that restrict the values stored in relations. When you store and modify field values, the constraint checks the validity of the values and generates an error message if the constraint is violated.

When the DEFINE CONSTRAINT statement executes, the constraint definition is added to the physical database. When you invoke the database using the PATHNAME qualifier, the constraint definition is also stored in the data dictionary.

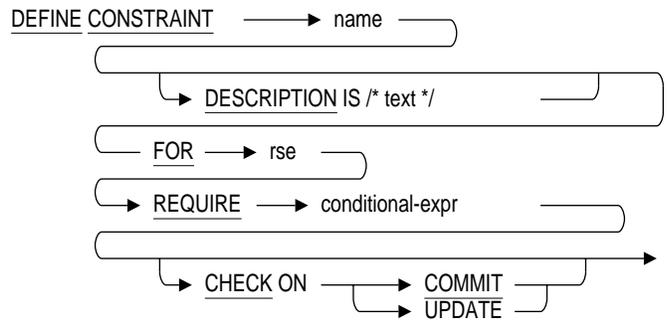
The DEFINE CONSTRAINT statement includes a record selection expression. Therefore, before you store values in a particular field, you can use the constraint definition to check values stored elsewhere in the database. A constraint extends the validation checking offered by the VALID IF clause of a field definition. In this case, it is more flexible than the VALID IF clause. For example:

- The VALID IF clause must be part of a DEFINE FIELD statement. Therefore, a VALID IF criterion of a globally defined field applies to all the fields that use the definition. You can define a constraint that refers to only one of several relations that use a global field definition.
- The VALID IF clause refers to a range of literal values. You cannot use this clause to check a value against values of other fields stored in the database. With the DEFINE CONSTRAINT statement, you can check values from one relation against other database values, either in the same or another relation.
- You can use the DEFINE CONSTRAINT statement to check for such conditions as existence, uniqueness, and nonexistence.

See the *VAX Rdb/VMS Guide to Database Design and Definition* for more information on using DEFINE CONSTRAINT. See Section 5.2 of this manual for more information about the VALID IF clause of the DEFINE FIELD statement.

DEFINE CONSTRAINT Statement

Format



Arguments

name

The name of the constraint. You use this name to refer to the constraint in other statements, such as `START_TRANSACTION`. When you choose a name, follow these rules:

- Use a name that is unique among all constraint names.
- Use any valid VMS name. However, the name cannot end in a dollar sign (\$) or underscore (_).
- Do not use any Rdb/VMS reserved words (see Appendix A).

text

A text string that adds a comment to the constraint definition.

FOR rse

A record selection expression that defines which records of which relations the constraint applies to. You cannot place constraints on views.

REQUIRE conditional-expr

A conditional expression that describes the constraint to be placed on the records and relations defined by the RSE.

DEFINE CONSTRAINT Statement

CHECK ON COMMIT

CHECK ON UPDATE [Default]

Specifies whether the constraint is to be evaluated when you issue a statement such as STORE or MODIFY that updates the relation (UPDATE), or when you issue the COMMIT statement to write the change to the database (COMMIT). The default is UPDATE. You can override this qualifier with the EVALUATING clause of the START_TRANSACTION statement.

Usage Notes

To define a constraint, you must have Rdb/VMS READ access to the database and Rdb/VMS READ and DEFINE access to all relations to which the constraint refers.

If the database is created with the DICTIONARY IS REQUIRED option, you must invoke the database by path name, rather than file name, before you issue this statement.

You cannot define a constraint for a relation or relations while there are active transactions accessing the relations. You must have EXCLUSIVE access to these relations.

Other users are allowed to be attached to the database when you issue the DEFINE CONSTRAINT statement.

You can define a constraint only after you have invoked the database. You must execute this statement in a read/write transaction. If you issue this statement when there is no active transaction, Rdb/VMS starts a read/write transaction implicitly.

Attempts to define a constraint will fail if that constraint or its affected relation is involved in a query at the same time. Users will have to detach from the database with a FINISH statement before you can define the constraint. When Rdb/VMS first accesses an object, such as the relation, a lock is taken out on that object and not released until the user exits the database. Attempts to update this object will get a LOCK CONFLICT ON CLIENT message due to the other user's optimized access to the object.

Similarly, while you define a constraint, users cannot execute queries involving that constraint until you have completed the transaction with a COMMIT or ROLLBACK statement for the DEFINE statement. The user will receive a LOCK CONFLICT ON CLIENT error message. While DDL operations are performed, normal data locking mechanisms are used against system relations. (System relations hold information on objects in the database.) Therefore,

DEFINE CONSTRAINT Statement

attempts to update an object will lock out attempts to query that object. These locks are held until the COMMIT or ROLLBACK of the DDL operation.

The WAIT/NOWAIT clause of the START_TRANSACTION statement does not affect attempts to update metadata with simultaneous queries. Even if you specify START_TRANSACTION WAIT for the metadata update transaction, you will get the following error message if a lock conflict exists:

```
%RDB-E-LOCK_CONFLICT, request failed due to locked resource; no-wait
parameter specified for transaction
-RDB-E-NO_META_UPDATE, metadata update failed
-RDMS-F-LCKCNFLCT, lock conflict on client
RDO>
```

However, a user's query will wait for a metadata update to complete with a ROLLBACK or COMMIT, even if the user specified NOWAIT on the START_TRANSACTION statement.

Rdb/VMS evaluates constraints by validating existing data against the RSE specified by the constraint. If there is no existing data for Rdb/VMS to validate, the constraint will be defined without being evaluated.

Rdb/VMS evaluates constraints at definition time; therefore, you cannot define a new constraint that violates an existing constraint.

You cannot execute the DEFINE CONSTRAINT statement when the RDB\$SYSTEM storage area is set to read-only. You must first set RDB\$SYSTEM to read/write. See the *VAX Rdb/VMS Guide to Database Maintenance and Performance* for more information on the RDB\$SYSTEM storage area.

Examples

Example 1

A constraint can check for the existence of a field value in another relation.

```
DEFINE CONSTRAINT DEPT_CODE_EXISTS
  FOR JH IN JOB_HISTORY
  REQUIRE ANY D IN DEPARTMENTS WITH
  D.DEPARTMENT_CODE = JH.DEPARTMENT_CODE.
```

The ANY operator is equivalent to saying "there exists." This constraint therefore means "For every record in the JOB_HISTORY relation, require that there exists a record in the DEPARTMENTS relation where the DEPARTMENT_CODE values match."

DEFINE CONSTRAINT Statement

Example 2

A constraint can also check that required fields are present.

```
DEFINE CONSTRAINT EMPLOYEE_ID_REQUIRED  
  FOR E IN EMPLOYEES  
  REQUIRE E.EMPLOYEE_ID NOT MISSING.
```

This constraint makes sure that an `EMPLOYEE_ID` value is stored for every record in the `EMPLOYEES` relation.

9.14 DEFINE DATABASE Statement

Creates database files, specifies a name for the database, and determines the physical characteristics of the database. If the data dictionary is installed on your system, you can use the DEFINE DATABASE statement to create an entity in the data dictionary where definitions of other database elements can be stored.

In its simplest form, the DEFINE DATABASE statement creates a single-file database. When the statement executes, Rdb/VMS creates the following files:

- A database (RDB) file that contains root file information as well as data
- A snapshot (SNP) file

If you use the DEFINE DATABASE statement to create a multifile database, Rdb/VMS creates the following files:

- A database (RDB) file that contains root file information
- Storage area (RDA) files that contain data
- Snapshot (SNP) files for the RDB file and for each RDA file

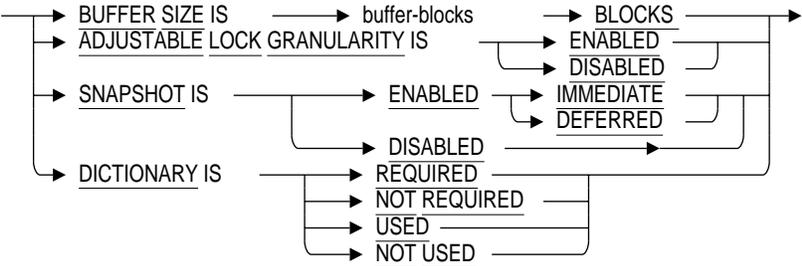
The presence or absence of a DEFINE STORAGE AREA clause within the DEFINE DATABASE statement determines whether the database is single file or multifile.

When the DEFINE DATABASE statement executes, Rdb/VMS:

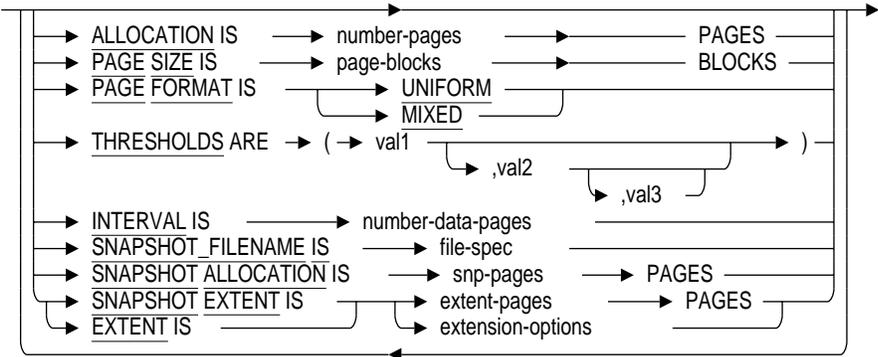
- Creates a database file, snapshot file, and, if the database is multifile, storage area files
- Creates a dictionary entity for the database, including all the system fields and relations, if VAX CDD/Plus software is installed and you did not specify the DICTIONARY IS NOT USED clause
- Assigns database and snapshot file parameters, and if the database is multifile, storage area parameters
- Creates a default access control list (ACL)
- Invokes the newly created database, using the database file name as the database handle

DEFINE DATABASE Statement

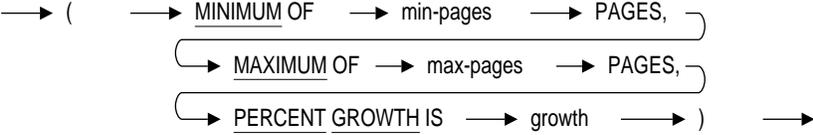
db-wide-options-2 =



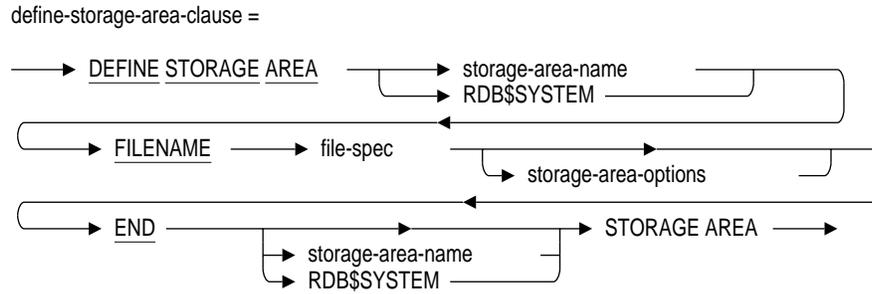
storage-area-options =



extension-options =



DEFINE DATABASE Statement



Defaults

Table 9–2 lists the defaults for the DEFINE DATABASE statement.

Table 9–2 Defaults for DEFINE DATABASE Statement

Parameter	Default
NUMBER OF USERS	50 users
NUMBER OF BUFFERS	20 buffers
NUMBER OF VAXCLUSTER NODES	16 nodes
COLLATING SEQUENCE	ASCII
NUMBER OF RECOVERY BUFFERS	20 buffers
BUFFER SIZE	3 times the page size parameter
ADJUSTABLE LOCK GRANULARITY	ENABLED
SNAPSHOT IS	ENABLED IMMEDIATE
DICTIONARY IS	NOT REQUIRED
DICTIONARY IS	USED
ALLOCATION	400 pages
PAGE SIZE	2 blocks
PAGE FORMAT	UNIFORM

(continued on next page)

DEFINE DATABASE Statement

Table 9–2 (Cont.) Defaults for DEFINE DATABASE Statement

Parameter	Default
THRESHOLDS	70, 85, 95 percent
INTERVAL	256 data pages
EXTENT	100 pages
SNAPSHOT EXTENT	
MINIMUM	100 pages
MAXIMUM	10,000 pages
PERCENT GROWTH	20 percent
SNAPSHOT ALLOCATION	100 pages

Arguments

file-spec

A file specification that names the database files. Put this file specification in quotation marks. Use either a full or partial file specification, or a logical name. If you use a simple file name, Rdb/VMS creates the database in the current default directory. In a single-file database, the file name specified becomes the file name for two files:

- The database root file. The default file type is RDB.
- The snapshot file. This file is used as a temporary file for read-only transactions. The default file type is SNP.

Type the name of the file-spec in uppercase letters when you define your database if you use the CDD/Plus data dictionary or may use it in the future. If you type the name of the file-spec in lowercase letters and try to use the DEFINE GENERIC command of the dictionary's CDO utility to create a directory name for the database in the dictionary, the command will fail.

In a multifile database, the file name specified becomes the file name for three files:

- The database root file. The default file type is RDB.
- The default storage area file. The default file type is RDA.
- The snapshot file. The default file type is SNP.

DEFINE DATABASE Statement

db-wide-options-1

db-wide-options-2

Characteristics of the database root file, in a single-file or multifile database. The exception to this is the SNAPSHOT IS ENABLED option. In a multifile database, the SNAPSHOT IS ENABLED option applies to *all* snapshot files associated with the storage area files. Thus, when you enable snapshots, you do so for *all* the storage area files.

Specify database-wide options to override the Rdb/VMS system defaults for a database.

storage-area-options

Storage area options apply to areas defined in the current DEFINE DATABASE statement. You can specify most storage area options for either single-file or multifile databases. However, the effects of these options differ, depending on whether the database is single file or multifile:

- In a multifile database, storage area options apply to storage areas.
- In a single-file database, storage area options apply to the database root file.

In a multifile database, if you specify storage area options outside a DEFINE STORAGE AREA clause, they serve as global defaults for the default storage area, RDB\$SYSTEM, and for any other storage areas you define within the DEFINE DATABASE statement. However, you can override these defaults by specifying a storage option within the DEFINE STORAGE AREA clause for the named storage area.

The three exceptions to this scheme of global defaults follow:

- If you specify a global default of MIXED page format, Rdb/VMS does not apply this default to the RDB\$SYSTEM storage area. RDB\$SYSTEM will be created with a UNIFORM page format.
- If you specify a global default for the INTERVAL or THRESHOLDS option, this default is simply ignored for storage areas that have a UNIFORM page format. The INTERVAL and THRESHOLDS options apply only to storage areas with a MIXED page format.
- The SNAPSHOT_FILENAME option will not serve as a global default for all storage areas when you put it outside the DEFINE STORAGE AREA clause. To control the name and placement of storage area snapshot files in a multifile database, use the SNAPSHOT_FILENAME option in individual DEFINE STORAGE AREA clauses.

DEFINE DATABASE Statement

DB_HANDLE IS db-handle

The name of the variable you will use to refer to the database. Do not use a file name for the db-handle.

DBKEY SCOPE IS COMMIT

During the session of the user who entered the DEFINE DATABASE statement, DBKEY SCOPE IS COMMIT specifies that the dbkey of each record used is guaranteed *not* to change *only* during each transaction. That is, if a record is erased, its dbkey is not reused by another database user until after a COMMIT statement is executed.

DBKEY SCOPE IS FINISH

During the session of this user who entered the DEFINE DATABASE statement, the DBKEY SCOPE IS FINISH option specifies that the dbkey of each record used is guaranteed not to change until this user detaches from the database (usually, by using the FINISH statement).

IN path-name

The data dictionary path name for the entity where the database metadata definitions are stored. Use this qualifier to store the definitions in a path different from the current default path.

Specify either:

- A full data dictionary path name, such as DISK1:[CDD]CORP.EMPS
- A relative data dictionary path name, such as EMPS

If you use a relative path name, the current default directory must include all the path name segments that precede the relative path name. For example, define CDD\$DEFAULT as DISK1:[CDD]CORP and then use the relative path name EMPS, as follows:

```
RDO> SHOW DICTIONARY
The current CDD dictionary is DISK1:[CDD]CORP
RDO> DEFINE DATABASE 'PERSONNEL' IN 'EMPS'.
```

If you do not specify a data dictionary path name for the database, Rdb/VMS appends the database file name to the current default data dictionary directory. The default directory is defined either by the CDD\$DEFAULT logical name or by the SET DICTIONARY statement. This new path name becomes the name of the data dictionary entity for the database definitions.

DEFINE DATABASE Statement

COLLATING_SEQUENCE IS sequence-name

Specifies a collating sequence to be used for all fields in the database. Sequence-name is a name of your choosing; use this sequence-name in any subsequent statements that refer to this collating sequence.

The VMS NCS utility provides a set of predefined collating sequences and also lets you define collating sequences of your own. The COLLATING_SEQUENCE clause accepts both predefined and user-defined NCS collating sequences.

If you do not specify a collating sequence, the default is ASCII (shown as “no collating sequence” in some displays).

ncs-name

Specifies the name of a collating sequence in the default NCS library, SYSS\$LIBRARY:NCS\$LIBRARY, or in the NCS library specified by the argument library-name. (In most cases, it is probably simplest to make the sequence-name the same as the ncs-name: for example, COLLATING_SEQUENCE IS FRENCH FRENCH.) You can view the collating sequence names by using the command NCS/LIST at DCL level.

The collating sequence can be either one of the predefined NCS collating sequences or one that you have defined yourself using NCS.

DESCRIPTION IS /* text */

Provides a comment for a collating sequence or database being defined.

FROM library-name

Specifies the name of an NCS library other than the default. The default NCS library is SYSS\$LIBRARY:NCS\$LIBRARY.

NUMBER OF USERS IS number-users

The maximum number of users allowed to access the database at one time. The default is 50 users. After the maximum number is reached, the next user who tries to invoke the database receives an error message and must wait. The largest number of users you can specify is 2032 and the fewest number of users is 1. Note that number of users is defined as the number of active attaches to the database. Thus, if a single process is running one program, but that program performs 12 concurrent INVOKE operations, to Rdb/VMS there are 12 active users.

NUMBER OF BUFFERS IS number-buffers

The number of buffers Rdb/VMS allocates per process using this database. Specify an unsigned integer between 2 and 32768. The default is 20 buffers.

DEFINE DATABASE Statement

NUMBER VAXCLUSTER NODES IS number-nodes

Sets the upper limit on the maximum number of VAXcluster nodes from which users can access the shared database. The default is 16 nodes. The range is 1 node to 64 nodes. The actual maximum limit is the current VMS VAXcluster limit.

NUMBER RECOVERY BUFFERS IS recovery-buffers

The number of database buffers used during the automatic recovery process that is initiated after a system or process failure. This recovery process uses the recovery-unit journal file. Specify an unsigned integer between 2 and 32768. The default is 20 buffers.

BUFFER SIZE IS buffer-blocks

The number of blocks Rdb/VMS allocates per buffer. Specify an unsigned integer greater than zero. If you do not specify this parameter, Rdb/VMS uses a buffer size that is three times the PAGE SIZE value.

Buffer size is a global parameter and the number of blocks per page (or buffer) is constrained to less than 64 blocks per page. The page size can vary by storage area for multifile databases, so you should determine the page size of each storage area based on the sizes of records that will be stored in each storage area.

When you choose the number of blocks per buffer, choose a number that is wholly divisible by all page sizes for all storage areas in your multifile database. For example, if your database has three storage areas with page sizes of 2, 3, and 4 blocks respectively, choosing a buffer size of 12 blocks will ensure optimal buffer utilization. If you choose a buffer size of 8, the storage area with a page size of 3 blocks will waste 2 blocks per buffer. Rdb/VMS reads as many pages as will fit into the buffer. In this case, Rdb/VMS reads two pages of 3 blocks apiece into the buffer, wasting 2 blocks.

ADJUSTABLE LOCK GRANULARITY IS ENABLED [Default]

ADJUSTABLE LOCK GRANULARITY IS DISABLED

Enables or disables adjustable locking granularity. Generally, enabling adjustable locking granularity results in fewer locks being used. However, when contention for database pages is high, performance may be impaired as locking granularity is adjusted to a lower level. If your application is query intensive, enable adjustable locking granularity. If your application processes specific records and performs a substantial number of update operations, you might want to disable adjustable locking granularity.

DEFINE DATABASE Statement

Disabling adjustable locking granularity may require that the VMS SYSGEN parameters for locks be increased.

For more information, see the *VAX Rdb/VMS Guide to Database Maintenance and Performance*.

SNAPSHOT IS ENABLED IMMEDIATE [Default]

SNAPSHOT IS ENABLED DEFERRED

The SNAPSHOT IS ENABLED IMMEDIATE option specifies that read/write transactions write copies of records to the snapshot file before those records are modified, regardless of whether a read-only transaction is active. The default is SNAPSHOT IS ENABLED IMMEDIATE.

The SNAPSHOT IS ENABLED DEFERRED option specifies that read/write transactions *not* write copies of records they modify to the snapshot file unless a read-only transaction is active. Read-only transactions that attempt to start after an active read/write transaction begins must wait for all active read/write users to complete their transactions.

You enable snapshot writing to all snapshot files for all storage areas when you specify the SNAPSHOT IS ENABLED clause.

SNAPSHOT IS DISABLED

Specifies that snapshot writing be disabled. You disable snapshot writing to all snapshot files for all storage areas when you specify the SNAPSHOT IS DISABLED clause.

Do not delete snapshot files unless you also delete the database itself. If you do not want snapshots enabled, disable them with the SNAPSHOT IS DISABLED clause.

DICTIONARY IS REQUIRED

DICTIONARY IS NOT REQUIRED [Default]

Determines whether the database must be invoked by path name for data definition changes to occur. If you specify the DICTIONARY IS REQUIRED option, the database must be invoked by path name to change metadata and the data dictionary will be maintained. If you specify the DICTIONARY IS NOT REQUIRED option, the database can be invoked by either file name or path name to change metadata.

DICTIONARY IS USED [Default]

DICTIONARY IS NOT USED

Determines whether the definition of the database and definitions of database elements will be stored in the data dictionary. If you specify the DICTIONARY

DEFINE DATABASE Statement

IS USED option, the definition of the database and definitions of database elements will be stored in the data dictionary. If you specify the DICTONARY IS NOT USED option, no definitions will be stored in the data dictionary.

You will receive an error message if you specify incompatible options, such as the DICTONARY IS REQUIRED clause and the DICTONARY IS NOT USED clause.

ALLOCATION IS number-pages

The number of database pages allocated to the database initially. Rdb/VMS automatically extends the allocation to handle the loading of data and subsequent expansion. The default is 400 pages. If you are loading a large database, a large allocation prevents the file from having to be extended many times.

Rdb/VMS allocates pages in groups of three, so if you specify that two pages be allocated for a storage area, for example, Rdb/VMS allocates three pages instead. Rdb/VMS also allocates a space area management (SPAM) entry page initially for the storage area. Therefore, because Rdb/VMS allocates pages in groups of three and also allocates a SPAM page, a total of four pages is allocated for the storage area even though you requested an allocation of only two pages.

PAGE SIZE IS page-blocks

The size in blocks of each database page. Page size is allocated in 512-byte blocks. The default is 2 blocks (1024 bytes). If your largest record is larger than approximately 950 bytes, allocate more blocks per page to prevent records from being fragmented.

The page size you specify must be smaller than the buffer size specified for the database or you will receive an error message.

PAGE FORMAT IS UNIFORM [Default]

PAGE FORMAT IS MIXED

Specifies whether a storage area contains UNIFORM or MIXED pages. You can use the PAGE FORMAT option with multifile databases only. In storage areas with UNIFORM page format, all pages in a specific logical area contain records from the same relation. In storage areas with MIXED page format, pages can hold records from different relations.

The default storage area, RDB\$SYSTEM, must have UNIFORM pages.

DEFINE DATABASE Statement

THRESHOLDS ARE (val1, val2, val3)

Specifies one, two, or three threshold values. The threshold values represent a fullness percentage on a data page and establish four possible ranges of guaranteed free space on the data pages. When a data page reaches the percentage defined by a given threshold value, the space area management (SPAM) entry for the data page is updated to reflect the new fullness percentage and its remaining free space.

The default thresholds are 70, 85, and 95 percent. If you specify only one or two values, unspecified values default to 100 percent. You can specify the THRESHOLDS option only for a storage area for a multifile database. The storage area page format must be MIXED.

INTERVAL IS number-data-pages

The number of data pages between SPAM pages in the physical storage area file, and thus the maximum number of data pages each SPAM page will manage. The default, and also the minimum interval, is 256 data pages. The first page of each storage area is a SPAM page. The interval you specify determines where subsequent SPAM pages are to be inserted, provided there are enough data pages in the storage file to require more SPAM pages.

The maximum SPAM interval you can specify is 4008 data pages for a database with a 2-block page size. Specifying a value greater than 4008 data pages for the SPAM interval causes data corruption.

You can specify the INTERVAL option only for a storage area for a multifile database. The storage area page format must be MIXED.

See the *VAX Rdb/VMS Guide to Database Maintenance and Performance* for information on formulas for calculating maximum and minimum SPAM intervals.

EXTENT IS extent-pages

SNAPSHOT EXTENT IS extent-pages

The number of pages of each extent. Use this parameter for simple control over the extent. For greater control, and particularly for multivolume databases, use the MIN, MAX, and PERCENT parameters instead. The default is 100 pages.

min-pages

The minimum number of pages of each extent. The default is 99 pages.

max-pages

The maximum number of pages of each extent. The default is 9,999 pages.

DEFINE DATABASE Statement

growth

The percent growth of each extent. The default is 20 percent growth.

SNAPSHOT_FILENAME IS file-spec

Provides a separate file specification for the snapshot file. Do not specify a file type other than SNP. Rdb/VMS will assign the extension SNP to the file specification, even if you specify an alternate extension. If you do not specify this clause for a storage area, the SNP file is automatically placed in the same directory as the storage area file and has the same name as the storage area file. You cannot specify a global default for the snapshot file name. Thus, in a multifile database, the SNAPSHOT_FILENAME option must be within a DEFINE STORAGE AREA clause.

The SNAPSHOT_FILENAME option cannot be specified for a single-file database.

SNAPSHOT ALLOCATION IS snp-pages

The number of pages allocated for the snapshot file. The default is 100 pages.

storage-area-name

The name of the storage area you want to create. You use this name to refer to the storage area in other statements. When you choose a name, follow these rules:

- Use a name that is unique among all storage area names in the database.
- Use any valid VMS name. However, the name cannot end in a dollar sign (\$) or underscore (_).
- Do not use any Rdb/VMS reserved words (see Appendix A).

RDB\$SYSTEM

The default storage area. If you directly specify RDB\$SYSTEM in the DEFINE STORAGE AREA clause, you can override the default characteristics for the main storage area. If you do not directly specify a DEFINE STORAGE AREA clause for RDB\$SYSTEM, the default storage area:

- Will be created with the same file name as the database file name
- Will have the Rdb/VMS storage area defaults, unless you specify storage area options globally, outside of a DEFINE STORAGE AREA clause

The exception to this is that even if you specify a global default of MIXED page format, Rdb/VMS does not apply this default to the RDB\$SYSTEM storage area. RDB\$SYSTEM will be created with UNIFORM page format.

DEFINE DATABASE Statement

The RDB\$SYSTEM storage area contains:

- System relations
- Relations not associated with other storage areas through DEFINE STORAGE MAP statements
- Relations explicitly associated with RDB\$SYSTEM through DEFINE STORAGE MAP statements
- Segmented strings, unless you assign a separate storage area for them with the SEGMENTED STRING STORAGE AREA clause

RDB\$SYSTEM must have a UNIFORM page format.

FILENAME file-spec

The storage area file that is associated with the named storage area. By default, this file has the default file type of RDA. Put this file specification in quotation marks. Use either a full or partial file specification, or a logical name. If you use a simple file name, Rdb/VMS creates the storage area file in the current default directory.

Use a name that is unique among all storage area file names defined for the database.

SEGMENTED STRING STORAGE AREA

The name of the storage area that will hold all segmented strings. For a single-file database or multfile database, if you do not explicitly define a storage area for segmented strings, segmented strings will be stored in the default storage area, RDB\$SYSTEM. If your database is a single-file database and you specify a storage area other than RDB\$SYSTEM, you will receive an error message because RDB\$SYSTEM is the only storage area in a single-file database.

The page format for the segmented string storage area can be UNIFORM or MIXED. However, Digital Equipment Corporation recommends that if you store segmented strings in a MIXED storage area, that area contain *only* segmented strings.

DEFINE DATABASE Statement

Usage Notes

By using the RDBVMS\$CREATE_DB logical name and the RDBVMS\$CREATE_DB identifier, you can restrict the ability of users to create databases on your system. For more information on the RDBVMS\$CREATE_DB logical name and identifier, see the chapter on defining database protection in the *VAX Rdb/VMS Guide to Database Design and Definition*.

If your system does not use the RDBVMS\$CREATE_DB logical name and identifier, all users on the system have the ability to create databases.

A table indicating default, minimum, and maximum values for database-wide parameters can be found in Appendix D. The same values for storage area parameters can be found in Appendix E.

In the following two cases, global defaults for multifile databases will be ignored:

- If you specify a global default of MIXED page format for all storage areas defined in the DEFINE DATABASE statement, Rdb/VMS does not apply this default to the RDB\$SYSTEM storage area. RDB\$SYSTEM will be created with UNIFORM page format.
- If you specify a global default for the INTERVAL or THRESHOLDS option, these defaults are ignored for storage areas that have a UNIFORM page format. The INTERVAL and THRESHOLDS options apply only to storage areas with a MIXED page format.

The order of precedence for storage area options is:

- 1 Local use of the option, within a DEFINE STORAGE AREA clause
- 2 Global (database-wide) use of the option, outside a DEFINE STORAGE AREA clause, but within the DEFINE DATABASE statement
- 3 The Rdb/VMS default for that option, if it is not specified locally or globally

You cannot issue a DEFINE DATABASE statement when a transaction is active. It is recommended that you make the DEFINE DATABASE statement the first statement of an RDO session.

Do not issue an INVOKE DATABASE statement after you use the DEFINE DATABASE statement. Rdb/VMS invokes the database automatically after it creates the database file.

DEFINE DATABASE Statement

Use system-wide concealed logical names instead of physical disk device names for file specifications in the DEFINE DATABASE statement. This will save you work if you later use the IMPORT statement on the database. When you use the IMPORT statement, Rdb/VMS tries to create storage areas with storage area and snapshot file specifications exactly as stored in the interchange (RBR) file. If those file specifications include disk devices or directory names that do not exist, the IMPORT operation will fail unless you specify DEFINE STORAGE AREA clauses with new file specifications for all storage area and snapshot files in the database. You can avoid this problem if you use concealed logical names to refer to the disk device and directory names in file specifications. Before you issue the IMPORT statement, redefine the logical names to refer to the appropriate disk device and directory names where you plan to import the database. For more information on using concealed logical names, see the *VAX Rdb/VMS Guide to Database Maintenance and Performance*.

If you use the SNAPSHOT IS ENABLED clause to enable snapshots on a multiframe database, writing to all snapshot files for all storage areas is enabled. If you use the SNAPSHOT IS DISABLED clause, writing to all snapshot files is disabled.

If you define a database in RDO, do not spawn a subprocess to run VAX DATATRIEVE and attempt to ready the same database through DATATRIEVE. Because the DEFINE DATABASE statement in RDO causes the current process to maintain locks, you must finish or commit the transaction before you spawn the DATATRIEVE subprocess.

When a database is created, the default is that the database can be opened automatically by any user. That is, any user can open a closed or previously unopened database by simply invoking it and executing a data manipulation language statement. This default can be changed by using the OPEN IS {AUTOMATIC | MANUAL } clause of the CHANGE DATABASE statement.

The only user allowed to be attached to the database when the DEFINE DATABASE statement is issued is the user who issues the statement.

Examples

Example 1

This example shows a statement that creates the single-file database, ACCT. The statement does the following:

- Creates the database file DISK1:[SMITH.RDB]ACCT.RDB

DEFINE DATABASE Statement

- Creates the snapshot file DISK1:[SMITH.RDB]ACCT.SNP
- Sets the allocation of the snapshot file to 200 pages
- Enables writing to the snapshot file.

Because no further option was specified, the default IMMEDIATE option will be used.

```
RDO> DEFINE DATABASE 'DISK1:[SMITH.RDB]ACCT'  
cont>     SNAPSHOT ALLOCATION IS 200 PAGES  
cont>     SNAPSHOT IS ENABLED.
```

Example 2

This statement creates a single-file database, disables snapshot writing, and sets the initial allocation page size to 1. This saves disk space.

```
RDO> DEFINE DATABASE "DISK2:[USER.TEST]PERSONNEL"  
cont>     SNAPSHOT IS DISABLED  
cont>     SNAPSHOT ALLOCATION IS 1.
```

Example 3

The following example defines a PERSONNEL database with the following characteristics:

- NUMBER OF VAXCLUSTER NODES is set to 20
Because the database will reside on a shared disk in a 20-node local area VAXcluster, increase this parameter from the default 16 nodes to 20 nodes.
- SNAPSHOT IS ENABLED DEFERRED
Because short read/write transactions occur frequently, but it is rare that many read-only users access the database when there are active read/write transactions, enable snapshots, but set the DEFERRED option.

```
RDO> DEFINE DATABASE "WORKDISK1:[TOP.DB]PERSONNEL"  
cont> IN "DISK1:[DICTIONARY.PERS]CORP"  
cont> NUMBER VAXCLUSTER NODES IS 20  
cont> SNAPSHOT IS ENABLED DEFERRED.
```

Example 4

The following example shows the definition of a database named LITERATURE. The FRENCH collating sequence (as defined in the NCS library) is to be used by default in controlling the behavior of sorting and Boolean operations.

DEFINE DATABASE Statement

```
RDO> DEFINE DATABASE LITERATURE
cont> COLLATING_SEQUENCE IS FRENCH FRENCH
RDO> DICTIONARY IS NOT USED.
RDO> SHOW COLLATING_SEQUENCE
User Collating Sequences in Database with db_handle  LITERATURE
      FRENCH
```

You can apply other collating sequences to specific global fields by performing the following operations: use the `DEFINE COLLATING_SEQUENCE` statement for each additional collating sequence you wish to use with this database, and use the `COLLATING_SEQUENCE IS` clause with the `DEFINE FIELD` or `CHANGE FIELD` statement for specific fields.

Example 5

The following RDO command procedure shows a part of a multifile database definition. It does not include all the storage area definitions for the database. This partial definition does the following:

- Defines database-wide characteristics
- Defines global storage area defaults
- Specifies local attributes for `RDB$SYSTEM`, the default storage area
- Defines a storage area for segmented strings
- Defines other storage areas

```
DEFINE DATABASE 'DB_DISK:MULTI_PERS'
! Define database-wide characteristics
  DESCRIPTION IS /* Sample multifile definition */
  NUMBER OF USERS IS 60
  NUMBER OF VAXCLUSTER NODES IS 22
  NUMBER OF RECOVERY BUFFERS IS 200
  DICTIONARY IS NOT USED
! Define global storage area characteristics
  ALLOCATION IS 500 PAGES
  PAGE FORMAT IS MIXED
! Specify local attributes for the default storage area
! Override the global default of MIXED page format
  DEFINE STORAGE AREA RDB$SYSTEM
    FILENAME 'DISK1:PERS_DEFAULT'
    PAGE FORMAT IS UNIFORM
    ALLOCATION IS 300 PAGES
    SNAPSHOT_FILENAME IS 'DISK2:PERS_DEFAULT'
  END RDB$SYSTEM STORAGE AREA
```

DEFINE DATABASE Statement

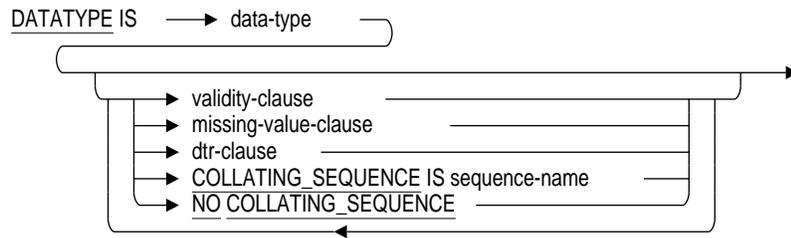
```
! Define storage area for segmented strings
  DEFINE STORAGE AREA PERS_SEGSTR
    FILENAME 'DISK1:PERS_SEGSTR'
    PAGE FORMAT IS UNIFORM
  END PERS_SEGSTR STORAGE AREA
  SEGMENTED STRING STORAGE AREA IS PERS_SEGSTR
! Definition of some sample storage areas
  DEFINE STORAGE AREA CANDIDATES
    FILENAME 'DISK3:CANDIDATES'
    PAGE FORMAT IS UNIFORM
    SNAPSHOT_FILENAME IS 'DISK4:CANDIDATES'
  END CANDIDATES STORAGE AREA

  DEFINE STORAGE AREA EMPIDS_LOW
    FILENAME 'DISK5:EMPIDS_LOW'
    SNAPSHOT_FILENAME IS 'DISK6:EMPIDS_LOW'
  END EMPIDS_LOW STORAGE AREA

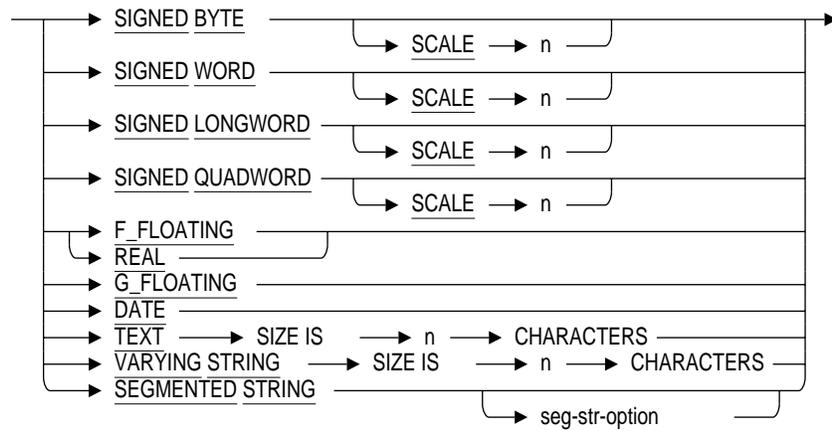
  DEFINE STORAGE AREA EMPIDS_MID
    FILENAME 'DISK7:EMPIDS_MID'
    SNAPSHOT_FILENAME IS 'DISK8:EMPIDS_MID'
  END EMPIDS_MID STORAGE AREA.
```


DEFINE FIELD Statement

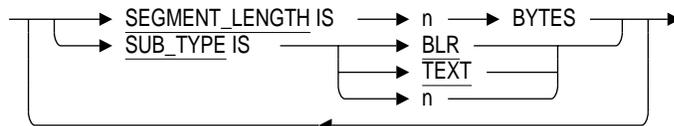
field-attributes =



data-type =



seg-str-option =



DEFINE FIELD Statement

validity-clause =

→ VALID IF → conditional-expr →

missing-value-clause =

→ MISSING_VALUE IS → fxd-pnt-num
→ quoted-string →

dtr-clause =

→	<u>QUERY_HEADER FOR</u>	→ DTR →	→ IS →	→ quoted-string →
		→ DATATRIEVE →	→ / →	→
		→	→	→
		→	→	→
→	<u>QUERY_NAME FOR</u>	→ DTR →	→ IS →	→ quoted-string →
		→ DATATRIEVE →	→ / →	→
→	<u>DEFAULT_VALUE FOR</u>	→ DTR →	→ IS →	→ fxd-pnt-num →
		→ DATATRIEVE →	→ / →	→ quoted-string →
		→	→	→
		→	→	→
→	<u>EDIT_STRING FOR</u>	→ DTR →	→ IS →	→ quoted-string →
		→ DATATRIEVE →	→	→

Arguments

name

The name of the field you want to create. When you choose a name, follow these rules:

- Use a name that is unique among all field names in the database.
- Use any valid VMS name. However, the name cannot end in a dollar sign (\$) or underscore (_).
- Do not use any Rdb/VMS reserved words (see Appendix A).
- Avoid using names that contain the dollar sign (\$) character. By convention, the dollar sign character is reserved for use by Digital Equipment Corporation software.

DEFINE FIELD Statement

FROM PATHNAME path-name

A full or relative data dictionary path name, enclosed in quotation marks, that specifies the source of the field definition. If you use a relative path name, the current default directory must include all the path name segments that precede the relative path name. You must invoke the database by path name, if you plan to copy a shareable field definition from the dictionary. You must specify a Common Dictionary Operator (CDO) utility path name and refer to a field that was created with CDO. You cannot use the FROM PATHNAME clause to specify a field that was created with the RDO DEFINE FIELD statement.

The name you give to the field must match the name of the field in the data dictionary. You cannot rename a shareable field.

text

A text string that adds a comment to the field definition.

field-attributes

A list of definitions that indicate what type of data you can store in the field and how Rdb/VMS uses that data. The field attributes include the data-type clause and the following optional clauses:

- VALID IF
- MISSING_VALUE
- DATATRIEVE support clauses, including:
 - DEFAULT_VALUE FOR DATATRIEVE
 - EDIT_STRING FOR DATATRIEVE
 - QUERY_HEADER FOR DATATRIEVE
 - QUERY_NAME FOR DATATRIEVE
- COLLATING_SEQUENCE
- NO COLLATING_SEQUENCE

The results can be unpredictable if you use multiple, conflicting clauses in field definitions. In the following example, Rdb/VMS recognizes only the second VALID IF clause:

```
RDO> DEFINE FIELD STAR VALID IF STAR > 0 VALID IF STAR < 100.
```

See Chapter 5 for a complete description of field attributes.

DEFINE FIELD Statement

Usage Notes

To define a field using the DEFINE FIELD statement, you must have the Rdb/VMS DEFINE privilege for the database.

If the database is created with the `DICTIONARY IS REQUIRED` option, you must invoke the database by path name, rather than file name, before you issue this statement.

You must execute this statement in a read/write transaction. If you issue this statement when there is no active transaction, Rdb/VMS starts a read/write transaction implicitly.

If you are going to define an index that uses a field defined as `VARYING STRING` or `TEXT`, the number of characters in that field must be less than or equal to 254. (In certain circumstances you may be restricted to less than 254.)

The `COLLATING_SEQUENCE` clause specifies the collating sequence to be used in the field you are defining. Before you use the `COLLATING_SEQUENCE` clause in a DEFINE FIELD statement, you must first specify the NCS collating sequence using the `DEFINE COLLATING_SEQUENCE` statement. The sequence-name argument in the `COLLATING_SEQUENCE` clause must be the same as the sequence-name in the `DEFINE COLLATING_SEQUENCE` statement.

The `NO COLLATING_SEQUENCE` clause specifies that this field will use the standard default collating sequence: that is, ASCII. Use `NO COLLATING_SEQUENCE` to override the collating sequence you have defined for the database using the `DEFINE DATABASE` statement.

Other users are allowed to be attached to the database when you issue the DEFINE FIELD statement.

You cannot execute the DEFINE FIELD statement when the `RDB$SYSTEM` storage area is set to read-only. You must first set `RDB$SYSTEM` to read/write. See the *VAX Rdb/VMS Guide to Database Maintenance and Performance* for more information on the `RDB$SYSTEM` storage area.

DEFINE FIELD Statement

Examples

Example 1

This statement gives a name to the field and specifies its data type.

```
DEFINE FIELD STANDARD_ID_NUMBER
    DATATYPE IS SIGNED LONGWORD.
```

Example 2

This definition provides validation criteria. When a user tries to enter a value other than "M" or "F" in a field that refers to this definition, Rdb/VMS returns an error message.

```
DEFINE FIELD SEX
    DATATYPE IS TEXT SIZE IS 1
    VALID IF SEX = "M" OR SEX = "F".
```

Example 3

This example copies the shareable field, STANDARD_DATE, from the dictionary.

```
DEFINE FIELD STANDARD_DATE
    FROM PATHNAME 'DISK1:[CDDPLUS.DEFS]PERS.STANDARD_DATE'.
```

Example 4

This example adds a DATATRIEVE edit string and default value to the name and data type.

```
DEFINE FIELD SALARY_AMOUNT
    DATATYPE IS SIGNED LONGWORD
    DEFAULT_VALUE FOR DATATRIEVE IS 0
    EDIT_STRING FOR DATATRIEVE IS "$$$$, $$9.99".
```

Example 5

The following example defines a field with the predefined NCS collating sequence SPANISH. Note that you must first execute the DEFINE COLLATING_SEQUENCE statement.

DEFINE FIELD Statement

```
RDO> INVOKE DATABASE FILENAME 'MF_PERSONNEL'  
RDO> DEFINE COLLATING_SEQUENCE SPANISH SPANISH.  
%RDO-W-NOCCDDUPDAT, database invoked by filename, the CDD will not be updated  
RDO> DEFINE FIELD LAST_NAME_SPANISH  
cont> DATATYPE IS TEXT SIZE IS 12 CHARACTERS  
cont> COLLATING_SEQUENCE IS SPANISH.  
RDO> SHOW FIELD LAST_NAME_SPANISH  
      LAST_NAME_SPANISH          text size is 12  
                                  Collating Sequence SPANISH
```

9.16 DEFINE INDEX Statement

Creates an index for a relation. An index allows Rdb/VMS direct access to the records in the relation, to avoid sequential searching. You can define either a sorted index or a hashed index. A sorted index uses the B-tree method of retrieval. A hashed index uses hash addressing for exact match retrievals.

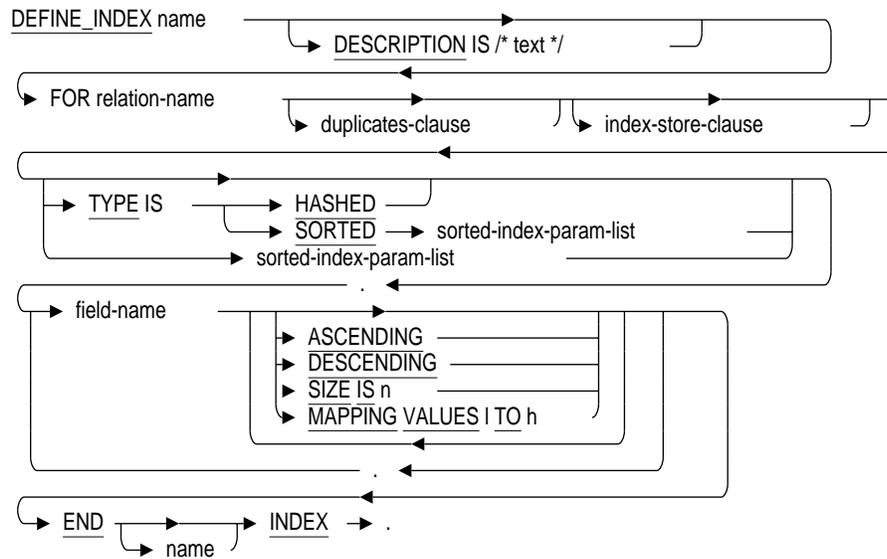
Optional arguments to the DEFINE INDEX statement let you specify:

- Physical characteristics of a sorted index structure, such as index node size, the initial fullness percentage of each node, and whether the index is primarily for update-intensive applications or query-intensive applications
- The type of index structure (sorted or hashed)
- Compression characteristics, including compressed key suffixes for text indexes and integer field compression for word or longword numeric fields
- A storage map for the index records

You define an index by listing the fields in a relation that make up the index. You can define more than one index for a relation, and you can define both types of index (sorted and hashed) for the same relation. The index can be either simple or multisegmented. A simple index is made up of one field; a multisegmented index is made up of two or more fields.

DEFINE INDEX Statement

Format

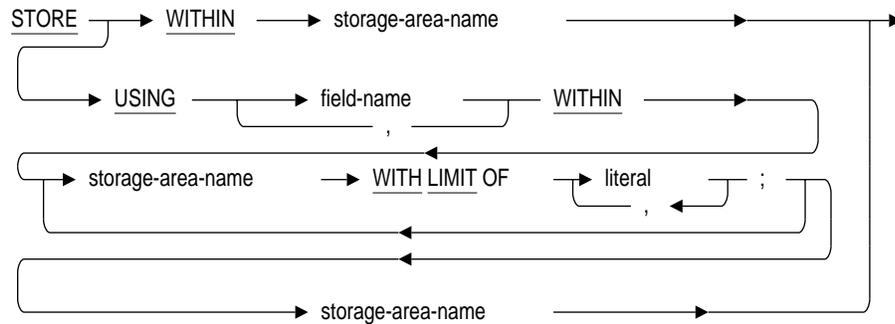


duplicates-clause =

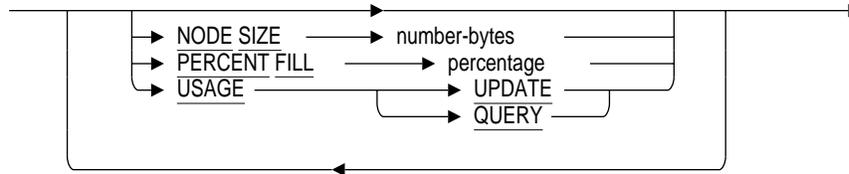


DEFINE INDEX Statement

index-store-clause



sorted-index-param-list =



Defaults

DUPLICATES ARE ALLOWED

TYPE IS SORTED

NODE SIZE number-bytes

If you omit the optional `NODE SIZE` clause, the default value is 430 bytes if the total index key size is 120 bytes or less; 960 bytes if the total index key size is more than 120 bytes.

PERCENT FILL

USAGE UPDATE

The `USAGE UPDATE` clause sets the initial fullness percentage for each node in the index node at 70 percent. Therefore, the `PERCENT FILL` default is also 70 percent.

DEFINE INDEX Statement

Arguments

name

The name of the index. You can use this name to refer to the index in other statements. When you choose a name, follow these rules:

- Use a name that is unique among all user-supplied names in the database.
- Use any valid VMS name. However, the name cannot end in a dollar sign (\$) or underscore (_).
- Do not use any Rdb/VMS reserved words (see Appendix A).

text

A text string that adds a comment to the index definition.

relation-name

The name of the relation that includes the index.

TYPE IS

Specifies whether Rdb/VMS creates a B-tree index structure (SORTED), or a hashed index structure (HASHED). If you specify HASHED, you cannot choose options from the sorted-index-param-list. Sorted indexes provide the best performance for queries that specify a range retrieval. Hashed indexes are effective only for exact match retrievals.

The default index type is SORTED.

field-name

The name of the field or fields that make up the index.

You can create a multisegmented index by naming two or more fields. All the fields must be part of the same relation. Separate multiple field names with periods.

The size of the field must be less than or equal to 255 characters.

If you are going to define an index that uses a field defined as VARYING STRING or TEXT, the number of characters in that field must be less than or equal to 254. (In certain circumstances you may be restricted to less than 254).

ASCENDING [Default]

An optional keyword that causes Rdb/VMS to create ascending index segments. If you omit the ASCENDING or DESCENDING keywords, ASCENDING is the default. To sort records in a particular way in the result of a particular query, specify the sort order in the SORTED BY clause.

DEFINE INDEX Statement

DESCENDING

An optional keyword that causes Rdb/VMS to create descending index segments. To guarantee that records will be sorted in a particular way in the result of a particular query, specify the sort order in the SORTED BY clause.

SIZE is n

A compression clause that specifies that the "first n" characters of a certain key are to be used in the index. These are specified with the DUPLICATES ARE ALLOWED clause. For example, if you wanted to place an index on a 100-byte field that is generally unique to the first 20 bytes, you could specify the first 20 bytes and save as much as 80 bytes per entry.

MAPPING VALUES l TO h

A compression clause for all-numeric fields that translates the field values into a more compactly encoded form. You can mix mapped and unmapped fields, but the most storage space is gained by building indexes of multiple fields of data type WORD or LONGWORD. Rdb/VMS attempts to pack all such fields into the smallest possible space.

l (low) to *h* (high) specifies the range of integers as the value of the index key.

The valid range of the compressed key:

- Cannot be zero
- The range $H - L$ is limited to $(2^{31}) - 4 \times (10^{\text{scale}})$
If the value of the key is less than zero or greater than $(2^{31}) - 4 \times (10^{\text{scale}})$, Rdb/VMS signals an exception.

duplicates-clause

A clause that specifies whether each value of the index key must be unique. If you try to store the same value twice in an indexed field defined as DUPLICATES NOT ALLOWED, Rdb/VMS returns an error message and does not store or modify the record.

index-store-clause

Creates a storage map definition for the index. It allows you to choose which storage area files will be used to store index entries. You can store all index entries for one relation in a single storage area, or you can partition the entries over multiple storage areas. If you do not specify an index-store-clause for a multifile database, the index is stored in the default storage area, RDB\$SYSTEM.

DEFINE INDEX Statement

STORE

The criteria for determining the location to store an index entry. This clause contains a number of subclauses. The clause can be used to spread data over multiple disks and to cause related data to be stored together in the database. If you do not specify the *STORE* clause, the index entries are stored in the default storage area, RDB\$SYSTEM.

USING field-name

The name of the field whose value will be used as a limit for partitioning the index across multiple storage areas. You must respecify this same field later in the index definition as one of the fields that makes up the index.

If the index key is multisegmented, you can include some or all of the fields that are joined in order to form the index key. If you include only a subset of the fields from the multisegmented index, you must specify the fields in the order in which they were specified when the index key was defined. For example, the multisegmented index EMP_FULL_NAME consists of the fields LAST_NAME, FIRST_NAME, and MIDDLE_INITIAL. In the *USING* clause, you can specify the following combinations of fields:

- LAST_NAME
- LAST_NAME and FIRST_NAME
- LAST_NAME, FIRST_NAME, and MIDDLE_INITIAL

You cannot specify only FIRST_NAME, or FIRST_NAME and MIDDLE_INITIAL.

Separate multiple field names with commas.

WITHIN storage-area-name

The name of the storage area in which you want the index stored. You must define this storage area with the DEFINE DATABASE statement before you refer to it in the index-store-clause.

If you define a hashed index, the storage area must have a MIXED page format.

WITH LIMIT OF literal

The maximum value for the index key that will reside in the specified storage area. For multisegmented index keys, specify a literal value for each field.

The number of literals in this clause must be the same as the number of fields in the *USING* clause. Repeat this clause to partition the index entries among multiple storage areas.

DEFINE INDEX Statement

When you are define a multisegmented index using multisegmented keys using the STORE USING . . . WITH LIMITS clauses, if the values for the first key are all the same, then set the limit for the first key at that value. By doing this, you insure that the value of the second key determines the storage area in which each record will be stored.

Note that the last storage area you specify *cannot* have a WITH LIMIT OF clause associated with it.

NODE SIZE number-bytes

The size of each index node. The number and level of the resulting index nodes depend on this value, the number and size of the index keys, and the value of the PERCENT FILL clause.

See the *VAX Rdb/VMS Guide to Database Maintenance and Performance* for information on estimating the valid range for a user-specified index node size.

PERCENT FILL number

Sets the *initial* fullness percentage for each node in the index structure being defined. The valid range is 1 percent to 100 percent. If the PERCENT FILL and USAGE clauses are both specified in the same DEFINE INDEX statement, the USAGE value is used.

USAGE UPDATE

Sets the initial fullness percentage for each node in the index node at 70 percent. Supplying PERCENT FILL and USAGE UPDATE is allowed in the syntax; however, the USAGE option takes precedence over an explicit PERCENT FILL value.

USAGE QUERY

Sets the initial fullness percentage value at 100 percent. Supplying PERCENT FILL and USAGE QUERY clauses together is allowed in the syntax; however, the USAGE option takes precedence over an explicit PERCENT FILL value.

Usage Notes

To define an index for a relation using the DEFINE INDEX statement, you need the Rdb/VMS DEFINE privilege for the relation.

The maximum total length of an index key is 255 bytes. The index key length is computed by adding 1 byte for the null flag indicator and the length of the field for *each* field in the index key. Thus the maximum length of any TEXT or VARYING STRING text field which may be used in an index key is 254. In a multisegmented index you may be restricted to less than 254.

DEFINE INDEX Statement

When the DEFINE INDEX statement executes, Rdb/VMS adds the index definition to the physical database. If you have invoked the database with the PATHNAME qualifier, the definition is also added to the data dictionary.

If you define a sorted index for a relation that contain no data, the root node for the index is not created until the first record is stored. When an RMU/VERIFY operation encounters a sorted index with no root node, it reports the index as empty.

You can define both a hashed index and a sorted index for the same field. Then, depending on the type of query you use, the Rdb/VMS optimizer will choose the appropriate method of retrieval. However, this incurs the additional overhead of maintaining two indexes. See the *VAX Rdb/VMS Guide to Database Design and Definition* for more detailed information on indexes.

If the database is created with the DICTONARY IS REQUIRED option, you must invoke the database by path name, rather than file name, before you issue the DEFINE INDEX statement.

You must execute this statement in a read/write transaction. If you issue this statement when there is no active transaction, Rdb/VMS starts a read/write transaction implicitly. When you define an index you must have exclusive access to the relation affected by this index. However, other users can access other parts of the database not affected by the index definition.

Attempts to define an index will fail if that index or its relation is involved in a query at the same time. Users must detach from the database with a FINISH statement before you can define the index. When Rdb/VMS first accesses an object, such as the relation, a lock is taken out on that object and not released until the user exits the database. Attempts to update this object will get a LOCK CONFLICT ON CLIENT message due to the other user's optimized access to the object.

Similarly, while you define an index, users cannot execute queries involving that index or its relation until you have completed the transaction with a COMMIT or ROLLBACK statement for the DEFINE statement. The user will receive a LOCK CONFLICT ON CLIENT error message. While DDL operations are performed, normal data locking mechanisms are used against system relations. (System relations hold information on objects in the database.) Therefore, attempts to update an object will lock out attempts to query that object. These locks are held until the COMMIT or ROLLBACK of the DDL operation.

DEFINE INDEX Statement

The WAIT/NOWAIT clause of the START_TRANSACTION statement does not affect attempts to update metadata with simultaneous queries. Even if you specify START_TRANSACTION WAIT for the metadata update transaction, you will get the following error message if a lock conflict exists:

```
%RDB-E-LOCK_CONFLICT, request failed due to locked resource; no-wait
parameter specified for transaction
-RDB-E-NO_META_UPDATE, metadata update failed
-RDMS-F-LCKCNFLCT, lock conflict on client
RDO>
```

However, a user's query will wait for a metadata update to complete with a ROLLBACK or a COMMIT statement, even if the user specified NOWAIT on the START_TRANSACTION statement.

You must specify a storage map for a hashed index. That is, the index-store-clause in the DEFINE INDEX statement is required if you specify HASHED in the TYPE IS clause. The storage area that you specify in the index-store-clause must have a MIXED page format.

You cannot define a hashed index in a single-file database. Because hashed indexes require a MIXED page format, you can define a hashed index only in a multifile database.

You cannot execute the DEFINE INDEX statement when the RDB\$SYSTEM storage area is set to read-only. You must first set RDB\$SYSTEM to read/write. See the *VAX Rdb/VMS Guide to Database Maintenance and Performance* for more information on the RDB\$SYSTEM storage area.

The following usage notes pertain to compressed indexes:

- All text compressed indexes require the DUPLICATES ARE ALLOWED clause, which is the default for the DEFINE INDEX statement. If the SIZE IS clause is specified, the field referred to by the clause must be of the TEXT or VARYING TEXT data type. The field must also be the same length or greater in length than the value specified in the SIZE IS clause.
- For integer field compressed indexes, the index field must be of data type WORD or LONGWORD. You can mix mapped and unmapped fields, but the most storage space is gained by building indexes of multiple fields of data type WORD or LONGWORD. Rdb/VMS attempts to pack all such fields into the smallest possible space.
- Compressed key suffixes also enable the user to use fields longer than 254 characters as index keys.

DEFINE INDEX Statement

- If any data values already stored are less than *l* or greater than *h*, the DEFINE INDEX statement will fail.
- A subsequent STORE or MODIFY operation that attempts to store a value less than *l* or greater than *h* will fail.

Examples

Example 1

The following example creates a simple relation index:

```
DEFINE INDEX EMP_EMPLOYEE_ID FOR EMPLOYEES
  DUPLICATES ARE NOT ALLOWED
  TYPE IS SORTED.
  EMPLOYEE_ID.
END EMP_EMPLOYEE_ID INDEX.
```

This statement names the index and the field that serves as the index key.

The DUPLICATES ARE NOT ALLOWED clause causes Rdb/VMS to return an error message if a user attempts to store an identification number (ID) that is already assigned.

Example 2

The following example creates a multisegmented index:

```
DEFINE INDEX EMP_FULL_NAME FOR EMPLOYEES
  DUPLICATES ARE ALLOWED
  TYPE IS SORTED.
  LAST_NAME.
  FIRST_NAME.
  MIDDLE_INITIAL.
END EMP_FULL_NAME INDEX.
```

This statement names three fields. Rdb/VMS concatenates these three fields to make the multisegmented index key.

Example 3

The following example defines the EMP_FULL_NAME index and causes the LAST_NAME segment to be defined in DESCENDING order.

```
DEFINE INDEX EMP_FULL_NAME FOR EMPLOYEES.
LAST_NAME DESCENDING.
FIRST_NAME ASCENDING.
MIDDLE_INITIAL.
END EMP_FULL_NAME INDEX.
```

DEFINE INDEX Statement

Having defined such an index on a particular field does not guarantee that Rdb/VMS will use that index in a particular query retrieval. To ensure a particular sort order is returned by a particular query, you have to specify that order in the RSE:

```
FOR E IN EMPLOYEES
    SORTED BY DESCENDING E.STATUS_CODE,
             ASCENDING E.LAST_NAME,
             DESCENDING E.EMPLOYEE_ID...
```

Example 4

The following example defines the JH_EMPLOYEE_ID index and sets each node size to 350 bytes and the initial fullness percentage of each node to 50 percent:

```
DEFINE INDEX JH_EMPLOYEE_ID FOR JOB_HISTORY
    DUPLICATES ARE ALLOWED
    TYPE IS SORTED
    NODE SIZE 350
    PERCENT FILL 50.
    EMPLOYEE_ID.
END JH_EMPLOYEE_ID INDEX.
```

See the example sections of Section 9.4 and Section 9.50 for related examples.

Example 5

This example defines several text and integer compressed indexes. In the second part of the example, the CITY and STATE fields must be of data type TEXT or VARYING TEXT, and they must be equal to or larger than 5 and 2 characters in length, respectively.

```
RDO> DEFINE INDEX DEGREE_YEAR_GIVEN FOR DEGREES DUPLICATES NOT ALLOWED.
cont> YEAR_GIVEN MAPPING VALUES 1950 TO 2000.
cont> END DEGREE_YEAR_GIVEN INDEX.
%RDB-E-NO_META_UPDATE, metadata update failed
-RDB-E-NO_DUP, index field value already exists; duplicates not
allowed for DEGREE_YEAR_GIVEN
RDO> DEFINE INDEX DEGREE_YEAR_GIVEN FOR DEGREES DUPLICATES ALLOWED.
cont> YEAR_GIVEN MAPPING VALUES 1950 TO 2000.
cont> END DEGREE_YEAR_GIVEN INDEX.
```

DEFINE INDEX Statement

```
RDO> FOR D IN DEGREES SORTED BY D.YEAR_GIVEN PRINT D.* END_FOR
EMPLOYEE_ID COLLEGE_CODE YEAR_GIVEN DEGREE DEGREE_FIELD
00209        MIT          1958   BA      Arts
00237                1962   BA      Arts
00224                1963   BA      Arts
00237        BATE        1963   MA      Applied Math
00187                1966   BA      Arts
00214                1966   BA      Arts
00191                1967   BA      Arts
00221                1967   BA      Arts
00231                1967   BA      Arts
00174        STAN        1969   MA      Applied Math
00197                1969   MA      Elect. Engrg.
.
.
.
```

```
RDO> DEFINE INDEX EMPLOYEES_CITY_STATE FOR EMPLOYEES.
cont> CITY SIZE 5.
cont> STATE SIZE 2.
cont> END EMPLOYEES_CITY_STATE INDEX.
RDO> FOR E IN EMPLOYEES
cont> SORTED BY E.CITY,E.STATE
cont> PRINT E.* END_FOR
```

```
EMPLOYEE_ID LAST_NAME      FIRST_NAME MIDDLE_INITIAL
ADDRESS_DATA_1 ADDRESS_DATA_2
CITY STATE POSTAL_CODE SEX
BIRTHDAY STATUS_CODE
00244 Boyd Ann B
71 Federal Hill Rd.
Acworth NH 03601 F
27-DEC-1943 00:00:00.00 1
00187 Lasch Stan P
59 Mt. Vernon St.
Acworth NH 03601 M
15-AUG-1956 00:00:00.00 1
00197 Danzig Chris
136 Beaver Brook Circle
Acworth NH 03601 F
21-JUN-1939 00:00:00.00 1
00218 Hall Lawrence
161 Pepperell Rd.
Alstead NH 03602 M
10-JUL-1933 00:00:00.00 1
.
.
.
```

DEFINE INDEX Statement

Example 6

The following example defines a hashed index, and uses the index-store-clause to partition the index into different storage areas:

```
DEFINE INDEX EMPLOYEES_HASH
  DESCRIPTION IS /* Hashed index for employees */
  FOR EMPLOYEES
  DUPLICATES ARE NOT ALLOWED
  STORE USING EMPLOYEE_ID
  WITHIN
    EMPIDS_LOW WITH LIMIT OF "00200";
    EMPIDS_MID WITH LIMIT OF "00400";
    EMPIDS_OVER
  TYPE IS HASHED.
  EMPLOYEE_ID.
END EMPLOYEES_HASH.
```

DEFINE PROTECTION Statement

9.17 DEFINE PROTECTION Statement

Adds an entry to the access control list (ACL) for a database, relation, view, or field.

An access control list is attached to each database, relation, view, and field. This list defines which users can access the database element and what operations each user can perform. Thus each entry in the access control list consists of two items of information:

- An identifier that specifies a user or set of users.
- A set of access rights. These rights specify what operations the user or users can perform on the database or relation.

When you first create a database, Rdb/VMS creates two ACL entries (ACEs), one for the creator of the database and one for all other users. Under the Rdb/VMS default protection scheme, the creator of the database is given all access rights to the database, including CONTROL, which enables the creator to grant CONTROL and any other access rights to other users. All other users of the database are given no access rights to the database. For any relations or views created under the Rdb/VMS default protection scheme, the creator of the relation or view is given all the access rights to the object, including CONTROL, and all other users are given no access rights to the object.

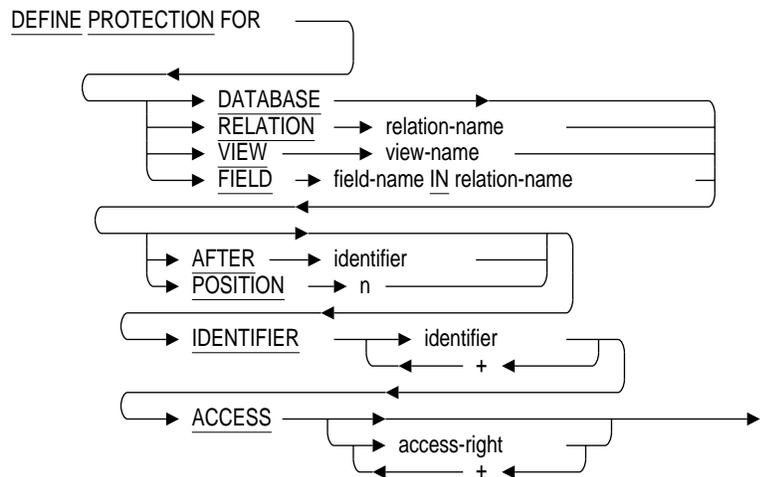
See the Usage Notes for information on how you can tailor the default protection for any new relations and views that are created in a database.

When you issue a DEFINE PROTECTION statement, the new ACL entry (ACE) takes effect after you exit from the database.

For a complete discussion of how to set up the protection on a database, see the *VAX Rdb/VMS Guide to Database Design and Definition*.

DEFINE PROTECTION Statement

Format



Defaults

AFTER identifier

POSITION n

If you do not specify a position or an identifier to locate the new ACL entry, Rdb/VMS places the newly created entry in the first position.

access-right

Rdb/VMS grants only the access rights specified in the DEFINE PROTECTION statement. All other rights are denied.

Arguments

relation-name

The name of the Rdb/VMS relation for which you want to insert an ACL entry.

view-name

The name of the Rdb/VMS view for which you want to insert an ACL entry.

field-name IN relation-name

The name of the local field in a specified relation for which you want to insert an ACL entry.

DEFINE PROTECTION Statement

AFTER identifier

A clause that locates the new ACL entry relative to an existing entry. When you specify an identifier, Rdb/VMS searches the access control list for an existing entry that matches. It then inserts the new entry after the existing one. If you use the AFTER clause, you cannot use the POSITION clause.

POSITION n

A clause that locates a new ACL entry relative to its position in the list. Use an unsigned integer of zero or greater to specify the position in the access control list where Rdb/VMS places a newly created entry. (Specifying zero is the same as using POSITION 1.) If you use the POSITION clause, you cannot use the AFTER clause.

When this statement executes, Rdb/VMS automatically reassigns sequence numbers to entries in the ACL, starting with number one.

identifier

Any valid VMS identifier in the identifier clause:

- UIC identifiers

A VMS user identification code (UIC) that identifies an entry within the ACL. A UIC has the form:

[g,m]

where *g* refers to a group number and *m* is the number of a member of that group. You can replace either or both with the wildcard character (*) to specify a particular member in all groups, all members in a particular group, or all members in all groups. Therefore:

[*,*] means all users

[ADMIN,*] means all users with the group ADMIN

[*,JONES] means all users with the member JONES, in any group

UIC identifiers depend on the user identification codes (UICs) that uniquely identify each user on the system. The UIC can be in either numeric format or alphanumeric format. The following are all valid UIC identifiers:

[SYSTEM3, K_JONES]

K_JONES

[341,311]

DEFINE PROTECTION Statement

You cannot specify more than one UIC identifier in a DEFINE PROTECTION statement.

- General identifiers

General identifiers are defined by the VMS system manager in the system rights database to identify groups of users on the system. The following are possible general identifiers:

DATAENTRY
SECRETARIES
MANAGERS

- System-defined identifiers

System-defined identifiers are automatically defined by the system when the rights database is created at system installation time. The following are all valid system-defined identifiers:

BATCH
NETWORK
INTERACTIVE
LOCAL
DIALUP
REMOTE

Identifiers are assigned depending on the type of login you execute.

For more information about these types of identifiers, see the *Guide to VMS System Security* or the *VMS DCL Dictionary*.

You can specify multiple identifiers in the identifier clause when combining an identifier with a system-defined identifier. However, you should regard the six system-defined identifiers as mutually exclusive. Do not attempt to use them in combination with each other. You can combine them with other identifiers (UICs and general identifiers). When you specify multiple identifiers, separate them with a plus sign (+).

If you specify multiple identifiers in the identifier clause, you define one ACE that determines the access rights that users will receive when they hold *all* the identifiers specified in the identifier clause. Users who do not hold all of the identifiers specified in the identifier clause will not be governed by the ACE that is created.

DEFINE PROTECTION Statement

For example, the first of the two RDO statements in the following example defines a single ACE for users with both the [CLERK,DAVIES] and REMOTE identifiers, not one ACE for users with the [CLERK,DAVIES] identifier and another ACE for users with the REMOTE identifier. The second statement shows the ACE created in the second position in the ACL, as specified in the DEFINE PROTECTION statement:

```
DEFINE PROTECTION FOR RELATION A1
POSITION 2
IDENTIFIER [CLERKS,DAVIES]+REMOTE
ACCESS READ+WRITE+MODIFY.
!
SHOW PROTECTION FOR RELATION A1
  ( IDENTIFIER=[DBS,RICK],ACCESS=READ+WRITE+MODIFY+ERASE+SHOW+DEFINE+CHANGE+
    DELETE+CONTROL+OPERATOR+ADMINISTRATOR+REFERENCES+SECURITY )
  ( IDENTIFIER=[CLERK,DAVIES]+REMOTE,ACCESS=READ+WRITE+MODIFY )
  ( IDENTIFIER=[*,*],ACCESS=READ+MODIFY )
```

To control which users have the ability to create databases, use the RDBVMS\$CREATE_DB logical name and RDBVMS\$CREATE_DB system rights identifier. See the *VAX Rdb/VMS Guide to Database Design and Definition* for more information on the RDBVMS\$CREATE_DB logical name and RDBVMS\$CREATE_DB system rights identifier.

ACCESS access-right

One of a list of Rdb/VMS access rights granted or denied the user identified in an ACL entry.

Access rights grant or deny access to:

- Rdb/VMS data manipulation statements
- Rdb/VMS data definition statements
- Rdb/VMS utility statements

Table 9–3, Table 9–4, and Table 9–5 list the Rdb/VMS access rights. Table 9–6 provides a task-oriented perspective on the privileges needed in the database ACL and the relation ACL to perform typical database tasks.

DEFINE PROTECTION Statement

Table 9–3 Data Manipulation Access Rights

Access Right	To Grant	To Deny
Read data	READ	NOREAD
Store data	WRITE	NOWRITE
Modify data	MODIFY	NOMODIFY
Erase data	ERASE	NOERASE

Table 9–4 DDL and Administrative Statements Controlled by Database ACL

Access Right	To Grant	To Deny
Define global field, relation, or storage map	DEFINE	NODEFINE
Change global field	CHANGE	NOCHANGE
Delete global field	DELETE	NODELETE
Run two-phase commit transaction on database	DISTRIBTRAN	NODISTRIBTRAN
Define, change, delete protection for database	CONTROL	NOCONTROL
[Reserved for future versions]	REFERENCES	NOREFERENCES
Specify, show, and review the audit trail of security events	SECURITY	NOSECURITY
[Reserved for future versions]	SHOW	NOSHOW
Open or close a database	ADMINISTRATOR	NOADMINISTRATOR
Any CHANGE DATABASE option	ADMINISTRATOR	NOADMINISTRATOR
Delete a database	ADMINISTRATOR	NOADMINISTRATOR

DEFINE PROTECTION Statement

Table 9–5 DDL and Administrative Statements Controlled by ACL for Each Relation, Local Field, or View in Statement

Access Right	To Grant	To Deny
Define index or storage map	DEFINE	NODEFINE
Define trigger ¹	READ, DEFINE	NOREAD, NODEFINE
Define view or constraint ²	READ, DEFINE	NOREAD, NODEFINE
Change relation	CHANGE	NOCHANGE
Delete relation, index, view, constraint, or storage map	DELETE	NODELETE
Define, change, delete protection for relation	CONTROL	NOCONTROL
[Reserved for future versions]	SHOW	NOSHOW
[Reserved for future versions]	OPERATOR	NOOPERATOR

¹A trigger requires READ and DEFINE on the subject relation; if the trigger specifies any form of update operation, the trigger requires READ, CONTROL, and the appropriate form of update access (ERASE, MODIFY, or WRITE) to the relations specified by the triggered action statements.

²A view or constraint requires READ and DEFINE access to each relation accessed by the view or constraint.

Table 9–6 Task-Oriented Summary of Required Privileges

To Perform the Following Operation	A User Needs in the Database ACL	A User Needs in the Relation ACL
Attach to a database	READ	Any privilege for the objects accessed
Display definitions	READ	READ on all relations accessed
Read data	READ	READ on all relations accessed
Store data	READ, WRITE	WRITE on relation in which rows are stored

(continued on next page)

DEFINE PROTECTION Statement

Table 9–6 (Cont.) Task-Oriented Summary of Required Privileges

To Perform the Following Operation	A User Needs in the Database ACL	A User Needs in the Relation ACL
Modify data	READ, MODIFY	READ on relation and MODIFY on either the relation or field to be modified
Erase rows	READ, ERASE	READ, ERASE on relation from which rows are erased
Create a relation ¹	READ, DEFINE	Does not apply to relations
Create an index	READ	DEFINE on relation for which index is created
Create a trigger ²	READ	READ, DEFINE on the subject relation
Create a view	READ	READ, DEFINE on all relations to which the view refers
Create a constraint	READ	READ, DEFINE on all relations to which the constraint refers
Create a storage map	READ	DEFINE on the relation to which storage map refers
Change a relation ³	READ	CHANGE on relation whose definition is changed
Change a storage map	READ	CHANGE on relation to which storage map refers
Delete a relation	READ	DELETE on relation being deleted

¹See Section 9.18 for complete information.

²A trigger requires READ and DEFINE on the subject relation; if the trigger specifies any form of update operation, the trigger requires READ, CONTROL, and the appropriate form of update access (ERASE, MODIFY, or WRITE) to the relations specified by the triggered action statements.

³See Section 9.6 for complete information.

(continued on next page)

DEFINE PROTECTION Statement

Table 9-6 (Cont.) Task-Oriented Summary of Required Privileges

To Perform the Following Operation	A User Needs in the Database ACL	A User Needs in the Relation ACL
Delete a view	READ	DELETE on view being deleted
Delete an index	READ	DELETE on relation associated with the index
Delete a database	ADMINISTRATOR	Not applicable
Delete a constraint	READ	DELETE on all relations to which constraint refers
Delete a storage map	READ	DELETE on relation to which storage map refers
Modify the ACL for the database entity	READ, CONTROL	Not applicable
Modify the ACL for a relation or view	READ	CONTROL on the relation or view whose ACL is being modified
Modify the ACL for a field	READ	CONTROL on the relation whose field's ACL is being modified
Use CHANGE DATABASE	ADMINISTRATOR	Not applicable
Use EXPORT	READ	READ on all relations
Use IMPORT	READ	READ on all relations
Use INTEGRATE	READ	READ on all relations and views
Run two-phase commit transaction on database	DISTRIBTRAN	Not applicable
Use RMU/SET AUDIT	SECURITY	Not applicable
Use RMU/SHOW AUDIT	SECURITY	Not applicable
Use RMU/LOAD/AUDIT	SECURITY	Not applicable

DEFINE PROTECTION Statement

Usage Notes

You must have the Rdb/VMS CONTROL privilege for an object to define protection for the object using the DEFINE PROTECTION statement.

Rdb/VMS denies all access rights not explicitly granted. Specify only those access rights that you want to grant.

If you specify two or more access rights, separate each with a plus sign (+), but do not embed any spaces. For example, READ+WRITE.

You can abbreviate keywords in the DEFINE and CHANGE PROTECTION statements.

If you specify the keyword ALL in the access clause, all access rights will be granted. You can use ALL to specify all access rights except for certain ones. For example, you can specify ACCESS = "ALL+NOCONTROL".

If the list of access rights exceeds one line in length, place the list in quotation marks and use the continuation character (-) at the end of the line. Otherwise, Rdb/VMS reads the carriage return as the end of the list, and an error results:

```
cont> ACCESS "DEFINE+CHANGE+DELETE -  
cont> +CONTROL+OPERATOR+ADMINISTRATOR".
```

If you are specifying additional access rights for an identifier that already exists in the ACL, be aware that the existing access control entry (ACE) may be deleted. If you are adding additional access rights to a particular identifier, you may wish to respecify all previously granted access rights in the new ACE.

The SHOW access right is not implemented, but it is displayed if you issue a SHOW PROTECTION statement for a database. Rdb/VMS does not return an error message if you attempt to deny the SHOW access right, and the SHOW access right will not be displayed in the SHOW PROTECTION statement if you attempt to deny the right. If you have the CONTROL access right, there is no way for you to deny yourself that privilege. Rdb/VMS does *not* allow you to perform any of the following tasks:

- Change your access control entry from CONTROL to NOCONTROL.
- Delete your only access control entry with CONTROL.
- Define a new access control entry without CONTROL privilege and place it above your current access control entry. The exception to this is if you have VMS SYSPRV privilege, it *is* then possible to deny yourself the CONTROL access right.

DEFINE PROTECTION Statement

It is not possible for you to deny yourself the READ access right.

For a particular user, Rdb/VMS grants data manipulation access rights to a relation only if those rights are granted in the ACL for *both* the database and the relation. That is, a user has WRITE privilege to the EMPLOYEES relation only if that user has WRITE privilege to both the PERSONNEL database and the EMPLOYEES relation. This means that protection at the database level should grant to each user or group of users all the data manipulation rights they may need for any relation.

The ADMINISTRATOR, OPERATOR, and SECURITY database privileges are the three Rdb/VMS role-oriented privileges. Users with these privileges have the ability to override ACLs for some objects to perform database operations. Similarly, users with certain VMS privileges also have the ability to override ACLs for some database objects. The Rdb/VMS role-oriented privileges are limited to the database in which they are granted, but the VMS privileges span all databases on the system.

Users with the Rdb/VMS or the VMS role-oriented privileges are implicitly granted other Rdb/VMS privileges. When you are granted implicit privileges to a database object as a result of an ACL override, you operate as if you actually hold the privilege, although you are not explicitly granted the privilege and it is not stored in the ACL.

Users with the ADMINISTRATOR database privilege or the VMS SYSPRV privilege can perform any data definition or data manipulation operation on any named object, including the database, regardless of the ACL for the object. The ADMINISTRATOR privilege is the most powerful privilege in Rdb/VMS, because it can override most privilege checks performed by Rdb/VMS. Users with the ADMINISTRATOR database privilege or the VMS SYSPRV privilege implicitly receive ALL privileges for all objects, except the SECURITY and OPERATOR database privileges.

Users with the OPERATOR database privilege or the VMS OPER privilege implicitly receive the Rdb/VMS READ, WRITE, MODIFY, and ERASE database privileges.

Users with the SECURITY database privilege or the VMS SECURITY privilege implicitly receive the Rdb/VMS READ, WRITE, MODIFY, and ERASE database privileges.

Users with the VMS BYPASS privilege implicitly receive ALL privileges except the Rdb/VMS ADMINISTRATOR, OPERATOR, and SECURITY database privileges and the CONTROL relation privilege.

DEFINE PROTECTION Statement

Users with the VMS READALL privilege receive implicit READ and SHOW database and relation privileges.

Table 9–7 shows which Rdb/VMS privileges can be overridden by the Rdb/VMS ADMINISTRATOR, OPERATOR, and SECURITY database privileges and the VMS SYSPRV, BYPASS, and READALL privileges. For each table entry, the question is whether users with the Rdb/VMS or VMS privilege specified in the columns at the top of the table implicitly receive the access rights associated with the Rdb/VMS privilege in the first column of the table. For example, the Y for the first entry in the table shows that the Rdb/VMS ADMINISTRATOR privilege overrides the Rdb/VMS CHANGE privilege and the N in the second entry shows that the Rdb/VMS OPERATOR privilege does not override the Rdb/VMS CHANGE privilege. An N/A entry in the table indicates that a privilege cannot override itself.

Table 9–7 Privilege Override Capability

Privilege	ADMINISTRATOR	OPERATOR	SECURITY	SYSPRV	BYPASS	READALL
CHANGE	Y	N	N	Y	Y	N
DEFINE	Y	N	N	Y	Y	N
ADMINISTRATOR	N/A	N	N	Y	N	N
CONTROL	Y	N	Y	Y	N	N
ERASE (database)	Y	Y	Y	Y	Y	N
ERASE (relation)	Y	N	N	Y	Y	N
DELETE	Y	N	N	Y	Y	N
WRITE (database)	Y	Y	Y	Y	Y	N
WRITE (relation)	Y	N	N	Y	Y	N
OPERATOR	N	N/A	N	N	N	N
REFERENCES	Y	N	N	Y	Y	N
SECURITY	N	N	N/A	N	N	N
READ (database)	Y	Y	Y	Y	Y	Y
READ (relation)	Y	N	N	Y	Y	Y
SHOW	Y	N	N	Y	Y	Y
MODIFY (database)	Y	Y	Y	Y	Y	N
MODIFY (relation)	Y	N	N	Y	Y	N
DISTRIBTRAN (database)	Y	N	N	Y	Y	N

To define protection for a database, you must first invoke the database. You must execute the DEFINE PROTECTION statement in a read/write

DEFINE PROTECTION Statement

transaction. If you issue this statement when there is no active transaction, Rdb/VMS starts a read/write transaction implicitly.

Rights on a field are determined by the rights defined on the field combined with those specified for the specific relation ACL.

A user with the MODIFY right on a relation automatically gets the MODIFY right on all fields in the relation. However, you can restrict the MODIFY right by defining it only on specific fields you want users to be able to modify and thus remove the right from the relation entry.

You can modify the data in a field if you have the MODIFY right on the field and the READ right on the relation and the database.

Rdb/VMS allows you to modify the default protection for any *new* relations or views that are created in a database:

- 1 The following example shows an ACL for a relation defined under the default Rdb/VMS protection scheme. The ACL consists of two access control list entries (ACEs). By default, the creator of the relation or view (with the identifier [DBS,RICK] in the first ACE) gets all the Rdb/VMS access rights to the object. All other users (represented by the [*,*] identifier in the second ACE) get no access rights.

```
RDO> SHOW PROTECTION FOR RELATION RELATION1
      ( IDENTIFIER=[ DBS , RICK ] , ACCESS=READ+WRITE+MODIFY+ERASE+SHOW+DEFINE+
        CHANGE+DELETE+CONTROL+OPERATOR+ADMINISTRATOR+REFERENCES )
      ( IDENTIFIER=[ * , * ] , ACCESS=NONE )
```

- 2 You cannot alter the Rdb/VMS default protection scheme for creators of new database objects. A creator of a new object always receives all the access rights to that object. However, you can alter the Rdb/VMS default protection scheme so that other users receive some access rights to new relations and views. To alter the Rdb/VMS default protection scheme, you must first define a VMS rights identifier called DEFAULT in the system rights database (if it is not already defined). Then, use the DEFINE PROTECTION FOR DATABASE statement to specify Rdb/VMS access rights for the DEFAULT identifier. The access rights specified for the DEFAULT identifier will be granted to all users except the creator of any new relations and views created in the database. For example, after the following statement is issued, all users except the creator of the database

DEFINE PROTECTION Statement

will receive the Rdb/VMS READ and MODIFY access rights to any new object created in the database:

```
RDO> DEFINE PROTECTION FOR DATABASE
cont> IDENTIFIER DEFAULT
cont> ACCESS READ+MODIFY.
```

- 3 You need to detach from the database to make the change in protection occur.

```
RDO> COMMIT
RDO> FINISH
```

- 4 The protection on existing relations and views in the database is not changed, but for any new relations and views created, users other than the new object's creator receive the protection specified by the DEFAULT identifier. In this example, the creator receives all the access rights to the new relation RELATION2 and all other users receive the READ and MODIFY access rights specified by the DEFAULT identifier.

```
RDO> INVOKE DATABASE FILENAME TEST1
RDO> DEFINE RELATION RELATION2.
cont>   FIELD1.
cont>   FIELD2.
cont> END RELATION.
RDO> SHOW PROTECTION FOR RELATION RELATION2
      ( IDENTIFIER=[ DBS,RICK ], ACCESS=READ+WRITE+MODIFY+ERASE+SHOW+DEFINE+
        CHANGE+DELETE+CONTROL+OPERATOR+ADMINISTRATOR+REFERENCES )
      ( IDENTIFIER=[ *,* ], ACCESS=READ+MODIFY )
```

Use only the DEFINE PROTECTION FOR DATABASE statement when you modify the Rdb/VMS default protection scheme by specifying access rights to be associated with the DEFAULT identifier. If you use any other statement, the Rdb/VMS default protection scheme is not changed.

You must explicitly define a new default protection scheme for each database in which you do not want to use the Rdb/VMS default protection scheme. If the VMS identifier DEFAULT does not exist in the system rights database, you receive the "unknown rights identifier" error message when you try to define the new default protection scheme. If the VMS identifier DEFAULT is removed from the system rights database after you have defined a new default protection scheme for an Rdb/VMS database, any new relations or views created after the VMS DEFAULT identifier is removed will receive the Rdb/VMS default protection.

You must execute the DEFINE PROTECTION statement in a read/write transaction. If you issue this statement when no transaction is active, Rdb/VMS starts a read/write transaction implicitly.

DEFINE PROTECTION Statement

Other users are allowed to be attached to the database when you issue the DEFINE PROTECTION statement.

You cannot execute the DEFINE PROTECTION statement when the RDB\$SYSTEM storage area is set to read-only. You must first set RDB\$SYSTEM to read/write. See the *VAX Rdb/VMS Guide to Database Maintenance and Performance* for more information on the RDB\$SYSTEM storage area.

Examples

Example 1

The following example grants access rights to a single user:

```
RDO> DEFINE PROTECTION FOR DATABASE
cont> POSITION 3
cont> IDENTIFIER [CLERKS,DAVIES]
cont> ACCESS "READ+WRITE+MODIFY+ERASE".
```

This statement performs the following actions:

- Specifies the location of the entry within the access control list. The new entry is in the third position and all subsequent entries are moved to the next higher position.
- Uses a UIC to identify the user who is granted access rights.
- Grants the specified access rights. Rdb/VMS denies all other rights.

Example 2

The following example grants access rights on a specified relation to a group of users:

```
RDO> DEFINE PROTECTION FOR RELATION SALARY_HISTORY
cont> AFTER [ANALYSTS,JOHNSON]
cont> IDENTIFIER [ANALYSTS,*]
cont> ACCESS
cont> "READ+WRITE+MODIFY+ERASE -
cont> +DEFINE+CHANGE+DELETE".
```

This statement performs the following actions:

- Names the relation SALARY_HISTORY. The new ACL entry will be applied to this relation.

DEFINE PROTECTION Statement

- Uses the AFTER clause to specify the location of the entry within the ACL. In this case, the new ACL entry appears after the entry for user identifier [ANALYSTS,JOHNSON].
- Identifies the set of users ([ANALYSTS,*]) who are granted the listed access rights. In this case, the new identifier is the same group identifier as the one that precedes it in the list. This means that for user JOHNSON in group ANALYSTS, Rdb/VMS will grant the privileges listed in the earlier entry. All other users in group ANALYSTS will fall through to the entry identified by [ANALYSTS,*]. Rdb/VMS grants these other members of group ANALYSTS the rights listed in this statement. In this way, the system gives general rights to a group and more specific rights to a single member of the group.

Example 3

The following example grants access rights on the specific field to a group of users:

```
RDO> DEFINE PROTECTION FOR FIELD SALARY_AMOUNT IN SALARY_HISTORY
cont>     AFTER [MANAGERS,SMITH]
cont>     IDENTIFIER [MANAGERS,*]
cont>     ACCESS MODIFY.
```

The following examples show how to specify the identifiers in a DEFINE PROTECTION statement.

Example 4

In this example, all users with the UIC matching [25,*] and running a batch job are granted the access rights listed.

```
RDO> DEFINE PROTECTION FOR DATABASE
cont> POSITION 4
cont> IDENTIFIER [25,*]+BATCH
cont> ACCESS READ+DEFINE+CHANGE+DELETE.
```

Example 5

All users associated with the general identifier DATAENTRY and using RDO interactively are granted update access rights.

```
RDO> DEFINE PROTECTION FOR DATABASE
cont> POSITION 5
cont> IDENTIFIER DATAENTRY+INTERACTIVE
cont> ACCESS READ+WRITE+MODIFY+STORE.
```

DEFINE PROTECTION Statement

Example 6

In the following example, user JONES gets the specified access rights.

```
RDO> DEFINE PROTECTION FOR DATABASE
cont> POSITION 6
cont> IDENTIFIER [RDB,JONES]
cont> ACCESS READ+WRITE+MODIFY+ERASE.
```

Example 7

In the following example, the user [SQL,HALSTON] receives the READ, WRITE, MODIFY, and ERASE rights. HALSTON also gets the DISTRIBTRAN right, which allows HALSTON to run the two-phase commit protocol on the current database.

```
RDO> DEFINE PROTECTION FOR DATABASE
cont> POSITION 2
cont> IDENTIFIER [SQL,HALSTON]
cont> ACCESS "READ+WRITE+MODIFY+ERASE+DISTRIBTRAN".
RDO>
RDO> SHOW PROTECTION FOR DATABASE
( IDENTIFIER=[SQL,RICK],ACCESS=READ+WRITE+MODIFY+ERASE+SHOW+DEFINE+CHANGE+
  DELETE+CONTROL+OPERATOR+ADMINISTRATOR+REFERENCES+SECURITY+DISTRIBTRAN)
( IDENTIFIER=[SQL,HALSTON],ACCESS=READ+WRITE+MODIFY+ERASE+DISTRIBTRAN)
( IDENTIFIER=[*,*],ACCESS=NONE)
```

9.18 DEFINE RELATION Statement

Creates a relation definition. A relation definition consists of a list of fields that make up an Rdb/VMS record.

When the DEFINE RELATION statement executes, Rdb/VMS:

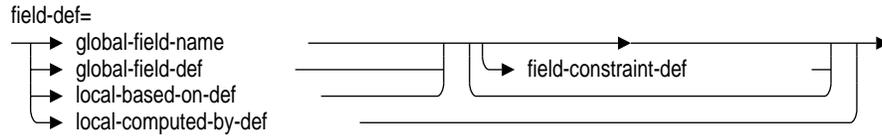
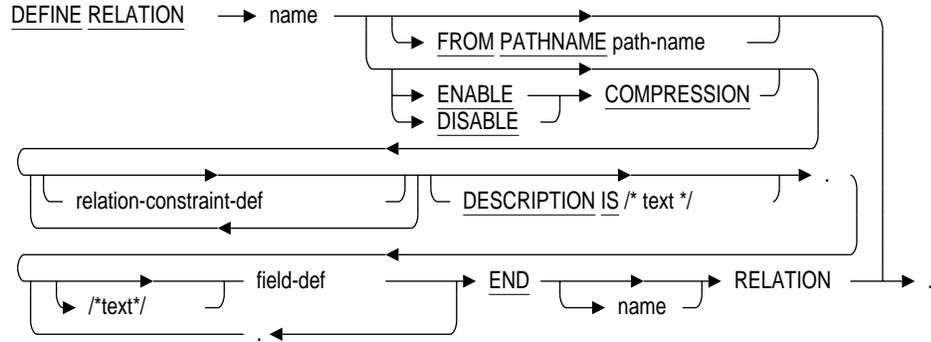
- Adds the relation definition to the physical database. If you invoke the database with the PATHNAME qualifier, the definition is also added to the data dictionary.
- Adds constraints associated with the relation definition to the physical database.
- Creates a default access control list (ACL) for the relation.

Note *The ENABLE/DISABLE COMPRESSION option in the DEFINE RELATION statement is obsolete. Use the DEFINE STORAGE MAP and CHANGE STORAGE MAP statements to control data compression. The COMPRESSION option in the DEFINE RELATION statement is maintained for compatibility with applications that used previous versions of Rdb/VMS. However, Digital Equipment Corporation recommends you use the DEFINE STORAGE MAP statement rather than the DEFINE RELATION statement to control data compression.*

You can copy a shareable relation definition from the data dictionary into the database by using the FROM PATHNAME clause of the DEFINE RELATION statement.

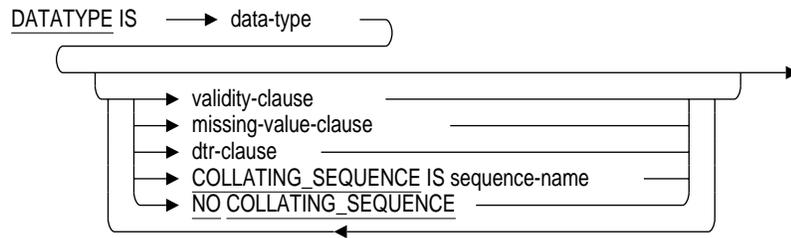
DEFINE RELATION Statement

Format

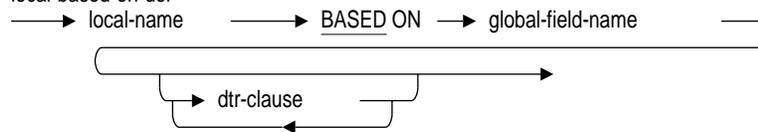


DEFINE RELATION Statement

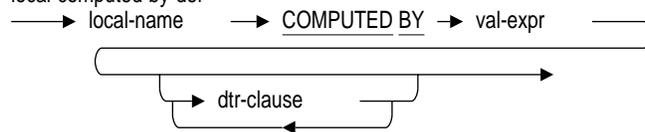
field-attributes =



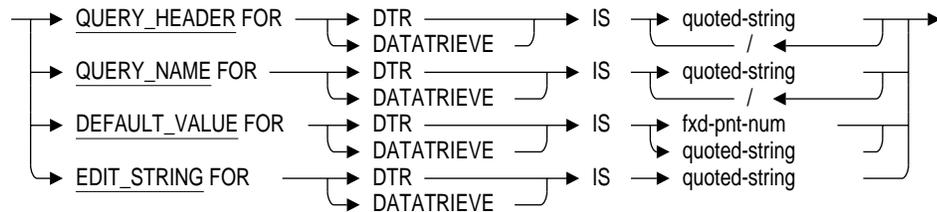
local-based-on-def=



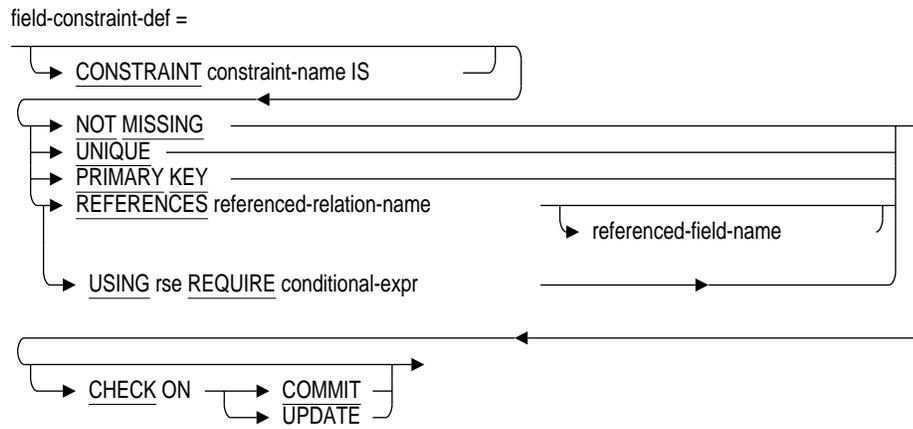
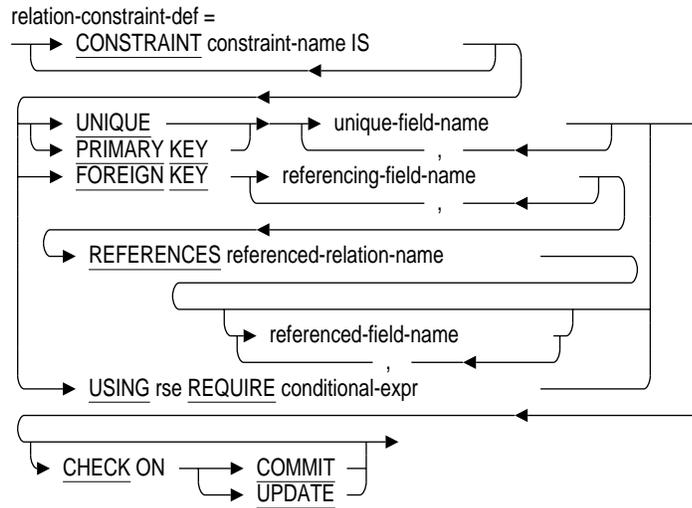
local-computed-by-def=



dtr-clause =



DEFINE RELATION Statement



DEFINE RELATION Statement

Arguments

name

The name of the relation definition you want to create. When you choose a name, follow these rules:

- Use a name that is unique among all relation and view names in the database.
- Use any valid VMS name. However, the name cannot end in a dollar sign (\$) or underscore (_).
- Do not use any Rdb/VMS reserved words (see Appendix A).

FROM PATHNAME *path-name*

A full or relative data dictionary path name, enclosed in quotation marks, that specifies the source of the relation definition. If you use a relative path name, the current default directory must include all the path name segments that precede the relative path name. You must invoke the database by path name if you plan to copy a shareable relation definition from the dictionary. You must specify a CDO path name and refer to a relation that was created with CDO. The relation definition that you copy from the dictionary must be a simple definition, that is, one that contains no nested records, no repeating groups (arrays), and no variants. Also, the data types of the individual field definitions that make up the dictionary record being copied must be compatible with supported Rdb/VMS data types.

You cannot use the FROM PATHNAME clause to specify a relation that was created with the RDO DEFINE RELATION statement unless it has been made shareable using the CDO ENTER command first. That is, you cannot use the FROM PATHNAME clause to define a relation unless the entity you name in FROM PATHNAME has been created using CDO. See the *VAX CDD/Plus User's Guide* for more information on how to use CDO DEFINE RECORD to define an Rdb/VMS relation.

ENABLE COMPRESSION [*Default*]

DISABLE COMPRESSION

Determines whether data compression is enabled or disabled for the relation.

text

A text string that adds a comment to the relation definition or the field definition.

DEFINE RELATION Statement

global-field-name

The name of the field that is part of the set of generic field definitions for the database. You can use a global field name in any one of three ways in the DEFINE RELATION statement:

- To refer to an existing global field by name. This includes the global definition in the relation.
- To refer to a new global field name and include a complete definition, including a DATATYPE clause. This includes the field definition in the relation and also enters the field definition in the global set of definitions for the database.
- To refer to an existing global field in a BASED ON clause. This causes the field to have a local name and a global definition.

field-attributes

A set of field attributes that explicitly defines a field for use in this relation definition.

Local field attributes are:

- Local field name, when you used the BASED ON clause
- DATATRIEVE support clauses
- COMPUTED BY clause

local-name

A name of a field that is recognized only within one relation. Local field attributes are defined only within a DEFINE RELATION statement and apply only to that relation's versions of the field definition. See Chapter 5 for a complete description of field attributes.

value-expr

A valid Rdb/VMS value expression. When you use the COMPUTED BY clause, the field derives its value from the value expression and the value for the field is calculated each time the field is accessed. The value expression can refer to any field in this relation or another relation in the database.

When you use the COMPUTED BY clause, any field mentioned in the value expression must already be defined in the relation.

DEFINE RELATION Statement

CONSTRAINT constraint-name

The name of a relation or field constraint associated with the relation that is being defined. This name must be unique within the database. The constraint name can be referred to in other statements such as CHANGE RELATION, SHOW CONSTRAINT, and START_TRANSACTION.

The clause "CONSTRAINT constraint-name IS" is optional. If you do not specify the keyword CONSTRAINT, Rdb/VMS provides a name for the constraint. However, Digital Equipment Corporation recommends that you always name field and relation constraints. The alternative is to have constraints named by the database system with names such as LAST_NAME_REQUIRE_0001.

UNIQUE unique-field-name

The name of a field in the relation that is a part of a unique key. This name can appear only once such that no two rows in the associated relation can have the same nonmissing values for the specified field or fields, and that no instances of the field or fields can be missing.

PRIMARY KEY unique-field-name

The name of a field in the relation that is a part of a primary key. This name can appear only once in the base relation. Rdb/VMS requires that the values in a primary key be unique and not missing; therefore, you need not specify the UNIQUE and NOT MISSING field constraints for a field that you designate a primary key. Only one primary key can be declared for a relation.

FOREIGN KEY

The name of a field or fields that you want to declare as a foreign key in the relation you are defining.

referencing-field-name

The name of a field in the relation that is part of a foreign key. This name can appear only once in the referencing definition, and must correspond to a field having the same ordinal position in any list of referenced fields. The data types of corresponding fields must be the same.

At the relation level, a constraint can have one or more referencing-field-names that correspond to a matching list of referenced field names.

REFERENCES

Declares that one or more fields comprise a single foreign key.

DEFINE RELATION Statement

referenced-relation-name

The name of the relation that defines the unique or primary key definition that is referred to by a foreign key of this relation. If no referenced field names are specified with this relation name, then the referenced relation must have an associated constraint that specifies a primary key. If there are referenced field names, the referenced relation must have a unique or primary key constraint defined that specifies a list of unique field names. The names can be different but the fields must be of the same data type, length, and scale.

See the Usage Notes for more information.

referenced-field-name

Specifies the name of a field in the foreign key relation that corresponds to the field with the same ordinal position within the list of fields referred to by the primary key relation. See the Usage Notes for more information.

NOT MISSING

Restricts field values such that none of the values for the specified field can assume either the defined value or the missing value for that field. You can explicitly declare the NOT MISSING clause only at the field level.

rse

A record selection expression that defines which records of which relations will be tested against the conditional expression. This rse cannot refer to any host variables.

conditional-expr

An expression that describes the optional conditions that must be satisfied before the record can be stored in the database.

CHECK ON COMMIT

CHECK ON UPDATE

Declares the time when the constraint is evaluated. The constraint can be evaluated when the update occurs (CHECK ON UPDATE) or when a COMMIT is issued (CHECK ON COMMIT). The EVALUATING clause of the START_TRANSACTION statement can override the CHECK ON clause.

Specifying CHECK ON COMMIT allows the use of interlocking constraints.

DEFINE RELATION Statement

Usage Notes

You need the Rdb/VMS DEFINE privilege for the database to define a relation. If you are defining a constraint as part of the relation definition, you must also have the Rdb/VMS DEFINE privilege for the relation referenced by the constraint.

If the database is created with the DICTONARY IS REQUIRED option, you must invoke the database by path name, rather than file name, before you issue this statement.

You can have a maximum of 2000 fields in a relation.

You must execute this statement in a read/write transaction. If you issue this statement when there is no active transaction, Rdb/VMS starts a read/write transaction implicitly.

Other users are allowed to be attached to the database when you issue the DEFINE RELATION statement.

You cannot execute the DEFINE RELATION statement when the RDB\$SYSTEM storage area is set to read-only. You must first set RDB\$SYSTEM to read/write. See the *VAX Rdb/VMS Guide to Database Maintenance and Performance* for more information on the RDB\$SYSTEM storage area.

The following usage notes pertain to relation-specific constraints:

- You can use relation-specific constraints
 - To maintain referential integrity by establishing a clear, visible set of rules
 - To attach the desired integrity rules directly to the definition of a relation
 - To avoid defining multiple, seemingly independent constraints to accomplish the same task

Note *Combinations of relation-specific constraints and appropriately defined triggers are not sufficient to guarantee that database integrity is preserved on updates. They must be used in conjunction with a common set of update procedures that assures completely reproducible and consistent retrieval and update strategies if integrity is to be preserved.*

DEFINE RELATION Statement

- Relation-specific constraints can be declared at the relation level or the field level or both. These constraints can specify that fields contain only certain values, primary key values, unique values, or that values can be restricted to nonmissing values. Multiple constraints can be declared at both the relation and field level.

If you want to specify multiple fields in a primary key, foreign key, or “unique” specification, you must use a relation-level constraint. For instance, if you wanted to say “CONSTRAINT NAME_PK IS PRIMARY KEY LAST_NAME, FIRST_NAME” you would have to make it a relation constraint, but if you just wanted to say “CONSTRAINT NAME_PK IS PRIMARY KEY LAST_NAME” you could specify it at either the relation level or field level.

Specifying constraints on the field level allows you to document your constraints directly with your field definitions as in the following example:

```
RDO> DEFINE RELATION X.  
cont> X_CODE DATATYPE TEXT SIZE IS 1 CONSTRAINT X_UK IS UNIQUE.  
cont> END RELATION.  
RDO> SHOW FIELDS FOR X  
Fields for relation X  
X_CODE text size is 1  
Constraint: X_UK  
unique
```

At both levels, you can specify definitions of unique, primary, and foreign keys, and foreign key references to unique or primary keys. You can also specify constraint evaluation time (either commit or update).

At the relation level, you can define constraints for multifield keys.

At the field level, you can restrict the values of fields to nonmissing values.

- Constraints should not specify fields defined as segmented strings, as only the segmented string ID will be referenced, not the actual segmented string.
- Within the relation definition, constraints can apply to the values in specific rows of a relation, to the entire contents of a relation, or to states existing between multiple relations.
- Within the relation definition, Rdb/VMS first defines new versions of fields. Then Rdb/VMS defines constraints and evaluates them. Therefore, if fields and constraints are defined within the same relation definition, constraints can use any of the fields defined in this relation before or after the constraint definition text.

DEFINE RELATION Statement

- Within the relation definition, you can specify constraints to be evaluated at either update time (after the UPDATE clause is executed) or at commit time (when a transaction is being committed). The evaluation time can be specified when the constraint is defined (default is update time) or when a transaction is initiated (named constraints only, default or specified evaluation time can be overridden). If the constraint is evaluated at update time and fails, the update operation is effectively rolled back. If the constraint fails at commit time, the update operation must be manually rolled back.
- Adding a constraint to the database causes the constraint criteria to be evaluated if there is data to validate. If existing data does not meet the criteria, an exception is produced and the definition fails.
- The REFERENCES clause can declare the one or more corresponding fields in the referenced relation that comprise a unique or primary key. If the corresponding fields are not declared, the referenced relation must include a PRIMARY KEY constraint at the relation level specifying the corresponding field or fields.

Examples

Example 1

The following example uses the DEFINE RELATION statement to create a relation:

```
DEFINE RELATION DEPARTMENTS .  
    DEPARTMENT_CODE .  
    DEPARTMENT_NAME .  
    MANAGER_ID BASED ON ID_NUMBER .  
END DEPARTMENTS RELATION .
```

This statement names the new relation, DEPARTMENTS, and specifies its fields. In the DEPARTMENTS relation definition:

- The DEPARTMENT_CODE and DEPARTMENT_NAME fields are already defined. The relation definition simply uses their names.
- The MANAGER_ID field is a local name, but it points to an existing global field definition. If the definition of the ID_NUMBER field changes, the MANAGER_ID field changes also.

DEFINE RELATION Statement

Example 2

The following example defines global fields in the DEFINE RELATION statement:

```
DEFINE RELATION FAMILY
    DESCRIPTION IS /* Family information */.
/* Employee ID */
    EMPLOYEE_ID BASED ON ID_NUMBER
    QUERY_NAME FOR DTR IS "EMP".
/* Married? M or S */
    MARITAL_STATUS
        DATATYPE TEXT SIZE 1
        VALID IF MARITAL_STATUS = "M" OR
            MARITAL_STATUS = "S".
/* Number of dependents */
    NUMBER_DEPENDENTS
        DATATYPE SIGNED WORD SCALE 0.
/* Percentage of income withheld */
    WITHHOLDING
        COMPUTED BY 0.20 / NUMBER_DEPENDENTS.
END FAMILY RELATION.
```

This DEFINE RELATION statement defines several new fields:

- The DESCRIPTION clause and the other text fields provide commentary for the relation definition and for each field.
- The EMPLOYEE_ID field is a local name for the global ID_NUMBER field. The QUERY_NAME clause overrides any QUERY_NAME clause in the ID_NUMBER field.
- The MARITAL_STATUS field uses the DATATYPE clause. Therefore, the MARITAL_STATUS field becomes a global field definition. It is entered in the list of global fields for the database, and other relations can use it by name.
- The NUMBER_DEPENDENTS field also becomes a global field definition.
- The WITHHOLDING field is a local field, defined in terms of the NUMBER_DEPENDENTS field.

Example 3

The following example copies a shareable relation definition from the data dictionary into the database:

```
DEFINE RELATION EMP_INFO
    FROM PATHNAME 'DISK1:[DICTIONARY]CORP.PERS.EMP_INFO'.
```

DEFINE RELATION Statement

Example 4

The following example defines constraints using a segment of the PERSONNEL database, with single-field keys. This example specifies primary and foreign keys.

```
RDO> DEFINE RELATION EMPLOYEES.  
cont>   EMPLOYEE_ID  
cont>     BASED ON ID_NUMBER  
cont>     PRIMARY KEY.  
cont>   LAST_NAME.  
cont>   FIRST_NAME.  
cont>   MIDDLE_INITIAL.  
cont>   ADDRESS_DATA_1.  
cont>   ADDRESS_DATA_2.  
cont>   CITY.  
cont>   STATE.  
cont>   POSTAL_CODE.  
cont>   SEX.  
cont>   BIRTHDAY  
cont>     BASED ON STANDARD_DATE.  
cont>   STATUS_CODE.  
cont> END EMPLOYEES RELATION.  
RDO>  
RDO> DEFINE RELATION JOB_HISTORY.  
cont>   EMPLOYEE_ID  
cont>     BASED ON ID_NUMBER  
cont>     CONSTRAINT JH_EMP_ID_FOREIGN  
cont>     REFERENCES EMPLOYEES EMPLOYEE_ID.  
cont>   JOB_CODE  
cont>     REFERENCES JOBS JOB_CODE.  
cont>   JOB_START  
cont>     BASED ON STANDARD_DATE.  
cont>   JOB_END  
cont>     BASED ON STANDARD_DATE.  
cont>   DEPARTMENT_CODE  
cont>     CONSTRAINT JH_D_CODE_FOREIGN  
cont>     REFERENCES DEPARTMENTS DEPARTMENT_CODE  
cont>   SUPERVISOR_ID  
cont>     BASED ON ID_NUMBER.  
cont> END JOB_HISTORY RELATION.  
RDO>  
RDO> DEFINE RELATION JOBS.  
cont>   JOB_CODE  
cont>     CONSTRAINT J_CODE_PRIMARY  
cont>     PRIMARY KEY.  
cont>   WAGE_CLASS.  
cont>   JOB_TITLE.
```

DEFINE RELATION Statement

```
cont>   MINIMUM_SALARY
cont>     BASED ON SALARY.
cont>   MAXIMUM_SALARY
cont>     BASED ON SALARY.
cont> END JOBS RELATION.
RDO>
RDO> DEFINE RELATION DEPARTMENTS.
cont>   DEPARTMENT_CODE
cont>     PRIMARY KEY.
cont>   DEPARTMENT_NAME.
cont>   MANAGER_ID
cont>     BASED ON ID_NUMBER.
cont>   BUDGET_PROJECTED
cont>     BASED ON BUDGET.
cont>   BUDGET_ACTUAL
cont>     BASED ON BUDGET.
cont> END DEPARTMENTS RELATION.
```

This example shows three single-field PRIMARY KEY constraints. The PRIMARY KEY constraints are the EMPLOYEE_ID field for the EMPLOYEES relation, the JOB_CODE field for the JOBS relation, and the DEPARTMENT_CODE field for the DEPARTMENTS relation. The JOB_HISTORY relation contains three foreign keys that refer to these primary keys.

Example 5

The following example defines constraints using a segment of the PERSONNEL database with multifield keys. This example uses some definitions not supplied with the sample database.

```
RDO> DEFINE FIELD LOC DATATYPE TEXT SIZE 10 MISSING_VALUE " ".
RDO> DEFINE FIELD DEPT DATATYPE TEXT SIZE 10 MISSING_VALUE " ".
RDO> DEFINE FIELD MGR DATATYPE TEXT SIZE 20.
RDO> DEFINE FIELD NAME DATATYPE TEXT SIZE 20.
RDO>
RDO> DEFINE RELATION WORK_STATION
cont> CONSTRAINT LOC_DEPT
cont> PRIMARY KEY LOC,DEPT.
cont> LOC BASED ON LOC.
cont> DEPT BASED ON DEPT.
cont> MGR BASED ON MGR.
cont> END WORK_STATION RELATION.
RDO>
```

DEFINE RELATION Statement

```
RDO> DEFINE RELATION WORKER
cont>     CONSTRAINT LOC_DEPT_FK
cont>     FOREIGN KEY LOCATION,DEPT
cont>     REFERENCES WORK_STATION LOC, DEPT
cont>     CONSTRAINT NAME_PK
cont>     PRIMARY KEY NAME.
cont>     LOCATION BASED ON LOC.
cont>     DEPT.
cont>     NAME BASED ON NAME.
cont> END WORKER RELATION.
RDO>
RDO> SHOW RELATION WORK_STATION
WORK_STATION
           CONSTRAINT LOC_DEPT
           PRIMARY KEY  LOC,DEPT
RDO> SHOW RELATION WORKER
WORKER
           CONSTRAINT NAME_PK
           PRIMARY KEY NAME
           CONSTRAINT LOC_DEPT_FK
           FOREIGN KEY LOCATION,DEPT
           REFERENCES WORK_STATION LOC, DEPT
```

In this example, the two fields LOC and DEPT constitute a key, and are defined as a PRIMARY KEY in a constraint for the WORK_STATION relation (both fields are nonmissing), and as a FOREIGN KEY constraint for the WORKER relation. The foreign key constraint specifies that the LOCATION or DEPT key from the WORKER relation must match a LOC or DEPT key from the WORK_STATION relation.

In addition, the field NAME is constrained to be a PRIMARY KEY constraint for the WORKER relation.

DEFINE SCHEDULE Statement (VAX Data Distributor)

9.19 DEFINE SCHEDULE Statement (VAX Data Distributor)

Creates a schedule definition and stores it in the transfer database. The schedule definition specifies when and how often a transfer should execute. A transfer can have only one schedule definition associated with it. Data Distributor returns an error if you attempt to define a schedule for a transfer that already has one.

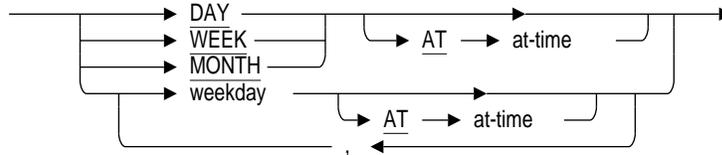
Note To use this RDO statement, you must have VAX Data Distributor installed on your system. See your system manager or your Digital Equipment Corporation representative if you need information on Data Distributor.

Format

```
DEFINE SCHEDULE → FOR → transfer-name )
(
  → START → start-date-time )
(
  → EVERY → every-delta-time
           → every-absolute-time )
(
  → RETRY → count → TIMES → RETRY EVERY → delta-time )
→ END → transfer-name ) → SCHEDULE → .
```

DEFINE SCHEDULE Statement (VAX Data Distributor)

every-absolute-time =



Arguments

transfer-name

The transfer for which this schedule is being defined. The transfer must already exist when you create a schedule definition for it. The `transfer-name` parameter is required.

start-date-time

The time to execute the initial transfer. You can specify an absolute or delta time. If you supply an absolute time, the initial transfer will be executed at the time specified. If you use a delta time, Data Distributor uses the current time and the delta-time value to calculate the actual transfer time. The `START` clause is optional. If you do not include a `START` clause, the start time is set to the current time so that the transfer will execute as soon as the transfer schedule definition is processed.

The `start-date-time` parameter uses the following absolute and delta-time formats. You can omit any of the trailing fields in the date or time. You can omit any of the fields in the middle of the format as long as you include the punctuation marks. For more information on date and time formats, see the entry on the `$BINTIM` VMS system service in the VMS documentation set.

Absolute time

dd-mmm-yyyy hh:mm:ss.cc

Delta time

dddd:hh:mm:ss.cc

DEFINE SCHEDULE Statement (VAX Data Distributor)

In the absolute and delta time designations, the abbreviations have the following meanings:

dddd	Number of days
dd	Day of the month
mmm	Month
yyyy	Year
hh	Hours
mm	Minutes
ss	Seconds
cc	Hundredths of a second

every-delta-time

Used to calculate the time to execute the next transfer. The EVERY clause is optional. If you specify the EVERY clause, you must specify either every-delta-time or every-absolute-time. If you omit the EVERY clause, the transfer occurs only once.

The valid specification for every-delta-time is the delta format used by the \$BINTIM VMS system service. The delta time you specify is the time from the completion of the last transfer sequence until the start of the next one. A transfer sequence is the set of transfers from the first attempt until one of the following conditions is met: the execution is successful, the maximum number of retries specified in the RETRY clause has been reached, or a fatal error has occurred (one for which retries are not attempted).

every-absolute-time

Used to calculate the time to execute the next transfer. The EVERY clause is optional. If you specify the EVERY clause, you must specify either every-absolute-time or every-delta-time. If you omit the EVERY clause, the transfer occurs only once.

Valid specifications for the every-absolute-time parameter are:

- DAY
The transfer executes on a daily basis.
- WEEK
The transfer occurs every week on the same day of the week as the initial transfer.

DEFINE SCHEDULE Statement (VAX Data Distributor)

- MONTH

The transfer occurs on a monthly cycle on the same day of the month as the initial transfer.

Note that if the initial transfer starts on the 31st of the month and the transfer schedule specifies EVERY MONTH, the next execution of the transfer will be on the last day of the following month, whether the last day is the 28th, 29th, 30th, or 31st.

- Weekday

The transfer executes on the weekdays that you list. If you list more than one weekday, separate the days with commas. Valid weekday specifications are:

SUNDAY
MONDAY
TUESDAY
WEDNESDAY
THURSDAY
FRIDAY
SATURDAY

at-time

Qualifies every-absolute-time by specifying the time of day at which the transfer should occur. This parameter is optional. If you specify DAY, WEEK, or MONTH, but do not specify an at-time, Data Distributor uses the time of day of the initial transfer. In the case of a list of weekdays, the at-time for each day can be specified. If the at-time for any weekday is not specified, the default is the time used for the previous entry in the list. For the first entry in the list, the time of day of the initial transfer is used, if none is specified.

count

The maximum number of times to retry a transfer if a transfer execution fails. If you do not include a retry count or if the count value is set to zero, Data Distributor does not attempt to retry the transfer until the next scheduled transfer time as indicated by the EVERY clause.

Note *Data Distributor does not retry a transfer if the transfer failed as a result of a fatal error. You can recover from a fatal error only by manual intervention.*

DEFINE SCHEDULE Statement (VAX Data Distributor)

delta-time

The length of time Data Distributor waits between the time of the last failed attempt and the time of the next retry attempt. The `RETRY EVERY delta-time` clause is optional and can only be specified if the `RETRY count TIMES` clause is included.

If you specify `RETRY count TIMES` in the schedule definition, but do not specify `RETRY EVERY delta-time`, Data Distributor attempts the retry immediately. See the description of the `start-date-time` parameter earlier in this section for valid `delta-time` values.

Usage Notes

To define a transfer schedule, you must have the same user identification code (UIC) as the user who defined the transfer. If you define a schedule for a transfer defined by a user with a different UIC, Data Distributor returns an error message and prevents the schedule from executing. If you have `BYPASS` privilege, you can define a schedule for a transfer associated with a different UIC. In this case, the UIC associated with the transfer does not change.

When you define a schedule for a transfer in the unscheduled state, that transfer is placed in the scheduled state. If the transfer is in the suspended state, it remains in the suspended state. To place the suspended transfer in the scheduled state, use the `START TRANSFER` statement. When the transfer executes, Data Distributor places the transfer in the active state.

You cannot modify a schedule definition. If you need to change a parameter value in a schedule definition, you must first delete the schedule using the `DELETE SCHEDULE` statement and then define a new schedule.

You must execute the `DEFINE SCHEDULE` statement outside of the scope of a transaction. If you issue this statement when a transaction is outstanding, Data Distributor returns an error message.

Examples

Example 1

This example initially executes the `NH_EMPLOYEES` transfer at 1:24 a.m. on June 24, 1988. The transfer will then execute every day at 5:30 p.m. If a nonfatal error occurs, the transfer execution will be retried three times with 30 minutes between the end of the failed transfer execution and the start of the next one.

DEFINE SCHEDULE Statement (VAX Data Distributor)

```
RDO> DEFINE SCHEDULE FOR NH_EMPLOYEES
cont> START 24-JUN-1988 01:24:00.00      <---- Absolute time
cont>   EVERY DAY AT 17:30                <---- Absolute time
cont>   RETRY 3 TIMES
cont>   RETRY EVERY 0 00:30:00
cont> END SCHEDULE.
```

Example 2

This example uses a delta-time specification to have 36 hours (1 day plus 12 hours) between the completion of one transfer execution and the beginning of the next. The example uses two delta times, `EVERY 1 12:00` and `RETRY EVERY 0 03:00:00`. `EVERY 1 12:00` states that there will be 36 hours (1 day plus 12 hours) between the completion of the first transfer and the beginning of the next. If transfer execution does not succeed on the first attempt, the transfer schedule permits Data Distributor to make up to five additional attempts to complete the transfer. `RETRY EVERY 0 03:00:00` means Data Distributor should retry the transfer three hours after a nonfatal failure until the transfer completes successfully or until the maximum of five retries has been attempted. If the transfer does not complete successfully after five retries, the next attempt will be 36 hours after the last failed attempt.

```
RDO> DEFINE SCHEDULE FOR NH_EMPLOYEES
cont> START 15-MAY-1988
cont>   EVERY 1 12:00                        <---- 1 day plus 12 hours
cont>   RETRY 5 TIMES
cont>   RETRY EVERY 0 03:00:00              <---- 3 hours
cont> END.
```

Example 3

The following transfer schedule causes Data Distributor to execute a transfer for the first time at 12 noon on the 15th of October 1988. Subsequent transfers will occur every day thereafter at 5:30 p.m. If an error occurs preventing a successful transfer, Data Distributor will retry every 30 minutes until the transfer is successful or the maximum of three retries has been reached.

```
RDO> DEFINE SCHEDULE FOR NH_EMPLOYEES
cont> START 15-OCT-1988 12:00
cont>   EVERY DAY AT 17:30                    <---- Every day
cont>   RETRY 3 TIMES
cont>   RETRY EVERY 0 00:30:00
cont> END SCHEDULE.
```

DEFINE SCHEDULE Statement (VAX Data Distributor)

Example 4

The following schedule definition assumes that you want to execute the transfer immediately after you complete the schedule definition and, therefore, it contains no START clause. If the system time is 20-OCT-1988 16:00:00 when you complete the DEFINE SCHEDULE statement, the next execution of the NH_EMPLOYEES transfer will start on 20-NOV-1988 16:00:00.

```
RDO> DEFINE SCHEDULE FOR NH_EMPLOYEES
cont> EVERY MONTH
cont> RETRY 3 TIMES
cont> RETRY EVERY 0 00:30:00
cont> END.
```

Example 5

This example calls for the NH_EMPLOYEES transfer to execute at 3:00 p.m. every Wednesday. The schedule definition calls for a maximum of three retries occurring at intervals of 30 minutes.

```
RDO> DEFINE SCHEDULE FOR NH_EMPLOYEES
cont> START 12-NOV-1988 15:00:00.00
cont> EVERY WEDNESDAY AT 15:00:00 <---- Wednesdays at 3 p.m.
cont> RETRY 3 TIMES
cont> RETRY EVERY 0 00:30:00
cont> END.
```

Example 6

The following example causes an initial transfer to execute on the 15th of July at 5 p.m., on every Monday after the 15th at 5 p.m., every Tuesday at 11 a.m., and every Friday at 11 a.m. Because a RETRY clause is not specified, even if a nonfatal failure occurs, the transfer will not execute until the next regularly scheduled time.

```
RDO> DEFINE SCHEDULE FOR NH_EMPLOYEES
cont> START 15-JUL-1988 17:00
cont> EVERY MONDAY, TUESDAY AT 11:00, FRIDAY
cont> END.
```

Example 7

This example causes a transfer to start at midnight on July 15th and on the 15th of every month thereafter at 7 p.m.

```
RDO> DEFINE SCHEDULE FOR NH_EMPLOYEES
cont> START 15-JUL-1988
cont> EVERY MONTH AT 19:00
cont> END.
```

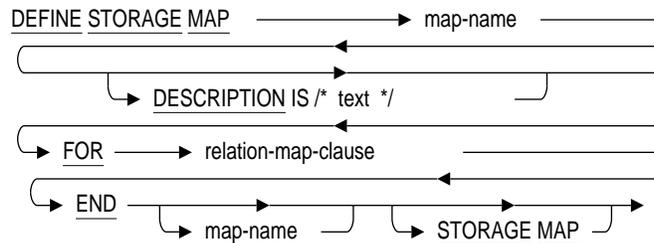
DEFINE STORAGE MAP Statement

9.20 DEFINE STORAGE MAP Statement

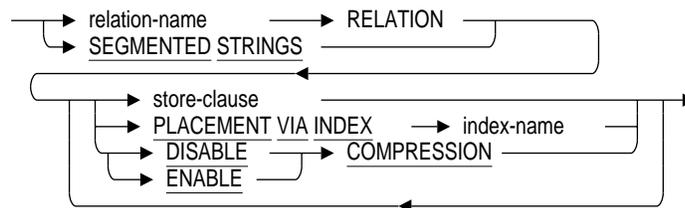
Creates a storage map for a relation. A storage map associates a relation with a particular storage area or areas. The DEFINE STORAGE MAP statement allows you to specify the following:

- Which storage areas the records in a relation will be stored in and how the records will be partitioned across the storage areas
- Whether an index will be used to select a target location for storing the record
- Whether data compression will be enabled or disabled when the records are stored in the relation
- Which storage areas segmented strings will be stored in

Format

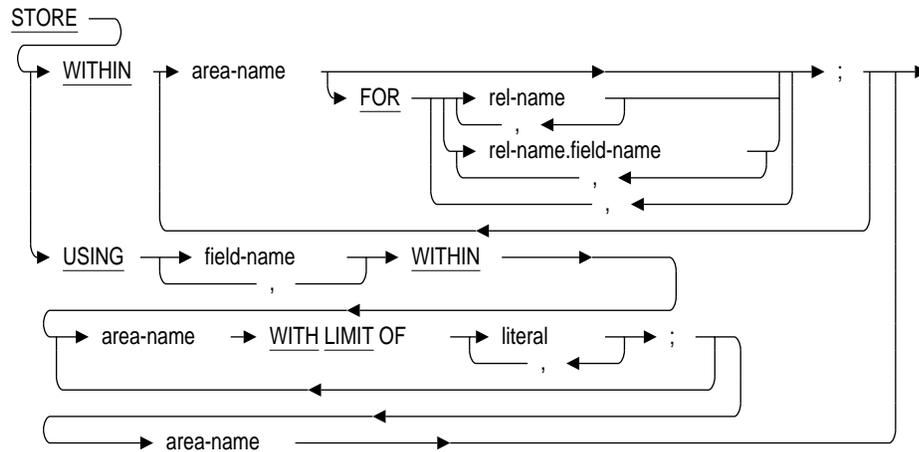


relation-map-clause =



DEFINE STORAGE MAP Statement

store-clause =



Arguments

map-name

The name of the storage map you want to create. When you choose a name, follow these rules:

- Use a name that is unique among all user-supplied names in the database.
- Use any valid VMS name. However, the name cannot end in a dollar sign (\$) or underscore (_).
- Do not use any Rdb/VMS reserved words (see Appendix A).

text

A text string that adds a comment to the storage map definition.

relation-name

The name of the relation to which the storage map will apply. The relation must already be defined in the database and cannot have a storage map already associated with it.

DEFINE STORAGE MAP Statement

SEGMENTED STRINGS

Specified when you want to store segmented strings in multiple storage areas. Use the store-clause to specify the storage areas in which you wish to store the segmented strings.

store-clause

The criteria for determining the location to store a record. This clause contains a number of subclauses. The store-clause can be used to spread data over multiple disks and to cause related data to be stored together in the database. The WITH LIMIT OF clause allows you to partition data over storage areas according to specific values. If you specify storage areas without including the WITH LIMIT OF clause, Rdb/VMS randomly stores records within the specified storage areas. However, if you also specify a PLACEMENT VIA INDEX clause, records will be stored according to the indicated path in the PLACEMENT VIA INDEX clause.

If you do not specify the store-clause, the relation associated with the storage map is stored in the default storage area.

WITHIN area-name

The name of the storage area in which you want records stored. You must define this storage area with the DEFINE DATABASE statement before you refer to it in the store-clause.

FOR rel-name

The name of the relation whose segmented strings you want to store in the specified storage area. If you want to store the segmented strings of more than one relation in the storage area, separate the names of the relations with commas.

FOR rel-name.field-name

The name of the relation and segmented string field that you want to store in the specified storage area. If you want to store more than one segmented string field in the storage area, separate the list items with commas.

USING field-name

The names of the fields whose values will be used as limits for partitioning the relation across multiple storage areas. Rdb/VMS compares values in the fields to the values in the WITH LIMIT OF clause to determine where to store the record initially.

DEFINE STORAGE MAP Statement

WITH LIMIT OF literal

The maximum value a field specified in the USING clause can have when it is initially stored in the specified storage area.

The number of literals specified in each WITH LIMIT OF clause must not exceed the number of fields specified in the USING clause.

Note that the last storage area you specify *cannot* have a WITH LIMIT OF clause associated with it.

When you define a multisegmented index using multiple keys and use the STORE USING . . . WITH LIMITS clauses, if the values for the first key are all the same, then set the limit for the first key at that value. By doing this, you ensure that the value of the second key determines the storage area in which each record will be stored.

PLACEMENT VIA INDEX

Directs Rdb/VMS to store a record in a way that optimizes access to that record through the indicated path.

If the index named is a hashed index, the storage area named must have a MIXED page format. If the hashed index definition and the storage map for the relation designate the same storage area, then the record is stored on the same page as the hashed index node. Otherwise, Rdb/VMS uses the same *relative* page within the data storage area as the target page.

If the index named is a sorted index, Rdb/VMS finds the dbkey of the next lowest record to the one being stored and uses the page number in the dbkey as the target page.

DISABLE COMPRESSION

ENABLE COMPRESSION [Default]

Specifies whether records in the relation will be compressed or uncompressed when stored. ENABLE COMPRESSION is the default.

Usage Notes

To define a storage map for a relation, you need the Rdb/VMS DEFINE privilege to the relation.

You must specify one or more of the clauses (store-clause, PLACEMENT VIA INDEX clause, or COMPRESSION clause) in the relation-map-clause, but you cannot repeat a clause.

You cannot create a storage map for a relation that already has records in it.

DEFINE STORAGE MAP Statement

You cannot delete an index if a storage map specifies the index in a `PLACEMENT VIA INDEX` clause.

When you partition records across storage areas according to values of a specified field (with the `WITH LIMIT OF` clause), you cannot include the `WITH LIMIT OF` clause on this last storage area. This last storage area holds all records whose values are greater than the limit specified on the previous storage area.

If the database is created with the `DICTIONARY IS REQUIRED` option, you must invoke the database by path name, rather than file name, before you issue this statement.

You must execute this statement in a read/write transaction. If there is no active transaction and you issue this statement, Rdb/VMS starts a read/write transaction implicitly.

Other users can be attached to the database when you issue the `DEFINE STORAGE MAP` statement.

Attempts to define a storage map will fail if that storage map or its relation is involved in a query at the same time. You must detach from the database with a `FINISH` statement before you can define the storage map. When Rdb/VMS first accesses an object, such as the relation, a lock is taken out on that object and not released until you exit the database. Attempts to update this object will get a `LOCK CONFLICT ON CLIENT` message due to the other user's optimized access to the object.

Similarly, while you define a storage map, users cannot execute queries involving that storage map until you have completed the transaction with a `COMMIT` or `ROLLBACK` statement for the `DEFINE` statement. The user will receive a `LOCK CONFLICT ON CLIENT` error message. While DDL operations are performed, normal data locking mechanisms are used against system relations. (System relations hold information on objects in the database.) Therefore, attempts to update an object will lock out attempts to query that object. These locks are held until the `COMMIT` or `ROLLBACK` of the DDL operation.

The `WAIT/NOWAIT` clause of the `START_TRANSACTION` statement does not affect attempts to update metadata with simultaneous queries. Even if you

DEFINE STORAGE MAP Statement

specify `START_TRANSACTION WAIT` for the metadata update transaction, you will get the following error message if a lock conflict exists:

```
%RDB-E-LOCK_CONFLICT, request failed due to locked resource; no-wait
parameter specified for transaction
-RDB-E-NO_META_UPDATE, metadata update failed
-RDMS-F-LCKCNFLCT, lock conflict on client
RDO>
```

However, a user's query will wait for a metadata update to complete with a `ROLLBACK` or `COMMIT` statement, even if the user specified `NOWAIT` on the `START_TRANSACTION` statement.

If a relation is named for more than one storage area, Rdb/VMS will select an area at random.

If multiple areas do not have any relations specified, the storage area to be used will be selected at random. However, `Rdb$SYSTEM` or whatever area was specified as the default at database creation will be used for the system relations' segmented strings.

Options normally used by storage maps, such as `ENABLE COMPRESSION` and `WITH LIMIT OF` clauses cannot be used with multiple storage area segmented strings.

The `CHANGE STORAGE MAP` statement can delete an area only if there are no segmented strings in the area. You cannot move a segment from one area to another. Changing the area or areas assigned to a relation will have no effect on current segmented strings, only on future storage.

You cannot execute the `DEFINE STORAGE MAP` statement when the `RDB$SYSTEM` storage area is set to read-only. You must first set `RDB$SYSTEM` to read/write. See the *VAX Rdb/VMS Guide to Database Maintenance and Performance* for more information on the `RDB$SYSTEM` storage area.

DEFINE STORAGE MAP Statement

Examples

Example 1

The following example shows how to define a storage map that stores all records from a relation in the named storage area:

```
RDO> DEFINE STORAGE MAP SALARY_HISTORY_MAP
cont> DESCRIPTION IS /* Map for salary history records */
cont> FOR SALARY_HISTORY RELATION
cont> STORE WITHIN SALARY_HISTORY
cont> END SALARY_HISTORY_MAP STORAGE MAP.
```

This statement stores all the records from the SALARY_HISTORY relation into the storage area, SALARY_HISTORY. SALARY_HISTORY is the name of a storage area that was created with the DEFINE DATABASE statement.

Example 2

The following example uses the PLACEMENT VIA INDEX clause to store records in a storage area according to a hashed index:

```
RDO> DEFINE STORAGE MAP DEPARTMENTS_MAP
cont> FOR DEPARTMENTS RELATION
cont> STORE WITHIN DEPARTMENTS
cont> PLACEMENT VIA INDEX DEPARTMENTS_INDEX
cont> END DEPARTMENTS_MAP STORAGE MAP.
```

Example 3

The following example partitions records from a relation over a number of storage areas according to the values of the specified field:

```
RDO> DEFINE STORAGE MAP EMPLOYEES_MAP
cont> DESCRIPTION IS /* employee records partitioned by employee_id */
cont> FOR EMPLOYEES RELATION
cont> STORE USING EMPLOYEE_ID
cont> WITHIN EMPIDS_LOW WITH LIMIT OF "00200";
cont> EMPIDS_MID WITH LIMIT OF "00400";
cont> EMPIDS_OVER
cont> END EMPLOYEES_MAP STORAGE MAP.
```

Note that the last storage area, EMPIDS_OVER, does not have a WITH LIMIT OF clause associated with it. All records with an EMPLOYEE_ID value over "00400" will be stored in the EMPIDS_OVER storage area. In the DEFINE STORAGE MAP statement, you cannot place a limit on the last storage area.

DEFINE STORAGE MAP Statement

Example 4

The following example defines a database and two relations that contain segmented strings. The two relations are mapped along with their segmented strings to separate storage areas.

```
DEFINE DATABASE SEGSTR_0
  DESCRIPTION IS
  /*
  This database is used to test the Rdb/VMS V4.0 enhanced
  segmented string support. Two relations that contain
  segmented strings are mapped along with their segmented
  string data to separate storage areas.
  */
  DICTIONARY IS NOT USED

  DEFINE STORAGE AREA RDB$SYSTEM
    FILENAME 'SEGSTR_0'
  END

  DEFINE STORAGE AREA SEGSTR_1
    FILENAME 'SEGSTR_1'
  END

  DEFINE STORAGE AREA SEGSTR_2
    FILENAME 'SEGSTR_2'
  END

  DEFINE STORAGE AREA SEGSTR_SS1
    FILENAME 'SEGSTR_SS1'
  END

  DEFINE STORAGE AREA SEGSTR_SS2
    FILENAME 'SEGSTR_SS2'
  END

  SEGMENTED STRING STORAGE AREA IS SEGSTR_SS2.

  DEFINE FIELD STANDARD_DATE
  DATATYPE DATE.

  DEFINE FIELD TITLE
  DATATYPE TEXT SIZE 50.

  DEFINE FIELD DIARY_ENTRY
  DATATYPE SEGMENTED STRING
  SUB_TYPE TEXT
  SEGMENT_LENGTH 200.
```

DEFINE STORAGE MAP Statement

```
DEFINE RELATION DAILY_DIARY
  DESCRIPTION /* Simple diary */.
  ENTRY_DATE      BASED ON STANDARD_DATE.
  TITLE.
  DIARY_ENTRY.
END.

DEFINE RELATION SPECIAL_EVENTS
  DESCRIPTION IS
  /* Special events - birthdays, anniversaries, etc. */.
  EVENT_DATE      BASED ON STANDARD_DATE.
  EVENT_NAME      BASED ON TITLE.
  EVENT_DESCRIPTION BASED ON DIARY_ENTRY.
  SPECIAL_INSTRUCTIONS BASED ON DIARY_ENTRY.
END.

DEFINE STORAGE MAP DAILY_DIARY_MAP
  DESCRIPTION IS
  /* Diary entries are randomly partitioned over two areas */
  FOR DAILY_DIARY
  STORE WITHIN SEGSTR_1; SEGSTR_2
END.

DEFINE STORAGE MAP SPECIAL_EVENTS_MAP
  DESCRIPTION IS
  /* Special events entries are stored in one area */
  FOR SPECIAL_EVENTS
  STORE WITHIN SEGSTR_1
END.

DEFINE STORAGE MAP DIARY_TEXT
  DESCRIPTION IS
  /* Keep segmented strings in separate areas */
  FOR SEGMENTED STRINGS
  STORE WITHIN
    SEGSTR_SS1 FOR DAILY_DIARY, SPECIAL_EVENTS.EVENT_DESCRIPTION;
    SEGSTR_SS2 FOR DAILY_DIARY;
    SEGSTR_1 FOR SPECIAL_EVENTS.SPECIAL_INSTRUCTIONS;
    RDB$SYSTEM      ! RDB$SYSTEM stores other segmented strings
END.
```

DEFINE TRANSFER Statement (VAX Data Distributor)

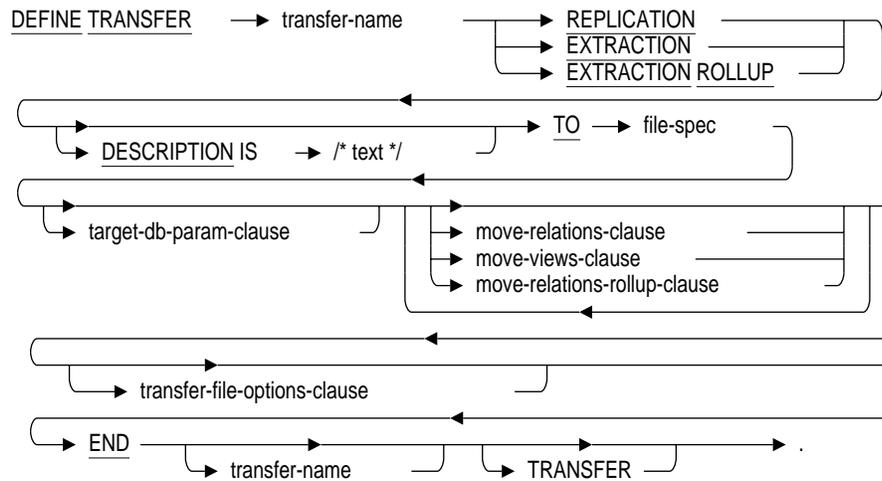
9.21 DEFINE TRANSFER Statement (VAX Data Distributor)

Creates a transfer definition and stores it in the transfer database. A transfer definition contains all the information necessary to select a set of records from the source database and to define a target database. A transfer definition also can contain the views to be defined on the target database and the command procedures to be executed before and after the execution of the transfer.

Note *To use this RDO statement, you must have VAX Data Distributor installed on your system. See your system manager or your Digital Equipment Corporation representative if you need information on Data Distributor.*

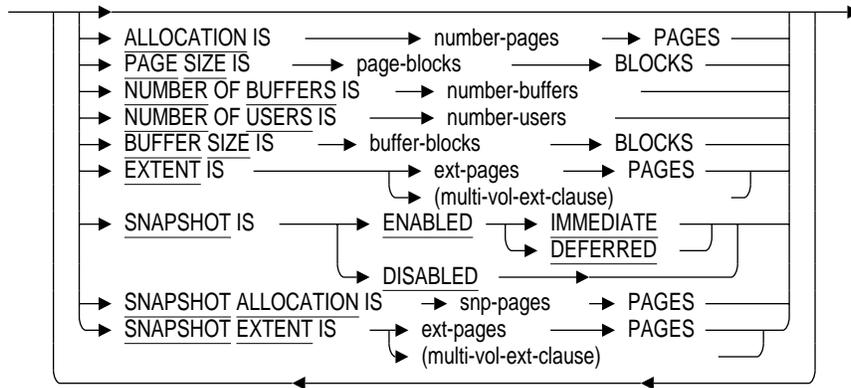
A source database for extraction or extraction rollup transfers can be any supported database, such as Rdb/VMS or VIDA. For replication transfers, the source database must be an Rdb/VMS database.

Format

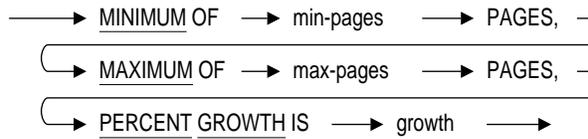


DEFINE TRANSFER Statement (VAX Data Distributor)

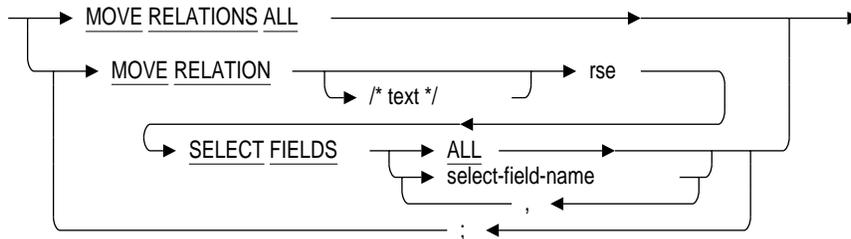
target-db-param-clause =



multi-vol-ext-clause =

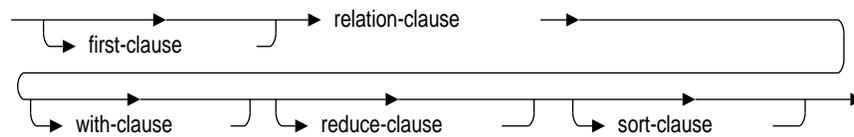


move-relations-clause =

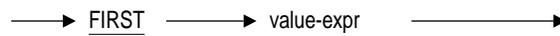


DEFINE TRANSFER Statement (VAX Data Distributor)

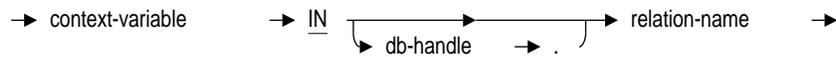
rse =



first-clause =



relation-clause =



with-clause =



reduce-clause =

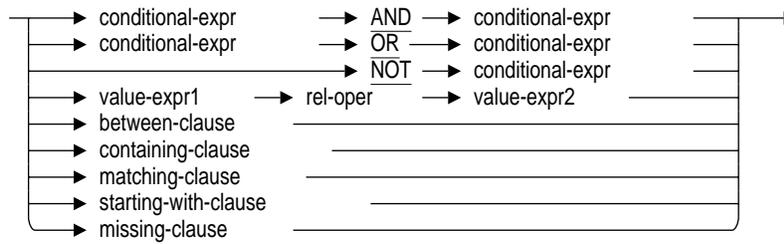


DEFINE TRANSFER Statement (VAX Data Distributor)

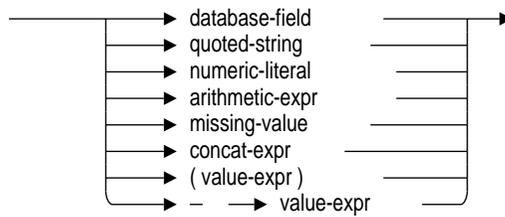
sort-clause =



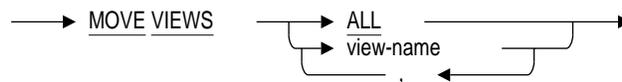
conditional-expr =



value-expr =

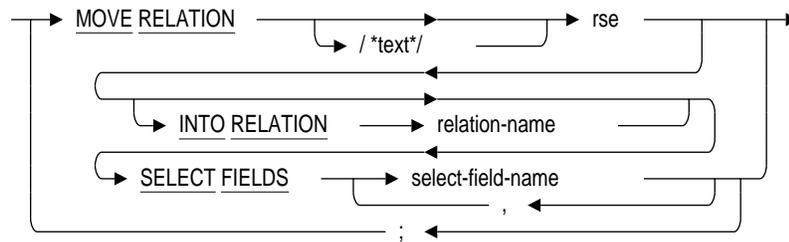


move-views-clause =

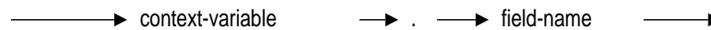


DEFINE TRANSFER Statement (VAX Data Distributor)

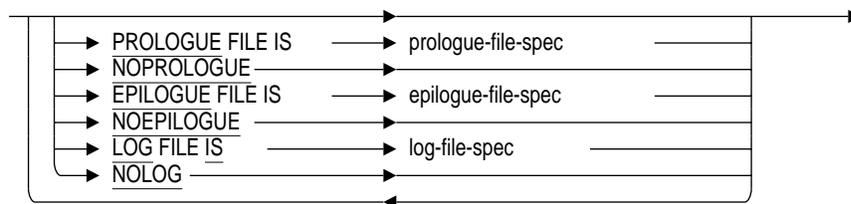
move-relations-rollup-clause =



select-field-name =



transfer-file-options-clause



Arguments

transfer-name

The name that identifies the transfer. The transfer name must be unique among the set of all transfers defined in the transfer database. Because a single transfer database serves all source databases on a node or VAXcluster, the transfer name must be unique across all source databases at a given site.

The transfer-name parameter is required.

DEFINE TRANSFER Statement (VAX Data Distributor)

REPLICATION

Specifies that Data Distributor create a replication database at the target site. When a replication transfer executes, Data Distributor creates a database at the target site during the initial execution and maintains a connection to that target database so the source can supply updates to the target. Subsequent transfers involve only those modifications that have been made to the selected records in the source database. (If the target and source database should become inconsistent with each other, you can use the REINITIALIZE TRANSFER statement to restore consistency.)

In addition, the execution of a replication transfer generally creates special-purpose relations in the source database and the target database to handle updating. Because the source database is modified with the addition of these system relations, the transfer definer's account needs the DEFINE access right to the source database itself.

Users are expected to access the replication target database only to retrieve data. Applications should not attempt to store, modify, or erase values in records of the replication database. Although data values should not be manipulated in a replication database, you can modify the target database by defining views and indexes, and adding access rights.

EXTRACTION

Specifies that Data Distributor create an extraction database at the target site. Because each extraction database is independent of the source database, users at the target directory can access extraction databases with read/write as well as read-only transactions. Each time an extraction mode transfer executes, Data Distributor creates a new version of the target database and transfers a complete copy of the selected records to the target node.

Indexes, constraints, and access rights are not transferred when the target database is created. Once the target database has been created, you can define indexes and constraints for it and modify the access rights.

EXTRACTION ROLLUP

Specifies that Data Distributor create an extraction rollup database at the target site by transferring selected relations and fields from one or more source databases into a single target database. An extraction rollup database is defined at the target site, and Data Distributor pulls data from the source databases into the target.

Each time an extraction rollup transfer is executed, Data Distributor creates a new version of the target database and transfers a complete copy of the selected records to the target node.

DEFINE TRANSFER Statement (VAX Data Distributor)

The EXTRACTION ROLLUP option in the DEFINE TRANSFER statement has the following characteristics and limitations:

- All extraction rollup transfers are *pull* transfers; that is, you define the extraction transfer at the target site to transfer data from a number of remote source databases to a single target database.
- Data Distributor issues an error message if you do not enter a MOVE RELATION clause or a SELECT FIELDS list in a move-relations-rollup-clause.
- You must refer to each source database in the DEFINE TRANSFER statement by its database handle.
- On the target database, relation names and field names must be unique. Consequently, all target relations of the same name must have the same definition, and all target fields of the same name must have the same data type. If the definitions or data types differ, Data Distributor issues an error message and the transfer fails.
- If any database specified in the transfer definition is on a node that is not available, Data Distributor places the transfer in a waiting-to-retry state. Data Distributor will retry the transfer at the time specified for retry in the schedule definition.
- If any of the source nodes specified in the transfer definition fail during the transfer, the transfer will also fail. Data Distributor will retry the transfer at the time specified for retry in the schedule definition.
- If the transfer fails, the retry will start at the beginning, not where the transfer left off.

Indexes, constraints, and access rights are not transferred when the target database is created. Once the target database has been created, you can define indexes and constraints for it and modify the access rights.

file-spec

Designates a file specification for the target database. The target database is an Rdb/VMS database. Data Distributor uses the file specification to create and access the target database. You can use either a full or partial file specification or a logical name. The file-spec parameter is required. If you do not specify the node, device, or directory, Data Distributor assumes the SYSS\$LOGIN of the user defining the transfer. If the file type is not RDB, the transfer will fail during the creation of the target database. The default file type is RDB.

DEFINE TRANSFER Statement (VAX Data Distributor)

When you specify the name of a remote node as part of the file specification, you can include an access control list (ACL) for login. However, the most effective method of allowing access to a target node uses proxy accounts.

Note that the file specification is resolved at run time. All logical names used for the file-spec parameter are resolved within the copy process after it executes the transfer definer's login initialization file, LOGIN.COM. If the target database name is a logical symbol name, when it is translated, the default directory is the one pointed to by SYS\$LOGIN.

Data Distributor makes no attempt to verify whether a database or file of the same name already exists in the target directory. When you issue a duplicate file specification, Data Distributor creates a second file with the same name, but a higher version number.

If the database system (Rdb/VMS) returns an error while attempting to create a database in the target directory, the copy process writes the error in the copy process log file and returns the error status to the transfer monitor.

text

A text string that allows you to associate a comment with the transfer definition. The text string, together with other parts of the transfer definition, appears in the SHOW TRANSFER DEFINITION display. The DESCRIPTION IS clause is optional.

target-db-param-clause

Enables you to specify values for the parameters that affect the creation of the target database. This clause uses syntax similar to the Rdb/VMS DEFINE DATABASE statement. However, Data Distributor does not support all the parameters that DEFINE DATABASE supports. Data Distributor uses the values provided in this clause to create the target database.

number-pages

The number of database pages initially allocated to the target database. Rdb/VMS automatically extends the allocation to handle the loading of data and subsequent expansion. The default is 400 pages. If you are transferring a large number of records, a large page allocation helps to prevent fragmented records.

page-blocks

The size in blocks of each database page in the target directory. Page size is allocated in 512-byte blocks. The default is two blocks (1024 bytes). If your largest record has more than approximately 950 bytes, you can specify more blocks for each page to prevent records from being fragmented.

DEFINE TRANSFER Statement (VAX Data Distributor)

number-buffers

The number of buffers Rdb/VMS allocates for each process using the target database. The value must be an unsigned integer between 2 and 32768. The default is 20 buffers.

number-users

The maximum number of users allowed to access the target database at one time. The default is 50 users. After the maximum is reached, the next user who tries to invoke the database receives an error message and must wait for access. The largest number of users you can specify is 508 and the smallest is 1.

buffer-blocks

The number of blocks Rdb/VMS allocates for each buffer. The value must be greater than zero. If you do not specify this parameter, Rdb/VMS uses a buffer size that is three times the page size value.

ENABLED IMMEDIATE

Specifies that read/write transactions write previously updated copies of records they modify to the snapshot file, regardless of whether a read-only transaction is active. Snapshot writing is enabled immediately by default. With snapshots enabled, users or applications can read from either the snapshot file or the database file itself.

ENABLED DEFERRED

Specifies that read/write transactions not write previously updated copies of records they modify to the snapshot file unless a read-only transaction is already active. Read-only transactions that attempt to start after an active read/write transaction must wait for all active read/write users to complete their transactions. With snapshots enabled, users or applications can read from either the snapshot file or the database file itself.

DISABLED

Specifies that snapshot writing be disabled. There is usually some performance gain when snapshot writing is disabled because Rdb/VMS does not need to perform write operations to the snapshot file during updates to the database.

You should not disable snapshots for replication databases. Although users are expected only to retrieve data from the replication database, retrieval can occur in either read-only or read/write transactions.

DEFINE TRANSFER Statement (VAX Data Distributor)

snp-pages

The number of pages allocated for the snapshot file. The default is 100 pages. After the target database has been created, you can use the RDO CHANGE DATABASE statement to truncate the snapshot file. To truncate the file, specify an allocation size smaller than the current snapshot file size.

ext-pages

The number of pages of each target database extent. This parameter provides direct control over the extent of the target database. For greater control, and particularly for multivolume databases, use the MINIMUM, MAXIMUM, and PERCENT GROWTH parameters of the multi-vol-ext-clause instead. The default extent is 100 pages.

min-pages

The minimum number of pages of each target database extent. The default is 100 pages.

max-pages

The maximum number of pages of each target database extent. The default is 10,000 pages.

growth

The percent of growth for each target database extent. The default is 20 percent growth.

move-relations-clause

Selects the records Data Distributor transfers to the target database during an extraction or a replication transfer. The records selected for replications are specified with a restricted form of a record selection expression (RSE). You must specify a move-relations-clause for each user relation moved to the target database. If you do not include a move-relations-clause, Data Distributor assumes the default, MOVE RELATIONS ALL.

When you specify a view in a MOVE RELATION clause, the records selected by that view are created as a relation in the target database. You can also transfer views using the move-views-clause.

You must have READ privilege for the relations of an Rdb/VMS source database that you specify in a MOVE RELATION clause or a MOVE RELATIONS ALL clause. For VIDA databases, Data Distributor does not check for READ privilege for views.

DEFINE TRANSFER Statement (VAX Data Distributor)

MOVE RELATIONS ALL

Allows you to have Data Distributor select all records and fields from all user relations, excluding views, that are in the source database at the time the transfer is defined. If you wish to move views, you must use the `move-views-clause` or the `MOVE RELATION` clause. Fields defined with a `COMPUTED_BY` clause are restricted to those that use the value expressions shown in the `value-expr` syntax diagram in this section. If a field is not defined by a supported value expression, you will receive a warning message, and the `COMPUTED_BY` fields will not be transferred.

If you define a transfer on a source database that is itself a target extraction database, do not use the `MOVE RELATIONS ALL` clause. An error will result during transfer execution because Data Distributor will attempt to define the `DDAL$VINTAGE` relation in the new target database twice. The second transfer would create a `DDAL$VINTAGE` relation in the target database. Then, using the `MOVE RELATIONS ALL` clause, it would attempt to copy the `DDAL$VINTAGE` relation that already exists in the source database. To avoid this error, specify explicitly each relation to be moved instead of using `MOVE RELATIONS ALL`.

Do not use the `MOVE RELATIONS ALL` statement for an extraction rollup transfer. If you use the `MOVE RELATIONS ALL` statement to define an extraction rollup transfer, the transfer will fail on execution. Instead, you must name the individual relations you want to transfer to the extraction rollup target.

rse

The record selection expression (RSE) used to identify records from the source database that Data Distributor includes in a record stream and transfers to the target database. The RSE specifies conditions that individual records must satisfy in order to be included in the transfer to the target database.

Data Distributor supports `COMPUTED_BY` fields named in an RSE if the fields are defined using the value expressions shown in the `value-expr` syntax diagram in this section. If the fields are not defined by a supported value expression, you will receive a fatal error message.

In extraction, extraction rollup, and replication transfers, you can use a `relation-clause` and `with-clause`. In extraction and extraction rollup transfers, you can also use a `first-clause`, `reduce-clause`, and `sort-clause`.

DEFINE TRANSFER Statement (VAX Data Distributor)

first-clause

Specifies how many records are selected from the record stream formed by the record selection expression (RSE). You can only use the first-clause for an extraction or an extraction rollup transfer. If you specify a first-clause for a replication transfer, Data Distributor issues an error message.

relation-clause

The context variables and relations to be included in the record stream or loop.

If you specify a view name in the relation-clause, the moved view becomes a relation in the target database.

context-variable

Supplies a temporary name to identify the record stream. Once you have associated a context-variable with a relation, you use the context-variable to refer to fields from that relation.

db-handle

A host variable or name used to refer to a specific database you have invoked.

In extraction or replication transfers, you should not use database handles in the RSE of a MOVE RELATION clause. Data Distributor returns an error message if you include a database handle in the RSE. However, in an extraction rollup transfer, you must use database handles in the RSE to specify the relation or Data Distributor will return an error message.

with-clause

Contains a conditional expression that allows you to specify conditions that must be true for a record to be included in a record stream.

conditional-expr

Conditions that must be true for a record to be included in a record stream.

reduce-clause

Allows you to eliminate duplicate values for fields in a record stream. You can use the reduce-clause for an extraction or an extraction rollup transfer. If you specify a reduce-clause for a replication transfer, Data Distributor issues an error message.

sort-clause

Allows you to sort the records in the record stream by the values of specific fields. You can sort the records according to a value expression called a sort key. The sort key determines the order of the records in the record stream. You can use the sort-clause for an extraction or an extraction rollup transfer.

DEFINE TRANSFER Statement (VAX Data Distributor)

If you specify a sort-clause for a replication transfer, Data Distributor issues an error message.

ASCENDING

Specifies that the records be sorted in ascending order. The default is ASCENDING.

DESCENDING

Specifies that Rdb/VMS sort the records in descending order.

value-expr

A value expression specifying the sort value. This value is called the sort key.

- database-field
- quoted-string
- numeric-literal
- arithmetic-expr
- missing-value
- concat-expr

select-field-name

Allows you to name a specific field or group of fields you want moved to the target database. You can specify only one SELECT clause for each MOVE clause. Data Distributor moves only the named fields to the target database. If you do not specify SELECT FIELDS ALL, the select-field-name clause is required.

Data Distributor supports COMPUTED_BY fields named in the select-field-name clause if the fields are defined using the value expressions shown in the value-expr syntax diagram in this section. If the fields are not defined by a supported value expression, you will receive a fatal error message.

SELECT FIELDS ALL

Enables you to include every field in a specific relation. Fields defined with a COMPUTED_BY clause are restricted to those that use the value expressions shown in the value-expr syntax diagram in this section. If a field is not defined by a supported value expression, you will receive a warning message, and the COMPUTED_BY fields will not be transferred. You cannot use the SELECT FIELDS ALL statement for an extraction rollup transfer.

DEFINE TRANSFER Statement (VAX Data Distributor)

move-views-clause

Selects the views Data Distributor defines on the target database during an extraction or a replication transfer. All the relations, views, and fields on which the transferred views are based must also be explicitly or implicitly named in the DEFINE TRANSFER statement. The transfer definition will fail if any underlying relation, view, or field is not named when you define the transfer.

If you specify a view name in this clause, the view definition is transferred to the target database. You can include only one *move-views-clause* in a DEFINE TRANSFER statement. However, you can specify two or more views in that *move-views-clause*.

Transferring views is limited by the following restrictions:

- **COMPUTED_BY fields**

Data Distributor transfers those COMPUTED_BY fields that depend only on the values of the relation in which the field is defined. Views that contain unsupported COMPUTED_BY fields cannot be transferred. If views that contain unsupported COMPUTED_BY fields are specified in the *move-views-clause* of a DEFINE TRANSFER statement, the transfer definition fails. You need to redefine the transfer, excluding the views that refer to the unsupported COMPUTED_BY fields.

- **READ privilege for the view**

For a view in an Rdb/VMS database to be transferred, Data Distributor requires that you have Rdb/VMS READ privilege for the view and for the underlying relations. If you do not have READ privilege for views you specify in the *move-views-clause* of a DEFINE TRANSFER statement, the transfer will fail. You need to obtain READ privilege for these views before including them in a DEFINE TRANSFER statement.

For VIDA databases, Data Distributor does not check for READ privilege for views.

You can also move views with the *move-relations-clause*. However, when you use the *move-relations-clause* to move a view, the view becomes a relation in the target database.

view-name

The name of the view Data Distributor transfers to the target during a transfer. If you do not specify a view name, Data Distributor assumes the default, MOVE VIEWS ALL.

DEFINE TRANSFER Statement (VAX Data Distributor)

move-relations-rollup-clause

Selects the records Data Distributor transfers into the target database from a number of source databases during an extraction rollup transfer. The *move-relations-rollup-clause* has the following restrictions:

- Data Distributor requires you to invoke each source database with a different database handle. You must then use these handles in the *move-relations-rollup-clause*.
- The *move-relations-rollup-clause* must be used with the EXTRACTION ROLLUP parameter.
Rollup transfers must be extractions. If you attempt to use the *move-relations-rollup-clause* with a replication transfer, the transfer definition will fail.
- The MOVE RELATIONS ALL and SELECT FIELDS ALL clauses are not allowed in rollup transfers.

transfer-file-options-clause

The DCL command procedures to be executed before or after transfers and the VMS file where logging information related to the transfer execution is located.

prologue-file-spec

The name of the prologue command procedure file to be run before the transfer executes. This file runs at the beginning of the first execution of the transfer and of all subsequent transfers. The prologue file is a DCL command file. You can include a version number and a remote node name in the file specification. The name is resolved when the transfer is defined.

The *prologue-file-spec* parameter is optional. If you do not include this parameter, Data Distributor assumes that NOPROLOGUE is in effect. The default file type is COM. The default directory for the prologue file is SYS\$LOGIN.

epilogue-file-spec

The name of the epilogue command procedure file to be run after the transfer executes. This file runs after the first execution of the transfer and after all subsequent transfers. The epilogue file is a DCL command file. You can include a version number or a remote node name in the file specification. The name is resolved when the transfer is defined.

DEFINE TRANSFER Statement (VAX Data Distributor)

The *epilogue-file-spec* parameter is optional. If you do not include this parameter, Data Distributor assumes that NOEPILOGUE is in effect. The default file type is COM. The default directory for the epilogue file is SYS\$LOGIN.

log-file-spec

The VMS file where the Data Distributor copy process writes any logging information related to the execution of the transfer. If you include a version number or a node name in the file specification, Data Distributor returns an error. Data Distributor creates a new log file every time a transfer executes. The name is resolved when the transfer is defined.

The *log-file-spec* parameter is optional. If you do not include this parameter, Data Distributor assumes that NOLOG is in effect. The default file type is LOG. The default directory for the log file is SYS\$LOGIN.

Usage Notes

You need the Rdb/VMS WRITE privilege for the transfer relation to use the DEFINE TRANSFER statement.

You must have the Rdb/VMS READ privilege for the relations and views of an Rdb/VMS source database that you specify in a DEFINE TRANSFER statement. Data Distributor returns an error message when you specify relations and views in a DEFINE TRANSFER statement for which you do not have READ privilege. If you specify a VIDA database, Data Distributor does not check for READ privilege for the relations and views.

After you issue the DEFINE TRANSFER statement, the transfer does not execute until one of the following events occurs:

- 1 The time specified in the corresponding DEFINE SCHEDULE statement arrives.
- 2 You issue a START TRANSFER NOW statement.

You must execute the DEFINE TRANSFER statement outside of the scope of a transaction. If you issue this statement when a transaction is outstanding, Data Distributor returns an error message.

You can refer to relations, fields, and views in a DEFINE TRANSFER statement either implicitly or explicitly. When you include either the MOVE RELATIONS ALL, SELECT FIELDS ALL, or MOVE VIEWS ALL clause, you refer to relations, fields, or views implicitly. Data Distributor uses the current definitions of the relations, fields, and views in the source database and

DEFINE TRANSFER Statement (VAX Data Distributor)

stores those names explicitly in the transfer definition. You refer to relations, fields, and views explicitly when you include the MOVE RELATION, SELECT FIELDS, or MOVE VIEWS clauses and supply specific relation, field, or view names.

Data Distributor transfers those COMPUTED_BY fields that depend only on the values of a relation in which the field is defined. You cannot transfer views that contain unsupported COMPUTED_BY fields. If views that contain unsupported COMPUTED_BY fields are specified in the move-views-clause of a DEFINE TRANSFER statement, the transfer definition fails. You need to redefine the transfer, excluding the views that refer to the unsupported COMPUTED_BY fields.

Data Distributor allows you to transfer views in two ways:

- Using the move-views-clause
Data Distributor transfers the definition of the view when you specify MOVE VIEWS view-name or MOVE VIEWS ALL. You must also explicitly or implicitly name in the DEFINE TRANSFER statement all the relations, views, or fields on which the transferred view definitions are based. The transfer will fail if any underlying relations, views, or fields are not named.
- Using the move-relations-clause
Data Distributor transfers a view by creating a relation in the target database containing the view's records when you use the MOVE RELATION clause. This method improves query performance in the target database. For example, you can achieve the effect of a CROSS clause by transferring a view that is a CROSS in the source.

You specify the relations you wish to move in the move-relations-clause and the views in the move-views-clause of the DEFINE TRANSFER statement. When you are doing an extraction rollup transfer, you specify the relations you wish to move in the move-relations-rollup-clause of the DEFINE TRANSFER statement.

The same clause can occur in a DEFINE TRANSFER statement no more than once. You cannot specify the same relation or view more than once in the move clauses. If a relation name or view name appears in more than one move clause, Data Distributor returns an error.

DEFINE TRANSFER Statement (VAX Data Distributor)

Individual move-relations clauses are separated by semicolons, as indicated in the syntax diagrams. You can implement the move-relations-clause in two ways:

- Specify MOVE RELATIONS ALL
- Specify a series of relations separated by semicolons

The move-relations-clause has the following characteristics and limitations:

- All fields in each move-relations-clause must reside in the same relation.
- You cannot name the same relation in more than one MOVE clause of a single transfer definition.

The move-relations-rollup-clause has the following characteristics and limitations:

- All fields in each move-relations-rollup-clause must reside in the same relation.
- You cannot name the same relation in more than one MOVE clause of a single transfer definition.
- For each relation of a move-relations-rollup-clause that you enter, you can supply a target relation specification in the form of an INTO RELATION clause. You can create one or many relations in the target database from several source database relations by renaming the target relations in the transfer definition.
- You cannot specify MOVE RELATIONS ALL or SELECT FIELDS ALL in a move-relations-rollup-clause. When you specify these clauses in the DEFINE TRANSFER statement, the transfer will fail at the time of definition.

If a transfer fails during execution, the transfer will begin again at one of the two following times:

- The next scheduled retry time if the failure results from a nonfatal error
- The next scheduled transfer time if the failure results from a fatal error

This next transfer will start from the beginning to ensure consistency with any updates that might have been made to the source database.

DEFINE TRANSFER Statement (VAX Data Distributor)

Data Distributor transfers only definitions for the database, fields, and relations specified in the transfer definition. If the source database includes additional database entities such as indexes, constraints, or access control lists, these are not transferred to the target database. You must either directly access the individual target database and explicitly define these entities or run command procedures to define the entities after the transfer executes.

When relations specified in the transfer definition are created in the target database, they are given the access control lists (ACLs) that have the Rdb/VMS default values. You can change the ACLs in the target database after it has been created.

In a transfer definition, you can use logical symbol names to name the source database file, the target database file, the copy process log file, and the prologue and the epilogue command procedure files. However, Data Distributor treats these file names differently:

- Source database file logical names

If the source file name is a logical symbol that translates to a standard VMS file name specification, Data Distributor performs the logical name translation. The SHOW TRANSFER DEFINITION statement shows the source file name in its translated form. The logical symbol can also translate into strings that, in turn, contain logical symbols. As long as the resulting string looks like a VMS file specification and the logical symbol is not a concealed device name, the translation process is repeated.

If the logical symbol translates into a VIDA access string (for example, /TYPE=VIDA/FILE=ACCESS_FILE/ . . .), Data Distributor does no further translations. Even if the VIDA name string contains logical symbol references, these references are not translated into VMS file specifications. You can use a logical name in place of the string to identify a VIDA database.

- Target database file logical name

If the target database name is a logical name, it is not translated by Data Distributor when it is stored in the transfer definition.

- Copy process log file logical name

Data Distributor translates the log file logical name in the same way as source file logical names, with the following exceptions:

- The log file specification is translated in RDO when the DEFINE TRANSFER statement is executed.

DEFINE TRANSFER Statement (VAX Data Distributor)

- These log file names must be standard VMS file specifications. Node names or version numbers are not allowed.
- Prologue and epilogue command procedure file logical names
Data Distributor translates the prologue and epilogue file logical names in the same way as the source file logical name, with the following exceptions:
 - The prologue and epilogue file specifications are translated in RDO when the DEFINE TRANSFER statement is executed.
 - The log file names must be standard VMS file specifications. Node names and version numbers are allowed.

Examples

Example 1

The following example defines a transfer, NH_EMPLOYEES, that creates an extraction database. The transfer definition selects New Hampshire employees address records for transfer to the remote node.

```
RDO> INVOKE DATABASE FILENAME PERSONNEL
RDO> DEFINE TRANSFER NH_EMPLOYEES EXTRACTION
cont> DESCRIPTION IS /* Employees who live in New Hampshire */
cont> TO NODE1::DISK1:[ADAMS]NH_EMP
cont> MOVE RELATION E IN EMPLOYEES WITH E.STATE = "NH"
cont> SELECT FIELDS E.EMPLOYEE_ID,
cont>     E.LAST_NAME,
cont>     E.FIRST_NAME,
cont>     E.MIDDLE_INITIAL,
cont>     E.ADDRESS_DATA_1,
cont>     E.ADDRESS_DATA_2,
cont>     E.CITY,
cont>     E.STATE,
cont>     E.POSTAL_CODE
cont> LOG IS NH_EMPLOYEES.LOG
cont> END.
```

Example 2

This example shows an extraction from a remote source database to a local target database. The DEFINE TRANSFER statement specifies the transfer of all personnel records to the target node.

DEFINE TRANSFER Statement (VAX Data Distributor)

```
RDO> INVOKE DATABASE FILENAME NODE1::DISK1:[DIR1]PERSONNEL
RDO> DEFINE TRANSFER PERS_EXAMP EXTRACTION
cont> DESCRIPTION IS /* Extraction of personnel records*/
cont> TO [LEARNER.PERS]PERS_COPY
cont> MOVE RELATIONS ALL
cont> LOG IS [LEARNER.PERS]PERS_EXAMP.LOG
cont> END.
```

Example 3

This example shows an extraction transfer from a remote source to a remote target database. The log file, PERS_EXAMP.LOG, is located on the source node.

```
RDO> INVOKE DATABASE FILENAME NODE1::DISK1:[DIR1]PERSONNEL
RDO> DEFINE TRANSFER PERS_EXAMP EXTRACTION
cont> DESCRIPTION IS /* Extraction of personnel records*/
cont> TO NODE2::DISK2:[PROTO]PERS_TARGET
cont> MOVE RELATIONS ALL
cont> LOG IS PERS_EXAMP.LOG
cont> END.
```

Example 4

This example of a replication transfer definition sends the entire colleges relation and the entire degrees relation to the target database. The replication database can be accessed for information about employees' college and degree histories. The transfer definition includes some database parameters for the target database.

```
RDO> INVOKE DATABASE FILENAME PERSONNEL
RDO> DEFINE TRANSFER COLLEGE_INFO REPLICATION
cont> DESCRIPTION IS /* Info about employees' colleges only */
cont> TO NODE1::DISK1:[BURTON]EMP_COLLEGE
cont> NUMBER OF USERS IS 25
cont> SNAPSHOT ALLOCATION IS 150 PAGES
cont> MOVE RELATION C IN COLLEGES
cont> SELECT FIELDS ALL;
cont> MOVE RELATION D IN DEGREES
cont> SELECT FIELDS ALL
cont> LOG IS COLLEGE_INFO.LOG
cont> END.
```

Example 5

This replication transfer specifies three fields from the CREDIT_CARDS source database for transfer to the target database. The fields are CARD_NUM, EXPIRATION_DATE, and VALID. Only invalid credit card records are selected for the transfer.

DEFINE TRANSFER Statement (VAX Data Distributor)

```
RDO> INVOKE DATABASE FILENAME CREDIT_CARDS
RDO> DEFINE TRANSFER BAD_CARDS REPLICATION
cont> DESCRIPTION IS /* Invalid credit cards */
cont> TO NODE1::DISK1:[CREDIT]BAD_CARDS
cont> MOVE RELATION C IN CARDS WITH C.VALID = "N"
cont> SELECT FIELDS C.CARD_NUM,
cont>     C.EXPIRATION_DATE,
cont>     C.VALID
cont> LOG IS BAD_CARDS.LOG
cont> END.
```

Example 6

The following example defines a replication transfer that sends the entire EMPLOYEES, JOB_HISTORY, SALARY_HISTORY, DEPARTMENTS, and JOBS relations. The definition also selects three views for transfer: CURRENT_INFO, CURRENT_JOB, and CURRENT_SALARY.

```
RDO> INVOKE DATABASE FILENAME PERSONNEL
RDO> DEFINE TRANSFER NH_PERSONNEL REPLICATION
cont> TO NODE1::DISK1:[DBADMIN]NHP.RDB
cont> MOVE RELATION E IN EMPLOYEES           SELECT FIELDS ALL;
cont> MOVE RELATION JH IN JOB_HISTORY        SELECT FIELDS ALL;
cont> MOVE RELATION SH IN SALARY_HISTORY     SELECT FIELDS ALL;
cont> MOVE RELATION D IN DEPARTMENTS        SELECT FIELDS ALL;
cont> MOVE RELATION J IN JOBS                SELECT FIELDS ALL
cont> MOVE VIEWS CURRENT_INFO, CURRENT_JOB, CURRENT_SALARY
cont> LOG FILE IS DISK1:[DBADMIN]NHP.LOG
cont> END.
```

Example 7

The following example moves the CURRENT_JOB view defined in the source database as a CROSS of the EMPLOYEES relation and the JOB_HISTORY relation. You can achieve the effect of a CROSS clause by transferring a view that is a CROSS in the source database. On the target database, the records selected by the view are created as a relation.

```
RDO> INVOKE DATABASE FILENAME PERSONNEL
RDO> DEFINE TRANSFER CJOB EXTRACTION
cont> TO NODE1::DISK1:[RAMESH]JOB_DB
cont> MOVE RELATION C IN CURRENT_JOB WITH C.JOB_CODE = "SEII"
cont>     SELECT FIELDS C.LAST_NAME, C.FIRST_NAME, C.ADDRESS
cont> LOG FILE IS CJOB.LOG
cont> END TRANSFER.
```

DEFINE TRANSFER Statement (VAX Data Distributor)

Example 8

The following example shows an extraction rollup transfer that invokes several source databases: PERSONNEL, WORKERS, and STAFF. The transfer selects fields in the EMPLOYEES relations from the three databases identified by the database handles DB1, DB2, and DB3 and located on three different nodes and in the JOBS relation from DB1. Then the transfer moves these relations to a single target database, LOCAL_DB.

```
RDO> INVOKE DATABASE DB1=FILENAME "NODE1::DISK1:[DIR1]PERSONNEL"
RDO> INVOKE DATABASE DB2=FILENAME "NODE2::DISK2:[DIR2]WORKERS"
RDO> INVOKE DATABASE DB3=FILENAME "NODE3::DISK3:[DIR1]STAFF"
RDO> DEFINE TRANSFER LABOR EXTRACTION ROLLUP
cont> TO NODE4::DISK4:[DIR4]LOCAL_DB
cont> MOVE RELATION E IN DB1.EMPLOYEES WITH E.EMP_CODE= "A+"
cont> INTO RELATION EXEMPT_EMPLOYEES
cont> SELECT FIELDS E.EMP.ID,E.SALARY,E.JOB_CODE;
cont> MOVE RELATION E IN DB2.EMPLOYEES WITH E.EMP_CODE= "A+"
cont> INTO RELATION EXEMPT_EMPLOYEES
cont> SELECT FIELDS E.EMP_ID,E.SALARY,E.JOB_CODE;
cont> MOVE RELATION E IN DB3.EMPLOYEES WITH E.EMP_CODE= "A+"
cont> INTO RELATION EXEMPT_EMPLOYEES
cont> SELECT FIELDS E.EMP.ID,E.SALARY,E.JOB_CODE;
cont> MOVE RELATION J IN DB1.JOBS WITH J.JOB_CODE= "J17"
cont> INTO RELATION ENTRY_JOBS
cont> SELECT FIELDS J.JOB_NAME,J.JOB_CODE
cont> LOG FILE IS DISK4:[DIR4]LOCAL.LOG
cont> END.
```

Example 9

The following EMPLOYEES transfer creates an extraction database. The transfer selects records using a first-clause, with-clause, reduce-clause, and sort-clause.

```
RDO> INVOKE DATABASE FILENAME PERSONNEL
RDO> DEFINE TRANSFER EMPLOYEES EXTRACTION
cont> TO NODE1::DISK1:[NELSON]EMPLOYEES_INFO
cont> MOVE RELATION FIRST 10 E IN EMPLOYEES
cont> WITH E.CITY = "BOSTON"
cont> REDUCED TO E.LAST_NAME, E.FIRST_NAME, E.ADDRESS
cont> SORTED BY DESCENDING E.LAST_NAME
cont> SELECT FIELDS E.LAST_NAME, E.FIRST_NAME, E.ADDRESS
cont> LOG FILE IS DISK1:[NELSON]EMPLOYEES.LOG
cont> END TRANSFER.
```

DEFINE TRANSFER Statement (VAX Data Distributor)

Example 10

The following extraction transfer includes two command files. The prologue command procedure establishes a connection to a remote node, and the epilogue command procedure terminates the connection. This example shows how to specify the prologue file and epilogue file; it does not show the command procedures.

```
RDO> INVOKE DATABASE FILENAME NODE1::DISK1:[DBADMIN.PERS]PERSONNEL
RDO> DEFINE TRANSFER PERS_SAMPLE EXTRACTION
cont> TO PERS_COPY
cont> MOVE RELATIONS ALL
cont> PROLOGUE FILE IS DIALUP.COM
cont> EPILOGUE FILE IS HANGUP.COM
cont> LOG IS PERS_SAMPLE.LOG
cont> END.
```

DEFINE TRIGGER Statement

9.22 DEFINE TRIGGER Statement

Creates a trigger for the specified relation. A **trigger** is a mechanism that associates a set of rules with an update operation. Each trigger is associated with a single subject relation, is evaluated at a specific time for a particular type of update on that relation, and specifies one or more “triggered” actions. Each triggered action consists of an optional condition and one or more statements to be evaluated either once or for each record of the relation being updated.

With triggers, you can define such useful operations as:

- **Cascading delete**
Erasing a record from one relation causes additional records to be erased from other relations that are related to the first by key values.
- **Cascading update**
Modifying a record from one relation causes additional records to be modified in other relations which are related to the first by key values. These modifications are commonly limited to the key fields themselves. You can use triggers with relation-specific constraints to preserve database integrity.
- **Summation update**
Modifying a record from one relation causes a value in a record of another relation to be modified by being incremented or decremented.

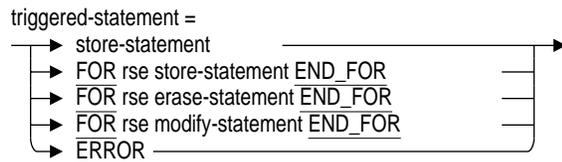
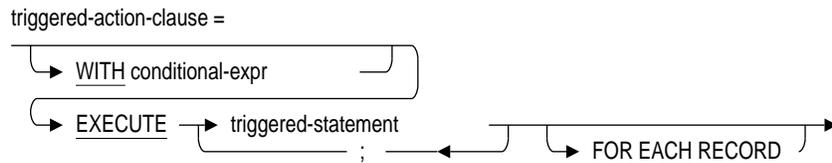
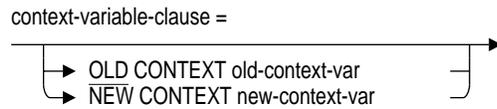
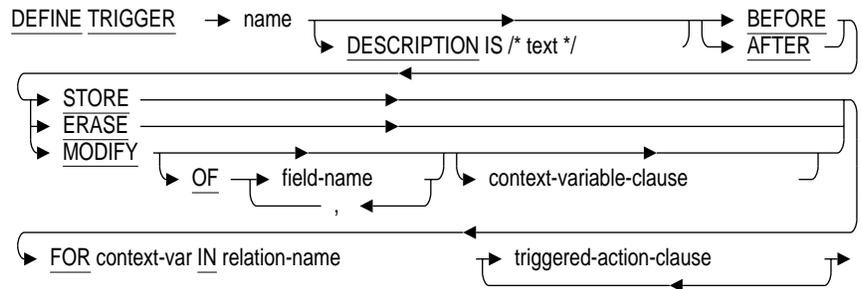
Note Combinations of relation-specific constraints and appropriately defined triggers are not sufficient to guarantee that database integrity is preserved on updates. They must be used in conjunction with a common set of update procedures that assures completely reproducible and consistent retrieval and update strategies if integrity is to be preserved.

The DEFINE TRIGGER statement adds a trigger definition to the physical database.

If the database has been invoked by path name, the definition will *not* be stored in the data dictionary. Therefore an inconsistency between the definitions in the database and the dictionary will exist, and the triggers must be redefined whenever the database metadata is restored from the dictionary with the INTEGRATE statement.

DEFINE TRIGGER Statement

Format



Arguments

name

The name of the trigger being defined. The name must be unique within the database.

DEFINE TRIGGER Statement

text

Text string used as a comment for the trigger definition.

field-name

The name of a field within the specified relation to be checked for modification. You can specify a list of field names separated by commas. You can also specify all the fields in a relation by specifying the asterisk (*) wildcard character.

Be sure that you specify *only* those fields whose data values are to be changed. See the Usage Notes for more information.

context-var

A temporary name defining the relation on which the trigger is defined. The context variable name is an alias for the record stream context of the triggering statement that caused this trigger to be executed. If you do not specify the FOR EACH RECORD clause, none of the context variables can be referenced in the triggered statement.

relation-name

The name of the relation for which this trigger is defined (subject relation).

WITH conditional-expr

A conditional expression that describes the optional condition that must be satisfied before the associated triggered statements are executed. This expression cannot refer to any host language variables.

FOR EACH RECORD

A clause that determines whether an action is evaluated once per triggering statement, or for each record of the subject relation that is updated by the triggering statement. If the FOR EACH RECORD clause is not specified, the triggered action is evaluated only once, and record values are not available to the triggered action.

old-context-var

A temporary name used to refer to the record values as they exist before a modify operation occurs. You use an old-context-var only when the trigger occurs after a modify operation. The old-context-var name must be different than the name given for context-var. If you do not specify the FOR EACH RECORD clause, this context variable cannot be referenced in the triggered statement.

DEFINE TRIGGER Statement

new-context-var

A temporary name used to reference the new record values about to be applied by the modify operation. You use a new-context-var only when the trigger occurs before a modify operation. The new-context-var name must be different than the name given for context-var. If you do not specify the FOR EACH RECORD clause, this context variable cannot be referenced in the triggered statement.

rse

A record selection expression that defines which records of which relations are affected by the triggered update action. This rse cannot refer to any host variables. The record selection expression includes a context variable that is a temporary name for the subject relation.

store-statement

A STORE statement.

erase-statement

An ERASE statement.

modify-statement

A MODIFY statement.

Usage Notes

To define a trigger, you need the Rdb/VMS READ and DEFINE privileges to the subject relation. If any triggered statement specifies some form of update operation, then CONTROL and the appropriate update privilege (ERASE, MODIFY, or WRITE) to the relations specified by the triggered action statements are also required.

Segmented strings cannot be updated with a triggered statement.

The trigger definition specifies when the trigger is to be evaluated relative to a specified database update operation (triggering statement) on the subject relation, and lists one or more triggered actions to be evaluated for that update operation. If the trigger specifies multiple triggered actions, each action is evaluated in the order in which it appears within the trigger definition.

The triggering statement can be:

- A STORE statement
- A FOR record selection expression enclosing a STORE statement

DEFINE TRIGGER Statement

- A FOR record selection expression enclosing an ERASE statement
- A FOR record selection expression enclosing a MODIFY statement
- A request to generate an exception

If the FOR record selection expression is not satisfied, the STORE, ERASE, or MODIFY statement will not be executed.

The trigger specification includes an action time, an update event, and an optional field list, which together determine when the trigger is to be evaluated. The action time can be specified as either before or after the triggering STORE, ERASE, or MODIFY statement (the type of update). For triggers evaluated on MODIFY statements, you can specify an optional list of fields (from the subject relation) to further stipulate that the trigger is to be evaluated only when any one of the fields listed is modified.

A triggered action consists of an optional condition clause, one or more triggered statements, and a frequency clause. Any specified condition must evaluate to true (cause records to be selected) for the triggered statements in the action to be executed. Each triggered statement in the action is executed in the order in which it appears within the triggered action definition.

In a BEFORE MODIFY or AFTER MODIFY trigger, be sure that you specify only the names of fields whose data values are to be changed, to avoid potentially unnecessary actions such as:

- Overlaying the data with itself within a record.
- Writing to the database (even though none of the fields in the record has actually changed values)
- Evaluating constraints that apply to fields in the MODIFY field list which have not changed values
- Evaluating MODIFY triggers that apply to fields in the MODIFY field list that have not changed values.
- Evaluating RDO VALID IF clauses for fields in the MODIFY field list that have not changed values.

If there is a possibility that any of the fields in a MODIFY field list will not actually be changed, the triggered actions for any pertinent MODIFY triggers should be changed accordingly.

DEFINE TRIGGER Statement

For those cases in which a triggered action performs an operation based on the changed value for a particular field, the action should include a conditional expression that prevents execution of the operation if no value change occurs. The conditional expression, which compares the old value with the new value, should appear as part of the triggered action's WITH clause.

The frequency clause, FOR EACH RECORD, determines whether an action is evaluated once per triggering statement, or for each record of the subject relation updated by the triggering statement. If the FOR EACH RECORD clause is not specified, the action is evaluated only once.

The subject relation context of the triggering statement has three possible states: current, old, and new. To differentiate between the current, old, and new states, you use context variables. The relation context variable is an alias for the current state of the subject relation. The old and new context variables are aliases for the old and new states of the subject relation. The old context variable is only available (valid) for AFTER MODIFY triggers and the new context variable is available (valid) only for BEFORE MODIFY triggers.

Defining a trigger requires READ and DEFINE access to the subject relation. If any triggered statement specifies some form of update operation, then READ, CONTROL and appropriate update access (ERASE, MODIFY, or WRITE) to the relations specified by the triggered action statements are also required.

Attempts to define a trigger will fail if that trigger affects relations that are involved in a query at the same time. Users must detach from the database with a FINISH statement before you can define the trigger. When Rdb/VMS first accesses an object, such as the relation, a lock is taken out on that object and not released until the user exits the database. Attempts to update this object will get a LOCK CONFLICT ON CLIENT message due to the other user's optimized access to the object.

Similarly, while you define a trigger, users cannot execute queries involving affected relations until you have completed the transaction with a COMMIT or ROLLBACK statement for the DEFINE statement. The user will receive a LOCK CONFLICT ON CLIENT error message. While DDL operations are performed, normal data locking mechanisms are used against system relations. (System relations hold information on objects in the database.) Therefore, attempts to update an object will lock out attempts to query that object. These locks are held until the COMMIT or ROLLBACK of the DDL operation.

DEFINE TRIGGER Statement

The WAIT/NOWAIT clause of the START_TRANSACTION statement does not affect attempts to update metadata with simultaneous queries. Even if you specify START_TRANSACTION WAIT for the metadata update transaction, you will get the following error message if a lock conflict exists:

```
%RDB-E-LOCK_CONFLICT, request failed due to locked resource; no-wait
parameter specified for transaction
-RDB-E-NO_META_UPDATE, metadata update failed
-RDMS-F-LCKCNFLCT, lock conflict on client
RDO>
```

However, a user's query will wait for a metadata update to complete with a ROLLBACK or COMMIT statement, even if the user specified NOWAIT on the START_TRANSACTION statement.

You cannot execute the DEFINE TRIGGER statement when the RDB\$SYSTEM storage area is set to read-only. You must first set RDB\$SYSTEM to read/write. See the *VAX Rdb/VMS Guide to Database Maintenance and Performance* for more information on the RDB\$SYSTEM storage area.

Only one trigger can specify one combination of action time and type of update statement defined for any relation, with the exception that multiple BEFORE MODIFY or AFTER MODIFY triggers can be defined as long as they all have exclusive, unique field lists. The trigger being defined checks for conflicts with the specified trigger for either update time and type, or in one of the field names on the list of fields to be modified. A triggered statement cannot affect the relation on which the trigger is defined such that the trigger would be recursively invoked.

Table 9-8 lists the six possible types of update action. The values from the record that is being affected by the triggering statement are available to the triggered actions with the FOR EACH RECORD frequency clause as shown in Table 9-8.

DEFINE TRIGGER Statement

Table 9–8 Record Data Updates

Action Time/Type of Update	Availability of Record Data
BEFORE STORE	Record data is not available.
AFTER STORE	Record data referred to by the relation context variable is available.
BEFORE ERASE	Record data referred to by the relation context variable is available.
AFTER ERASE	Record data is not available.
BEFORE MODIFY	Old values of record data referred to by the relation context variable are available. New values of record data referred to by the new context variable are available.
AFTER MODIFY	New values of record data referred to by the relation context variable are available. Old values of record data referred to by the old context variable are available.

Note that if the FOR EACH RECORD clause is not specified, the triggered action is evaluated only once, and record values are not available to the triggered action.

The following examples illustrate the rules in Table 9–8 about the availability of record data being stored, erased, or modified:

- A BEFORE STORE trigger action for the EMPLOYEES relation cannot create a record in the JOB_HISTORY relation for the EMPLOYEE_ID to be stored, because the information in the record to be stored is not yet available. However, an AFTER STORE trigger action can use the EMPLOYEE_ID of the record being stored to create a record in the JOB_HISTORY relation.
- A BEFORE ERASE trigger action for the EMPLOYEES relation can delete records in the JOB_HISTORY relation with the EMPLOYEE_ID of the record to be erased. However, an AFTER ERASE trigger action cannot delete any JOB_HISTORY records with that EMPLOYEE_ID, because the information from the erased record is no longer available.

DEFINE TRIGGER Statement

- The following syntax illustrates the use of old and new values of record data in a BEFORE MODIFY trigger action:

```
RDO> DEFINE TRIGGER COMMISSION_CODE_CASCADE_UPDATE
cont> BEFORE MODIFY OF COMMISSION_CODE NEW CONTEXT NEW_CC
cont> FOR OLD_CC IN SALES EXECUTE
cont>   FOR T IN TOTAL_SALES
cont>     WITH T.COMMISSION_CODE = OLD_CC.COMMISSION_CODE
cont>       MODIFY T
cont>         USING T.COMMISSION_CODE = NEW_CC.COMMISSION_CODE
cont>       END_MODIFY
cont>     END_FOR
cont> FOR EACH RECORD.
```

If the COMMISSION_CODE field in the SALES relation changes, the trigger in this example causes a change in all affected TOTAL_SALES records also.

- The following syntax illustrates the use of old and new values of record data in an AFTER MODIFY trigger action:

```
RDO>DEFINE TRIGGER UPDATE_SALES_TOTAL_ON_CHANGE1
cont> AFTER MODIFY OF SALES_AMOUNT OLD CONTEXT OS
cont>   FOR S IN SALES
cont>     EXECUTE
cont>       FOR T IN TOTAL_SALES
cont>         MODIFY T USING
cont>           T.SALES_AMOUNT =
cont>             T.SALES_AMOUNT - OS.SALES_AMOUNT + S.SALES_AMOUNT
cont>         END_MODIFY
cont>       END_FOR
cont> FOR EACH RECORD.
```

The trigger in this example updates the sales total with the difference between the new and old values for an already existing sales record.

Once a trigger has been selected for evaluation, Rdb/VMS evaluates each pertinent triggered action in succession. The execution of a triggered action statement may cause other triggers to be selected for invocation; however, if a trigger is selected recursively by direct or indirect execution of one of its actions, an exception is produced. Once all triggered actions have been exhausted, another pertinent trigger may be selected for evaluation (BEFORE MODIFY and AFTER MODIFY triggers only).

You must execute the DEFINE TRIGGER statement in a read/write transaction. If you issue this statement when there is no transaction is active, Rdb/VMS starts a read/write transaction implicitly.

DEFINE TRIGGER Statement

Examples

Example 1

The following example defines a trigger that performs a cascading delete triggered by the deletion of an employee record.

Each associated employee record (from the relations that have foreign keys referring to the primary key in the EMPLOYEES relation) is deleted.

```
RDO> DEFINE TRIGGER EMPLOYEE_ID_CASCADE_DELETE
cont>     BEFORE ERASE
cont>     FOR E IN EMPLOYEES
cont>     EXECUTE
cont>         FOR D IN DEGREES WITH
cont>             D.EMPLOYEE_ID = E.EMPLOYEE_ID
cont>             ERASE D
cont>         END_FOR;
cont>         FOR JH IN JOB_HISTORY WITH
cont>             JH.EMPLOYEE_ID = E.EMPLOYEE_ID
cont>             ERASE JH
cont>         END_FOR;
cont>         FOR R IN RESUMES WITH
cont>             R.EMPLOYEE_ID = E.EMPLOYEE_ID
cont>             ERASE R
cont>         END_FOR;
cont>         FOR SH IN SALARY_HISTORY WITH
cont>             SH.EMPLOYEE_ID = E.EMPLOYEE_ID
cont>             ERASE SH
cont>         END_FOR
cont>     FOR EACH RECORD.
cont>!     Also, if an employee is terminated and that employee is
cont>!     the manager of a department, set the manager_id null for
cont>!     that department.
cont>     FOR D IN DEPARTMENTS WITH D.MANAGER_ID = E.EMPLOYEE_ID
cont>     MODIFY D
cont>         USING D.MANAGER_ID = RDB$MISSING (D.MANAGER_ID)
cont>     END_MODIFY
cont>     END_FOR
cont> FOR EACH RECORD.
```

Example 2

The following example defines a trigger that performs a cascading update triggered by the modification of an employee record.

If the status code in the WORK_STATUS relation changes, the trigger also causes a change in all affected EMPLOYEES records also.

DEFINE TRIGGER Statement

If you do not specify “OF STATUS CODE” (the field name), this definition causes the trigger to be executed whenever *any* field in the WORK_STATUS relation is changed. In this example, the STATUS_CODE of the EMPLOYEES relation is changed to the STATUS_CODE of the WORK_STATUS relation only when the STATUS_CODE field of the WORK_STATUS relation is modified.

```
RDO> DEFINE TRIGGER STATUS_CODE_CASCADE_UPDATE
cont>   BEFORE MODIFY OF STATUS_CODE NEW CONTEXT NEW_WS
cont>   FOR OLD_WS IN WORK_STATUS
cont>   EXECUTE
cont>     FOR E IN EMPLOYEES
cont>     WITH E.STATUS_CODE = OLD_WS.STATUS_CODE
cont>     MODIFY E
cont>     USING E.STATUS_CODE = NEW_WS.STATUS_CODE
cont>     END_MODIFY
cont>   END_FOR
cont>   FOR EACH RECORD.
```

Example 3

The following example shows a trigger that updates the ARCHIVE relation as records are deleted from the CANDIDATES relation.

```
RDO> !First define the ARCHIVE relation and its fields.
RDO> DEFINE RELATION ARCHIVE.
cont>   LAST_NAME DATATYPE TEXT SIZE IS 14.
cont>   FIRST_NAME DATATYPE TEXT SIZE IS 10.
cont>   MIDDLE_INITIAL DATATYPE TEXT SIZE IS 1.
cont>   CANDIDATE_STATUS DATATYPE VARYING STRING SIZE IS 255.
cont> END ARCHIVE RELATION.
RDO> !Then define the trigger to create an audit trail of deletions
RDO> ! from the CANDIDATES relation in the ARCHIVE relation.
RDO> DEFINE TRIGGER ARCHIVE_CANDIDATE
cont> BEFORE ERASE
cont>   FOR C IN CANDIDATES
cont>   EXECUTE
cont>     STORE A IN ARCHIVE
cont>     USING A.LAST_NAME = C.LAST_NAME;
cont>           A.FIRST_NAME = C.FIRST_NAME;
cont>           A.MIDDLE_INITIAL = C.MIDDLE_INITIAL;
cont>           A.CANDIDATE_STATUS = C.CANDIDATE_STATUS
cont>     END_STORE
cont>   FOR EACH RECORD.
RDO> ! Now print the data in the CANDIDATES relation.
RDO> FOR C IN CANDIDATES
cont> PRINT C.*
cont> END_FOR
LAST_NAME          FIRST_NAME    MIDDLE_INITIAL
CANDIDATE_STATUS
Wilson             Oscar           M
Available part time Oct.-Dec. and full time starting in January

Schwartz           Trixie         R
Available second week November. Don't contact her at current job.
```

DEFINE TRIGGER Statement

```
Boswick          Fred          W
Engineering Dept. not impressed. No offer made.
RDO> ! Erase a record in the CANDIDATES relation.
RDO> FOR C IN CANDIDATES
cont> WITH C.LAST_NAME = "Wilson"
cont> ERASE C
cont> END_FOR
RDO> !Verify that the erasure has been made from the
RDO> !CANDIDATES relation and that the ERASE statement
RDO> ! has triggered an update in the ARCHIVE relation.
RDO> FOR A IN ARCHIVE
cont> PRINT A.*
cont> END_FOR
LAST_NAME      FIRST_NAME  MIDDLE_INITIAL
CANDIDATE_STATUS
Wilson         Oscar        M
Available part time Oct.-Dec. and full time starting in January
RDO> FOR C IN CANDIDATES
cont> PRINT C.*
cont> END_FOR
LAST_NAME      FIRST_NAME  MIDDLE_INITIAL
CANDIDATE_STATUS
Schwartz       Trixie       R
Available second week November. Don't contact her at current job.
Boswick          Fred          W
Engineering Dept. not impressed. No offer made.
RDO>
```

Example 4

The following RDO command procedure example defines a MODIFY trigger with two actions. The first action is defined to account for the situation where the triggering MODIFY statement has not actually changed the value for the pertinent field (EMPLOYEE_ID). This trigger causes a cascading update of the EMPLOYEES relation's EMPLOYEE_ID value to the JOB_HISTORY table. The WITH clause stipulates that the cascading update will occur only when the EMPLOYEE_ID value actually changes. The example also logs each MODIFY operation to the LOG relation.

DEFINE TRIGGER Statement

```
!
! Invoke the database:
INVOKE DATABASE FILENAME 'PERSONNEL'
!
! Define the global fields for the LOG relation:
START_TRANSACTION READ_WRITE
!
DEFINE FIELD TYPE
DATATYPE IS TEXT 10.
%RDO-W-NOCDDUPDAT, database invoked by filename, the CDD will not be updated
!
DEFINE FIELD REL_NAME
DATATYPE IS TEXT 31.
!
! Define the LOG relation:
DEFINE RELATION LOG.
TYPE.
REL_NAME.
END LOG RELATION.
!
! Define trigger TRIG1:
DEFINE TRIGGER TRIG1
  AFTER MODIFY OF EMPLOYEE_ID OLD CONTEXT OLD_EMP
  FOR NEW_EMP IN EMPLOYEES
  WITH NEW_EMP.EMPLOYEE_ID <> OLD_EMP.EMPLOYEE_ID EXECUTE
    FOR JH IN JOB_HISTORY
    WITH JH.EMPLOYEE_ID = OLD_EMP.EMPLOYEE_ID
    MODIFY JH USING JH.EMPLOYEE_ID = NEW_EMP.EMPLOYEE_ID
    END_MODIFY
  END_FOR
FOR EACH RECORD
  EXECUTE
    STORE L IN LOG USING
      L.TYPE = "Modify";
      L.REL_NAME = "EMPLOYEES"
    END_STORE
FOR EACH RECORD.
!
! Test the trigger by changing the EMPLOYEE_ID of "00164" to "98765":
FOR E IN EMPLOYEES WITH E.EMPLOYEE_ID = "00164"
  MODIFY E USING E.EMPLOYEE_ID = "98765"
  END_MODIFY
END_FOR
!
! The trigger causes a record to be stored in the LOG relation:
FOR L IN LOG
PRINT L.TYPE,
      L.REL_NAME
END_FOR
TYPE          REL_NAME
Modify        EMPLOYEES
!
```

DEFINE TRIGGER Statement

Example 5

The following example does not use the sample personnel database supplied with Rdb/VMS. In the example, the EMPLOYEE_ID field is defined as a longword; in the sample personnel database, the EMPLOYEE_ID field is defined as text. This example defines a trigger that assigns the next available employee identification number to the EMPLOYEE_ID field of the record that has just been stored.

```
RDO> DEFINE TRIGGER SET_EMPLOYEE_ID
      AFTER STORE
      FOR E IN EMPLOYEES EXECUTE
        FOR E1 IN EMPLOYEES WITH E1.RDB$DB_KEY = E.RDB$DB_KEY
          MODIFY E1 USING
            E1.EMPLOYEE_ID = ((MAX E2.EMPLOYEE_ID OF E2 IN EMPLOYEES)+1)
          END_MODIFY
        END_FOR
      FOR EACH RECORD.
```

```
RDO> !Storing these records assigns the next available number to the
RDO> !EMPLOYEE_ID field no matter what value for the field is given
RDO> !in the store statement. (Since the EMPLOYEE_ID field is required
RDO> !in this example, you must store a dummy value, however, or you
RDO> !receive a constraint violation.)
```

```
473 Ringer, Joe
                                     ?
17-NOV-1858 00:00:00.00 N
474 Ringer, Moe
                                     ?
17-NOV-1858 00:00:00.00 N
```

Example 6:

The following example shows two triggers that perform a summation update when a particular field of a sales record is updated. The first trigger adds a new sales amount to the sales total; the second trigger updates the sales total with the difference between the new and old values for an already existing sales record. This example does not use the sample personnel database supplied with Rdb/VMS.

DEFINE TRIGGER Statement

```
RDO> DEFINE TRIGGER UPDATE_SALES_TOTAL_ON_NEW_SALE
cont>     AFTER STORE
cont>     FOR S IN SALES
cont>     EXECUTE
cont>         FOR T IN TOTAL_SALES
cont>             MODIFY T USING
cont>                 T.SALES_AMOUNT = T.SALES_AMOUNT + S.SALES_AMOUNT
cont>             END_MODIFY
cont>         END_FOR
cont>     FOR EACH RECORD.
RDO>
RDO> DEFINE TRIGGER UPDATE_SALES_TOTAL_ON_CHANGE
cont>     AFTER MODIFY OF SALES_AMOUNT OLD CONTEXT OS
cont>     FOR S IN SALES
cont>     EXECUTE
cont>         FOR T IN TOTAL_SALES
cont>             MODIFY T USING
cont>                 T.SALES_AMOUNT = T.SALES_AMOUNT -
cont>                 OS.SALES_AMOUNT + S.SALES_AMOUNT
cont>             END_MODIFY
cont>         END_FOR
cont>     FOR EACH RECORD.
RDO>
```

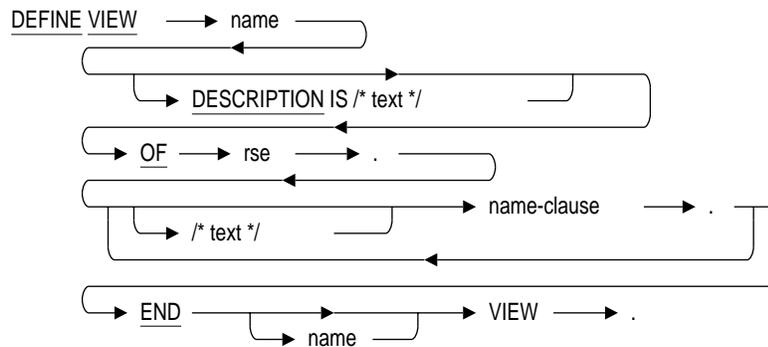
9.23 DEFINE VIEW Statement

Creates a view definition. A **view** is a relation whose data is not physically stored. Rather, it is a virtual structure that refers to records stored in other relations. You can include in a view definition combinations of records and fields from other relations and other views in the database. You define a view by specifying:

- A record selection expression to name the criteria for selecting the relations, records, and fields for the view
- A set of fields from those relations

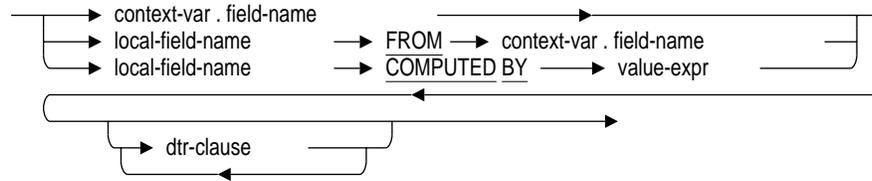
When the DEFINE VIEW statement executes, Rdb/VMS adds the view definition to the physical database. If you have invoked the database with the PATHNAME qualifier, the definition is also stored in the data dictionary.

Format

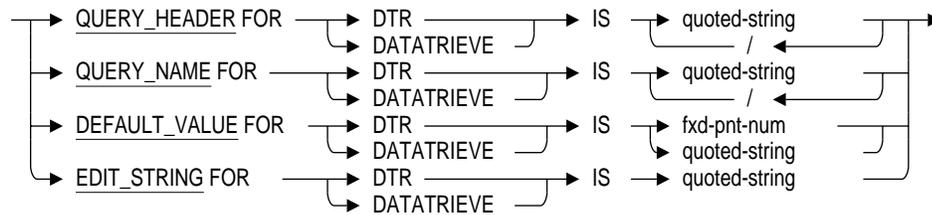


DEFINE VIEW Statement

name-clause =



dtr-clause =



Arguments

name

Name of the view definition you want to create. When you choose a name, follow these rules:

- Use a name that is unique among all view and relation names in the database.
- Use any valid VMS name. However, the name cannot end in a dollar sign (\$) or underscore (_).
- Do not use any Rdb/VMS reserved words (see Appendix A).

rse

A record selection expression that defines which rows of which relations Rdb/VMS includes in the view. See Chapter 4 for a complete description of record selection expressions.

DEFINE VIEW Statement

name-clause

A field that you want to include in the view. There are three ways to include a field in a view definition:

- Name the field from one of the component relations, using the context variable declared in the RSE.
- Give the field a local name and use the FROM clause to associate that name with a field from one of the component relations.
- Give the field a local name and use the COMPUTED BY clause to calculate the field value.

value-expr

A valid Rdb/VMS value expression used to calculate the value of the view's field.

Usage Notes

You need the Rdb/VMS READ and DEFINE privileges to the referenced relations to use the DEFINE VIEW statement.

If the database is created with the DICTONARY IS REQUIRED option, you must invoke the database by path name, rather than file name, before you issue this statement.

You must execute this statement in a read/write transaction. If you issue this statement when there is no active transaction, Rdb/VMS starts a read/write transaction implicitly.

A multirelation view cannot refer to more than 32 relations.

Other users are allowed to be attached to the database when you issue the DEFINE VIEW statement.

You cannot execute the DEFINE VIEW statement when the RDB\$SYSTEM storage area is set to read-only. You must first set RDB\$SYSTEM to read/write. See the *VAX Rdb/VMS Guide to Database Maintenance and Performance* for more information on the RDB\$SYSTEM storage area.

DEFINE VIEW Statement

Examples

Example 1

You can define a view of a single relation.

```
DEFINE VIEW EMP_NAME
  DESCRIPTION IS /* A view for an employee's full name only */
  OF E IN EMPLOYEES.
  E.FIRST_NAME.
  MIDDLE_INITIAL.
  E.LAST_NAME.
END EMP_NAME VIEW.
```

This command file specifies a view definition derived from a single relation, using three fields.

Example 2

You can also define a view using more than one relation.

```
DEFINE VIEW CURRENT_SALARY
  OF SH IN SALARY_HISTORY CROSS
  E IN EMPLOYEES OVER EMPLOYEE_ID WITH
  SH.SALARY_END MISSING.
  E.LAST_NAME.
  E.FIRST_NAME.
  E.EMPLOYEE_ID.
  SH.SALARY_START.
  SH.SALARY_AMOUNT.
END VIEW.
```

This command file defines a view from the EMPLOYEES and SALARY_HISTORY relations. It uses the RSE to join the relations and limit the view to current salaries. Then it selects the fields derived from each relation. These field definitions are used as is, with the same names.

Example 3

You can give local field names to a view.

```
DEFINE VIEW EMP_JOB OF E IN EMPLOYEES
  CROSS JH IN JOB_HISTORY OVER EMPLOYEE_ID
  CROSS J IN JOBS OVER JOB_CODE
  WITH JH.JOB_END MISSING.
  CURRENT_ID FROM E.EMPLOYEE_ID.
  CURRENT_NAME FROM E.LAST_NAME.
  CURRENT_JOB FROM J.JOB_TITLE.
  SUPERVISOR FROM JH.SUPERVISOR_ID.
END EMP_JOB VIEW.
```

DEFINE VIEW Statement

The definition in this command file does the following:

- Joins the EMPLOYEES and JOB_HISTORY relations. This join links employees to job history records.
- Joins the JOB_HISTORY and JOBS relations. This join lets the view contain job titles, instead of job codes.
- Uses the MISSING value expression. This clause specifies that only the current job history records, where the JOB_END field is missing, should be included in the view.
- Derives the view field definitions from the source relations but gives them local names.

The following query uses the view defined in the preceding example:

```
&RDB&  START_TRANSACTION READ_ONLY
&RDB&  FOR CE IN EMP_JOB
      GET
        ID = CE.CURRENT_ID;
        NAME = CE.CURRENT_NAME;
        JOB = CE.CURRENT_JOB;
        SUPER = CE.SUPERVISOR;
      END_GET
&RDB&  END_FOR
&RDB&  COMMIT
```

Example 4

The COMPUTED BY field calculates the field in the view using a field or fields from a component relation:

```
DEFINE VIEW SS_DEDUCTION OF E IN EMPLOYEES
  CROSS SH IN SALARY_HISTORY OVER EMPLOYEE_ID
  WITH SH.SALARY_END MISSING.
  E.EMPLOYEE_ID.
  SH.SALARY_AMOUNT.
  SS_AMOUNT COMPUTED BY (SH.SALARY_AMOUNT * 0.065).
END SS_DEDUCTION VIEW.
```

This view definition computes a new virtual field from the SALARY_AMOUNT field of SALARY_HISTORY each time it executes. You cannot define a

DEFINE VIEW Statement

COMPUTED BY field that refers to another local field in the view. The following is incorrect:

```
DEFINE VIEW WEEKLY_SALS OF SH IN SALARY_HISTORY.  
  LOC_SALARY FROM SH.SALARY_AMOUNT.  
  WEEKLY_SALARY COMPUTED BY (LOC_SALARY / 52).  
END WEEKLY_SALS VIEW.
```

Instead, refer to the actual field itself:

```
DEFINE VIEW WEEKLY_SALS OF SH IN SALARY_HISTORY.  
  LOC_SALARY FROM SH.SALARY_AMOUNT.  
  WEEKLY_SALARY COMPUTED BY (SH.SALARY_AMOUNT / 52).  
END WEEKLY_SALS VIEW.
```

DELETE COLLATING_SEQUENCE Statement

9.24 DELETE COLLATING_SEQUENCE Statement

Deletes the named collating sequence.

You cannot delete a collating sequence if the database or any global field in the database uses that collating sequence.

Format

```
DELETE COLLATING_SEQUENCE → sequence-name.
```

Arguments

sequence-name

Specifies the sequence-name argument you used when creating the collating sequence in the DEFINE COLLATING_SEQUENCE statement.

Usage Notes

You must have the DELETE privilege to the database to use the DELETE COLLATING_SEQUENCE statement.

You must execute this statement in a read/write transaction. If you issue this statement when there is no active transaction, Rdb/VMS starts a read/write transaction implicitly.

Other users can be attached to the database when you issue the DELETE COLLATING_SEQUENCE statement.

You cannot execute the DELETE COLLATING_SEQUENCE statement when the RDB\$SYSTEM storage area is set to read-only. You must first set RDB\$SYSTEM to read/write. See the *VAX Rdb/VMS Guide to Database Maintenance and Performance* for more information on the RDB\$SYSTEM storage area.

DELETE COLLATING_SEQUENCE Statement

Examples

Example 1

The following example creates a collating sequence using the predefined collating sequence French. It then shows the defined collating sequence by using the SHOW COLLATING_SEQUENCE statement.

The example then deletes the collating sequence using the DELETE COLLATING_SEQUENCE statement. The SHOW COLLATING_SEQUENCE statement then shows that the collating sequence no longer exists.

```
RDO> INVOKE DATABASE FILENAME 'MF_PERSONNEL'
RDO> DEFINE COLLATING_SEQUENCE FRENCH FRENCH.
%RDO-W-NOCDUPDAT, database invoked by filename, the CDD will not be
updated
RDO> !
RDO> SHOW COLLATING_SEQUENCE
User Collating Sequences in Database with filename MF_PERSONNEL
      FRENCH
RDO> !
RDO> DELETE COLLATING_SEQUENCE FRENCH.
RDO> !
RDO> SHOW COLLATING_SEQUENCE
User Collating Sequences in Database with filename MF_PERSONNEL
      No Collating Sequences were found
```

Example 2

The following example shows that you cannot delete a collating sequence if an existing global field or database is defined using the collating sequence.

```
RDO> INVOKE DATABASE FILENAME 'MF_PERSONNEL'
RDO> DEFINE COLLATING_SEQUENCE SPANISH SPANISH.
%RDO-W-NOCDUPDAT, database invoked by filename, the CDD will not be
updated
RDO> DEFINE FIELD LAST_NAME_SPANISH DATATYPE IS TEXT SIZE IS 12 CHARACTERS
cont> COLLATING_SEQUENCE IS SPANISH.
RDO> SHOW FIELD LAST_NAME_SPANISH
      LAST_NAME_SPANISH          text size is 12
                                Collating Sequence SPANISH

RDO> !
RDO> SHOW COLLATING_SEQUENCE
User Collating Sequences in Database with filename MF_PERSONNEL
      SPANISH
```

DELETE COLLATING_SEQUENCE Statement

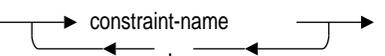
```
RDO> !
RDO> ! You cannot delete the collating sequence because the
RDO> ! field LAST_NAME_SPANISH, defined using SPANISH, still exists:
RDO> !
RDO> DELETE COLLATING_SEQUENCE SPANISH.
%RDB-E-NO_META_UPDATE, metadata update failed
-RDMS-F-COLUSEDFLD, the collating sequence named SPANISH is used in
field LAST_NAME_SPANISH
RDO> !
RDO> ! Delete the field:
RDO> !
RDO> DELETE FIELD LAST_NAME_SPANISH.
RDO> !
RDO> ! Now you can delete the collating sequence:
RDO> !
RDO> DELETE COLLATING_SEQUENCE SPANISH.
RDO> !
RDO> SHOW COLLATING_SEQUENCE
User Collating Sequences in Database with filename MF_PERSONNEL
No Collating Sequences were found
```

DELETE CONSTRAINT Statement

9.25 DELETE CONSTRAINT Statement

Deletes one or more constraint definitions. When the DELETE CONSTRAINT statement executes, Rdb/VMS deletes the constraint definition from the physical database. If you use the PATHNAME qualifier when you invoke the database, the DELETE CONSTRAINT statement also deletes the constraint definition from the data dictionary.

Format

DELETE CONSTRAINT 

Argument

constraint-name

The name of the constraint definition you want to delete.

Usage Notes

To delete a constraint, you must have the Rdb/VMS READ privilege to the database and the Rdb/VMS DELETE privilege to all relations to which the constraint refers.

If the database is created with the DICTONARY IS REQUIRED option, you must invoke the database by path name, rather than file name, before you issue this statement.

You must execute this statement in a read/write transaction. If there is no active transaction and you issue this statement, Rdb/VMS starts a read/write transaction implicitly.

You cannot delete a constraint when there are active transactions accessing the relation or relations involved.

Other users are allowed to be attached to the database when you issue the DELETE CONSTRAINT statement.

DELETE CONSTRAINT Statement

Attempts to delete a constraint will fail if that constraint is involved in a query at the same time. Users will have to detach from the database with a FINISH statement before you can delete the constraint. When Rdb/VMS first accesses an object, such as the constraint, a lock is taken out on that object and not released until the user exits the database. Attempts to update this object will get a LOCK CONFLICT ON CLIENT message due to the other user's optimized access to the object.

Similarly, while you delete a constraint, users cannot execute queries involving that constraint until you have completed the transaction with a COMMIT or ROLLBACK statement for the DELETE statement. The user will receive a LOCK CONFLICT ON CLIENT error message. While DDL operations are performed, normal data locking mechanisms are used against system relations. (System relations hold information on objects in the database.) Therefore, attempts to update an object will lock out attempts to query that object. These locks are held until the commit or rollback of the DDL operation.

The WAIT/NOWAIT clause of the START_TRANSACTION statement does not affect attempts to update metadata with simultaneous queries. Even if you specify START_TRANSACTION WAIT for the metadata update transaction, you will get the following error message if a lock conflict exists:

```
%RDB-E-LOCK_CONFLICT, request failed due to locked resource; no-wait
parameter specified for transaction
-RDB-E-NO_META_UPDATE, metadata update failed
-RDMS-F-LCKCNFLCT, lock conflict on client
RDO>
```

However, a user's query will wait for a metadata update to complete with a ROLLBACK or COMMIT statement, even if the user specified NOWAIT on the START_TRANSACTION statement.

You cannot execute the DELETE CONSTRAINT statement when the RDB\$SYSTEM storage area is set to read-only. You must first set RDB\$SYSTEM to read/write. See the *VAX Rdb/VMS Guide to Database Maintenance and Performance* for more information on the RDB\$SYSTEM storage area.

DELETE CONSTRAINT Statement

Examples

Example 1

The following example deletes a single constraint definition:

```
INVOKE DATABASE PATHNAME 'PERSONNEL'  
DELETE CONSTRAINT EMP_NUM_CON.
```

This statement deletes EMP_NUM_CON from the physical database and the data dictionary.

Example 2

You can also delete more than one constraint definition at once.

```
INVOKE DATABASE PATHNAME 'PERSONNEL'  
DELETE CONSTRAINT SAL_CON,SS_NUM_CON.
```

This statement deletes the two constraint definitions from the physical database and the data dictionary.

Example 3

The following example shows how the deletion of constraints sometimes depends on deleting relations that depend on those constraints. In this case, to delete the foreign key constraint in the COLLEGES relation, it is necessary first to delete the DEGREES relation and the COLLEGE_CODE_CASCADE_UPDATE trigger because the COLLEGES relation has constraints that depend on them.

```
RDO> INVOKE DATABASE PATHNAME "PERSONNEL"  
RDO> DELETE RELATION COLLEGES.  
%RDB-E-NO_META_UPDATE, metadata update failed  
-RDMS-F-CONEXI, relation COLLEGES is referenced in constraint DEGREES_FOREIGN2  
-RDMS-F-RELNOTDEL, relation COLLEGES has not been deleted  
RDO> DELETE CONSTRAINT DEGREES_FOREIGN2.  
%RDB-E-NO_META_UPDATE, metadata update failed  
-RDMS-F-CONDELVIAREL, constraint DEGREES_FOREIGN2 can only be deleted by  
changing or deleting relation DEGREES  
RDO> DELETE RELATION DEGREES.  
RDO> DELETE RELATION COLLEGES.  
%RDB-E-NO_META_UPDATE, metadata update failed  
-RDMS-F-TRGEXI, relation COLLEGES is referenced in trigger  
COLLEGE_CODE_CASCADE_UPDATE  
-RDMS-F-RELNOTDEL, relation COLLEGES has not been deleted  
RDO> DELETE TRIGGER COLLEGE_CODE_CASCADE_UPDATE.  
RDO> DELETE RELATION COLLEGES.  
RDO> COMMIT
```

DELETE CONSTRAINT Statement

This statement deletes the definition for COLLEGES from the database file for PERSONNEL and from the data dictionary.

DELETE DATABASE Statement

FILENAME file-spec

A file specification that names the database files. Place this file specification in quotation marks. You can use a full or partial file specification. In the second case, Rdb/VMS uses the standard VMS defaults.

Usage Notes

You must have the ADMINISTRATOR privilege to the database to use the DELETE DATABASE statement.

You cannot delete a database when there are active users for that database, including yourself. That is, issue the DELETE DATABASE statement before you invoke the database or after you issue the FINISH statement.

The following sequence of statements requires a pause between the FINISH statement and the DELETE statement:

```
RDO> DEFINE DATABASE TESTDB.  
RDO> FINISH  
RDO> DELETE DATABASE PATHNAME TESTDB.
```

The pause is required because the DELETE statement is issued before the Rdb/VMS monitor has detached from the database. In a DCL command file, you can use the DCL WAIT statement to ensure a pause.

Examples

Example 1

The following statement deletes the database root file, all storage area files, and all accompanying snapshot files:

```
RDO> DELETE DATABASE FILENAME 'DISK2:[ACCOUNTING]MF_PERSONNEL'.
```

Example 2

You can also delete a database, its snapshot file, and its data dictionary definitions.

```
RDO> DELETE DATABASE PATHNAME 'DISK1:[DICTIONARY]CORP.MIS.PERSONNEL'.
```

This statement deletes:

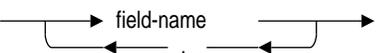
- The RDB file and the SNP file referred to in the data dictionary definition for the database (specified in the DEFINE DATABASE statement).
- The data dictionary entity DISK1:[DICTIONARY]CORP.MIS.PERSONNEL.

DELETE FIELD Statement

9.27 DELETE FIELD Statement

Deletes one or more field definitions. When the DELETE FIELD statement executes, Rdb/VMS deletes the field definition from the physical database. You can delete any named field that was defined globally, whether it was defined with a DEFINE FIELD statement or with a DEFINE RELATION statement. If you invoke the database using the PATHNAME qualifier, Rdb/VMS also deletes the field definition from the data dictionary.

Format

DELETE FIELD  .

Argument

field-name

The name of the field definition you want to delete.

Usage Notes

You must have the Rdb/VMS DELETE privilege for a field to delete the field with the DELETE FIELD statement.

If you want to delete a field that is part of a relation definition, you must use the CHANGE RELATION statement to delete the field from the relation definition, or you must delete the entire relation.

You cannot issue a DELETE FIELD statement for a field that is used in a constraint, index, or view definition. Rdb/VMS returns an error message and does not delete the field. To delete a field used in a constraint, index, or view, first delete the constraint, index, or view definition, then delete the field, and finally redefine the constraint, index, or view.

If the database is created with the **DICTIONARY IS REQUIRED** option, you must invoke the database by path name, rather than file name, before you issue this statement.

DELETE FIELD Statement

You must execute this statement in a read/write transaction. If you issue this statement when there is no active transaction, Rdb/VMS starts a read/write transaction implicitly.

Other users are allowed to be attached to the database when you issue the DELETE FIELD statement. However, attempts to delete a field will fail if that field is involved in a query at the same time. Users will have to detach from the database with a FINISH statement before you can delete the field. When Rdb/VMS first accesses an object, such as the field, a lock is taken out on that object and not released until the user exits the database. Attempts to update this object will get a LOCK CONFLICT ON CLIENT message due to the other user's optimized access to the object.

Similarly, while you delete a field, users cannot execute queries involving that table until you have completed the transaction with a COMMIT or ROLLBACK statement for the DELETE statement. The user will receive a LOCK CONFLICT ON CLIENT error message. While DDL operations are performed, normal data locking mechanisms are used against system relations. (System relations hold information on objects in the database.) Therefore, attempts to update an object will lock out attempts to query that object. These locks are held until the commit or rollback of the DDL operation.

The WAIT/NOWAIT clause of the START_TRANSACTION statement does not affect attempts to update metadata with simultaneous queries. Even if you specify START_TRANSACTION WAIT for the metadata update transaction, you will get the following error message if a lock conflict exists:

```
%RDB-E-LOCK_CONFLICT, request failed due to locked resource; no-wait
parameter specified for transaction
-RDB-E-NO_META_UPDATE, metadata update failed
-RDMS-F-LCKCNFLCT, lock conflict on client
RDO>
```

However, a user's query will wait for a metadata update to complete with a ROLLBACK or COMMIT statement, even if the user specified NOWAIT on the START_TRANSACTION statement.

You cannot execute the DELETE FIELD statement when the RDB\$SYSTEM storage area is set to read-only. You must first set RDB\$SYSTEM to read/write. See the *VAX Rdb/VMS Guide to Database Maintenance and Performance* for more information on the RDB\$SYSTEM storage area.

DELETE FIELD Statement

Examples

Example 1

The following statement deletes a single field definition:

```
RDO> INVOKE DATABASE PATHNAME 'PERSONNEL'  
RDO> START_TRANSACTION READ_WRITE  
RDO> DELETE FIELD TEMP_NUM.  
RDO> COMMIT
```

This sequence deletes the field definition from the database and the data dictionary.

Example 2

You can delete more than one field definition in a single statement.

```
RDO> INVOKE DATABASE PATHNAME 'PERSONNEL'  
RDO> START_TRANSACTION READ_WRITE  
RDO> DELETE FIELD TEMP_NUM, MONEY.  
RDO> COMMIT
```

This sequence deletes the definitions for the TEMP_NUM and MONEY fields from the physical database and the data dictionary.

Example 3

If a field is used in a relation, you need two operations to delete it:

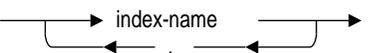
- Change the relation by deleting the field.
- Delete the global field definition.

```
RDO> INVOKE DATABASE FILENAME 'PERSONNEL'  
RDO> START_TRANSACTION READ_WRITE  
RDO> DELETE FIELD STATUS_NAME.  
%RDO-W-NOCDUPDAT, database invoked by filename, the CDD will not be updated  
%RDB-E-NO_META_UPDATE, metadata update failed  
%RDMS-F-RELEXI, field STATUS_NAME is used in relation WORK_STATUS  
-RDMS-F-FLDNOTDEL, field STATUS_NAME has not been deleted  
RDO> CHANGE RELATION WORK_STATUS.  
cont> DELETE STATUS_NAME.  
cont> END.  
RDO> DELETE FIELD STATUS_NAME.  
RDO> COMMIT
```

9.28 DELETE INDEX Statement

Deletes one or more index definitions. When the DELETE INDEX statement executes, Rdb/VMS deletes the index definition from the physical database. If you invoke the database using the PATHNAME qualifier, Rdb/VMS also deletes the index definition from the data dictionary.

Format

```
DELETE INDEX  .
```

Argument

index-name

The name of the index definition you want to delete.

Usage Notes

To delete an index for a relation, you need the Rdb/VMS DELETE privilege to the relation.

If the database is created with the DICTONARY IS REQUIRED option, you must invoke the database by path name, rather than file name, before you issue this statement.

You cannot delete an index definition when there are active transactions accessing the relations involved. Attempts to delete an index will fail if that index is involved in a query at the same time. Users will have to detach from the database with a FINISH statement before you can delete the index. When Rdb/VMS first accesses an object, such as the index, a lock is taken out on that object and not released until the user exits the database. Attempts to update this object will get a LOCK CONFLICT ON CLIENT message due to the other user's optimized access to the object.

Similarly, while you delete an index, users cannot execute queries involving that index until you have completed the transaction with a COMMIT or ROLLBACK statement for the DELETE statement. The user will receive a LOCK CONFLICT ON CLIENT error message. While DDL operations are

DELETE INDEX Statement

performed, normal data locking mechanisms are used against system relations. (System relations hold information on objects in the database.) Therefore, attempts to update an object will lock out attempts to query that object. These locks are held until the COMMIT or ROLLBACK of the DDL operation.

The WAIT/NOWAIT clause of the START_TRANSACTION statement does not affect attempts to update metadata with simultaneous queries. Even if you specify START_TRANSACTION WAIT for the metadata update transaction, you will get the following error message if a lock conflict exists:

```
%RDB-E-LOCK_CONFLICT, request failed due to locked resource; no-wait
parameter specified for transaction
-RDB-E-NO_META_UPDATE, metadata update failed
-RDMS-F-LCKCNFLCT, lock conflict on client
RDO>
```

However, a user's query will wait for a metadata update to complete with a ROLLBACK or COMMIT statement, even if the user specified NOWAIT on the START_TRANSACTION statement.

You must execute this statement in a read/write transaction. If you issue this statement when there is no active transaction, Rdb/VMS starts a read/write transaction implicitly.

You cannot delete an index if a storage map specifies this index in a PLACEMENT VIA INDEX clause.

You cannot execute the DELETE INDEX statement when the RDB\$SYSTEM storage area is set to read-only. You must first set RDB\$SYSTEM to read/write. See the *VAX Rdb/VMS Guide to Database Maintenance and Performance* for more information on the RDB\$SYSTEM storage area.

Examples

Example 1

The following example deletes a single index:

```
RDO> INVOKE DATABASE PATHNAME "PERSONNEL"
RDO> DELETE INDEX DEG_COLLEGE_CODE.
RDO> COMMIT
```

This statement deletes the index from the physical database and the definition from the data dictionary.

DELETE INDEX Statement

Example 2

You can delete multiple index definitions in one statement.

```
RDO> INVOKE DATABASE PATHNAME "PERSONNEL"  
RDO> DELETE INDEX EMP_LAST_NAME, SH_EMPLOYEE_ID.  
RDO> COMMIT
```

This statement deletes the indexes from the physical database and their definitions from the data dictionary.

DELETE PATHNAME Statement

9.29 DELETE PATHNAME Statement

The DELETE PATHNAME statement deletes the data dictionary entity that contains the database definitions. It does not delete the database, snapshot, or storage area files.

Format

`DELETE PATHNAME` → `path-name` → `.`

Argument

path-name

The data dictionary path name for the dictionary entity where the database definitions are stored. Specify either a full data dictionary path name or a relative data dictionary path name and enclose the path name in quotation marks. If you use a relative path name, the current default data dictionary directory must be defined to include all the path name segments that precede the relative path name.

Usage Notes

To use the DELETE PATHNAME statement, you must have the Rdb/VMS DELETE privilege for the CDD/Plus dictionary database in which the data dictionary entity specified by the path name is stored. You must also have the Rdb/VMS READ privilege for the Rdb/VMS database that you have invoked by path name.

Examples

Example 1

The data dictionary entity DISK1:[DICTIONARY]CORP.MIS.PERSONNEL is deleted in the following example. The DELETE PATHNAME statement does not delete the database or its snapshot files.

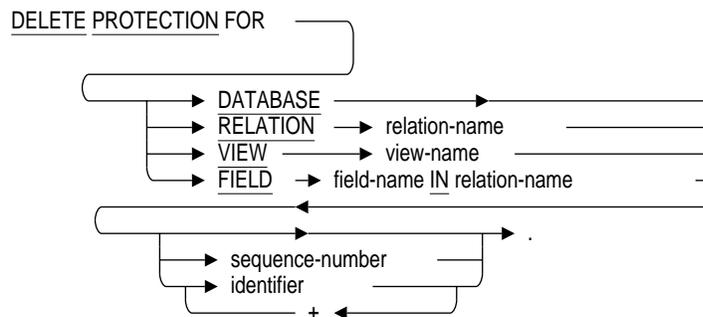
```
RDO> DELETE PATHNAME 'DISK1:[DICTIONARY]CORP.MIS.PERSONNEL'.
```

DELETE PROTECTION Statement

9.30 DELETE PROTECTION Statement

Deletes an entry from the access control list (ACL) for a database or relation. An ACL entry specifies the operations a user may perform on the associated database element. Rdb/VMS access control lists are stored in the database file.

Format



Default

sequence-number

If you do not specify an identifier, Rdb/VMS deletes the first entry in the specified ACL.

Arguments

relation-name

Name of the relation for which you want to change the ACL.

field-name IN relation-name

The name of the local field in a specified relation for which you want to delete an ACL entry.

sequence-number

A number that identifies the entry within the specified ACL whose protection you want to change.

DELETE PROTECTION Statement

Rdb/VMS returns an error message if it does not find an entry for the position you specify in the DELETE PROTECTION statement. For example, if you specify position 6 and only four access control entries are in the access control list, Rdb/VMS returns an error message. Rdb/VMS does not delete the last entry, the fourth entry, in the list.

You cannot specify both an identifier and a sequence number in the same statement.

identifier

Any valid VMS identifier in the identifier clause:

- UIC identifiers

A VMS user identification code (UIC) that identifies an entry within the access control list. A UIC has the form:

[g,m]

where *g* refers to a group number and *m* is the number of a member of that group. You can replace either or both with the wildcard character (*) to specify a particular member in all groups, all members in a particular group, or all members in all groups.

- General identifiers

General identifiers are defined by the VMS system manager in the system rights database to identify groups of users on the system.

- System-defined identifiers

System-defined identifiers are automatically defined by the system when the rights database is created at system installation time.

Identifiers are assigned depending on the type of login you execute.

For more information about these types of identifiers, see Section 9.17.

You cannot specify both an identifier and a sequence number in the same statement.

DELETE PROTECTION Statement

Usage Notes

You must have the Rdb/VMS CONTROL privilege to use the DELETE PROTECTION statement.

You can delete protection even when there are active users. Deleting protection will not affect active users until they exit their session and invoke the database the next time.

You must execute this statement in a read/write transaction. If you issue this statement when there is no active transaction, Rdb/VMS starts a read/write transaction implicitly.

You cannot execute the DELETE PROTECTION statement when the RDB\$SYSTEM storage area is set to read-only. You must first set RDB\$SYSTEM to read/write. See the *VAX Rdb/VMS Guide to Database Maintenance and Performance* for more information on the RDB\$SYSTEM storage area.

Examples

Example 1

The following example deletes an ACL entry identified by a sequence number:

```
RDO> DELETE PROTECTION FOR DATABASE 3.
```

All the entries following the deleted entry are given the next lower sequence number.

Example 2

You can also identify an ACL entry with a UIC.

```
RDO> DELETE PROTECTION FOR RELATION JOB_HISTORY [DATAENTRY,JONES].
```

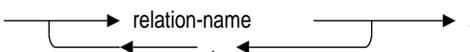
This statement deletes the entry identified by [DATAENTRY,JONES].

DELETE RELATION Statement

9.31 DELETE RELATION Statement

Deletes one or more relation definitions and all associated relation-specific constraints. When the DELETE RELATION statement executes, Rdb/VMS deletes the relation definition, the storage map (if one exists), the constraints associated with the relation definition, and the data stored in that relation from the physical database. If you use the PATHNAME qualifier when you invoke the database, the DELETE RELATION statement also deletes the relation definition from the data dictionary.

Format

DELETE RELATION 

Argument

relation-name

The name of the relation definition you want to delete.

Usage Notes

You must have the Rdb/VMS DELETE privilege to a relation to delete the relation with the DELETE RELATION statement.

If the database is created with the DICTIONARY IS REQUIRED option, you must invoke the database by path name, rather than file name, before you issue this statement.

You cannot delete a relation when there are other active transactions involving the relation. That is, you must have EXCLUSIVE access to the relation. Attempts to delete a relation will fail if that relation is involved in a query at the same time. Users will have to detach from the database with a FINISH statement before you can delete the relation. When Rdb/VMS first accesses an object, such as the relation, a lock is taken out on that object and not released until the user exits the database. Attempts to update this object will get a LOCK CONFLICT ON CLIENT message due to the other user's optimized access to the object.

DELETE RELATION Statement

Similarly, while you delete a relation, users cannot execute queries involving that relation until you have completed the transaction with a COMMIT or ROLLBACK statement for the DELETE statement. The user will receive a LOCK CONFLICT ON CLIENT error message. While DDL operations are performed, normal data locking mechanisms are used against system relations. (System relations hold information on objects in the database.) Therefore, attempts to update an object will lock out attempts to query that object. These locks are held until the COMMIT or ROLLBACK of the DDL operation.

The WAIT/NOWAIT clause of the START_TRANSACTION statement does not affect attempts to update metadata with simultaneous queries. Even if you specify START_TRANSACTION WAIT for the metadata update transaction, you will get the following error message if a lock conflict exists:

```
%RDB-E-LOCK_CONFLICT, request failed due to locked resource; no-wait
parameter specified for transaction
-RDB-E-NO_META_UPDATE, metadata update failed
-RDMS-F-LCKCNFLCT, lock conflict on client
RDO>
```

However, a user's query will wait for a metadata update to complete with a ROLLBACK or COMMIT statement, even if the user specified NOWAIT on the START_TRANSACTION statement.

If a view definition refers to a relation you want to delete, you must delete that view definition before you delete the relation.

If a constraint in the database references a relation, you cannot delete that relation until you delete the constraint that references the relation.

You must execute the DELETE RELATION statement in a read/write transaction. If you issue this statement when there is no active transaction, Rdb/VMS starts a read/write transaction implicitly.

Other users are allowed to be attached to the database when you issue the DELETE RELATION statement.

You cannot execute the DELETE RELATION statement when the RDB\$SYSTEM storage area is set to read-only. You must first set RDB\$SYSTEM to read/write. See the *VAX Rdb/VMS Guide to Database Maintenance and Performance* for more information on the RDB\$SYSTEM storage area.

DELETE RELATION Statement

Examples

Example 1

The following example deletes a single relation definition from the database:

```
RDO> INVOKE DATABASE PATHNAME "PERSONNEL"  
RDO> DELETE RELATION COLLEGES.  
RDO> COMMIT
```

Example 2

The following example deletes a single relation definition and its associated relation-specific constraints from the database. To delete the COLLEGES relation, it is necessary first to delete the DEGREES relation and the COLLEGE_CODE_CASCADE_UPDATE trigger because the COLLEGES relation has constraints that depend on them.

```
RDO> INVOKE DATABASE PATHNAME "PERSONNEL"  
RDO> DELETE RELATION COLLEGES.  
%RDB-E-NO_META_UPDATE, metadata update failed  
-RDMS-F-CONEXI, relation COLLEGES is referenced in constraint DEGREES_FOREIGN2  
-RDMS-F-RELNOTDEL, relation COLLEGES has not been deleted  
RDO> DELETE CONSTRAINT DEGREES_FOREIGN2.  
%RDB-E-NO_META_UPDATE, metadata update failed  
-RDMS-F-CONDELVIAREL, constraint DEGREES_FOREIGN2 can only be deleted by  
changing or deleting relation DEGREES  
RDO> DELETE RELATION DEGREES.  
RDO> DELETE RELATION COLLEGES.  
%RDB-E-NO_META_UPDATE, metadata update failed  
-RDMS-F-TRGEXI, relation COLLEGES is referenced in trigger COLLEGE_CODE_CASCADE_  
UPDATE  
-RDMS-F-RELNOTDEL, relation COLLEGES has not been deleted  
RDO> DELETE TRIGGER COLLEGE_CODE_CASCADE_UPDATE.  
RDO> DELETE RELATION COLLEGES.  
RDO> COMMIT
```

This statement deletes the definition for COLLEGES from the database file for PERSONNEL and from the data dictionary.

Example 3

This example deletes two relation definitions in a single statement:

```
RDO> INVOKE DATABASE PATHNAME "PERSONNEL"  
RDO> DELETE RELATION DEGREES, COLLEGES.  
RDO> COMMIT
```

This statement deletes both the DEGREES and COLLEGES relation from the database file for PERSONNEL and from the data dictionary.

DELETE SCHEDULE Statement (VAX Data Distributor)

9.32 DELETE SCHEDULE Statement (VAX Data Distributor)

Deletes the schedule definition associated with a transfer definition. You can delete a schedule definition only when the associated transfer is in the *suspended* state. See Section 9.67 for information about how to suspend a transfer. If you issue a START TRANSFER statement after deleting the schedule, the transfer will be placed in the unscheduled state.

Note To use this RDO statement, you must have VAX Data Distributor installed on your system. See your system manager or your Digital Equipment Corporation representative if you need information on Data Distributor.

Format

`DELETE SCHEDULE` → `FOR` → `transfer-name` → `.`

Argument

transfer-name

Identifies the transfer whose schedule is to be deleted. The `transfer-name` parameter is required.

Usage Notes

You must execute the DELETE SCHEDULE statement outside of the scope of a transaction. If you issue this statement when a transaction is outstanding, Data Distributor returns an error message.

If a transfer has a schedule definition, DELETE TRANSFER also deletes the corresponding schedule definition. If you want to delete a transfer schedule, the schedule definition must be associated with your UIC.

DELETE SCHEDULE Statement (VAX Data Distributor)

Examples

Example 1

This example places the NH_EMPLOYEES transfer in a suspended state and then deletes the associated schedule.

```
RDO> STOP TRANSFER NH_EMPLOYEES  
RDO> DELETE SCHEDULE FOR NH_EMPLOYEES.
```

DELETE STORAGE MAP Statement

9.33 DELETE STORAGE MAP Statement

Deletes a storage map definition. When the DELETE STORAGE MAP statement executes, Rdb/VMS deletes the storage map definition from the physical database.

Format

`DELETE STORAGE MAP` → `map-name` → .

Argument

map-name

The name of the storage map definition you want to delete.

Usage Notes

To use the DELETE STORAGE MAP statement to delete a storage map for a relation, you must have the Rdb/VMS DELETE privilege to the relation.

If the database is created with the DICTIONARY IS REQUIRED option, you must invoke the database by path name, rather than file name, before you issue this statement.

You must execute this statement in a read/write transaction. If there is no active transaction and you issue this statement, Rdb/VMS starts a read/write transaction implicitly.

You cannot delete a storage map that refers to a relation that contains data. If you attempt to do so, you will receive an error message. However, you can delete the relation once the necessary views and constraints have been deleted, and the underlying storage map will be deleted with the relation and its data.

Other users are allowed to be attached to the database when you issue the DELETE STORAGE MAP statement.

Attempts to delete a storage map will fail if that storage map is involved in a query at the same time. Users will have to detach from the database with a FINISH statement before you can delete the storage map. When Rdb/VMS first accesses an object, such as the storage map, a lock is taken out on that

DELETE STORAGE MAP Statement

object and not released until the user exits the database. Attempts to update this object will get a LOCK CONFLICT ON CLIENT message due to the other user's optimized access to the object.

Similarly, while you delete a storage map, users cannot execute queries involving that storage map until you have completed the transaction with a COMMIT or ROLLBACK statement for the DELETE statement. The user will receive a LOCK CONFLICT ON CLIENT error message. While DDL operations are performed, normal data locking mechanisms are used against system relations. (System relations hold information on objects in the database.) Therefore, attempts to update an object will lock out attempts to query that object. These locks are held until the commit or rollback of the DDL operation.

The WAIT/NOWAIT clause of the START_TRANSACTION statement does not affect attempts to update metadata with simultaneous queries. Even if you specify START_TRANSACTION WAIT for the metadata update transaction, you will get the following error message if a lock conflict exists:

```
%RDB-E-LOCK_CONFLICT, request failed due to locked resource; no-wait
parameter specified for transaction
-RDB-E-NO_META_UPDATE, metadata update failed
-RDMS-F-LCKCNFLCT, lock conflict on client
RDO>
```

However, a user's query will wait for a metadata update to complete with a ROLLBACK or COMMIT statement, even if the user specified NOWAIT on the START_TRANSACTION statement.

You cannot execute the DELETE STORAGE MAP statement when the RDB\$SYSTEM storage area is set to read-only. You must first set RDB\$SYSTEM to read/write. See the *VAX Rdb/VMS Guide to Database Maintenance and Performance* for more information on the RDB\$SYSTEM storage area.

Examples

Example 1

The following example deletes the storage map TEST_MAP:

```
RDO> DELETE STORAGE MAP TEST_MAP.
```

DELETE TRANSFER Statement (VAX Data Distributor)

9.34 DELETE TRANSFER Statement (VAX Data Distributor)

Deletes a transfer definition. If a transfer has a schedule definition, DELETE TRANSFER also deletes the corresponding schedule definition.

Note To use this RDO statement, you must have VAX Data Distributor installed on your system. See your system manager or your Digital Equipment Corporation representative if you need information on Data Distributor.

Format

`DELETE TRANSFER` → `transfer-name` → .

Argument

transfer-name

The transfer definition to be deleted. The transfer-name parameter is required.

Usage Notes

You must have the Rdb/VMS ERASE privilege to the transfer relation to use the DELETE TRANSFER statement.

You must execute the DELETE TRANSFER statement outside of the scope of a transaction. If you issue this statement when a transaction is outstanding, Data Distributor returns an error message.

You can delete a transfer only if the transfer is in the suspended state. See Section 9.67 for information about how to suspend a transfer.

When you delete a transfer definition, the associated target database does not change in any way.

If you want to delete a transfer definition, the transfer definition must be associated with your UIC.

DELETE TRANSFER Statement (VAX Data Distributor)

Examples

Example 1

The following example suspends the MA_EMPLOYEES transfer and then deletes its definition:

```
RDO> STOP TRANSFER MA_EMPLOYEES  
RDO> DELETE TRANSFER MA_EMPLOYEES.
```

9.35 DELETE TRIGGER Statement

Deletes one or more trigger definitions from the database.

Format

```
DELETE TRIGGER trigger-name .
```

Arguments

trigger-name

The name of the trigger to be deleted.

Usage Notes

To delete a trigger, you must have the Rdb/VMS DELETE privilege to the relation for which the trigger is defined.

Attempts to delete a trigger will fail if that trigger is involved in a query at the same time. Users will have to detach from the database with a FINISH statement before you can delete the trigger. When Rdb/VMS first accesses an object, such as the trigger, a lock is taken out on that object and not released until the user exits the database. Attempts to update this object will get a LOCK CONFLICT ON CLIENT message due to the other user's optimized access to the object.

Similarly, while you delete a trigger, users cannot execute queries involving that trigger until you have completed the transaction with a COMMIT or ROLLBACK statement for the DELETE statement. The user will receive a LOCK CONFLICT ON CLIENT error message. While DDL operations are performed, normal data locking mechanisms are used against system relations. (System relations hold information on objects in the database.) Therefore, attempts to update an object will lock out attempts to query that object. These locks are held until the commit or rollback of the DDL operation.

DELETE TRIGGER Statement

The WAIT/NOWAIT clause of the START_TRANSACTION statement does not affect attempts to update metadata with simultaneous queries. Even if you specify START_TRANSACTION WAIT for the metadata update transaction, you will get the following error message if a lock conflict exists:

```
%RDB-E-LOCK_CONFLICT, request failed due to locked resource; no-wait  
parameter specified for transaction  
-RDB-E-NO_META_UPDATE, metadata update failed  
-RDMS-F-LCKCNFLCT, lock conflict on client  
RDO>
```

However, a user's query will wait for a metadata update to complete with a ROLLBACK or COMMIT statement, even if the user specified NOWAIT on the START_TRANSACTION statement.

You cannot execute the DELETE TRIGGER statement when the RDB\$SYSTEM storage area is set to read-only. You must first set RDB\$SYSTEM to read/write. See the *VAX Rdb/VMS Guide to Database Maintenance and Performance* for more information on the RDB\$SYSTEM storage area.

Examples

Example 1

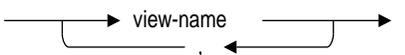
```
RDO> DELETE TRIGGER EMPLOYEE_ID_CASCADE_DELETE.
```

This example shows the deletion of a trigger named EMPLOYEE_ID_CASCADE_DELETE.

9.36 DELETE VIEW Statement

Deletes one or more view definitions. When the DELETE VIEW statement executes, Rdb/VMS deletes the view definition from the physical database. If you invoke the database using the PATHNAME qualifier, Rdb/VMS also deletes the view definition from the data dictionary.

Format

`DELETE VIEW`  .

Argument

view-name

The name of the view definition you want to delete.

Usage Notes

To delete a view with the DEFINE VIEW statement, you must have the Rdb/VMS DELETE privilege to the view.

If the database is created with the `DICTIONARY IS REQUIRED` option, you must invoke the database by path name, rather than file name, before you issue this statement.

You must execute this statement in a read/write transaction. If you issue this statement when there is no active transaction, Rdb/VMS starts a read/write transaction implicitly.

Other users are allowed to be attached to the database when you issue the DELETE VIEW statement.

Deleting a view does not affect active users until you commit your transaction, and the users detach from the database and attach to the database again.

You cannot execute the DELETE VIEW statement when the RDB\$SYSTEM storage area is set to read-only. You must first set RDB\$SYSTEM to read/write. See the *VAX Rdb/VMS Guide to Database Maintenance and Performance* for more information on the RDB\$SYSTEM storage area.

DELETE VIEW Statement

Examples

Example 1

The following example deletes a single view definition:

```
RDO> INVOKE DATABASE PATHNAME "PERSONNEL"  
RDO> START_TRANSACTION READ_WRITE  
RDO> DELETE VIEW CURRENT_INFO.  
RDO> COMMIT
```

This statement deletes the view definition CURRENT_INFO from the physical database and the data dictionary.

Example 2

You can also delete more than one view definition with one statement. This example shows that you cannot delete a view when other views refer to it. You must delete the referencing views before you can delete the underlying views.

```
RDO> INVOKE DATABASE PERS = PATHNAME 'PERSONNEL'  
RDO> START_TRANS READ_WRITE  
RDO> DELETE VIEW CURRENT_JOB.  
%RDMS-F-VIEWINVIEW, view, CURRENT_JOB, is referenced by view, CURRENT_INFO  
-RDMS-F-VIEWNOTDEL, view, CURRENT_JOB, has not been deleted  
RDO> DELETE VIEW CURRENT_INFO.  
RDO> DELETE VIEW CURRENT_SALARY.  
RDO> DELETE VIEW CURRENT_JOB.  
RDO> SHOW RELATIONS  
User Relations in Database with db_handle PERS  
COLLEGES  
DEGREES  
DEPARTMENTS  
EMPLOYEES  
JOBS  
JOB_HISTORY  
RESUMES  
SALARY_HISTORY  
WORK_STATUS  
RDO> COMMIT
```

This sequence shows how to delete a set of views, some of which depend on others. In this case CURRENT_INFO is a view definition that combines fields from two other views, CURRENT_JOB and CURRENT_SALARY. Although you can define views based on other views, performance improves when you base view definitions on the base relations themselves.

9.37 EDIT Statement

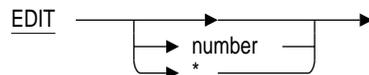
Calls an editor that lets you edit the RDO statements you have issued within a terminal session. By default, Rdb/VMS uses the VAX EDT editor. You can use EDT in the normal fashion to modify your previous RDO statements, construct your next statement or group of statements, or include a file with other statements.

If you have the VAX Text Processing Utility (VAXTPU) installed on your system, you can invoke VAXTPU with the EDIT statement in an RDO session. To use VAXTPU in an RDO session, define the logical name RDO\$EDIT in the following way:

```
$ DEFINE RDO$EDIT "TPU"
```

Then, when you type EDIT in an RDO session, VAXTPU is invoked. If RDO\$EDIT is not defined, or is defined to be something other than VAXTPU, then the EDT editor will be invoked when you issue the EDIT statement. If RDO cannot find the VAXTPU shareable image, EDT will be invoked. To use your personal VAXTPU section file with RDO EDIT, you must define the logical name TPU\$SECTION to be your personal section file.

Format



Defaults

If you do not include a number or a wildcard, RDO places the last statement in the editing buffer.

If you do not use the SET EDIT KEEP statement, EDIT * puts the last 20 statements in your editing buffer.

EDIT Statement

Arguments

number

The number of previous statements you want to edit. This must be an integer. If *number* = 0, then RDO calls the editor with an empty main editing buffer.

*

The wildcard character. If you use the wildcard, RDO includes in the editing buffer *n* previous commands, where *n* is the number specified in SET EDIT KEEP *n*. The default is 20 statements.

Usage Notes

If you have invoked a database, you have the necessary privileges to use the EDIT statement.

You can use the editor you choose with your usual initialization file to modify your previous RDO statements, construct your next statement or group of statements, or include a file with other statements.

Examples

Example 1

The following sequence demonstrates the correction of a misspelled statement.

1 Make a mistake:

```
RDO> FOR J IN JOSB
cont> PRINT J.JOB_TITLE END_FOR
%RDO-F-RELNOTDEF, Relation JOSB is not defined in database
RDO>
```

2 Invoke the EDT editor:

```
RDO> EDIT
```

3 When in the editor, change “JOSB” to “JOBS.” See the *EDT Editor Manual* for more information on using EDT.

EDIT Statement

- 4 Exit from the editor. RDO automatically executes the contents of the editing buffer.

```
* EXIT
Associate Programmer
Clerk
Deputy Clerk
Department Manager
Dept. Supervisor
.
.
.
```

END_SEGMENTED_STRING Statement

9.38 END_SEGMENTED_STRING Statement

Marks the end of a block that starts with a CREATE_SEGMENTED_STRING statement or a START_SEGMENTED_STRING statement.

Format

END_SEGMENTED_STRING \longrightarrow *ss-handle*

Argument

ss-handle

The handle given to the segmented string in the CREATE_SEGMENTED_STRING or START_SEGMENTED_STRING statement.

Usage Note

If you have invoked a database, you have the necessary privileges to use the END_SEGMENTED_STRING statement.

Examples

See the examples in Section 9.9 and Section 9.62.

9.39 END_STREAM Statement

Ends a stream. The `END_STREAM` statement closes a currently open stream, but does not affect any other stream that you have open. The `END_STREAM` statement:

- Stops the retrieval of rows from the named stream
- Sets to null your current position in the stream

If you close a record stream, you can use the `START_STREAM` statement to establish a new record stream with the same name.

You can use the `END_STREAM` statement to close a declared or undeclared stream.

Format

```

END_STREAM → stream-name →
                ↙         ↘
                on-error
    
```

Arguments

stream-name

The stream that you want to close. If you are closing a declared stream, this name must be the same name used in the associated `DECLARE_STREAM` statement. If you are closing an undeclared stream, this name must be the same name used in the associated undeclared `START_STREAM` statement.

on-error

The `ON ERROR` clause. This clause specifies a host language statement or Rdb/VMS data manipulation statement to be performed if an Rdb/VMS error occurs.

END_STREAM Statement

Usage Notes

If you have invoked a database, you have the necessary privileges to use the END_STREAM statement.

If you are using declared streams, you can have more or fewer END_STREAM statements than declared START_STREAM statements in your program, as long as the structure of the program ensures that exactly one END_STREAM statement is executed for each START_STREAM statement that is executed.

If you are using declared streams, you can issue several END_STREAM statements in a module. As long as you use the same declared stream name in each END_STREAM statement, the END_STREAM statements will all refer to the same stream.

Examples

Example 1

The following example closes a stream named STREAM_X:

```
RDO> END_STREAM STREAM_X
```

Example 2

The following example closes a stream named OLD_STREAM in a COBOL program:

```
&RDB& END_STREAM OLD_STREAM
```

9.40 ERASE Statement

Erases records from a relation. Before you use the ERASE statement, start a read/write transaction and establish a record stream using a context variable with a FOR statement or a START_STREAM statement.

Format

ERASE → context-var 

Arguments

context-var

A temporary name specified in an RSE for name recognition. You must define the context variable in a START_STREAM statement or in a FOR loop.

See Chapter 4 for a complete description of context variables.

on-error

The ON ERROR clause. This clause specifies a host language statement or Rdb/VMS data manipulation statement to be performed if an Rdb/VMS error occurs.

Usage Notes

You need the Rdb/VMS READ and ERASE privileges to the relation and the Rdb/VMS ERASE privilege to the database to use the ERASE statement.

You cannot erase records from a view that was formed with one of the following clauses:

- WITH clause of an RDO record selection expression
- REDUCED TO clause of an RDO record selection expression
- CROSS clause of an RDO record selection expression
- UNION clause of an SQL select expression

ERASE Statement

Examples

Example 1

Assume you wish to erase all the records in the COLLEGES relation:

```
RDO> START_TRANSACTION READ_WRITE RESERVING
cont> COLLEGES FOR EXCLUSIVE WRITE
RDO> FOR C IN COLLEGES ERASE C END_FOR
RDO> PRINT COUNT OF C IN COLLEGES
0
RDO> COMMIT
```

This statement uses the loop established by the FOR statement and erases all the records from the COLLEGES relation.

Example 2

To erase all the information about an employee, you need to erase from several relations within a single transaction:

```
        DISPLAY "Enter the ID number of employee".
        DISPLAY "whose records you want to delete:  "
          WITH NO ADVANCING.
        ACCEPT EMP-ID.

&RDB&   START_TRANSACTION READ_WRITE
&RDB&   RESERVING EMPLOYEES      FOR PROTECTED WRITE,
&RDB&   JOB_HISTORY             FOR PROTECTED WRITE,
&RDB&   SALARY_HISTORY          FOR PROTECTED WRITE,
&RDB&   DEGREES                 FOR PROTECTED WRITE

&RDB& FOR E IN EMPLOYEES WITH E.EMPLOYEE_ID = EMP-ID
&RDB& ON ERROR
&RDB&   ROLLBACK
&RDB&   GO TO ERROR-PAR
&RDB& END_ERROR

&RDB& FOR JH IN JOB_HISTORY
&RDB&   WITH JH.EMPLOYEE_ID = E.EMPLOYEE_ID
&RDB& ON ERROR
&RDB&   ROLLBACK
&RDB&   GO TO ERROR-PAR
&RDB& END_ERROR
&RDB& ERASE JH
&RDB& ON ERROR
&RDB&   ROLLBACK
&RDB&   GO TO ERROR-PAR
&RDB& END_ERROR
&RDB& END_FOR
```

ERASE Statement

```
&RDB& FOR SH IN SALARY_HISTORY
&RDB&     WITH SH.EMPLOYEE_ID = E.EMPLOYEE_ID
&RDB&     ON ERROR
&RDB&     ROLLBACK
&RDB&     GO TO ERROR-PAR
&RDB&     END_ERROR
&RDB&     ERASE SH
&RDB&     ON ERROR
&RDB&     ROLLBACK
&RDB&     GO TO ERROR-PAR
&RDB&     END_ERROR
&RDB&     END_FOR

&RDB& FOR D IN DEGREES
&RDB&     WITH D.EMPLOYEE_ID = E.EMPLOYEE_ID
&RDB&     ON ERROR
&RDB&     ROLLBACK
&RDB&     GO TO ERROR-PAR
&RDB&     END_ERROR
&RDB&     ERASE D
&RDB&     ON ERROR
&RDB&     ROLLBACK
&RDB&     GO TO ERROR-PAR
&RDB&     END_ERROR
&RDB&     END_FOR

&RDB& ERASE E
&RDB&     ON ERROR
&RDB&     ROLLBACK
&RDB&     GO TO ERROR-PAR
&RDB&     END_ERROR
&RDB&     END_FOR

&RDB& COMMIT
```

This program segment erases the records for an employee from all the relations that contain employee information. Note that all the erase operations are included in one transaction so that no employee records are only partially erased.

ERASE Statement

Errors

Your program can check for and handle the following errors, using the name listed here in the form RDB\$_error-name. The following list includes two types of errors:

- E Expected errors. These are run-time errors. Your program should check for these errors and provide error handlers.
- U Unexpected errors. Preprocessors detect these errors when the program is preprocessed, so your program does not need to check for them. Callable RDO, however, detects these errors at run time. A Callable RDO program should check for them.

DEADLOCK (E)

request failed due to resource deadlock

INTEG_FAIL (U)

constraint *name* failed

LOCK_CONFLICT (E)

NO WAIT request failed because resource was locked

NO_PRIV (U)

privilege denied by database protection

READ_ONLY_FIELD (U)

attempt to update read only field *field-name*

READ_ONLY_REL (U)

relation *name* was reserved for READ, updates disallowed

READ_ONLY_TRANS (U)

attempt to update from a read-only transaction

READ_ONLY_VIEW (U)

view *name* cannot be updated

REQ_NO_TRANS (U)

attempt to continue request after COMMIT/ROLLBACK

9.41 Execute (@) Statement

Executes statements that are contained in a command file. The at sign (@) means execute in RDO, just as in DCL. When you type @ and the name of an indirect command file, RDO executes the statements in the file as if you had typed them one at a time at the RDO prompt. The command file must be a VMS text file that contains RDO statements.

Format

@file-spec

Argument

file-spec

The name of an indirect command file. You can use either a full VMS file specification, a file name, or a logical name. If you use a file name, RDO looks in the current default VMS directory for a file by that name. The file must contain valid RDO statements.

The default file type for an indirect command file is RDO.

Usage Notes

You do not need any special Rdb/VMS privileges to use the EXECUTE statement.

You can use the SET VERIFY statement to have the commands in the file displayed on the screen as they execute.

RDO recognizes a special RDO command file called RDOINI.RDO, which contains RDO statements to be issued before RDO displays the RDO> prompt. If this file exists, RDO executes the commands in that file first, before displaying the prompt and accepting your input. If you have defined the logical name RDOINI to point to a general initialization file, RDO uses this file. Otherwise, it looks for RDOINI.RDO in the current default directory.

Execute (@) Statement

Examples

Example 1

You can use an indirect command file to execute a frequently used query.

```
$ TY EMPADDR.RDO
!
! This command file generates information for a mailing list.
!
INVOKE DATABASE FILENAME 'DISK2:[DEPT32]MF_PERSONNEL.RDB'
SET OUTPUT 'DISK2:[DEPT32]MAILLIST.DOC'
FOR E IN EMPLOYEES
  PRINT E.FIRST_NAME, E.MIDDLE_INITIAL, E.LAST_NAME,
        E.ADDRESS_DATA_1, E.ADDRESS_DATA_2, E.CITY, E.STATE,
E.POSTAL_CODE
END_FOR
$ RUN SYS$SYSTEM:RDO
RDO> @EMPADDR
```

Example 2

You can use a logical name to run a command file.

```
$ SET DEFAULT DISK1:[FORESTER]
$ DEFINE COUNT "DISK2:[DEPT3]COUNT.RDO"
$ TYPE DISK2:[DEPT3]COUNT.RDO
!
! This command file counts the records in
! each relation in the MF_PERSONNEL database.
!
SET NOVERIFY
SET OUTPUT COUNT.LOG
INVOKE DATABASE FILENAME 'DISK2:[DEPT3]MF_PERSONNEL.RDB'

PRINT "  "
PRINT "Statistics for the MF_PERSONNEL database follow: "
PRINT "  "
PRINT "Count of Employees -----> ", COUNT OF X IN EMPLOYEES
PRINT "Count of Jobs -----> ", COUNT OF J IN JOBS
PRINT "Count of Degrees -----> ", COUNT OF D IN DEGREES
PRINT "Count of Salary_History --> ", COUNT OF SH IN SALARY_HISTORY
PRINT "Count of Job_History -----> ", COUNT OF JH IN JOB_HISTORY
PRINT "Count of Work_Status -----> ", COUNT OF W IN WORK_STATUS
PRINT "Count of Departments -----> ", COUNT OF D IN DEPARTMENTS
PRINT "Count of Colleges -----> ", COUNT OF C IN COLLEGES
PRINT "Count of Resumes -----> ", COUNT OF R IN RESUMES
PRINT "  "
PRINT "Statistics Complete for Database - written to COUNT.LOG"
PRINT "  "
PRINT "  "
```

Execute (@) Statement

```
$ !  
$ RUN SYS$SYSTEM:RDO  
RDO> @COUNT
```

Statistics for the PERSONNEL database follow:

Count of Employees ----->	100
Count of Jobs ----->	15
Count of Degrees ----->	166
Count of Salary_History -->	729
Count of Job_History ----->	274
Count of Work_Status ----->	3
Count of Departments ----->	26
Count of Colleges ----->	16
Count of Resumes ----->	0

Statistics Complete for Database - written to COUNT.LOG

```
RDO>
```

Notice that the default VMS directory is not the same as the directory that contains the command file. You define the process logical name to point to the correct command file. Then you use the logical name as the object of the @ command.

EXIT Statement

9.42 EXIT Statement

Causes RDO to exit. The EXIT statement allows you to end an RDO session. It returns you to the DCL prompt.

Format

EXIT

Usage Notes

You do not need any special Rdb/VMS privileges to use the EXIT statement.

You can also exit an RDO session by typing CTRL/Z.

If there is a transaction open and you have made changes to a database or the data dictionary, RDO will not automatically exit. Rather, RDO allows you to commit your changes.

Examples

Example 1

The following example shows how to end an RDO session. If there is a transaction open and you have made changes to the database or data dictionary when you issue the EXIT statement, RDO does not exit.

```
RDO> EXIT
There are uncommitted changes to a database or the CDD.
Would you like a chance to COMMIT these changes [NO] ?
```

If you type YES, RDO returns the RDO> prompt. Then you can type COMMIT, ROLLBACK, or any other RDO statement.

If you type NO, or any other response, RDO completes the exit.

EXPORT Statement

Arguments

database-file-spec

The file specification for the database you want to export. Use either a full or partial file specification or a logical name. If you use a simple file name, Rdb/VMS looks for the database in the current default directory. The default file type is RDB.

interchange-file-spec

The file specification for a file in which EXPORT places an intermediate version of the database. Use either a full or partial file specification or a logical name. If you use a simple file name, Rdb/VMS places the interchange file in the current default directory. The default file type is RBR.

This parameter can also refer to a magnetic tape volume. If you are exporting the database to tape, use the device name as part of the file specification.

WITH EXTENSIONS [Default]

WITH NOEXTENSIONS

Specifies whether the exported format of the database is compatible with a pre-Version 3.0 Rdb/VMS database (WITH NOEXTENSIONS), or with an Rdb/VMS database created with Version 3.0 Rdb/VMS software or higher (WITH EXTENSIONS). The default is WITH EXTENSIONS. The WITH EXTENSIONS option also directs Rdb/VMS to preserve other parameters about the physical structure of the exported database. Use WITH NOEXTENSIONS in the following situations:

- When you migrate a database to Rdb/ELN
- When you export a database that you plan to restore on pre-Version 3.0 Rdb/VMS software

When you specify the WITH NOEXTENSIONS option, null flags for missing value fields are not backed up; therefore, numeric missing values are replaced by zeros and character missing values are replaced by blanks.

Note also that when you specify the WITH NOEXTENSIONS option, features of Version 3.0 and higher Rdb/VMS databases are not backed up. Storage maps, triggers, and collating sequences, for example, are not backed up when you specify the WITH NOEXTENSIONS option.

Note *The WITH NOEXTENSIONS option is not compatible with CDD/Plus dictionary databases (CDD\$DATABASE.RDB). If you attempt to export a*

EXPORT Statement

CDD\$DATABASE.RDB database, RDO issues an error message stating that the *NOEXTENSIONS* option is not valid for CDD/Plus dictionary databases.

WITH DATA [Default]

WITH NODATA

Specifies whether the RBR file created by the EXPORT statement includes the data and metadata contained in the database, or the metadata only. WITH DATA is the default.

When you specify the WITH NODATA option, the EXPORT statement copies the metadata, but not the data, from a source database to an RBR file. Use the IMPORT statement to generate an empty database whose metadata is identical to that of the source database.

Note The WITH NODATA option is not compatible with CDD/Plus dictionary databases (*CDD\$DATABASE.RDB*). If you attempt to export a *CDD\$DATABASE.RDB* database, RDO issues an error message stating that the NODATA option is not valid for CDD/Plus dictionary databases.

Usage Notes

You must have the Rdb/VMS READ privilege to the database relations to use the EXPORT statement.

You cannot use the EXPORT statement on a database that has over 65,535 segments in one segmented string field. However, the RMU/BACKUP command does not have this limitation on the number of segments within a segmented string.

The EXPORT statement does not check for a corrupt database. If the database is corrupt, the EXPORT and IMPORT statements create a valid database, but the contents of that database may not be identical to the original. Use the RMU/VERIFY command before you export the database to check the database contents.

Make a backup copy of your database if you intend to delete the original database after you use the EXPORT statement.

The EXPORT statement does not allow Rdb/VMS system fields in user-defined relations. Rdb/VMS returns an error message and the export operation fails if you use Rdb/VMS system fields in relations.

For long-term storage, export the database to disk and then backup the RBR file to tape or optical disk.

EXPORT Statement

See the *VAX Rdb/VMS Guide to Database Maintenance and Performance* for a complete discussion of when to use the IMPORT, EXPORT, and CHANGE DATABASE statements.

Examples

Example 1

The following example shows how to export a database to a disk file:

```
RDO> EXPORT
cont> 'DEPT3:PERSONNEL.RDB' INTO
cont> 'EXPORT$DISK:[EXPORT]PERSONNEL.RBR'
```

This statement creates an interchange file for the PERSONNEL database from the database file identified by DEPT3:PERSONNEL.RDB. DEPT3 in this case is a logical name for the device and directory where the database is located. The copy is stored on EXPORT\$DISK in the directory [EXPORT]. By default, the database is exported WITH EXTENSIONS.

Example 2

The following example shows how to export a database to a magnetic tape volume:

```
$ INITIALIZE MUA0:
_Label: PERS
$
$ MOUNT MUA0:
_Label: PERS
_Log name:
$ RDO
RDO> EXPORT 'DEPT3:PERSONNEL.RDB' INTO 'MUA0:PERSONNEL.RBR'
```

This statement creates an intermediate copy of the database on the magnetic tape volume labeled PERS, mounted on device MUA0:.

Example 3

The following example shows how to use the EXPORT statement to migrate an Rdb/VMS multifile database to an Rdb/ELN system.

```
RDO> EXPORT 'MF_PERS.RDB' INTO 'ELN_PERS.RBR' WITH NOEXTENSIONS
```

Because Rdb/ELN must see the exported database as a pre-V3.0 database, you must use the NOEXTENSIONS option.

EXPORT Statement

Example 4

The following example shows how to export the metadata in a database without its data:

```
RDO> EXPORT
cont> 'DEPT3:PERSONNEL.RDB' INTO
cont> 'EXPORT$DISK:[EXPORT]PERSONNEL.RBR'
cont> WITH NODATA
```

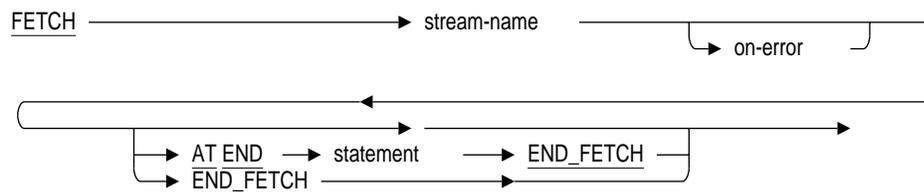
This statement creates an interchange file for the PERSONNEL database from the database file identified by DEPT3:PERSONNEL.RDB. The WITH NODATA option specifies that the interchange file, PERSONNEL.RBR, contains only the metadata that defines the structure of PERSONNEL.RDB.

FETCH Statement

9.44 FETCH Statement

Advances the pointer for a record stream to the next record of a relation. Once you have established and opened a stream with the `START_STREAM` statement, use the `FETCH` statement to establish the first record in a relation as the current record. After that, each `FETCH` statement makes the next record in the stream the current one. The `FETCH` statement only advances the pointer in a record stream. You must use other data manipulation statements to manipulate each record. For instance, you might use `FETCH` to advance the pointer and the `GET` statement to assign values from that record to host language variables.

Format



Arguments

stream-name

Names the stream in which you want to advance the stream pointer. Rdb/VMS moves the pointer to the next record that it can return.

Because the `FETCH` statement defines the current record in a record stream, you must use a `FETCH` statement before any other Rdb/VMS data manipulation statement when you are using streams.

on-error

The `ON ERROR` clause. This clause specifies the action to be taken if an Rdb/VMS error occurs during the `FETCH` operation.

statement

Any valid host language or Rdb/VMS statement. The program executes the statement specified in the `AT END` clause when no records remain to be

FETCH Statement

processed in the record stream. If you use the AT END clause, you must also include END_FETCH.

Usage Note

If you have invoked a database, you have the necessary privileges to use the FETCH statement.

Examples

Example 1

This COBOL program fragment advances a stream pointer in an open stream.

```
CONTROL-PAR.  
    PERFORM INIT THRU INIT-END.  
    PERFORM LOOP UNTIL DONE = "Y".  
    PERFORM GET-OUT.  
    STOP RUN.  
  
INIT.  
&RDB& START_TRANSACTION READ_WRITE  
&RDB& START_STREAM WORKERS USING C IN CURRENT_INFO.  
  
INIT-END.  
  
    EXIT.  
  
LOOP.  
  
&RDB&  FETCH WORKERS  
&RDB&      ON ERROR  
          GO TO STREAM-ERROR  
&RDB&      END_ERROR  
&RDB&      AT END  
          MOVE "Y" TO DONE  
          GO TO GET-OUT  
  
&RDB&  END_FETCH  
&RDB&  GET  
&RDB&      LAST = C.LAST;  
&RDB&      DEPARTMENT = C.DEPARTMENT;  
&RDB&      SALARY = C.SALARY  
&RDB&  END_GET  
    DISPLAY LAST, DEPARTMENT, SALARY.  
  
STREAM-ERROR.  
  
    DISPLAY "Error in START_STREAM".  
    STOP RUN.  
  
GET-OUT.
```

FETCH Statement

```
&RDB& END_STREAM WORKERS  
&RDB& COMMIT  
&RDB& FINISH.
```

This program fragment does the following:

- Starts a stream and gives it the name WORKERS. The RSE specifies a set of records, in this case the whole CURRENT_INFO view.
- Uses COBOL statements to set up a loop and give the ending condition for the loop.
- Uses the FETCH and GET statements to retrieve one record on each pass through the loop and place three fields from that record into host language variables.
- Uses AT END and ON ERROR to handle end-of-stream and error conditions.
- Uses the COBOL DISPLAY statement to display the variables each time through the loop.

Errors

Your program can check for and handle the following errors, using the name listed here in the form RDB\$_error-name. The following list includes two types of errors:

- E Expected errors. These are run-time errors. Your program should check for these errors and provide error handlers.
- U Unexpected errors. Preprocessors detect these errors when the program is preprocessed, so your program does not need to check for them. Callable RDO, however, detects these errors at run time. A Callable RDO program should check for them.

DEADLOCK(E)

request failed due to resource deadlock

LOCK_CONFLICT (E)

NO WAIT request failed because resource was locked

NO_CUR_REC (U)

stream has no current record for retrieval or update

NO_PRIV (U)

privilege denied by database protection

FETCH Statement

STREAM_EOF (E)

attempt to fetch past end of stream

WRONUMARG (U)

illegal number of arguments on call to Rdb system

FINISH Statement

9.45 FINISH Statement

Explicitly declares a database closed when you are done working with it. The FINISH statement with no parameter also commits all active transactions. When you are using multiple databases, you can use the FINISH statement to close databases to save system resources. It is recommended that you use the FINISH statement explicitly before exiting from a program.

Format



Argument

db-handle

A host language variable that identifies the database to be accessed. Use the db-handle that you associated with the database in the INVOKE statement.

In Callable RDO, use the substitution marker !VAL to specify the db-handle.

on-error

The ON ERROR clause. This clause specifies a host language statement or Rdb/VMS data manipulation statement to be performed if an Rdb/VMS error occurs.

Usage Notes

If you have invoked a database, you have the necessary privileges to use the FINISH statement.

When you use the INVOKE statement to declare databases and then use the START_TRANSACTION statement, the Rdb/VMS preprocessors automatically attach your process to all the databases you have invoked. This involves some overhead and record locking. If the FINISH statement occurs explicitly in the run-time flow of the program, Rdb/VMS detaches from the database specified in the FINISH statement. If another START_TRANSACTION occurs, Rdb/VMS attaches to the databases again. Therefore, if your program uses databases

FINISH Statement

sequentially, you can use the FINISH statement to close each database as you finish working with it. In this way, Rdb/VMS attaches only the databases you need at any one time. You should be aware, however, that attaching and detaching also consume system resources.

Examples

Example 1

The following program fragment shows how to use the INVOKE and FINISH statement to work with two databases:

```
&RDB&  INVOKE DATABASE WORKERS =
&RDB&  FILENAME 'DISK2:[DEPT4]EMPLOYEES'
&RDB&  INVOKE DATABASE PARTS =
&RDB&  FILENAME 'DISK2:[DEPT4]PARTS'
.
.
.
!  [This part of the program can combine data from
    both databases]
.
.
&RDB&  FINISH WORKERS
.
.
.
!  [This part of the program uses only
    the PARTS database]
.
.
&RDB&  FINISH PARTS
```

Between the second INVOKE statement and the first FINISH statement, you can access both databases at once.

FOR Statement

You can put parentheses around the host language variable name.

See the *VAX Rdb/VMS Guide to Using RDO, RDBPRE, and RDML* for information on using request handles.

TRANSACTION_HANDLE var

A keyword followed by a host language variable. A transaction handle identifies each instance of a database attach. If you do not declare the transaction handle explicitly, Rdb/VMS uses the default transaction handle.

In Callable RDO, use !VAL as a marker for host language variables.

You can put parentheses around the host language variable name.

Note *Normally, you do not need to use this argument. The ability to declare a transaction handle is provided for compatibility with other database products and future releases of Rdb/VMS.*

Do not use this argument in interactive RDO.

rse

Any valid record selection expression. See Chapter 4 for a complete discussion of record selection expressions.

on-error

The ON ERROR clause. This clause specifies the action to be taken if an error occurs while Rdb/VMS is compiling the RSE.

statement

Any valid Rdb/VMS data manipulation statement or host language statement except INVOKE, COMMIT, or ROLLBACK.

No statement within the FOR loop can redefine the context variable that was defined by the RSE in the FOR statement.

Usage Notes

You need the Rdb/VMS READ privilege to the records in a record stream to use the FOR statement.

You can nest FOR loops as an alternative to the CROSS clause to perform a join operation. However, the performance of the CROSS clause is usually faster.

FOR Statement

In RDO, each FOR loop has a stream associated with it. When you use nested FOR loops, the actions of one stream may affect how the other streams work. In the following example, the first RDO statement shows a query that causes concurrent read and write activity to occur in relation RELATION3. *Queries that cause concurrent read and write activity should be avoided because they can return incorrect results.*

```
RDO> START_TRANSACTION READ_WRITE
cont> FOR R IN RELATION3 WITH R.FIELD2=1
cont>     PRINT R.*, R.RDB$DB_KEY
cont>     FOR T IN RELATION3 WITH T.FIELD1=R.FIELD1 AND T.FIELD2=0
cont>         ERASE T
cont>     END_FOR
cont> END_FOR
```

The same problem can be created in an application by opening two streams on the same relation with one stream reading data and the other modifying it. You can avoid a problem with concurrent read and write activity in a single relation by using query syntax carefully or performing the actions in the query serially.

You can fix the previous query by replacing it with the following two queries, which perform the actions in the previous query serially:

```
RDO> FOR R IN RELATION3 WITH R.FIELD2=1
cont>     PRINT R.*, R.RDB$DB_KEY
cont> END_FOR
RDO> FOR T IN RELATION3 WITH T.FIELD2=0 AND
cont>     (ANY R IN RELATION3 WITH R.FIELD1=T.FIELD1 AND R.FIELD2=1)
cont>     ERASE T
cont> END_FOR
```

If you are interested in printing the records that will be erased, the following query works:

```
RDO> FOR T IN RELATION3 WITH T.FIELD2=0 AND
cont>     (ANY R IN RELATION3 WITH R.FIELD1=T.FIELD1 AND R.FIELD2=1)
cont>     PRINT T.*, T.RDB$DB_KEY
cont>     ERASE T
cont> END_FOR
```

FOR Statement

Examples

Example 1

The following example creates a record stream with a FOR statement in RDO:

```
RDO> START_TRANSACTION READ_ONLY
RDO>
RDO> FOR D IN DEPARTMENTS WITH D.DEPARTMENT_CODE = "SEUR"
cont>   PRINT D.DEPARTMENT_CODE,
cont>         D.DEPARTMENT_NAME,
cont>         D.MANAGER_ID
cont> END_FOR
RDO>
RDO> COMMIT
```

These statements:

- Create a record stream defined by a record selection expression
- Retrieve three field values from each record in that stream

Example 2

Here is a similar sequence using the FOR statement in BASIC:

```
&RDB& START_TRANSACTION READ_ONLY
&RDB&
&RDB& FOR E IN EMPLOYEES CROSS
&RDB& S IN SALARY_HISTORY OVER EMPLOYEE_ID
&RDB&   WITH E.EMPLOYEE_ID = EMPLOYEE_ID
&RDB&   AND S.SALARY_END MISSING
&RDB&   ON ERROR
&RDB&       GOTO 3000
&RDB&   END_ERROR
&RDB&   GET
&RDB&       LAST_NAME = E.LAST_NAME;
&RDB&       FIRST_NAME = E.FIRST_NAME;
&RDB&       SALARY = S.SALARY_AMOUNT
&RDB&   END_GET
&RDB& END_FOR
&RDB& COMMIT
```

This program fragment retrieves the current salary for an employee specified by the value of the EMPLOYEE_ID variable. The example:

- Establishes a record stream that consists of the record in the EMPLOYEES relation with the ID number that the user supplies in the host language variable EMPLOYEE_ID, joined with the corresponding current SALARY_HISTORY record

FOR Statement

- Points to an error-handling subroutine, in case of errors from Rdb/VMS
- Assigns the values from the FIRST_NAME and LAST_NAME fields of the EMPLOYEES relation and the SALARY_AMOUNT field of the SALARY_HISTORY relation to host language variables

Errors

Your program can check for and handle the following errors, using the name listed here in the form RDB\$_error-name. The following list includes two types of errors:

- E Expected errors. These are run-time errors. Your program should check for these errors and provide error handlers.
- U Unexpected errors. Preprocessors detect these errors when the program is preprocessed, so your program does not need to check for them. Callable RDO, however, detects these errors at run time. A Callable RDO program should check for them.

ARITH_EXCEPT (U)

truncation of a numeric value at runtime

DEADLOCK (E)

request failed due to resource deadlock

LOCK_CONFLICT (E)

NO WAIT request failed because resource was locked

NO_PRIV (U)

privilege denied by database protection

WRONUMARG (U)

illegal number of arguments on call to Rdb system

FOR Statement with Segmented Strings

9.47 FOR Statement with Segmented Strings

Sets up a record stream that consists of segments from a segmented string field. Because a single segmented string field value is made up of multiple segments, a record stream that includes a segmented string field is nested. The outer loop retrieves records that include the field, and the inner loop retrieves the segments of each field value one at a time. Therefore, a FOR statement that retrieves segmented strings looks like a set of nested FOR statements.

Rdb/VMS defines a special name to refer to the segments of a segmented string. This value expression is equivalent to a field name; it names the fields or segments of the string. Furthermore, because segments can vary in length, Rdb/VMS also defines a name for the length of a segment. The statement inside the segmented string FOR loop must use these names to refer to the segments of the string. These names are:

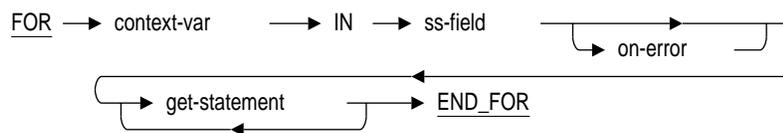
RDB\$VALUE

The value stored in a segment of a segmented string

RDB\$LENGTH

The length in bytes of a segment

Format



Arguments

context-var

A context variable. See Chapter 4 for a complete description of context variables.

FOR Statement with Segmented Strings

ss-field

A qualified field name that refers to a field defined with the SEGMENTED STRING data type. Note that this field name, like all field names in a FOR statement, must be qualified by its own context variable. This second context variable must match the variable declared in the outer FOR statement. See the Examples section.

on-error

The ON ERROR clause. This clause specifies the action to be taken if an Rdb/VMS error occurs while Rdb/VMS is trying to set up retrieval of the next segmented string.

get-statement

Any valid Rdb/VMS DML or host language statement except INVOKE, COMMIT, or ROLLBACK. The GET statement can reference only the RDB\$VALUE and RDB\$LENGTH fields.

Usage Notes

If you have invoked a database, you have the necessary privileges to use the FOR Statement with Segmented Strings.

Do not declare the segmented string field in your program. The data type generated for a segmented string field in a relation is that of the segmented string identifier, which is the value that actually is stored in a segmented string field. For example, if you want to include the following RDO statements in your program, you should not declare the field RESUME. (However, you would need to replace the RDO PRINT statement with your host language display statement.)

```
FOR R IN RESUMES  
    PRINT R.RESUME  
END_FOR;
```

FOR Statement with Segmented Strings

Examples

Example 1

The following example creates a stream whose records contain segmented string fields:

```
RDO> FOR R IN RESUMES
cont>   FOR S IN R.RES_SEG
cont>   PRINT S.RDB$LENGTH, S.RDB$VALUE
cont>   END_FOR
cont> END_FOR
```

This statement looks like a nested FOR loop:

- The outer loop sets up a record stream using the context variable R. The same context variable qualifies the field name, SS_FIELD, as in every FOR statement.
- The inner loop retrieves the segments of the string field one at a time.
- The context variable S identifies the segments.
- The special segmented string value expressions, RDB\$VALUE and RDB\$LENGTH, are qualified by S, the context variable associated with the field.

Errors

Your program can check for and handle the following errors, using the name listed here in the form RDB\$_error-name. The following list includes two types of errors:

- E Expected errors. These are run-time errors. Your program should check for these errors and provide error handlers.
- U Unexpected errors. Preprocessors detect these errors when the program is preprocessed, so your program does not need to check for them. Callable RDO, however, detects these errors at run time. A Callable RDO program should check for them.

ARITH_EXCEPT (U)

truncation of a numeric value at runtime

DEADLOCK (E)

request failed due to resource deadlock

FOR Statement with Segmented Strings

LOCK_CONFLICT (E)

NO WAIT request failed because resource was locked

NO_PRIV (U)

privilege denied by database protection

WRONUMARG (U)

illegal number of arguments on call to Rdb system

9.48 GET Statement

Assigns values from data records in a record stream to host language variables in RDBPRE (BASIC, COBOL, and FORTRAN) programs. You can use the GET statement in three ways:

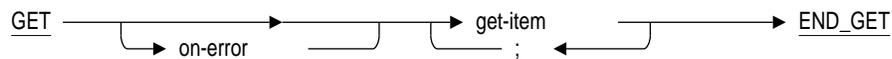
- When you establish a record stream with the FOR or START_STREAM statement, you use the GET statement to assign values from the current record in the stream to variables in your program. For the START_STREAM statement, you also need a FETCH statement to indicate the current record in the stream.
- You can use GET . . . RDB\$DB_KEY in a STORE . . . END_STORE block to retrieve the dbkey of the record just stored. For example:

```
&RDB&   STORE E IN EMPLOYEES USING E.EMPLOYEE_ID = 15231;
&RDB&                                     E.LAST_NAME = "Smith";
&RDB&           GET MY_DB_KEY = E.RDB$DB_KEY;
&RDB&           END_GET
&RDB&   END_STORE
```

(MY_DB_KEY is a user-defined host language variable.)

- You can also use the GET statement alone, without a FOR, FETCH, or STORE statement, to retrieve the result of a statistical function. The record stream is formed by the record selection expression within the statistical expression.

Format



get-item =



GET Statement

Arguments

get-item

The GET statement includes an assignment statement that specifies a host language variable and a database value. The database value is assigned to the host language variable from the Rdb/VMS value expression or statistical expression.

If you list a field for which the value is null, and there is a MISSING_VALUE clause for that field, Rdb/VMS supplies the missing value.

If there is no MISSING_VALUE clause, and missing field values are allowed, then Rdb/VMS supplies the following values when you retrieve the field:

- Zeros for numeric fields
- Blanks for text fields
- The VMS base date and time for date fields (17-NOV-1858 00:00:00.00)

on-error

The ON ERROR clause. This clause specifies the action to be taken if an Rdb/VMS error occurs during the GET operation.

value-expr

A valid Rdb/VMS value expression. The value expression can include the “<context-variable> . RDB\$DB_KEY” expression. During a STORE . . . END_STORE block, this expression lets you retrieve the dbkey of the record just stored. See Section 9.52 for related information on the DBKEY SCOPE FINISH clause. Also see Chapter 3 for more information on value expressions.

record-descr

A valid data dictionary record descriptor matching all of the fields of the relation. You can use a host language statement to include the relation definition in your program. Each field of the record descriptor must match exactly the field names and data types of the fields in the Rdb/VMS relation referred to by the context variable.

host-variable

A valid variable name declared in the host language program. See the *VAX Rdb/VMS Guide to Using RDO, RDBPRE, and RDML* for a complete description of host language variables.

GET Statement

statistical-expr

A statistical expression. It calculates values based on a value expression for every record in a record stream. See Chapter 3 for a complete list of the Rdb/VMS statistical expressions.

Usage Notes

If you have invoked a database, you have the necessary privileges to use the GET statement.

You cannot use the GET statement in RDO. RDO uses the PRINT statement to display values on the terminal.

You cannot use the concatenation operator in a GET statement. To concatenate fields in preprocessed programs, first use the GET statement to retrieve the individual fields into separate host language variables. Then concatenate these variables in a host language statement using the host language concatenation operator.

Any context variable used in a value expression or in a context-var.* expression must be within its scope. That is, these expressions must be within a FOR or STORE statement. However, you can use the GET statement alone (without a FOR or STORE statement) to retrieve the result of a statistical function.

Examples

Example 1

The following COBOL example retrieves values from named fields in a relation:

```
&RDB& START_TRANSACTION READ_WRITE
&RDB& FOR E IN EMPLOYEES
&RDB&     GET
&RDB&         LAST-NAME = E.LAST_NAME;
&RDB&         FIRST-NAME = E.FIRST_NAME;
&RDB&         MIDDLE-INITIAL = E.MIDDLE_INITIAL
&RDB&     END_GET
&RDB& END_FOR
&RDB& COMMIT
```

This program fragment retrieves field values from each record in the EMPLOYEES relation. It assumes that the program has declared the three host language variables, LAST-NAME, FIRST-NAME, and MIDDLE-INITIAL, with the appropriate data types.

GET Statement

Example 2

The following set of statements performs a join of two relations and uses the GET statement to retrieve a value from each:

```
&RDB& START_TRANSACTION READ_WRITE
&RDB& FOR JH IN JOB_HISTORY CROSS D IN DEPARTMENTS
&RDB&     OVER DEPARTMENT_CODE
&RDB&     WITH JH.JOB_END MISSING
&RDB&     GET
&RDB&         ID-NUMBER = JH.EMPLOYEE_ID;
&RDB&         DEPT-NAME = D.DEPARTMENT_NAME
&RDB&     END_GET
&RDB& END_FOR

&RDB& COMMIT
```

Example 3

The following BASIC program fragment retrieves the result of a statistical function:

```
      INPUT "State:  ", STATE
&RDB& START_TRANSACTION READ_ONLY
&RDB&     GET
&RDB&     NUMBER-EMPLOYEES = COUNT OF E IN EMPLOYEES
&RDB&                       WITH E.STATE = STATE
&RDB&     END_GET

      PRINT "Number of employees in "; STATE; " is "; NUMBER-EMPLOYEES &
&RDB& COMMIT
```

This statement retrieves the number of employees who live in the specified state and assigns that number to the variable, NUMBER-EMPLOYEES.

Example 4

The following FORTRAN program fragment shows how the database key (dbkey) of a record just stored can be retrieved into a host language variable, using GET . . . RDB\$DB_KEY, in a STORE . . . END_STORE block. For this complete program, see the examples in Section 9.52.

GET Statement

```
&RDB& DATABASE FILENAME 'PERSONNEL' DBKEY SCOPE IS FINISH
&RDB& START_TRANSACTION READ_WRITE

&RDB&   STORE P IN EMPLOYEES USING P.EMPLOYEE_ID =15231;
&RDB&   P.LAST_NAME = "Santoli";
&RDB&   GET MY_DB_KEY = P.RDB$DB_KEY;
&RDB&   END_GET
&RDB& END_STORE

&RDB& COMMIT

&RDB& START_TRANSACTION READ_WRITE
&RDB&   FOR J IN EMPLOYEES WITH J.RDB$DB_KEY = MY_DB_KEY
&RDB&     GET
&RDB&       1 JC = J.EMPLOYEE_ID;
&RDB&       2 JT = J.LAST_NAME;
&RDB&       3 END_GET
&RDB&       4 END_FOR
&RDB&     WRITE (6,60020) JC, JT
&RDB&     .
&RDB&     .
&RDB&     .
```

Errors

Your program can check for and handle the following errors, using the name listed here in the form RDB\$_error-name. The following list includes two types of errors:

- E Expected errors. These are run-time errors. Your program should check for these errors and provide error handlers.
- U Unexpected errors. Preprocessors detect these errors when the program is preprocessed, so your program does not need to check for them. Callable RDO, however, detects these errors at run time. A Callable RDO program should check for them.

DEADLOCK (E)

request failed due to resource deadlock

LOCK_CONFLICT (E)

NO WAIT request failed because resource was locked

NO_CUR_REC (U)

stream has no current record for retrieval or update

NO_PRIV (U)

privilege denied by database protection

GET Statement

NO_RECORD (U)

access by DBKEY failed—record previously deleted

OBSOLETE_METADATA (U)

request references undefined fields or relations

REQ_NO_TRANS (U)

attempt to continue request after COMMIT/ROLLBACK

SEGMENT (E)

segmented string is not complete

SEGSTR_EOF (E)

fetch past end of segmented string

SEGSTR_NO_OP (U)

operation is not valid on segmented strings

SEGSTR_NO_READ (U)

segmented string is open for write, read prohibited

WRONUMARG (U)

illegal number of arguments on call to Rdb system

9.49 HELP Statement

Gives you access to assistance on all Rdb/VMS statements, components, and concepts.

When you type HELP:

- A menu of topics on which assistance is available replaces the RDO> prompt.
- After the menu scrolls by, the cursor remains at a “Topic?” prompt. To get help on a topic, type one of the items at the “Topic?” prompt. Many of the topics have further levels of assistance, indicated by a “Subtopic?” prompt.
- To move back to the next higher level, press the RETURN key. For example, pressing RETURN at the “Subtopic?” prompt brings you to the “Topic?” prompt, and pressing RETURN again returns you to the RDO> prompt.
- To see the list of additional topics at any level, type a question mark (?) and press RETURN.
- To leave HELP, type CTRL/Z, or press RETURN at the “Topic?” prompt.

Most HELP entries in RDO have a similar structure. The main screen gives a brief description of the topic, followed by a simple example. In many cases, this screen gives you all the information you need to execute the statement.

The main screen also displays a list of “Additional information available.” This list usually includes three entries:

More

More details on the topic

Format

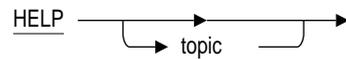
The complete syntax for the statement

Examples

Complete, annotated examples

HELP Statement

Format



Argument

topic

The Rdb/VMS statement or concept on which you need help.

Usage Note

You can use the HELP statement without invoking a database, so you do not need any special Rdb/VMS privileges to use the statement.

Examples

Example 1

The following statement accesses online HELP for the RDO GET statement:

```
RDO> HELP GET
```

9.50 IMPORT Statement

Restructures an interchange (RBR) file to a database (RDB) file according to the parameters you specify. The RDO IMPORT statement replaces the RDO RESTORE statement.

You can use the EXPORT and IMPORT statements to:

- Restructure an Rdb/VMS single-file database into a multifile database
- Restructure a multifile database
- Migrate an Rdb/VMS database to an Rdb/ELN database management system, or an Rdb/ELN database to an Rdb/VMS database management system
- Create a version-independent copy of the database for archiving purposes
- Create an empty target database that uses the same data definitions as a source database by copying the metadata, but not the data, to the target
- Change database and storage area characteristics that you cannot change with the CHANGE DATABASE statement

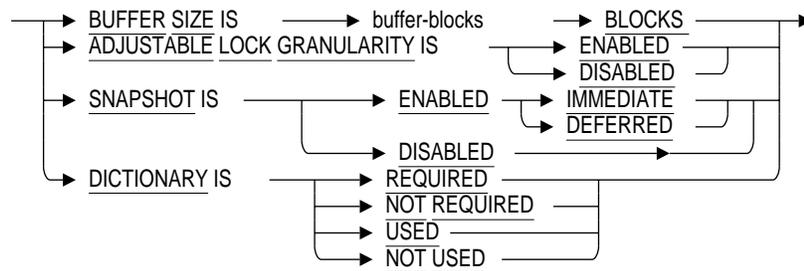
The IMPORT and EXPORT statements are not intended for regular backups of the database. For regular backups and restorations of Rdb/VMS databases, use the RMU/BACKUP and RMU/RESTORE commands.

An EXPORT statement translates the definitions and data in a database into an intermediate form in a special type of Record Management System (RMS) sequential file. The IMPORT statement reads the records in this file and uses them to create an Rdb/VMS database that will usually be identical to the one that the EXPORT statement used. The Rdb/VMS database will not be identical to the one that the EXPORT statement used if you issued the EXPORT statement with the NOEXTENSIONS clause or if you are importing the database into another version of Rdb/VMS. If the data dictionary is installed on your system and you do not specify the DICTIONARY IS NOT USED clause, the IMPORT statement also creates a data dictionary entity and copies the database definitions to it.

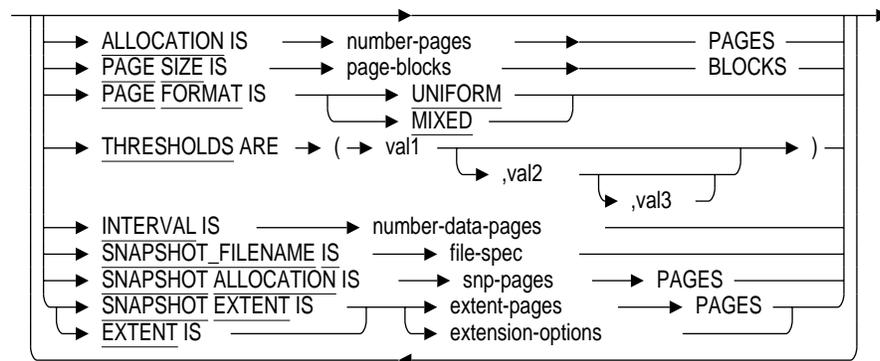
If you use the NODATA option, the IMPORT statement creates an Rdb/VMS database whose *metadata* is identical to that found in the source database used by the EXPORT statement, but the duplicate database contains no data. The NODATA option is not compatible with CDD/Plus dictionary databases. See the description under the NODATA option.

IMPORT Statement

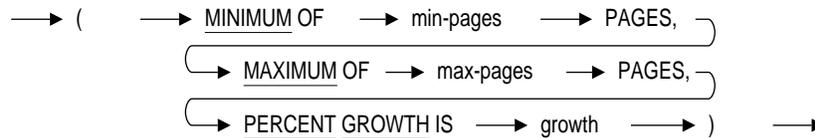
db-wide-options-2 =



storage-area-options =

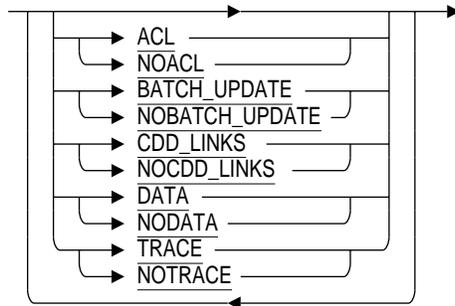


extension-options =

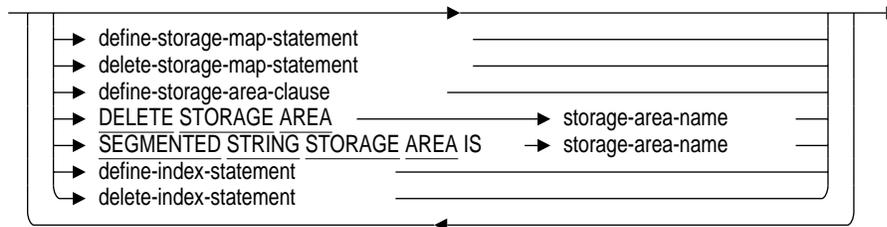


IMPORT Statement

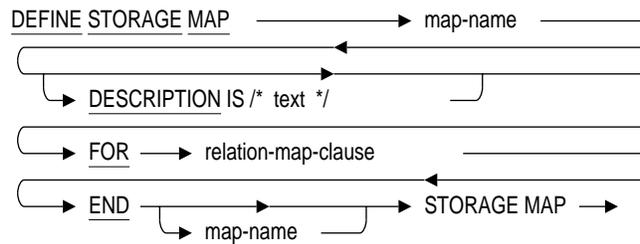
import-options =



metadata-options =

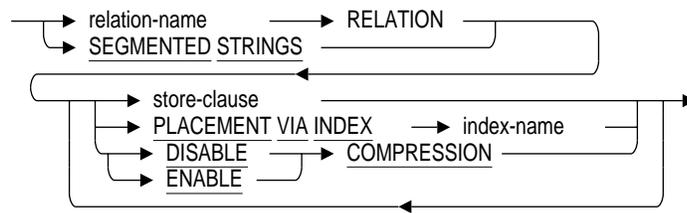


define-storage-map-statement =

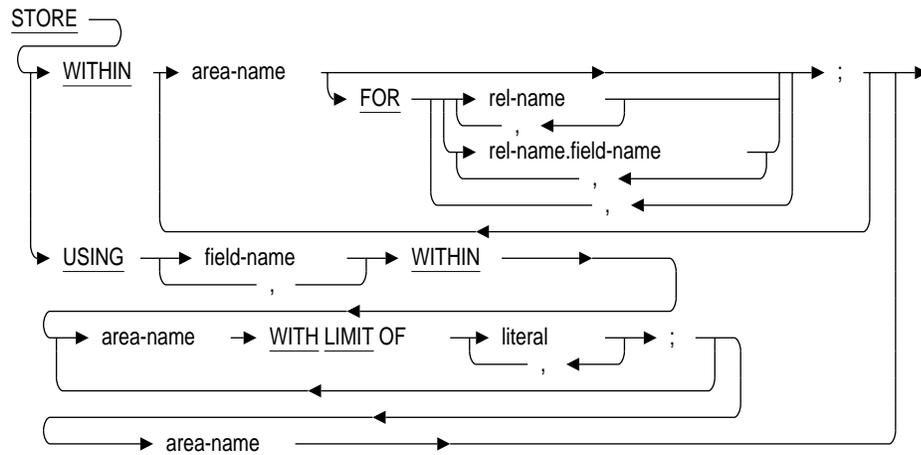


IMPORT Statement

relation-map-clause =

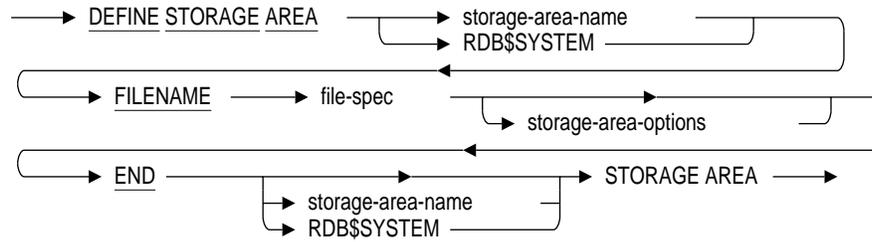


store-clause =



IMPORT Statement

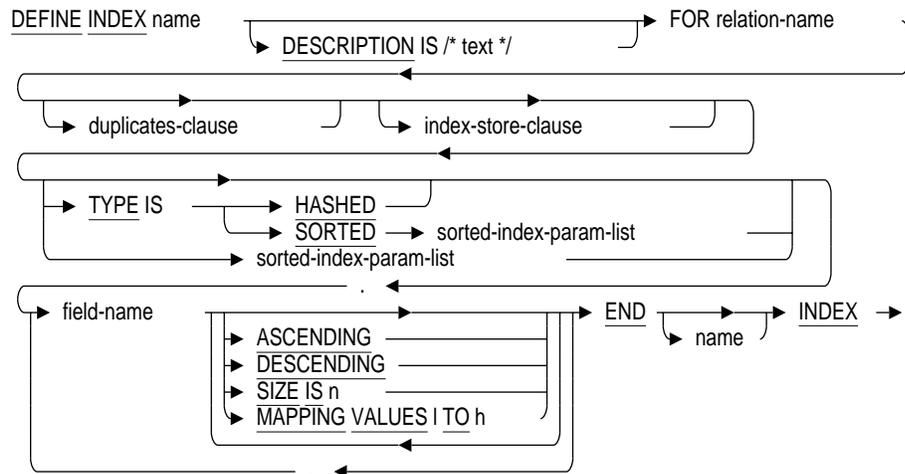
define-storage-area-clause =



delete-storage-map-statement =



define-index-statement =

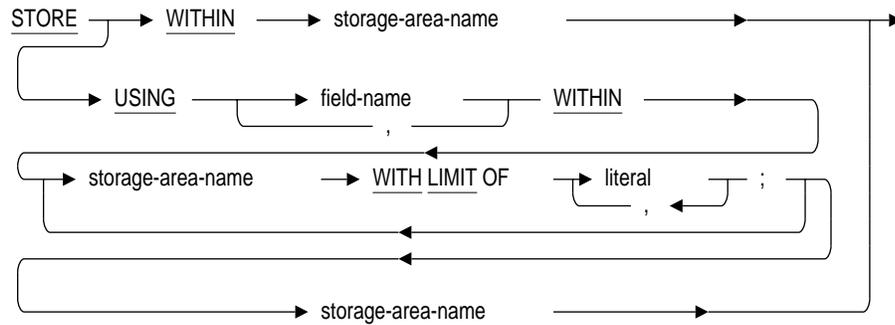


IMPORT Statement

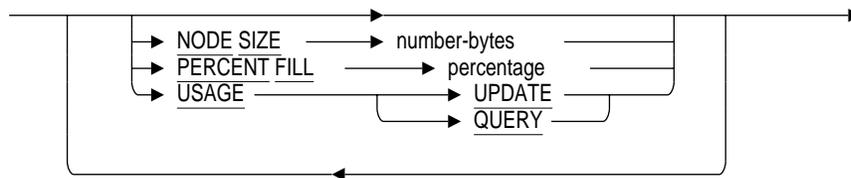
duplicates-clause =



index-store-clause



sorted-index-param-list =



delete-index-statement =



IMPORT Statement

Arguments

interchange-file-spec

The name of the interchange file that the IMPORT statement uses as a source to create the new database. Use either a full or partial file specification or a logical name. If you use a simple file name, Rdb/VMS looks for the interchange file in the current default directory. You must specify a file that was created by the Rdb/VMS EXPORT statement or by the Rdb/ELN Backup and Restore Utility (EBRP). The default file type is RBR.

database-file-spec

The name of the Rdb/VMS database file you want to create from the interchange file. Use either a full or partial file specification or a logical name. If you use a simple file name, Rdb/VMS creates the database in the current default directory. The default file type is RDB.

DB_HANDLE IS db-handle

A host language variable or name that you associate with the database. Use a database handle when you access more than one database at a time.

DBKEY SCOPE IS COMMIT

During the session of the user who entered the IMPORT statement, specifies that the dbkey of each record used is guaranteed not to change only during each transaction this user may execute.

DBKEY SCOPE IS FINISH

During the session of the user who entered the IMPORT statement, specifies that the dbkey of each record used is guaranteed not to change until this user ends the RDO session or executes a FINISH statement.

db-wide-options

Characteristics of the database root file, in a single-file or multfile database. The exception to this is the SNAPSHOT IS ENABLED/DISABLED option. In a multfile database, the SNAPSHOT option applies to all snapshot files associated with the storage area files. Thus, when you enable snapshots, you do so for the database root file and for *all* storage area files.

Specifying database-wide options is a way of overriding the Rdb/VMS system defaults for a database.

IMPORT Statement

IN path-name

The data dictionary path name for the dictionary directory where the database definition is stored. Use this qualifier to store the dictionary definitions for the database in a dictionary directory other than the default directory.

Specify either:

- A full dictionary path name such as DISK1:[DICTIONARY]CORP.EMPS
- A relative dictionary path name such as EMPS

If you use a relative path name, CDD\$DEFAULT must be defined to include all the path name segments that precede the relative path name.

COLLATING_SEQUENCE IS sequence-name

Specifies a collating sequence to be used for all fields in the database. Sequence-name is a name of your choosing; use this sequence-name in any subsequent statements that refer to this collating sequence.

The VMS NCS utility provides a set of predefined collating sequences and also lets you define collating sequences of your own. The COLLATING_SEQUENCE clause accepts both predefined and user-defined NCS collating sequences.

If you do not specify a collating sequence, the default is ASCII (shown as “no collating sequence” in some displays).

DESCRIPTION IS /* text */

Provides a description for the collating sequence.

ncs-name

Specifies the name of a collating sequence in the default NCS library, SYSS\$LIBRARY:NCS\$LIBRARY, or in the NCS library specified by the argument library-name. (In most cases, it is probably simplest to make the sequence-name the same as the ncs-name: for example, COLLATING_SEQUENCE IS FRENCH FRENCH.) You can view the collating sequence names by using the command NCS/LIST at DCL level.

The collating sequence can be either one of the predefined NCS collating sequences or one that you have defined yourself using NCS.

FROM library-name

Specifies the name of an NCS library other than the default. The default NCS library is SYSS\$LIBRARY:NCS\$LIBRARY.

IMPORT Statement

DESCRIPTION IS /* text */

A text string that adds a comment to the imported database.

NUMBER USERS IS number-users

The maximum number of users allowed to access the database at one time. The default is 50 users. After the maximum is reached, the next user who tries to invoke the database receives an error message and must wait. The largest number of users you can specify is 2032 and the smallest number is 1. With VAX ACMS, each attached server can support multiple users. Thus, the actual number of terminal users can be greater than 2032 if you are using VAX ACMS.

Note that the number of users is defined as the number of active attaches to the database. Thus, if a single process is running one program that performs 12 INVOKE . . . FINISH operations, to Rdb/VMS there are 12 active users.

NUMBER BUFFERS IS number-buffers

The number of buffers Rdb/VMS allocates per process using this database. Specify an unsigned integer between 2 and 32768. The default is 20 buffers.

NUMBER VAXCLUSTER NODES IS number-nodes

Sets the upper limit on the maximum number of VAXcluster nodes from which users can access the shared database. The default is 16 nodes. The range is 1 node to 64 nodes. The actual maximum limit is the current VMS VAXcluster limit.

NUMBER RECOVERY BUFFERS IS recovery-buffers

The number of database buffers used during the automatic recovery process that is initiated after a system or process failure. This recovery process uses the recovery-unit journal (RUJ) file. Specify an unsigned integer between 2 and 32768. The default is 20 buffers.

BUFFER SIZE IS buffer-blocks

The number of blocks per buffer. Buffer size is allocated in 512-byte blocks. Specify an unsigned integer greater than zero. If you do not specify this parameter, Rdb/VMS uses a buffer size that is three times the PAGE SIZE value. Since at least one page must fit in one buffer, the buffer size must be equal to or greater than the page size.

ADJUSTABLE LOCK GRANULARITY IS ENABLED [Default]

ADJUSTABLE LOCK GRANULARITY IS DISABLED

Enables or disables adjustable locking granularity. Generally, enabling adjustable locking granularity results in fewer locks being used. However,

IMPORT Statement

when contention for database pages is high, performance may be impaired as locking granularity is adjusted to a lower level. If your application is query intensive, enable adjustable locking granularity. If your application processes specific records and performs a substantial number of update operations, you might want to disable adjustable locking granularity.

Disabling adjustable locking granularity may require that the VMS SYSGEN parameters for locks be increased.

For more information, see the *VAX Rdb/VMS Guide to Database Maintenance and Performance*.

SNAPSHOT IS ENABLED IMMEDIATE [Default]

SNAPSHOT IS ENABLED DEFERRED

The SNAPSHOT IS ENABLED IMMEDIATE option specifies that read/write transactions write copies of records to the snapshot file before those records are modified, regardless of whether a read-only transaction is active.

The SNAPSHOT IS ENABLED DEFERRED option specifies that read/write transactions *not* write copies of records they modify to the snapshot file unless a read-only transaction is active. Read-only transactions that attempt to start after an active read/write transaction begins must wait for all active read/write users to complete their transactions.

You enable snapshot writing to all snapshot files for all storage areas when you specify the SNAPSHOT IS ENABLED clause.

SNAPSHOT IS DISABLED

Disables snapshot writing. You disable snapshot writing to all snapshot files for all storage areas when you specify the SNAPSHOT IS DISABLED clause.

Do not delete snapshot files unless you also delete the database itself using the RDO DELETE DATABASE command. Do not use the VMS DELETE command. If you do not want snapshots enabled, disable them with the SNAPSHOT IS DISABLED clause.

DICTIONARY IS REQUIRED

DICTIONARY IS NOT REQUIRED [Default]

Determines whether the database must be invoked by path name for data definition changes to occur. If you specify the DICTIONARY IS REQUIRED option, the database must be invoked by path name to change metadata and the data dictionary will be maintained. If you specify the DICTIONARY IS NOT REQUIRED option, the database can be invoked by either file name or path name to change metadata.

IMPORT Statement

DICTIONARY IS USED [Default]

DICTIONARY IS NOT USED

Determines whether the definition of the database and definitions of database elements will be stored in the data dictionary. If you specify the **DICTIONARY IS USED** option, the definition of the database and definitions of database elements will be stored in the data dictionary. If you specify the **DICTIONARY IS NOT USED** option, no definitions will be stored in the data dictionary.

You receive an error message if you specify incompatible options, such as the **DICTIONARY IS REQUIRED** clause and the **DICTIONARY IS NOT USED** clause.

storage-area-options

You can specify most storage area options for either single-file or multifile databases. However, the effects of these options differ, depending on whether the database is single file or multifile:

- In a multifile database, storage area options apply to storage areas.
- In a single-file database, storage area options apply to the database root file.

In a multifile database, if you specify a storage area option outside a **DEFINE STORAGE AREA** clause, the option serves as a global default for all storage areas in the database. The exception to this is if you specify the **SNAPSHOT_FILENAME** option outside the **DEFINE STORAGE AREA** clause, the snapshot file name applies only to the snapshot file associated with the database root file; not to all snapshot files.

Note that certain storage area options apply only to multifile databases. You cannot specify these options for a single-file database.

ALLOCATION IS number-pages

The number of database pages allocated to the database initially. Rdb/VMS automatically extends the allocation to handle the loading of data and subsequent expansion. The default is 400 pages. If you are loading a large database, a large allocation prevents the file from having to be extended many times.

Rdb/VMS allocates pages in groups of three, so if you specify that two pages be allocated for a storage area, for example, Rdb/VMS allocates three pages instead. Rdb/VMS also allocates a space area management (SPAM) entry page initially for the storage area. Therefore, because Rdb/VMS allocates pages in groups of three and also allocates a SPAM page, a total of four pages are

IMPORT Statement

allocated for the storage area even though you requested an allocation of only two pages.

PAGE SIZE IS page-blocks

The size in blocks of each database page. Page size is allocated in 512-byte blocks. The default is 2 blocks (1024 bytes). If your largest record is larger than approximately 950 bytes, allocate more blocks per page to prevent records from being fragmented.

The page size you specify must be smaller than the buffer size specified for the database or you will receive an error message.

PAGE FORMAT IS UNIFORM [Default]

PAGE FORMAT IS MIXED

Specifies whether a storage area contains UNIFORM or MIXED pages. You can use the PAGE FORMAT option with multifile databases only. In storage areas with UNIFORM page format, all pages in a specific logical area contain records from the same relation. In storage areas with MIXED page format, pages can hold records from different relations.

The default storage area, RDB\$SYSTEM, must have UNIFORM pages.

THRESHOLDS ARE (val1, val2, val3)

Specifies one, two, or three threshold values. The threshold values represent a fullness percentage on a data page and establish four possible ranges of guaranteed free space on the data pages. When a data page reaches the percentage defined by a given threshold value, the space area management (SPAM) entry for the data page is updated to reflect the new fullness percentage and its remaining free space.

The default thresholds are 70, 85, and 95 percent. If you specify only one or two values, unspecified values default to 100 percent. You can specify the THRESHOLDS option only for a storage area for a multifile database. The storage area page format must be MIXED.

INTERVAL IS number-data-pages

The number of data pages between SPAM pages in the physical storage area file, and thus the maximum number of data pages each SPAM page will manage. The default, and also the minimum interval, is 256 data pages. The first page of each storage area is a SPAM page. The interval you specify determines where subsequent SPAM pages are to be inserted, provided there are enough data pages in the storage file to require more SPAM pages.

IMPORT Statement

The maximum SPAM interval you can specify is 4008 data pages for a database with a 2-block page size. Specifying a value greater than 4008 data pages for the SPAM interval causes data corruption.

You can specify the INTERVAL option only for a storage area for a multifile database. The storage area page format must be MIXED.

See the *VAX Rdb/VMS Guide to Database Maintenance and Performance* for information on formulas for calculating maximum and minimum SPAM intervals.

SNAPSHOT_FILENAME IS file-spec

Provides a separate file specification for the snapshot file. Do not specify a file type other than SNP. If you do not specify this clause for a storage area, the SNP file is automatically placed in the same directory as the storage area file and has the same name as the storage area file. You cannot specify a global default for the snapshot filename. Thus, in a multifile database, the SNAPSHOT_FILENAME option must be within a DEFINE STORAGE AREA clause.

SNAPSHOT ALLOCATION IS snp-pages

The number of pages allocated for the snapshot file. The default is 100 pages.

EXTENT IS extent-pages

SNAPSHOT EXTENT IS extent-pages

The number of pages of each extent. Use this parameter for simple control over the page extents. For greater control, and particularly for multivolume databases, use the MIN, MAX, and PERCENT parameters instead. The default is 100 pages.

extension-options

Specifies the MIN, MAX, and percent growth of each database file extent. Enclose the parameter list in parentheses.

min-pages

The minimum number of pages of each extent. The default is 100 pages.

max-pages

The maximum number of pages of each extent. The default is 10,000 pages.

growth

The percent growth of each extent. The default is 20 percent growth.

IMPORT Statement

ACL [Default]

NOACL

Determines whether the previously defined access control lists are used in the imported version of the database (ACL), or whether the access control lists from the interchange file are not used (NOACL). The NOACL option overrides the access control lists specified in the original database and uses the system default access control lists. The NOACL option makes you the owner of the imported database. Use the NOACL option if the access control lists from the original database would prevent you from accessing the imported database.

BATCH_UPDATE [Default]

NOBATCH_UPDATE

Determines whether the IMPORT operation runs in batch-update mode (BATCH_UPDATE) or in exclusive share mode (NOBATCH_UPDATE). No recovery-unit journaling is performed in batch-update mode. The IMPORT operation runs faster if you specify the BATCH_UPDATE option instead of the NOBATCH_UPDATE option. However, with the NOBATCH_UPDATE option, you can recover your database in the event of a failure during the IMPORT operation.

If you use the NOBATCH_UPDATE option, the IMPORT operation executes in several transactions. Thus, a partial restoration of the database can occur. The IMPORT statement uses exclusive transactions to store data and create indexes. The first exclusive transaction defines the relation and its fields. The second transaction stores user data in the relation. The third transaction redefines indexes. A fourth, optional transaction executes if an ACL for the relation must be defined. When you choose to import the database in nobatch-update mode, you can benefit from using the recovery-unit journal file if an error occurs.

CDD_LINKS [Default]

NOCDD_LINKS

Determines whether the dictionary relationships are preserved in the imported database. If you specify CDD_LINKS, the IMPORT operation attempts to reconnect the fields and records to the dictionary path name entities to which they refer. If the dictionary entity no longer exists, you will receive a warning message. If you specify NOCDD_LINKS, the IMPORT operation runs without preserving dictionary relationships in the imported database. Fields and records are not reconnected to the dictionary path name entities to which they refer.

You receive an error message if you specify the CDD_LINKS clause and the DICTONARY IS NOT USED clause.

IMPORT Statement

DATA [Default]

NODATA

Specifies whether the RDB file created by the IMPORT statement includes the data and metadata contained in the database, or the metadata only. DATA is the default.

When you specify the NODATA option, you import the metadata that defines a database from an RBR file and exclude the data. Duplicating the metadata of a database while excluding the data offers the following benefits:

- You can use established, tested metadata to create a database to store new data. Standardized metadata can be created once but used in multiple databases.
- You can use the duplicated metadata to test the database structure. You can experiment with storage areas and storage maps, and by entering sample data, you can test other aspects of database structure.
- If a database needs testing by someone outside of your group, you can submit the database structure without exposing any sensitive data. Also, if the database is very large, you need not submit multiple reels of tape to the tester.

Note *The NODATA option is not compatible with CDD/Plus dictionary databases (CDD\$DATABASE.RDB). An RBR file, created by an EXPORT statement with the DATA option (the default) and generated from a CDD\$DATABASE.RDB file, cannot be used with the NODATA option to the IMPORT statement. RDO will issue an error message stating that the NODATA option is not valid for CDD/Plus dictionary databases.*

TRACE

NOTRACE

Specifies whether usage statistics are logged by IMPORT. NOTRACE is the default.

Some actions taken by the IMPORT statement can consume significant amounts of I/O, and CPU time. These actions include the following operations:

- Loading data
- Defining indexes
- Defining constraints

IMPORT Statement

When you specify the TRACE option with the IMPORT statement, RDO writes a message to your terminal screen when each operation begins, and writes a summary of DIO, CPU, and PAGE FAULT statistics when the operation completes. When the IMPORT statement finishes execution, a summary of all DIO, CPU, and PAGE FAULT statistics is displayed. The display also includes information on access to the RBR file, database creation, and loading of data.

metadata-options

Lets you define or delete storage maps, storage areas, and indexes.

define-storage-map-statement

See the Arguments section in Section 9.20 for a description of arguments for this statement. This statement defines a new storage map, or replaces the definition of an existing storage map contained in the interchange file.

Note The **DEFINE STORAGE MAP** statement is not terminated with a period when it is included within the **IMPORT** statement.

delete-storage-map-statement

See the Arguments section in Section 9.33 for a description of arguments for this statement. This statement prevents use of a storage map definition from the interchange file. The relation that would have used the storage map definition is placed in the system default storage area.

Note The **DELETE STORAGE MAP** statement is not terminated with a period when it is included within the **IMPORT** statement.

define-storage-area-clause

See the Arguments section in Section 9.14 for a description of arguments for this clause. This clause adds a new storage area or replaces the definition of an existing storage area contained in the interchange file.

DELETE STORAGE AREA storage-area-name

Prevents a storage area that was defined in the interchange file from being created in the new database. You must also delete or redefine all storage maps and indexes that refer to this storage area. You cannot delete the RDB\$SYSTEM storage area.

SEGMENTED STRING STORAGE AREA

The name of the storage area that will hold all segmented strings. In a multifile database, if you do not define a storage area for segmented strings, they will be stored in the default storage area, RDB\$SYSTEM.

IMPORT Statement

The page format for the segmented string storage area can be UNIFORM or MIXED. However, Digital Equipment Corporation recommends that if you store segmented strings in a MIXED storage area, the storage area contain *only* segmented strings.

define-index-statement

See the Arguments section in Section 9.16 for a description of arguments for this statement. This statement defines a new index or replaces an existing index definition contained in the interchange file.

Note The **DEFINE INDEX** statement is not terminated with a period when it is included within the **IMPORT** statement.

delete-index-statement

See the Arguments section in Section 9.28 for a description of the argument for this statement. This statement prevents creation of an index defined in the interchange file.

Note The **DELETE INDEX** statement is not terminated with a period when it is included within the **IMPORT** statement.

Usage Notes

You must have the Rdb/VMS ADMINISTRATOR privilege to use the **IMPORT** statement.

The **IMPORT** statement creates a new database that inherits characteristics from the source database. Only the characteristics that you change will differ from the original database.

The order of precedence for storage area options is:

- 1 Local use of the option, within a **DEFINE STORAGE AREA** clause
- 2 Database-wide use of the option, outside a **DEFINE STORAGE AREA** clause, but within the **IMPORT** statement
- 3 The Rdb/VMS default for that option, if it is not specified locally or globally

Note that storage area options are *not* inherited as defaults from the exported database through the interchange file.

IMPORT Statement

There are two cases in which global defaults for multifile databases will be ignored. They are the following:

- If you specify a global default of MIXED page format for all storage areas defined in the IMPORT statement, Rdb/VMS does not apply this default to the RDB\$SYSTEM storage area. RDB\$SYSTEM will be created with UNIFORM page format.
- If you specify a global default for the INTERVAL or THRESHOLDS option, these defaults are simply ignored for storage areas that have a UNIFORM page format. The INTERVAL and THRESHOLDS options apply only to storage areas with a MIXED page format.

By default, the IMPORT statement:

- Creates the new database
- Performs the metadata operations in a read/write transaction
- Stores user data and indexes in a batch-update transaction
- Leaves you attached to the database, with a database handle equal to the name of the database

If the IMPORT operation fails, the database is closed and any associated data dictionary entity is deleted.

Use systemwide concealed logical names instead of physical disk device names for file specifications. This will save you work during IMPORT operations. When you use the IMPORT statement, Rdb/VMS tries to create storage areas with storage area and snapshot file specifications exactly as stored in the interchange file. If those file specifications include disk devices or directory names that do not exist, the IMPORT operation will fail unless you specify DEFINE STORAGE AREA clauses with new file specifications for all storage area and snapshot files in the database. You can avoid this problem if you use concealed logical names to refer to the disk device and directory names in file specifications. Before you issue the IMPORT statement, redefine the logical names to refer to the appropriate disk device and directory names where you plan to import the database. For more information on using concealed logical names, see the *VAX Rdb/VMS Guide to Database Maintenance and Performance*.

IMPORT Statement

DEFINE STORAGE AREA, DEFINE STORAGE MAP, and DEFINE INDEX statements within an IMPORT statement can refer to storage areas, storage maps, and indexes that exist in the original database. When they refer to existing elements, each statement replaces the existing definition in the interchange file with the new definition specified in the IMPORT statement.

If you use the IMPORT statement to restructure a CDD/Plus dictionary, be sure to use the DICTONARY IS NOT USED clause to prevent RDO from using the dictionary in any way:

```
RDO> IMPORT CDDPLUS.RBR INTO CDD$DATABASE
cont> DICTONARY IS NOT USED
.
.
.
cont> END IMPORT.
```

See the *VAX Rdb/VMS Guide to Database Maintenance and Performance* for a complete discussion of when to use the IMPORT, EXPORT, and CHANGE DATABASE statements.

Examples

Example 1

The following example imports a database. It also specifies the number of buffers and the length of each buffer in the imported database.

```
RDO> IMPORT
cont> 'DISK1:PERSONNEL.RBR'
cont> INTO 'DEPT3:NEW_PERSONNEL'
cont> NUMBER OF BUFFERS IS 10
cont> BUFFER SIZE IS 10 BLOCKS
cont> END IMPORT.
```

Example 2

The following example imports a database without including its data. Using the NODATA option with the IMPORT statement creates an Rdb/VMS database whose metadata is identical to the metadata in the intermediate RBR file.

```
RDO> IMPORT
cont> 'DISK1:PERSONNEL.RBR'
cont> INTO 'DEPT3:NEW_PERSONNEL'
cont> NODATA
cont> END IMPORT.
```

IMPORT Statement

Example 3

The following example imports a database from a magnetic tape.

```
$ MOUNT MUA0:
_Label: PERS
_Log name:
$ RDO
RDO> IMPORT
cont> 'MUA0:PERSONNEL' INTO
cont> 'DEPT3:PERSONNEL'
cont> END IMPORT.
```

This statement reads the export copy of the database from the magnetic tape volume labeled PERS, mounted on device MUA0:. It creates a new database in DEPT3:PERSONNEL.RDB, where DEPT3 is a logical name for a device and directory.

Example 4

The following example uses the RMU/DUMP command to check the current value of the node count, and then uses the IMPORT statement with the NUMBER VAXCLUSTER NODES clause to lower the node count parameter. In this case, the Local Area VAXcluster contains 22 nodes; the default maximum for the VAXCLUSTER NODES parameter is 16 nodes. This example sets the upper limit of user access from VAX nodes at 20.

```
$ RMU/DUMP ACTING
.
.
.
Maximum node count is 22
.
.
.
$ RDO
RDO> EXPORT ACTING.RDB INTO ACCOUNTING_TEST.RBR
RDO> IMPORT ACCOUNTING_TEST.RBR INTO ACCOUNTING_TEST.RDB
cont> NUMBER OF VAXCLUSTER NODES IS 20
cont> END IMPORT.
```

Example 5

The following example uses the IMPORT statement to restructure a single-file database into a multifile database. This example does not include all the storage areas and storage maps that are in the sample multifile MF_PERSONNEL database.

IMPORT Statement

```
$ RDO
!
! Invoke a command file that has the IMPORT statement in it
!
RDO> @IMPORT_PERS.RDO
!
!
IMPORT 'DISK$BACK:PERSONNEL.RBR' INTO 'DB_DISK:MULTI_PERSONNEL.RDB'
!
! Specify database-wide characteristics
!
    NUMBER OF USERS IS 40
    NUMBER VAXCLUSTER NODES 12
    SNAPSHOT IS ENABLED IMMEDIATE
    DICTIONARY IS REQUIRED
    NUMBER RECOVERY BUFFERS 200
!
!
! Specify global defaults to override Rdb/VMS defaults
! for all storage areas
!
    ALLOCATION IS 500 PAGES
    PAGE FORMAT IS MIXED
    THRESHOLDS ARE (55,65,75)
    INTERVAL IS 300
!
!
! Define the default storage area
!
DEFINE STORAGE AREA RDB$SYSTEM
    FILENAME DISK1:PERS_DEFAULT
    PAGE FORMAT IS UNIFORM
    SNAPSHOT_FILENAME IS DISK2:PERS_DEFAULT
END RDB$SYSTEM STORAGE AREA
!
!
! Define storage area for segmented strings
!
DEFINE STORAGE AREA PERS_SEGSTR
    FILENAME DISK1:PERS_SEGSTR
    PAGE FORMAT IS MIXED
    SNAPSHOT_FILENAME IS DISK2:PERS_SEGSTR
END PERS_SEGSTR STORAGE AREA
!
! Put the segmented strings in the named storage area
!
SEGMENTED STRING STORAGE AREA IS PERS_SEGSTR
!
```

IMPORT Statement

```
!  
! Define other storage areas  
!  
! Storage area parameters specified within the DEFINE STORAGE AREA  
! clause override the global defaults and the Rdb/VMS defaults  
!  
DEFINE STORAGE AREA EMPIDS_LOW  
  FILENAME DISK3:EMPIDS_LOW  
  SNAPSHOT_FILENAME IS DISK4:EMPIDS_LOW  
END EMPIDS_LOW STORAGE AREA  
!  
!  
DEFINE STORAGE AREA EMPIDS_MID  
  FILENAME DISK5:EMPIDS_MID  
  SNAPSHOT_FILENAME IS DISK6:EMPIDS_MID  
END EMPIDS_MID STORAGE AREA  
!  
!  
DEFINE STORAGE AREA EMPIDS_OVER  
  FILENAME DISK7:EMPIDS_OVER  
  SNAPSHOT_FILENAME IS DISK8:EMPIDS_OVER  
END EMPIDS_OVER STORAGE AREA  
!  
!  
DEFINE STORAGE AREA EMP_INFO  
  FILENAME DISK9:EMP_INFO  
  ! Local definition overrides the global default  
  THRESHOLDS ARE (65,75,85)  
  INTERVAL IS 400  
  SNAPSHOT_FILENAME IS DISK10:EMP_INFO  
END EMP_INFO STORAGE AREA  
!  
!  
!  
DEFINE INDEX EMPLOYEES_HASH  
  DESCRIPTION IS /* hashed index for employees relation */  
  FOR EMPLOYEES  
  STORE USING EMPLOYEE_ID  
  WITHIN  
    EMPIDS_LOW WITH LIMIT OF "00200";  
    EMPIDS_MID WITH LIMIT OF "00500";  
    EMPIDS_OVER  
  TYPE IS HASHED.  
  EMPLOYEE_ID.  
END EMPLOYEES_HASH INDEX  
!
```

IMPORT Statement

```
!  
DEFINE STORAGE MAP EMP_MAP  
  DESCRIPTION IS /* Employees records partitioned by EMPLOYEE_ID */  
  FOR EMPLOYEES RELATION  
  STORE USING EMPLOYEE_ID  
  WITHIN  
    EMPIDS_LOW WITH LIMIT OF "00200";  
    EMPIDS_MID WITH LIMIT OF "00500";  
    EMPIDS_OVER  
  PLACEMENT VIA INDEX EMPLOYEES_HASH  
END EMP_MAP STORAGE MAP  
!  
!  
! End the IMPORT statement  
!  
END IMPORT.
```

Example 6

The following example shows how to use the **DICTIONARY IS NOT USED** clause to prevent metadata from being written to the data dictionary. The **IMPORT** statement creates a database called **RALLY\$COMMERCE**. Because the **DICTIONARY IS NOT USED** clause is specified, no metadata is written to the data dictionary during the **IMPORT** operation.

```
RDO> IMPORT RALLY$COMMERCE.RBR INTO RALLY$COMMERCE  
cont>     NOCDD LINKS  
cont>     NOACL  
cont>     DICTIONARY IS NOT REQUIRED  
cont>     DICTIONARY IS NOT USED  
cont> END IMPORT.
```

Example 7

This example imports a database and uses the **TRACE** option to display **DIO**, **CPU**, and **PAGE FAULT** statistics.

```
RDO> !  
RDO> ! First export the PERSONNEL database  
RDO> !  
RDO> EXPORT PERSONNEL INTO LOCAL_PERSONNEL  
RDO>  
RDO> !  
RDO> ! Now IMPORT a new version with TRACE turned on  
RDO> !
```

IMPORT Statement

```
RDO> IMPORT LOCAL_PERSONNEL
cont>     INTO PERSONNEL_T
cont>     DICTIONARY IS NOT USED
cont>     TRACE
cont> END IMPORT.
Exported by Rdb/VMS V4.0-0 Import/Export utility
A component of Rdb/VMS V4.0-0
Previous name was PERSONNEL
It was logically exported on 2-APR-1990 16:23
Database NUMBER OF USERS is 50
Database NUMBER OF VAXCLUSTER NODES is 16
Database NUMBER OF DBR BUFFERS is 20
Database SNAPSHOT is ENABLED
Database SNAPSHOT is IMMEDIATE
Database BUFFER SIZE is 6 blocks
Database NUMBER OF BUFFERS is 20
IMPORTing STORAGE AREA: RDB$SYSTEM
IMPORTing relation CANDIDATES
Completed CANDIDATES. DIO = 79, CPU = 0:00:00.46, FAULTS = 79
IMPORTing relation COLLEGES
Completed COLLEGES. DIO = 74, CPU = 0:00:00.41, FAULTS = 1
Starting INDEX definition COLL_COLLEGE_CODE
Completed COLL_COLLEGE_CODE. DIO = 25, CPU = 0:00:00.10, FAULTS = 2
IMPORTing relation DEGREES
Completed DEGREES. DIO = 84, CPU = 0:00:00.63, FAULTS = 3
Starting INDEX definition DEG_COLLEGE_CODE
Completed DEG_COLLEGE_CODE. DIO = 29, CPU = 0:00:00.19, FAULTS = 0
Starting INDEX definition DEG_EMP_ID
Completed DEG_EMP_ID. DIO = 24, CPU = 0:00:00.17, FAULTS = 0
IMPORTing relation DEPARTMENTS
Completed DEPARTMENTS. DIO = 68, CPU = 0:00:00.33, FAULTS = 0
IMPORTing relation EMPLOYEES
Completed EMPLOYEES. DIO = 98, CPU = 0:00:00.66, FAULTS = 0
Starting INDEX definition EMP_EMPLOYEE_ID
Completed EMP_EMPLOYEE_ID. DIO = 34, CPU = 0:00:00.25, FAULTS = 1
IMPORTing relation JOBS
Completed JOBS. DIO = 70, CPU = 0:00:00.33, FAULTS = 0
IMPORTing relation JOB_HISTORY
Completed JOB_HISTORY. DIO = 97, CPU = 0:00:00.90, FAULTS = 0
Starting INDEX definition JH_EMPLOYEE_ID
Completed JH_EMPLOYEE_ID. DIO = 49, CPU = 0:00:00.37, FAULTS = 4
IMPORTing relation RESUMES
Completed RESUMES. DIO = 51, CPU = 0:00:00.23, FAULTS = 1
IMPORTing relation SALARY_HISTORY
Completed SALARY_HISTORY. DIO = 125, CPU = 0:00:01.59, FAULTS = 70
Starting INDEX definition SH_EMPLOYEE_ID
Completed SH_EMPLOYEE_ID. DIO = 50, CPU = 0:00:00.51, FAULTS = 94
IMPORTing relation WORK_STATUS
Completed WORK_STATUS. DIO = 70, CPU = 0:00:00.39, FAULTS = 29
IMPORTing view CURRENT_SALARY
IMPORTing view CURRENT_JOB
IMPORTing view CURRENT_INFO
```

IMPORT Statement

```
Starting CONSTRAINT definition COLLEGE_CODE_REQUIRED
Completed COLLEGE_CODE_REQUIRED. DIO = 40, CPU = 0:00:00.19, FAULTS = 2
Starting CONSTRAINT definition DEPT_CODE_REQUIRED
Completed DEPT_CODE_REQUIRED. DIO = 23, CPU = 0:00:00.06, FAULTS = 0
Starting CONSTRAINT definition EMPLOYEE_ID_REQUIRED
Completed EMPLOYEE_ID_REQUIRED. DIO = 18, CPU = 0:00:00.04, FAULTS = 1
Starting CONSTRAINT definition JH_EMP_ID_EXISTS
Completed JH_EMP_ID_EXISTS. DIO = 27, CPU = 0:00:00.17, FAULTS = 4
Starting CONSTRAINT definition JOB_CODE_REQUIRED
Completed JOB_CODE_REQUIRED. DIO = 19, CPU = 0:00:00.09, FAULTS = 0
Starting CONSTRAINT definition SH_EMP_ID_EXISTS
Completed SH_EMP_ID_EXISTS. DIO = 27, CPU = 0:00:00.23, FAULTS = 0
Completed IMPORT. DIO = 2637, CPU = 0:00:16.30, FAULTS = 1011
RDO>
```

INTEGRATE DATABASE Statement

9.51 INTEGRATE DATABASE Statement

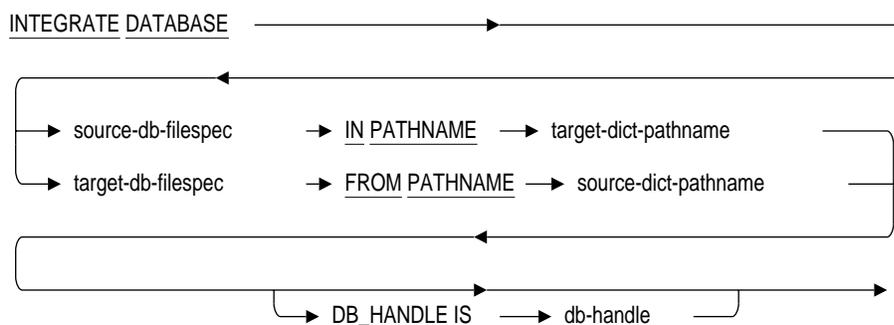
Moves data definitions between an Rdb/VMS database and the data dictionary. You can use the INTEGRATE DATABASE statement in either of these ways:

- To copy the data definitions from an existing database into a data dictionary entity (INTEGRATE DATABASE IN PATHNAME)
- To copy the data definitions from a CDD\$DATABASE data dictionary entity to a database (INTEGRATE DATABASE FROM PATHNAME)

If the data dictionary was not installed when a particular database was created, you can use the INTEGRATE statement to build the dictionary definitions from the database.

You can also periodically use the INTEGRATE statement to ensure that the database and data dictionary definitions remain synchronized with each other.

Format



Arguments

source-db-filespec

A full or partial VMS file specification, enclosed in quotation marks, that specifies the source of the database definitions. The source-db-filespec must be equivalent to the database file name that is recorded in the data dictionary entity for an existing data dictionary entity.

INTEGRATE DATABASE Statement

target-dict-pathname

The data dictionary path name for the dictionary entity in which the database definition will be created again. You can specify either a full data dictionary path name or a relative data dictionary path name. Enclose the data dictionary path name in quotation marks.

target-db-filespec

A full or partial VMS file specification, enclosed in quotation marks, that specifies where the data dictionary definition will be created again. The *target-db-filespec* must be equivalent to the database file name that is recorded in the data dictionary entity for a database that has existing data definitions.

source-dict-pathname

The dictionary path name for the CDD\$DATABASE dictionary entity from which the database definition will be copied. You can specify a full dictionary path name or a relative path name. Enclose the dictionary path name in quotation marks.

db-handle

A name that you assign to the database.

Usage Notes

You must have the Rdb/VMS READ privilege for a database to use the INTEGRATE DATABASE statement.

You must have UPDATE privilege to the dictionary to issue the IN PATHNAME option of the INTEGRATE DATABASE statement.

The INTEGRATE DATABASE statement invokes the database and, if needed, creates the specified data dictionary entity. When you use the IN PATHNAME option, if the data dictionary entity already exists, it will be updated to reflect the data definitions from the database. When you use the FROM PATHNAME option, if the data dictionary entity does not exist, Rdb/VMS returns an error.

You cannot issue the INTEGRATE DATABASE statement if there is an active transaction. The INTEGRATE DATABASE statement implicitly starts a read/write transaction for an Rdb/VMS database, and a read-only transaction for a VIDA database.

If the INTEGRATE DATABASE statement successfully completes, the database remains invoked, and the transaction is still active. You must explicitly complete the transaction with a COMMIT or ROLLBACK statement. If you issue the ROLLBACK statement, the data dictionary entity is retained.

INTEGRATE DATABASE Statement

If the INTEGRATE DATABASE statement fails, Rdb/VMS rolls back the transaction, closes the database, and deletes any data dictionary entity that was created.

If you use the FROM PATHNAME option to copy the data definitions from a dictionary entity into a database, data, as well as metadata, may be deleted from the database. If certain fields or relations are not defined in the dictionary, when the metadata from the database is replaced by the dictionary metadata, those fields or relations will be deleted from the database. Rdb/VMS signals an error if the integrate operation will delete metadata from the database. At this point, you can use the ROLLBACK statement to cancel the integrate operation.

If a database definition was created from the data dictionary, the data definition will contain information that refers to the data dictionary definition. If you use the IN PATHNAME option and the original data dictionary definition is missing, the database definition will be modified to eliminate the reference to the data dictionary.

Although you can issue the INTEGRATE DATABASE statement when there are active users updating the database, this is not recommended. Use the RMU/DUMP/USERS command to make sure that no other users are updating the database definitions before you issue the INTEGRATE DATABASE statement.

Examples

Example 1

The following statement creates the data dictionary definitions for the PERSONNEL database in 'DISK1:[DICTIONARY]CORP.PERSONNEL':

```
RDO> INTEGRATE DATABASE
cont> 'DISK2:[DEPT3]PERSONNEL'
cont> IN PATHNAME 'DISK1:[DICTIONARY]CORP.PERSONNEL'
RDO> COMMIT
```

Example 2

This example shows how to specify a database handle for a database. Because one database is already invoked, you must specify a database handle with the INTEGRATE DATABASE statement.

INTEGRATE DATABASE Statement

```
RDO> SHOW DATABASES
Database with filename personnel
RDO> INTEGRATE DATABASE 'DISK2:[DEPT3]ACCT'
cont> IN PATHNAME
cont> 'DISK1:[DICTIONARY]CORP.ACCT' DB_HANDLE IS CHECKS
RDO> COMMIT
RDO> SHOW DATABASES
Database with filename personnel
Database with db_handle CHECKS in file acct
```

Example 3

The following example shows the steps you use to create the definitions in the same data dictionary entity. Assume the corrupt data dictionary entity is in 'DISK1:[DICTIONARY]CORP.PERSONNEL'. In the example, the RMU/DUMP/USERS command displays no active users of the database.

```
$ RMU/DUMP/USERS PERSONNEL
No active users
$ RUN SYS$SYSTEM:RDO
RDO> INTEGRATE DATABASE 'PERSONNEL'
cont> IN PATHNAME 'DISK1:[DICTIONARY]CORP.PERSONNEL'
RDO> COMMIT
```

Example 4

This example shows how to override the database definitions with the data dictionary definitions. In this example, the definitions stored in the data dictionary DISK1:[DICTIONARY]CORP.NEW.PERSONNEL replace the definitions in the target database, PERSONNEL.

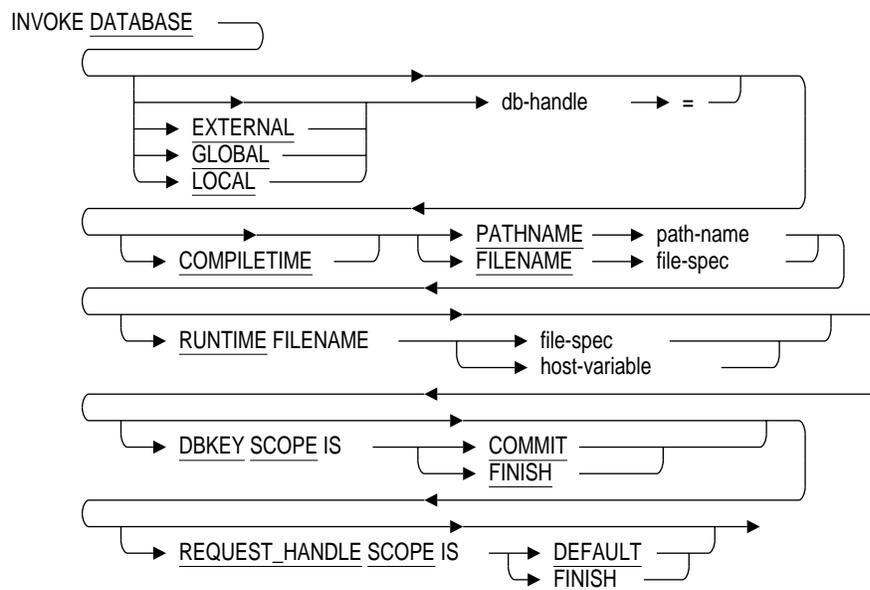
```
RDO> INTEGRATE DATABASE 'PERSONNEL'
cont> FROM PATHNAME 'DISK1:[DICTIONARY]CORP.NEW.PERSONNEL'
RDO> COMMIT
```

INVOKE DATABASE Statement

9.52 INVOKE DATABASE Statement

Specifies the name of the database to be accessed in a program or by RDO. You must issue an INVOKE DATABASE statement before you can use any other statement to refer to data in that database. Using INVOKE DATABASE is equivalent to declaring an external subroutine; it declares the database to the program.

Format



Arguments

EXTERNAL
GLOBAL [Default]
LOCAL

The scope of the database handle in programs. You can make a handle local to the module in which it is declared (LOCAL), or global to all modules that declare the database (GLOBAL, EXTERNAL).

INVOKE DATABASE Statement

db-handle

A host language variable that you associate with the name of the database. Use database handles when you are accessing more than one database at a time. Do not declare a host language variable explicitly for the database handle. The Rdb/VMS preprocessors declare the variable for you.

COMPILETIME

The source of the database definitions when the program is compiled. This can be either a data dictionary path name or a VMS file specification. If you specify only the COMPILETIME keyword and omit the RUNTIME keyword, Rdb/VMS uses the compile-time identifier for both compiling and running the program.

RUNTIME

The source of the database definitions when the program is run. This can be either a VMS file specification or a host language variable. If you do not specify this parameter, Rdb/VMS uses the compile-time identifier for both compiling and running the program.

PATHNAME path-name

A full or relative data dictionary path name, enclosed in quotation marks, that specifies a data dictionary entity that is the source of the database definitions. If you specify the PATHNAME qualifier, changes you make to database data definitions are entered in both the data dictionary and the database file.

If the DICTONARY IS REQUIRED option was specified when the database was defined or changed, you *must* invoke the database with the PATHNAME qualifier if you make changes to data definitions. If you invoke the database with the FILENAME qualifier and issue a data definition statement, you will get an error message.

FILENAME file-spec

A full or partial VMS file specification, enclosed in quotation marks, that specifies the source of the database definitions. If you specify the FILENAME qualifier, changes you make to database data definitions are entered only in the database file, not the data dictionary.

host-variable

A valid host language variable that equates to a database file specification.

INVOKE DATABASE Statement

DBKEY SCOPE IS COMMIT [Default] FINISH

The DBKEY SCOPE clause controls when the database key (dbkey) of an erased record may be reused by Rdb/VMS. When the DBKEY SCOPE is COMMIT, Rdb/VMS cannot reuse the dbkey of an erased record to store another record until the transaction that erased the original record completes (by entering COMMIT).

The DBKEY SCOPE COMMIT clause specifies that the dbkey of each record used is guaranteed to remain consistent *only* during each transaction. With the DBKEY SCOPE IS FINISH clause, Rdb/VMS cannot reuse the dbkey (to store another record) until the user who erased the original record detaches from the database (by using the FINISH statement).

Furthermore, the DBKEY SCOPE FINISH clause specifies that the dbkey of each record used is guaranteed not to change until this user detaches from the database (usually, with FINISH). With the DBKEY SCOPE FINISH clause, an RDBPRE program can complete one or several transactions and, while still attached to the database (after the INVOKE and before the FINISH statements), use the dbkey obtained during a STORE operation to directly access those records. During the STORE operation, the program can write the value of <context-variable>.RDB\$DB_KEY to a host language variable.

Example:

```
&RDB&   DATABASE FILENAME 'PERSONNEL' DBKEY SCOPE IS FINISH
&RDB&   START_TRANSACTION READ_WRITE

&RDB&   STORE P IN EMPLOYEES USING P.EMPLOYEE_ID =15231;
&RDB&       P.LAST_NAME = "Santoli";
&RDB&       GET
&RDB&           MY_DB_KEY = P.RDB$DB_KEY;
&RDB&       END_GET
&RDB&   END_STORE

&RDB&   COMMIT

&RDB&   START_TRANSACTION READ_WRITE
&RDB&   FOR J IN EMPLOYEES WITH J.RDB$DB_KEY = MY_DB_KEY
&RDB&       GET
1           JC = J.EMPLOYEE_ID;
2           JT = J.LAST_NAME;
3       END_GET
4   END_FOR
```

INVOKE DATABASE Statement

```
WRITE (6,60020) JC, JT
.
.
.
```

A complete example that demonstrates this feature is shown later in this section.

For programs like the previous one that directly access records using dbkey values saved earlier in the STORE . . . END_STORE block:

- The DBKEY SCOPE FINISH clause is useful because if the record accessed earlier was indeed subsequently erased, a message will indicate that this record is no longer available.
- With the DBKEY SCOPE COMMIT clause, it is possible for a program that performs direct access using dbkeys to access a different (new) record at that database page and line number even if the original record (whose dbkey was saved in a host language variable) was erased. The program has no indication that it is no longer working with the original record it intended to access and perhaps use.

REQUEST_HANDLE SCOPE IS DEFAULT [Default] FINISH

The REQUEST_HANDLE SCOPE clause is used only with RDBPRE and RDML programs, not with RDO or RDB\$INTERPRET. In a program where a request occurs within a higher level language loop, the REQUEST_HANDLE SCOPE clause determines whether system and user request handles are set to zero in the FINISH statement within the loop.

The default is DEFAULT, where the values of the request handles are not set to zero when the FINISH statement executes.

With the REQUEST_HANDLE SCOPE IS FINISH clause, the values of the request handles are set to zero when the FINISH statement executes, as in the following FORTRAN program example:

```
PROGRAM EXAMPLE
C
  IMPLICIT INTEGER*4 (A-Z)
  CHARACTER*15 LNAME
C
C***First we invoke the database.***
C
&RDB&      INVOKE DATABASE FILENAME 'personnel'
&RDB&      REQUEST_HANDLE SCOPE IS FINISH
C
```

INVOKE DATABASE Statement

```
C***Go into the database twice***
C
      DO J = 1, 2
C
C***Now get the first three employees in EMPLOYEES***
C
&RDB&      START_TRANSACTION READ_ONLY
C
&RDB&      FOR FIRST 3 E IN EMPLOYEES
&RDB&          GET
&RDB&              LNAME = E.LAST_NAME
&RDB&          END_GET
&RDB&              TYPE *,LNAME
&RDB&      END_FOR
C
&RDB&      ROLLBACK
C
C***Now we loop back and do it again.***
C
      TYPE *, 'DONE FOR - ', J
&RDB&      FINISH
      END DO
      TYPE *, 'ALL DONE!'
      STOP
C
      END
```

Note *If the database is invoked more than once in your program, the REQUEST_HANDLE SCOPE clause works only if it occurs with the first INVOKE statement that attaches to the database.*

The SQL FINISH statement initializes all request handles in all compilation units in a program. The RDBPRE and RDML preprocessors allow programs to define and manipulate request handles. If you do not want your request handles to be reinitialized, then you must use RDML or RDBPRE (not SQL) to do the attach, and you must use REQUEST_HANDLE SCOPE IS DEFAULT.

For more information on request handles, see the *VAX Rdb/VMS Guide to Using RDO, RDBPRE, and RDML*.

INVOKE DATABASE Statement

Usage Notes

You must have the Rdb/VMS READ privilege for a database to invoke it using the INVOKE DATABASE statement.

In programs, you must use quotation marks around path names and file names. In RDO, quotation marks are optional, unless you are invoking a database on a remote network node.

The INVOKE DATABASE statement is not terminated with a period.

The INVOKE DATABASE statement, unlike other data manipulation statements, does not include an END clause.

Rdb/VMS allows you to perform all database operations with or without using the data dictionary. Because an Rdb/VMS database stores both database definitions and the actual data values in the database file itself, you can use Rdb/VMS without storing data definitions in the data dictionary.

However, Digital Equipment Corporation recommends that you store, or plan to store, all database definitions in the data dictionary as well as in the database file for the following reasons:

- Compatibility with other Digital information management products
- Compatibility with future data dictionary requirements for Digital information management software

It is possible to enforce use of the data dictionary among users of the database. If you specify the DICTONARY IS REQUIRED clause on the DEFINE DATABASE or CHANGE DATABASE statement, users must invoke the database with the PATHNAME qualifier to change metadata.

Examples

Example 1

The following example invokes a database in RDO:

```
RDO> INVOKE DATABASE PATHNAME 'DISK1:[DICTIONARY]CORP.MIS.PERSONNEL'
```

To invoke the PERSONNEL database in RDO, this example uses the data dictionary definitions in DISK1:[DICTIONARY]CORP.MIS.PERSONNEL.

INVOKE DATABASE Statement

Example 2

The INVOKE DATABASE statement must be part of every program.

```
&RDB& INVOKE DATABASE FILENAME 'DISK2:[DEPT3]PERSONNEL'
```

This statement declares the database defined by the file specification DISK2:[DEPT3]PERSONNEL. The preprocessor then uses this definition when processing the program, and Rdb/VMS uses the database file DISK2:[DEPT3]PERSONNEL.RDB when the program runs.

Example 3

The following BASIC example invokes a database using an EXTERNAL database handle:

```
&RDB& INVOKE DATABASE EXTERNAL  
&RDB& PERSONNEL = PATHNAME 'DISK1:[DICTIONARY]CORP.MIS.PERSONNEL'
```

This example shows a global declaration. Other modules in the program can refer to the database using the handle PERSONNEL; they must use the same database handle for the same database. The program *does not* declare the variable.

- GLOBAL and EXTERNAL are synonymous. These make the database handle global to all modules that declare the database. Rdb/VMS creates a PSECT that contains one longword using the name supplied with the overlay attribute by the user.
- The default database scope is GLOBAL.
- The preprocessor generates declarations for all database handles.

Example 4

The following COBOL example uses the COMPILETIME and RUNTIME options:

```
&RDB& INVOKE DATABASE LOCAL PERSONNEL =  
&RDB& COMPILETIME PATHNAME  
&RDB& 'DISK1:[DICTIONARY]ACCT.PERSONNEL'  
&RDB& RUNTIME FILENAME  
&RDB& 'USERDISK3:[DEPT3]PERSONNEL'
```

This example declares the database using two different sources:

- At compile time, the preprocessor uses the database definitions in the data dictionary.

INVOKE DATABASE Statement

- At run time, Rdb/VMS uses the definition in the file USERDISK3:[DEPT3]PERSONNEL.RDB.

Example 5

The following VAX FORTRAN DML program:

- Invokes the sample PERSONNEL database and uses the DBKEY SCOPE FINISH clause
- Stores a record and obtains its newly assigned dbkey value
- Displays the EMPLOYEE_ID and LAST_NAME fields from the record and the record's dbkey
- Commits the transaction
- Directly retrieves the record stored in the previous transaction using the known dbkey pointer
- Displays the two fields again
- Detaches from the database with the FINISH statement

```
*
* STORE2DML.RFO. Demonstrate:
* 1. DBKEY SCOPE FINISH
* 2. STORE that uses RDB$DB_KEY to retain DBKEY
*   of record being stored
* 3. Subsequent access by the stored dbkey before COMMIT
* 4. Subsequent access by the stored dbkey after COMMIT
*   and before FINISH
*   IMPLICIT NONE
*   CHARACTER*8 MY_DB_KEY
*   CHARACTER*20 JT
*   CHARACTER*5 JC
*
&RDB&  DATABASE FILENAME 'PERSONNEL' DBKEY SCOPE IS FINISH
*
&RDB&  START_TRANSACTION READ_WRITE
*
* Store a record and get its dbkey
*
```

INVOKE DATABASE Statement

```
&RDB& STORE P IN EMPLOYEES USING
&RDB&     P.EMPLOYEE_ID = 15231;
&RDB&     P.LAST_NAME = "Santoli";
&RDB&     P.SEX = "M" ;
&RDB&     P.STATUS_CODE = "0";
&RDB&     GET
&RDB&         MY_DB_KEY = P.RDB$DB_KEY;
&RDB&     END_GET
&RDB& END_STORE

&RDB& FOR J IN EMPLOYEES WITH J.RDB$DB_KEY = MY_DB_KEY
&RDB&     GET
&RDB&         1         JC = J.EMPLOYEE_ID;
&RDB&         2         JT = J.LAST_NAME;
&RDB&         3         END_GET
&RDB&         4     END_FOR

&RDB&     WRITE (6,60020) JC, JT

&RDB& COMMIT

&RDB& START_TRANSACTION READ_WRITE
&RDB& FOR J IN EMPLOYEES WITH J.RDB$DB_KEY = MY_DB_KEY
&RDB&     GET
&RDB&         1         JC = J.EMPLOYEE_ID;
&RDB&         2         JT = J.LAST_NAME;
&RDB&         3         END_GET
&RDB&         4     END_FOR

&RDB&     WRITE (6,60020) JC, JT

&RDB& COMMIT

&RDB&     PRINT *, ' '
60020  FORMAT(1X, A5, 2X, A20)

&RDB&     END
```

When this sample program is run against the PERSONNEL database, the following output is produced:

```
15231   Santoli
15231   Santoli
```

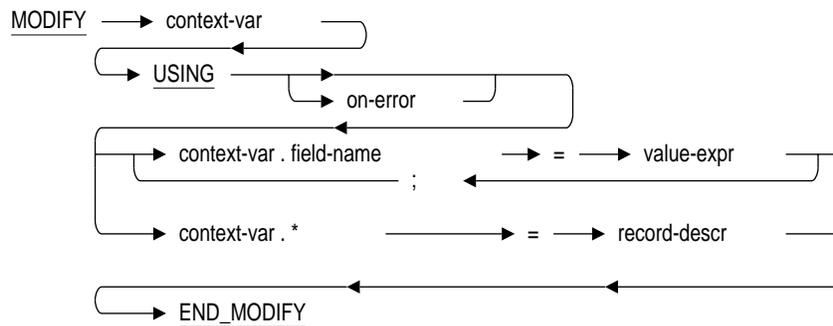
The second printed line demonstrates that the same record was retrieved directly with the dbkey obtained when the record was stored.

MODIFY Statement

9.53 MODIFY Statement

Changes the value in a field in one or more records from a relation or open stream. Before you use the MODIFY statement, you must start a read/write transaction and establish a record stream with a FOR or START_STREAM statement. The context variables you refer to in a MODIFY statement must be the same as those defined in the FOR or START_STREAM statement.

Format



Arguments

on-error

The ON ERROR clause. This clause specifies a host language statement to be performed if an Rdb/VMS error occurs.

context-var

A context variable declared in the FOR or START_STREAM statement. The MODIFY statement must appear after FOR or START_STREAM and before END_FOR or END_STREAM.

field-name

The name of the field to be modified.

value-expr

An Rdb/VMS value expression that specifies the new value for the modified field.

MODIFY Statement

record-descr

A valid data dictionary record descriptor that matches all the fields of the relation. You can use a host language statement to include the relation definition in your program. Each field of the record descriptor must match exactly the field names and data types of the fields in the Rdb/VMS relation referred to by the context variable.

Usage Notes

You need the Rdb/VMS READ and MODIFY privileges to the relation and the Rdb/VMS MODIFY privilege to the database to use the MODIFY statement.

You *cannot* modify records from a view that:

- Refers to more than one relation
- Is defined using one or more of the following clauses:
 - WITH clause of an RDO record selection expression
 - CROSS clause of an RDO record selection expression
 - REDUCED TO clause of an RDO record selection expression

You can modify fields in only one relation at a time. That is, all the context variables on the left side of the assignment in a MODIFY statement must refer to the same relation.

You can modify fields using a CROSS clause as long as the fields you are modifying are not involved in the join operation.

You cannot modify segmented strings.

Examples

Example 1

The following example modifies a field value in a record:

```
DISPLAY "Enter employee's ID number: " WITH NO ADVANCING.  
ACCEPT ID.  
DISPLAY "Enter new status code: " WITH NO ADVANCING.  
ACCEPT STATUS-CODE.  
  
&RDB& START_TRANSACTION READ_WRITE
```

MODIFY Statement

```
&RDB&  FOR E IN EMPLOYEES WITH E.EMPLOYEE_ID = ID
&RDB&    MODIFY E USING
&RDB&      ON ERROR
&RDB&        GO TO ERROR-PAR
&RDB&      END_ERROR
&RDB&      E.STATUS_CODE = STATUS-CODE
&RDB&    END_MODIFY
&RDB&  END_FOR

&RDB&  COMMIT
```

This COBOL example prompts for an identification number and a new status code and modifies the corresponding employee record.

Example 2

You can update more than one relation in a single FOR loop.

```
      INPUT "What is employee's ID number"; ID_NUMBER

&RDB&  START_TRANSACTION READ_WRITE RESERVING
&RDB&      JOB_HISTORY,
&RDB&      SALARY_HISTORY FOR
&RDB&      PROTECTED WRITE

&RDB&  FOR JH IN JOB_HISTORY CROSS
&RDB&      SH IN SALARY_HISTORY OVER EMPLOYEE_ID
&RDB&      WITH JH.EMPLOYEE_ID = ID_NUMBER
&RDB&      AND JH.JOB_END MISSING
&RDB&      AND SH.SALARY_END MISSING
&RDB&      MODIFY JH USING
&RDB&          JH.JOB_CODE = NEW-JOB
&RDB&      END_MODIFY

&RDB&      MODIFY SH USING
&RDB&          SH.SALARY_AMOUNT = NEW-AMOUNT
&RDB&      END_MODIFY
&RDB&  END_FOR

&RDB&  COMMIT
```

This BASIC example updates an employee record by changing his or her job code and salary amount. In the example:

- The RSE in the FOR loop joins the JOB_HISTORY and SALARY_HISTORY relations and specifies the conditions for the records. This record stream includes only the records where the end dates are missing (the current records) and where the EMPLOYEE_ID field values match.
- The two MODIFY statements are included within a single FOR loop.

MODIFY Statement

Errors

Your program can check for and handle the following errors, using the name listed here in the form RDB\$_error-name. The following list includes two types of errors:

- E Expected errors. These are run-time errors. Your program should check for these errors and provide error handlers.
- U Unexpected errors. Preprocessors detect these errors when the program is preprocessed, so your program does not need to check for them. Callable RDO, however, detects these errors at run time. A Callable RDO program should check for them.

ARITH_EXCEPT (U)

truncation of a numeric value at runtime

BAD_SEGSTR_HANDLE (U)

invalid handle for segmented string

DEADLOCK (E)

request failed due to resource deadlock

INTEG_FAIL (E)

constraint *name* failed

LOCK_CONFLICT (E)

NO WAIT request failed because resource was locked

NO_CUR_REC (U)

stream has no current record for retrieval or update

NO_DUP (E)

update would cause duplicates on unique index *name*

NO_PRIV (U)

privilege denied by database protection

NO_SEGSTR_CLOSE (U)

segmented string must be closed before being stored

NOT_VALID (U)

validation failure on field *name*

MODIFY Statement

OBSOLETE_METADATA (U)

request references undefined fields or relations

READ_ONLY_FIELD (U)

attempt to update read-only field *name*

READ_ONLY_REL (U)

relation *name* was reserved for READ, updates disallowed

READ_ONLY_TRANS (U)

attempt to update from a read-only transaction

READ_ONLY_VIEW (U)

view *name* can not be updated

REQ_NO_TRANS (U)

attempt to continue request after COMMIT/ROLLBACK

SEGSTR_NO_WRITE (U)

segmented string is open for read, write prohibited

WRONUMARG (U)

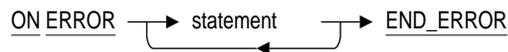
illegal number of arguments on call to Rdb system

9.54 ON ERROR Clause

Specifies the statement the host language will perform if an error occurs during the execution of the associated host language or Rdb/VMS data manipulation statement.

Format

ON ERROR → statement → END_ERROR



Argument

statement

Any valid host language statement or Rdb/VMS data manipulation statement. Separate multiple statements with the appropriate statement delimiter or terminator according to the rules of the host language.

Usage Notes

If you have invoked a database, you have the necessary privileges to use the ON_ERROR clause.

An ON ERROR clause directly following the FOR statement in a FOR . . . END_FOR block handles errors from the FOR statement, not from other Rdb/VMS data manipulation statements within the FOR . . . END_FOR block. To handle errors that occur in an Rdb/VMS data manipulation statement that is contained within a FOR . . . END_FOR block, include the ON ERROR clause after the data manipulation statement.

The ON ERROR clause is supported only for the high-level language preprocessors. It is *not supported* for RDO or Callable RDO.

ON ERROR Clause

Examples

Example 1

This BASIC program uses LIB\$MATCH_COND to handle errors that occur in the MODIFY statement:

```
30
&RDB& START_TRANSACTION READ_WRITE

45 ! Input data
!

PRINT
INPUT "enter status code"; STATUS_CODE
PRINT
INPUT "enter status name"; STATUS_NAME
PRINT
INPUT "enter status type"; STATUS_TYPE

50 ! Modify the record
!
&RDB& FOR W IN WORK_STATUS
&RDB&     MODIFY W USING
&RDB&     ON ERROR
&RDB&         GOSUB 1000
&RDB&     END_ERROR
&RDB&     W.STATUS_CODE = STATUS_CODE;
&RDB&     W.STATUS_NAME = STATUS_NAME;
&RDB&     W.STATUS_TYPE = STATUS_TYPE
&RDB&     END_MODIFY
PRINT
PRINT STATUS_CODE, STATUS_NAME, STATUS_TYPE
&RDB& END_FOR
GO TO 5000

1000 MATCH_COND =
LIB$MATCH_COND(RDB$STATUS,RDB$_NOT_VALID)

SELECT MATCH_COND

CASE 1
PRINT "You have entered invalid data. Try again."
CALL SYS$PUTMSG(RDB$MESSAGE_VECTOR)
&RDB& ROLLBACK
GOTO 20

CASE 0
PRINT "Unknown error"
CALL SYS$PUTMSG(RDB$MESSAGE_VECTOR)
&RDB& ROLLBACK
GO TO 5000
```

ON ERROR Clause

```
END SELECT
RETURN
5000
&RDB&  FINISH
      END
```

This excerpt from a BASIC program:

- Prompts for input for the MODIFY statement.
- Starts a FOR loop.
- Branches to the call to LIB\$MATCH_COND if an error occurs in the MODIFY statement. This run-time library procedure matches the condition value that Rdb/VMS returns to the program (RDB\$STATUS) with the condition code RDB\$_NOT_VALID. If the return status is RDB\$_NOT_VALID, the program prints the appropriate message and loops back to prompt again. If some other error code is detected, “Unknown error” is displayed and the program stops.

PLACE Statement

on-error

The ON ERROR clause. This clause specifies a host language statement or Rdb/VMS data manipulation statement to be performed if an Rdb/VMS error occurs.

field-name

The names of the fields that appear in the index referenced in the DEFINE STORAGE MAP statement's PLACEMENT VIA INDEX clause. All index segments must be specified or the results of the PLACE statement will be of little use. Field-names not used in the index, if listed, will be ignored.

value-expr

A valid Rdb/VMS value expression that specifies the value that would be stored.

record-descr

A valid data dictionary record descriptor that matches all the fields in the relation. You can use a host language statement to include the relation definition in your program. Each field of the record descriptor must match exactly the field names and data types of the fields in the Rdb/VMS relation referred to by the context variable.

host-var

A user-defined host language variable into which you may store the dbkey of the record that would be stored. Use the GET . . . RDB\$DB_KEY construct to assign the value of the dbkey to the host language variable.

Usage Notes

You must have the Rdb/VMS WRITE privilege to the database and relation to use the PLACE statement.

The PLACE statement returns the dbkey of a specified record. When used in a FOR statement in a program, the PLACE statement builds a list of unordered dbkeys for all specified records. You can then use the SORT utility to create a sorted list of dbkeys and use this sorted list of dbkeys to store the records. When you store records sorted by dbkey, you are writing records to database pages in sequence with all records for a page written to the page while it is in the buffer. Because less random I/O is involved when you store records in this way, a significant performance improvement can occur during your load procedure.

PLACE Statement

The **PLACE** statement can result in significant performance improvements in database load procedures that specify the **PLACEMENT VIA INDEX** clause for a hashed index. Use it only with records for which a hashed index has been defined.

You must execute the **PLACE** statement in a read/write transaction.

Examples

Example 1

In the following example, the **EMPLOYEE_ID** field is in a hashed index and was stored using the **PLACEMENT VIA INDEX** clause of the **DEFINE STORAGE MAP** statement. The **PLACE** statement returns the dbkey of the specified record:

```
RDO> PLACE E IN EMPLOYEES
cont>     USING E.EMPLOYEE_ID = "94789"
cont>     PRINT E.RDB$DB_KEY
cont> END_PLACE
```


PRINT Statement

Arguments

value-expr

A valid Rdb/VMS value expression. See Chapter 3 for more information on value expressions.

context-var

A context variable you have defined in a FOR or START_STREAM statement. Use a context variable and a wildcard character (*) to display all the fields for each record in the record stream. If you use the wildcard character, RDO uses the field names as column headers on the display.

Usage Notes

If you have invoked a database, you have the necessary privileges to use the PRINT statement.

The PRINT statement displays column headers for each field in the database. Rdb/VMS uses the field name as the column name. However, Rdb/VMS does not display column headers for statistical expressions, segmented strings, or literals. If you use the PRINT statement with a CROSS clause, Rdb/VMS also displays the context variable with the field name in the column header. The following example shows the column header output from the PRINT statement:

```
RDO> FOR E IN EMPLOYEES
cont>   CROSS JH IN JOB_HISTORY
cont>     WITH E.EMPLOYEE_ID = JH.EMPLOYEE_ID
cont>     AND E.EMPLOYEE_ID = "00201"
cont>     PRINT
cont>       E.EMPLOYEE_ID,
cont>       JH.JOB_CODE,
cont>       JH.JOB_START
cont> END_FOR
```

E.EMPLOYEE_ID	JH.JOB_CODE	JH.JOB_START
00201	APGM	4-JUN-1977 00:00:00.00
00201	APGM	15-APR-1979 00:00:00.00
00201	APGM	28-MAY-1980 00:00:00.00
00201	APGM	1-JUL-1975 00:00:00.00

The PRINT statement is valid only in RDO. To retrieve values in an RDBPRE or RDML program, use the GET statement to assign database values to host language variables.

PRINT Statement

Examples

Example 1

The following RDO query displays all the field values from each record in the EMPLOYEES relation:

```
RDO> START_TRANSACTION READ_ONLY
RDO>
RDO> FOR E IN EMPLOYEES
cont> PRINT E.*
cont> END_FOR
RDO>
RDO> COMMIT
```

You can use the wildcard character to display all fields in the following cases:

- Within a stream
- Within an RSE that has a CROSS clause

However, you cannot use the wildcard character with the PRINT statement to display segmented strings, statistical expressions, or literals.

Example 2

The following RDO query displays field values from each record in the EMPLOYEES relation:

```
RDO> START_TRANSACTION READ_ONLY
RDO>
RDO> FOR E IN EMPLOYEES
cont> PRINT E.LAST_NAME, E.FIRST_NAME, E.MIDDLE_INITIAL
cont> END_FOR
RDO>
RDO> COMMIT
```

Example 3

The following example retrieves values from fields in two relations:

```
RDO> START_TRANSACTION READ_ONLY
RDO>
RDO> FOR E IN EMPLOYEES CROSS JH IN JOB_HISTORY
cont> OVER EMPLOYEE_ID
cont> WITH JH.JOB_END MISSING
cont> PRINT E.LAST_NAME, JH.JOB_CODE
cont> END_FOR
RDO>
RDO> COMMIT
```

PRINT Statement

This RDO statement performs a join of two relations and uses the PRINT statement to display a value from each. The result is the last name and current job code of each employee.

Example 4

The following statement retrieves the number of employees who live in Massachusetts, using the COUNT statistical expression:

```
RDO> START_TRANSACTION READ_ONLY
RDO>
RDO> PRINT COUNT OF E IN EMPLOYEES WITH E.STATE = "MA"
RDO>
RDO> COMMIT
```

Example 5

You can use the PRINT statement to calculate arithmetic expressions. For example:

```
RDO> INVOKE DATABASE FILENAME "WORK$DISK:PERSONNEL"
RDO> PRINT 1+2
          3
RDO> PRINT "1+2"
          1+2
RDO> EXIT
```

Example 6

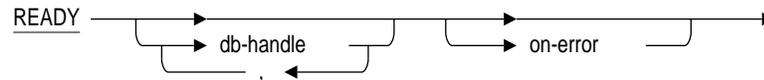
For experimental purposes, you can use the PRINT statement within a STORE . . . END_STORE block to display the dbkey of the record just stored:

```
RDO> STORE E IN EMPLOYEES USING
cont> E.EMPLOYEE_ID = "15231";
cont> E.LAST_NAME = "Smith";
cont> PRINT E.RDB$DB_KEY
cont> END_STORE
          RDB$DB_KEY
          21:339:0
```

9.57 READY Statement

In programs, explicitly declares your intention to access one or more databases and causes an attach to the database. The READY statement is not available in RDO.

Format



Arguments

db-handle

A host language variable used to refer to a specific database your program invokes.

Use a database handle when you want to work with multiple databases. You can activate each database as you need it and then use the FINISH statement to close it when you are done. In this way, you do not have to allocate system resources to keep all the required databases active throughout the program.

on-error

The ON ERROR clause. This clause specifies host language statements to be performed if an error occurs during the READY operation. See Section 9.54 for details.

Usage Notes

You need the Rdb/VMS READ privilege for the database to use the READY statement.

If you issue a READY statement without specifying a database handle, Rdb/VMS opens all databases declared in that module.

You do not have to use the READY statement to access a database. A database is readied automatically the first time you refer to it. However, you do need an INVOKE DATABASE statement.

READY Statement

You can use the `READY` statement to test the availability of a database. For example, you may want to check availability before your program prompts a user for input.

Examples

Example 1

The following program fragments demonstrate the use of the `READY` statement to open a database. The program fragments:

- Use the `INVOKE DATABASE` statement to declare the `PERSONNEL` database
- Declare a database handle `PERS` for `PERSONNEL`
- Open the `PERSONNEL` database with the `READY` statement
- Close the database with the `FINISH` statement

C Program

```
include <stdio.h>
DATABASE PERS = FILENAME "WORK$DISK:PERSONNEL";

.
.
.

main ()
{
READY PERS;

.
.
.

FINISH PERS;
```

Pascal Program

```
program empupdate;
DATABASE PERS = FILENAME 'WORK$DISK:PERSONNEL';

.
.
.

begin
READY PERS;
```

READY Statement

```
.  
. .  
. .  
FINISH PERS;
```

Example 2

The following program fragments demonstrate how to open two databases within the same program. The program fragments:

- Use the INVOKE DATABASE statement to declare two databases, PERSONNEL and PAYROLL
- Declare database handles for both databases
- Open both databases
- Close each database

C Program

```
include <stdio.h>  
DATABASE PERS = FILENAME "WORK$DISK:PERSONNEL";  
DATABASE PAY  = FILENAME "WORK$DISK:PAYROLL";  
  
main ()  
{  
  
    .  
    .  
    .  
  
    READY PERS;  
  
    .  
    .  
    .  
  
    FINISH PERS;  
  
    .  
    .  
    .  
  
    READY PAY;  
  
    .  
    .  
    .  
  
    FINISH PAY;  
  
    .  
    .  
    .  
}
```

READY Statement

```
READY PERS, PAY;
```

```
.  
. .  
.
```

```
FINISH PERS, PAY;
```

Pascal Program

```
program new_employee;  
DATABASE PERS = FILENAME 'WORK$DISK:PERSONNEL';  
DATABASE PAY  = FILENAME 'WORK$DISK:PAYROLL';
```

```
.  
. .  
.
```

```
READY PERS;
```

```
.  
. .  
.
```

```
FINISH PERS;
```

```
.  
. .  
.
```

```
READY PAY;
```

```
.  
. .  
.
```

```
FINISH PAY;
```

```
.  
. .  
.
```

```
READY PERS, PAY;
```

```
.  
. .  
.
```

```
FINISH PERS, PAY;
```

REINITIALIZE TRANSFER Statement (VAX Data Distributor)

9.58 REINITIALIZE TRANSFER Statement (VAX Data Distributor)

Reinitializes a replication transfer. You use this statement when the source database and the replication target database are no longer consistent with each other.

Note To use this RDO statement, you must have VAX Data Distributor installed on your system. See your system manager or your Digital Equipment Corporation representative if you need information on Data Distributor.

Format

REINITIALIZE TRANSFER \longrightarrow transfer-name \longrightarrow

Argument

transfer-name

The replication transfer that you want to reinitialize. The transfer-name parameter is required.

Usage Notes

If you want to reinitialize a particular transfer, the transfer definition must be associated with your UIC.

You can reinitialize a transfer only when the transfer is in the suspended state. To suspend a transfer, first issue a STOP TRANSFER statement. You must execute the REINITIALIZE TRANSFER statement outside the scope of a transaction. If you issue this statement when a transaction is outstanding, Data Distributor returns an error message.

After you enter the REINITIALIZE statement, the transfer remains in the suspended state. You must then issue a START TRANSFER statement to restart the transfer.

REINITIALIZE TRANSFER Statement (VAX Data Distributor)

The REINITIALIZE statement forces the next execution of a replication transfer to create a new replication database in the target directory. You must reinitialize a transfer when someone restores a previous version of the source database.

Examples

Example 1

This example places the CARS_LAX transfer in a suspended state and then reinitializes it. The schedule and definition for this replication transfer remain in place. Once the transfer has been reinitialized, you can restart it.

```
RDO> STOP TRANSFER CARS_LAX  
RDO> REINITIALIZE TRANSFER CARS_LAX  
RDO> START TRANSFER CARS_LAX
```

9.59 ROLLBACK Statement

Terminates a transaction and undoes all changes that have been made to the database since the program's most recent `START_TRANSACTION` statement. The `ROLLBACK` statement also:

- Closes all open streams
- Releases all readied relations
- Releases all locks

The `ROLLBACK` statement affects:

- All databases participating in the current transaction
- All changes to data made with Rdb/VMS data manipulation statements (`ERASE`, `MODIFY`, and `STORE`)
- All changes to data definitions made with Rdb/VMS data definition statements (`CHANGE`, `DEFINE`, and `DELETE`)

Format

```
ROLLBACK (TRANSACTION_HANDLE var) on-error
```

Arguments

TRANSACTION_HANDLE var

A keyword followed by a host language variable. A transaction handle identifies each instance of a database attach. If you do not declare the transaction handle explicitly, Rdb/VMS attaches an internal identifier to the transaction.

In Callable RDO, use `!VAL` as a substitution marker for host language variables. See the *VAX Rdb/VMS Guide to Using RDO, RDBPRE, and RDML* for information about the use of `!VAL`.

ROLLBACK Statement

You can put parentheses around the host language variable name.

Note Normally, you do not need to use this argument. The ability to declare a transaction handle is provided for compatibility with other database products and future releases of Rdb/VMS.

Do not use this argument in interactive RDO.

on-error

The ON ERROR clause. This clause specifies a host language statement to be performed if an Rdb/VMS error occurs.

Usage Notes

If you have invoked a database, you have the necessary privileges to use the ROLLBACK statement.

The ROLLBACK operation closes all open streams, whether they were formed by a FOR or START_STREAM statement. Therefore, the ROLLBACK statement performs the END_STREAM statement implicitly, and you do not need an explicit END_STREAM statement after a ROLLBACK statement.

Examples

Example 1

The following example shows how to roll back changes made during a transaction in a COBOL program:

```
GET-ID-NUMBER.  
    DISPLAY "Enter employee ID number:  "  
        WITH NO ADVANCING.  
    ACCEPT EMPLOYEE-ID.  
CHANGE-SALARY.  
    DISPLAY "Enter new salary amount:  "  
        WITH NO ADVANCING.  
    ACCEPT SALARY-AMOUNT.  
  
&RDB&    START_TRANSACTION READ_WRITE  
&RDB&    FOR S IN SALARY_HISTORY WITH  
&RDB&        S.EMPLOYEE_ID = EMPLOYEE-ID  
&RDB&        AND S.END_DATE MISSING  
  
&RDB&        MODIFY USING  
&RDB&            S.SALARY_AMOUNT = SALARY-AMOUNT  
&RDB&        END_MODIFY  
&RDB&    END_FOR
```

ROLLBACK Statement

```
DISPLAY EMPLOYEE-ID, SALARY-AMOUNT.  
DISPLAY "Is this figure correct? [Y or N] "  
    WITH NO ADVANCING.  
ACCEPT ANSWER.  
IF ANSWER = "Y" THEN  
&RDB&    COMMIT  
    ELSE  
&RDB&    ROLLBACK  
DISPLAY "Please enter the new salary amount again."  
GO TO CHANGE-SALARY.
```

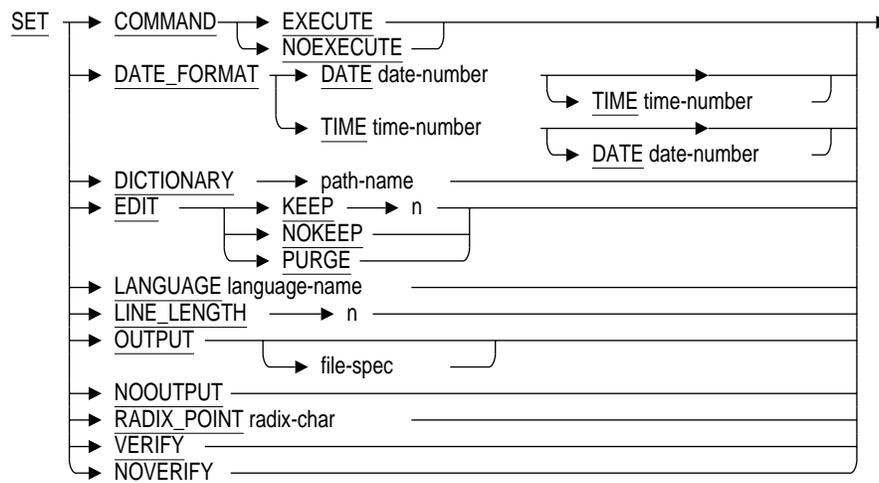
SET Statement

9.60 SET Statement

Changes the characteristics of your RDO terminal session. You can control:

- The execution of RDO data manipulation statements
- The display format for date and time values
- The default directory in the data dictionary
- The number of statements to be included in the editing buffer
- The language to be used for date and time input and display
- The line length for RDO output
- The file in which the session is recorded
- The character representing the radix point in output displays
- The display of statements from a command file

Format



SET Statement

Arguments

COMMAND EXECUTE [Default]

Instructs RDO to execute the data manipulation statements you issue in an interactive RDO session.

The SET COMMAND EXECUTE statement is available only in RDO.

COMMAND NOEXECUTE

Instructs RDO not to execute the data manipulation statements you issue in an interactive RDO session. You can use the NOEXECUTE option in conjunction with RDMS\$DEBUG_FLAGS to examine the estimated cost and access strategy associated with a query. Although the query will not be executed, the access strategies will still be displayed. See the *VAX Rdb/VMS Guide to Database Maintenance and Performance* for information on using RDMS\$DEBUG_FLAGS.

The SET COMMAND NOEXECUTE statement is available only in RDO.

DATE_FORMAT

Specifies the display format for either date values, time values, or both.

You must specify a numeric argument with the DATE and TIME portions of SET DATE_FORMAT. This numeric argument is the same as the numeric portion of certain VMS Run-Time Library formats. The formats are documented in the *VMS RTL Library (LIBS) Manual*.

Note that SET DATE_FORMAT DATE and SET DATE_FORMAT TIME change only the output for the date or time formats. If you want to change the input format, use the logical name LIB\$DT_INPUT_FORMAT. See the *VMS RTL Library (LIBS) Manual* for more information on LIB\$DT_INPUT_FORMAT.

DATE_FORMAT date-number

Specifies the display format for date values.

The date-number argument corresponds to numbers in the date format logical names listed in tables in the *VMS RTL Library (LIBS) Manual*.

For example, LIB\$DATE_FORMAT_006 is one of the logical names in the table. The logical name specifies the format in which the eighth day of May in the year 1957 would be displayed as 8 May 57. Note that the latter part of the logical name is the number 006.

SET Statement

If you wanted to specify the 8 May 57 format using the SET DATE_FORMAT statement, you would use the numeric part of the LIB\$DATE_FORMAT_006 logical name, 6. You do not have to enter any leading zeros that the number might have.

If you do not specify a date format, the default is dd-mmm-yyyy.

TIME *time-number*

Specifies the display format for time values.

You must enter a number for the time-number argument. This number corresponds to numbers in the time format logical names listed in tables in the *VMS RTL Library (LIB\$) Manual*.

For example, the table contains the logical name LIB\$TIME_FORMAT_020. The logical name specifies the format in which the eighth hour, fourth minute, and thirty-second second of a day would be displayed as 8 h 4 min 32 s. Note that the latter part of the logical name is the number 020.

If you wanted to specify the 8 h 4 min 32 s format for the SET DATE_FORMAT TIME statement, you would use the numeric part of the LIB\$TIME_FORMAT_020 logical name, 20. You do not have to enter any leading zeros that the number might have.

If you do not specify a time format, the default is hh:mm:ss.cc.

DICTIONARY *path-name*

Changes your default data dictionary directory to the path name you specify.

EDIT

The SET EDIT statement allows you to control the size of the editing buffer that you create when you use the EDIT statement with a wildcard as the parameter:

- SET EDIT KEEP *n*

Instructs RDO to save the previous *n* statements. For example, assume you have specified SET EDIT KEEP 5. When you type EDIT *, RDO places the previous five statements in the editing buffer. The default is 20 statements.

- SET EDIT NOKEEP

This statement is equivalent to SET EDIT KEEP 0. If you use this form of the statement, and if you type EDIT or EDIT *, your editing buffer is empty. This form of the statement saves system resources when you are running command procedure files rather than an interactive process.

SET Statement

- **SET EDIT PURGE**

This statement retains the value of the KEEP parameter but purges all the previous statements. As with SET EDIT NOKEEP, if you use SET EDIT PURGE and the EDIT or EDIT *, your editing buffer will be empty. Unlike SET EDIT NOKEEP, however, SET EDIT PURGE causes RDO to accumulate subsequent statements to place in the editing buffer when you issue EDIT statements later in the interactive session.

LANGUAGE language-name

Specifies the language to be used for translation of month names and abbreviations in date and time input and display. The language-name also determines the translation of other language-dependent text, such as the translation for the date literals YESTERDAY, TODAY, and TOMORROW.

If you do not specify a language, the default is the language specified by the logical name SYSSLANGUAGE.

LINE_LENGTH n

Sets the line length for RDO output. Use this option to specify an alternate width when output is going to a file or alternate output device. You can specify any number for n up to 65536.

You can put a pair or set of SET LINE_LENGTH statements into a nested FOR loop to change the line length during the loop.

OUTPUT file-spec

Names the default target for output. The default file type is LIS.

If you specify the OUTPUT option with a file name, RDO writes its output to a log file that you specify. The output is also written to SYSSOUTPUT, which is usually the terminal. If you do not specify a file name, RDO writes the output only to SYSSOUTPUT. The log file contains both statements and results. The SET OUTPUT statement lets you check the results of command procedures run in batch mode. You can suspend writing to the output file by specifying SET NOOUTPUT or by specifying another output file.

RADIX_POINT radix-char

Changes the output displaying the radix point to the specified character. The radix point is the symbol that separates units from decimal fractions. For example, in the number 98.6, the period is the radix point.

If you do not specify an alternate character, the default is either the period (.), or the value specified by the system logical name SYSSRADIX_POINT.

SET Statement

VERIFY

Displays indirect command procedure files at your terminal as you run them.

NOVERIFY [Default]

Does not display indirect command procedure files.

Usage Notes

If you have invoked a database, you have the necessary privileges to use the SET statement.

The SET LANGUAGE statement does not affect the collating sequences used for sorting and comparing data. The DEFINE COLLATING_SEQUENCE statement specifies alternate collating sequences.

The SET RADIX POINT statement changes the radix point only in the output display. It does not change the input character; the input character must always be a period.

Table 9–9 shows the logical names you can use with the SET statement to specify the language, radix point, and date format for output displays. The logical names are documented in the *VMS RTL Library (LIB\$) Manual*.

If you want to change the input format for dates, you must use the logical name LIB\$DT_INPUT_FORMAT documented in the *VMS RTL Library (LIB\$) Manual*. The SET DATE statement in RDO changes only the date format for output displays.

Table 9–9 Logical Names for Internationalization of SET Statements

SQL SET Statement	RDO SET Statement	Related System Logical Name
CURRENCY SIGN	Not applicable	SYSSCURRENCY
DATE FORMAT DATE <i>date-number</i>	DATE_FORMAT DATE <i>date-number</i>	LIB\$DT_FORMAT
DATE FORMAT TIME <i>time-number</i>	DATE_FORMAT TIME <i>time-number</i>	LIB\$DT_FORMAT

(continued on next page)

SET Statement

Table 9-9 (Cont.) Logical Names for Internationalization of SET Statements

SQL SET Statement	RDO SET Statement	Related System Logical Name
DIGIT SEPARATOR	Not applicable	SYSSDIGIT_SEP
LANGUAGE	LANGUAGE	SYSSLANGUAGE
RADIX POINT	RADIX_POINT	SYSSRADIX_POINT

The RDO DEFINE TRANSFER statement accepts only the VMS standard date and time formats. It does not accept any date formats set by defining LIB\$DT_INPUT_FORMAT.

Examples

Example 1

The following example shows how you can set up the characteristics of your terminal session. (This example does not include the SET statements used for internationalization. They are described in the second example.)

```
$ DEFINE RDOINI "DISK2:[DEPT3]RDOINI.RDO"
$ TYPE DISK2:[DEPT3]RDOINI.RDO
!
! This is the RDOINI file. It sets up your RDO session.
!
SET OUTPUT LOG.LIS
SET DICTIONARY 'DISK1:[DICTIONARY]CORP.DEPT3'
SET EDIT KEEP 10
INVOKE DATABASE PERS =
  PATHNAME 'PERSONNEL'
PRINT " "
PRINT "Rdb/VMS is ready to grant your every wish, using a "
SHOW DATABASES
$ RDO

Rdb/VMS is ready to grant your every wish, using a
Database with db_handle PERS in file PERSONNEL
RDO> FINISH
RDO> EXIT
```

SET Statement

```
$ TYPE LOG.LIS
SET DICTIONARY 'DISK1:[DICTIONARY]CORP.DEPT3'
SET EDIT KEEP 10
INVOKE DATABASE PERS =
  PATHNAME 'PERSONNEL'
PRINT " "

PRINT "Rdb/VMS is ready to grant your every wish, using a"
  Rdb/VMS is ready to grant your every wish, using a
SHOW DATABASES
Database with db_handle PERS in file PERSONNEL
PRINT " "

FINISH
EXIT
```

These statements work like this:

- Because this command procedure file is the RDOINI file, it runs automatically when you invoke RDO.
- The SET OUTPUT statement opens a file called LOG.LIS in the current default directory. From this point on, all the input and output, including error messages, appear in this file.
- The SET DICTIONARY statement changes the default data dictionary directory.
- The SET EDIT KEEP statement specifies that you get 10 previous statements in the editing buffer when you type EDIT *.
- The INVOKE DATABASE statement invokes the PERSONNEL database with the database handle of PERS.
- The PRINT and SHOW DATABASES statements display a message to the user.

Example 2

The following example shows the SET statements you can use to internationalize your application.

SET Statement

```
RDO> !
RDO> ! First, use the PRINT statement to see the default output:
RDO> !
RDO> FOR FIRST 5 E IN SALARY_HISTORY
RDO> PRINT E.SALARY_START
RDO> END_FOR
SALARY_START
  5-JUL-1980 00:00:00.00
 14-JAN-1983 00:00:00.00
   2-MAR-1981 00:00:00.00
 21-SEP-1981 00:00:00.00
   3-NOV-1981 00:00:00.00
RDO> !
RDO> ! The SET DATE_FORMAT statement customizes the output of the date
RDO> ! format. The output will appear in the form 5 JUL 80, as specified
RDO> ! by the date-number argument 6:
RDO> !
RDO> SET DATE_FORMAT DATE 6
RDO> !
RDO> FOR FIRST 5 E IN SALARY_HISTORY
RDO> PRINT E.SALARY_START
RDO> END_FOR
SALARY_START
   5 JUL 80
  14 JAN 83
   2 MAR 81
  21 SEP 81
   3 NOV 81
RDO> !
RDO> ! The SET DATE_FORMAT TIME statement customizes
RDO> ! the output of the time format. The output will appear
RDO> ! in the form 16 h 30 min 0 s, as specified by the
RDO> ! time-number argument 20:
RDO> !
RDO> SET DATE_FORMAT TIME 20
RDO> !
RDO> FOR FIRST 5 E IN SALARY_HISTORY
RDO> PRINT E.SALARY_START
RDO> END_FOR
SALARY_START
  0 h 0 min 0 s
  0 h 0 min 0 s
RDO> !
RDO> ! Note that the previous date example has dropped
RDO> ! the time output, and the previous time example has
RDO> ! dropped the date output.
RDO> !
```

SET Statement

```
RDO> ! If you want the display to continue to show
RDO> ! BOTH date and time, you must specify
RDO> ! both arguments with the SET DATE_FORMAT statement:
RDO> !
RDO> SET DATE_FORMAT DATE 6 TIME 20
RDO> !
RDO> FOR FIRST 5 E IN SALARY_HISTORY
RDO> PRINT E.SALARY_START
RDO> END_FOR
SALARY_START
  5 JUL 80 0 h 0 min 0 s
 14 JAN 83 0 h 0 min 0 s
   2 MAR 81 0 h 0 min 0 s
 21 SEP 81 0 h 0 min 0 s
   3 NOV 81 0 h 0 min 0 s
RDO> !
RDO> ! Set the language to FRENCH and see how the output changes.
RDO> !
RDO> SET LANGUAGE FRENCH
RDO> FOR FIRST 10 E IN SALARY_HISTORY
RDO> PRINT E.SALARY_START
RDO> END_FOR
SALARY_START
  5 jul 80 0 h 0 min 0 s
 14 jan 83 0 h 0 min 0 s
   2 mar 81 0 h 0 min 0 s
 21 sep 81 0 h 0 min 0 s
   3 nov 81 0 h 0 min 0 s
   1 jul 82 0 h 0 min 0 s
 27 jan 81 0 h 0 min 0 s
   1 jul 75 0 h 0 min 0 s
 29 déc 78 0 h 0 min 0 s
   2 fév 80 0 h 0 min 0 s
RDO> !
RDO> ! The next two examples show how to change the radix point from a
RDO> ! period to a comma using the SET RADIX_POINT statement. Notice
RDO> ! how the output changes:
RDO> !
RDO> FOR FIRST 5 E IN SALARY_HISTORY
cont> PRINT E.SALARY_AMOUNT
cont> END_FOR
SALARY_AMOUNT
 26291.00
  51712.00
 26291.00
 50000.00
 11676.00
```

SET Statement

```
RDO> !
RDO> SET RADIX_POINT ,
RDO> FOR FIRST 5 E IN SALARY_HISTORY
RDO> PRINT E.SALARY_AMOUNT
RDO> END_FOR
  SALARY_AMOUNT
      26291,00
      51712,00
      26291,00
      50000,00
      11676,00
RDO> ROLLBACK
```

SHOW Statements

9.61 SHOW Statements

Displays information about the Rdb/VMS database and database entities. The SHOW TRANSFER statement lists information about the VAX Data Distributor transfer operations. Table 9–10 lists the statements and describes their functions.

Note that you must invoke a database before you enter a SHOW statement; three exceptions are the SHOW DATABASES ALL, SHOW DICTIONARY, and SHOW VERSIONS statements, which do not require an invoked database. If you do not invoke a database before you enter all other SHOW statements, you will get the following error:

```
$ RDO
RDO> show fields for salary_history
%RDO-F-INVNOTDON, no DATABASE invoked yet, please issue a DATABASE command
```

Table 9–10 Rdb/VMS SHOW Statements

Statement	Description
SHOW ALL	Lists information that is equivalent to a combination of SHOW DATABASES, RELATIONS, FIELDS, INDEXES, COLLATING_SEQUENCE, CONSTRAINTS, STORAGE AREAS, STORAGE MAPS, and TRIGGERS for the most recently invoked database.
SHOW COLLATING_SEQUENCE	Lists collating sequences for invoked databases and fields in the most recently invoked database.
SHOW CONSTRAINTS	Lists constraint names and definitions for named relation or for all relations in most recently invoked database.
SHOW DATABASES	Shows all invoked databases, with database handles and file names. Using the optional ALL clause displays the path names of all databases defined at the current dictionary directory.

(continued on next page)

SHOW Statements

Table 9-10 (Cont.) Rdb/VMS SHOW Statements

Statement	Description
SHOW DATE_FORMAT	Displays the value for the date-number argument and time-number argument of the SET DATE_FORMAT statement.
SHOW DICTIONARY	Shows current default directory in data dictionary.
SHOW FIELDS	Lists globally defined fields in most recently invoked database, with data types. Adding optional clauses lets you list information about a particular globally defined field or list attributes about one or more fields in a specified relation.
SHOW INDEXES	Lists index names and definitions for named relation or for all relations in most recently invoked database. Includes DUPLICATES ARE ALLOWED information.
SHOW LANGUAGE	Displays the language that will be used for date and time input and display.
SHOW PRIVILEGES	Displays your access control list (ACL) entry when Rdb/VMS matches your user identification code (UIC) with the identifier specified in the ACL entry.
SHOW PROTECTION	Lists all ACL entries for the specified database, relation, field, or view.
SHOW RADIX_POINT	Displays the character that will be used as the radix point in output displays.
SHOW RELATIONS	Lists the user relations and relation-specific constraint sin the most recently invoked database. Using the SYSTEM clause displays the Rdb/VMS system relations. Other optional clauses let you list information about a specific relation, or all relations, or all relations in all databases you invoked in RDO.
SHOW STORAGE AREAS	Lists names of all storage areas in most recently invoked database. Other optional clauses let you display information about a particular named storage area.

(continued on next page)

SHOW Statements

Table 9-10 (Cont.) Rdb/VMS SHOW Statements

Statement	Description
SHOW STORAGE MAPS	Lists names of storage maps in most recently invoked database. Other optional clauses let you display information about a particular storage map.
SHOW STREAMS	Lists all open streams or indicates that no streams are active.
SHOW TRANSACTION	Lists the transaction mode, the transaction sequence number, and the session ID number for the current RDO transaction.
SHOW TRANSFER	Displays information about the VAX Data Distributor transfer definition, schedule definition or the transfer status, or both.
SHOW TRIGGERS	Displays names and definitions of the specified triggers for a specified database or all invoked databases.
SHOW VERSIONS	Displays all version numbers of the Rdb/VMS components you are running.

See the subsequent SHOW statement sections for details on each statement.

Note The definitions appearing in the SHOW examples in the following sections are in RDO format because it is assumed that RDO was the source language used to create the definitions. If you use the RDO SHOW statement to view definitions created with SQL, the definitions may be displayed in SQL format (the source language of the definitions) instead of in RDO format.

SHOW ALL Statement

9.61.1 SHOW ALL Statement

Displays information about the system and user relations, fields, indexes, collating sequences, constraints, storage areas, storage maps, and triggers for the most recently invoked database. If the optional `IN ALL DATABASES` clause is used, the `SHOW ALL` statement displays this information for all databases you have invoked in RDO.

Format



Argument

db-handle

A host language variable that you associate with the name of the database in the `INVOKE DATABASE` statement.

Usage Notes

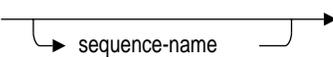
You must have the Rdb/VMS `READ` privilege to the database to use the `SHOW ALL` statement.

SHOW COLLATING_SEQUENCE Statement

9.61.2 SHOW COLLATING_SEQUENCE Statement

Displays the collating sequences for databases and fields.

Format

SHOW COLLATING_SEQUENCE 

Argument

sequence-name

Specify the name of the collating sequence you wish to display.

Usage Notes

You must have the Rdb/VMS READ privilege to the database to use the SHOW COLLATING_SEQUENCE statement.

Examples

Example 1

The following example displays the collating sequence for the database TEST:

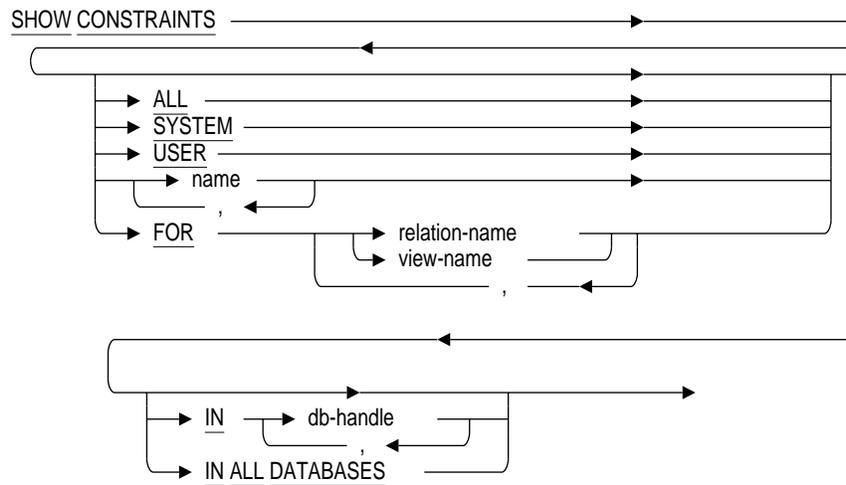
```
RDO> DEFINE DATABASE TEST DICTIONARY IS NOT USED
cont> COLLATING_SEQUENCE IS FRENCH FRENCH.
RDO> SHOW COLLATING_SEQUENCE
User Collating Sequences in Database with db_handle TEST
FRENCH
```

SHOW CONSTRAINTS Statement

9.61.3 SHOW CONSTRAINTS Statement

Displays constraint names and definitions for the invoked database, for a specified relation or view, or for all invoked databases.

Format



Arguments

relation-name

The name of the relation for which you want to list constraint definitions.

view-name

The name of the view for which you want to list constraint definitions.

db-handle

A host language variable that you associate with the name of the database in the `INVOKE DATABASE` statement.

SHOW CONSTRAINTS Statement

Usage Notes

You must have the Rdb/VMS READ privilege for the database to use the SHOW CONSTRAINTS statement.

The optional ALL, SYSTEM, and USER clauses are permitted syntactically; however, the display for each clause is identical. All user-defined constraints are listed.

Examples

Example 1

The following example lists the constraints defined for the EMPLOYEES relation:

```
RDO> SHOW CONSTRAINTS FOR EMPLOYEES
Constraints for relation  EMPLOYEES
EMPLOYEE_ID_REQUIRED

                                FOR E IN EMPLOYEES
                                REQUIRE NOT E.EMPLOYEE_ID MISSING.

JH_EMP_ID_EXISTS

                                FOR JH IN JOB_HISTORY
                                REQUIRE ANY E IN EMPLOYEES WITH
                                E.EMPLOYEE_ID = JH.EMPLOYEE_ID
                                CHECK ON COMMIT.

SH_EMP_ID_EXISTS

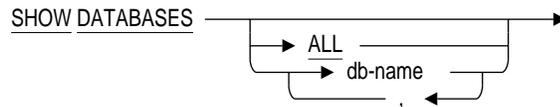
                                FOR SH IN SALARY_HISTORY
                                REQUIRE ANY E IN EMPLOYEES WITH
                                E.EMPLOYEE_ID = SH.EMPLOYEE_ID
                                CHECK ON COMMIT.
```

SHOW DATABASES Statement

9.61.4 SHOW DATABASES Statement

Displays the full file specifications for the currently invoked databases. If you invoked the database by path name, the path name for the currently invoked database is displayed. If you used a database handle when you invoked the database, the database handle name is displayed.

Format



Arguments

ALL

Directs Rdb/VMS to display the path name of all databases defined at the current directory in the data dictionary. You do not have to invoke a database before you use the SHOW DATABASES ALL statement.

db-name

The name of the database. If the database resides on a different device, in a different directory, or both, specify the complete file specification.

Usage Notes

You must have the Rdb/VMS READ privilege for a database to display information about the database with the SHOW DATABASES statement.

SHOW DATABASES Statement

Examples

Example 1

In the following example, the SHOW DATABASES statement is used to display information about the invoked databases:

```
RDO> INVOKE DATABASE DB1 = FILENAME 'WORK$DISK:PERSONNEL'  
RDO> INVOKE DATABASE DB2 = FILENAME 'WORK$DISK:BENEFITS'  
  
.  
.  
.  
RDO> SHOW DATABASES  
Database with db_handle DB1 in file WORK$DISK:PERSONNEL  
Database with db_handle DB2 in file WORK$DISK:BENEFITS  
RDO>
```

Example 2

If the database is accessed remotely, the SHOW DATABASES statement will display the remote node name.

```
RDO> DATABASE FILE 'VAXA"SMITH SECRET"::DISK1:[SMITH.DATABASES]PM'  
RDO> SHOW DATABASE RDB$DBHANDLE  
Database with db_handle RDB$DBHANDLE (default handle)  
File: DISK1:[SMITH.DATABASES]PM.RDB;1  
This database is accessed remotely on node VAXA  
Default segmented string storage area: RDB$SYSTEM  
Number of users: 4  
Number of nodes: 16  
Buffer Size (blocks/buffer): 6  
Number of Buffers: 20  
Number of Recovery Buffers: 20  
Snapshots are Enabled Immediate  
Dictionary Not Required  
ACL based protections
```

SHOW DATE_FORMAT Statement

9.61.5 SHOW DATE_FORMAT Statement

Displays the value for the output date-number and time-number argument of the SET DATE_FORMAT statement.

SHOW DATE_FORMAT also displays the current date input format, which cannot be set from within RDO.

Format

SHOW DATE_FORMAT

Usage Notes

You can use the SHOW DATE_FORMAT statement without invoking a database, so you do not need any special Rdb/VMS privileges to use the statement.

Examples

Example 1

The following example shows the SET and SHOW DATE_FORMAT statements.

```
RDO> !
RDO> ! Display the default date and time formats:
RDO> !
RDO> SHOW DATE_FORMAT
Output date and time formats are:
    DATE = 1 (for example: 21-JUN-1990)
    TIME = 1 (for example: 14:14:00.43)
Input date and time format is:
    DD-MONTH-YYYY4 HH:MM:SS.CC2
```

SHOW DATE_FORMAT Statement

```
RDO> !
RDO> ! Exit and return to DCL level.
RDO> !
RDO> EXIT
$ !
$ ! Use the LIB$DT_FORMAT logical name to set the date and time
$ ! format:
$ !
$ DEFINE LIB$DT_FORMAT LIB$DATE_FORMAT_012, LIB$TIME_FORMAT_011
$ !
$ ! Invoke RDO and display the changed date and time formats:
$ !
$ RUN SYS$SYSTEM:RDO
RDO> !
RDO> SHOW DATE_FORMAT
Output date and time formats are:
    DATE = 12 (for example: JUNE 21, 1990)
    TIME = 11 (for example: 02:14 PM)
Input date and time format is:
    DD-MONTH-YYYY4 HH:MM:SS.CC2
RDO> !
RDO> ! In addition to using the LIB$DT_FORMAT logical name at
RDO> ! DCL level, you can use the RDO SET DATE_FORMAT statement
RDO> ! to set the date and time formats. This example changes
RDO> ! them to 6 and 8, respectively:
RDO> !
RDO> SET DATE_FORMAT DATE 6 TIME 8
RDO> !
RDO> ! Use the RDO SHOW DATE_FORMAT statement to see the date and
RDO> ! time format:
RDO> !
RDO> SHOW DATE_FORMAT
Output date and time formats are:
    DATE = 6 (for example: 21 JUN 90)
    TIME = 8 (for example: 14:14)
Input date and time format is:
    DD-MONTH-YYYY4 HH:MM:SS.CC2
RDO> !
RDO> EXIT
```

SHOW DICTIONARY Statement

9.61.6 SHOW DICTIONARY Statement

Displays the current default directory in the data dictionary.

Format

SHOW DICTIONARY

Usage Notes

You can use the `SHOW DICTIONARY` statement without invoking a database, so you do not need any special Rdb/VMS privileges to use the statement.

Examples

Example 1

The following example displays information about the current default within the data dictionary:

```
RDO> SHOW DICTIONARY
The current CDD dictionary is  DISK1:[DICTIONARY]DEPT3
```

SHOW FIELDS Statement

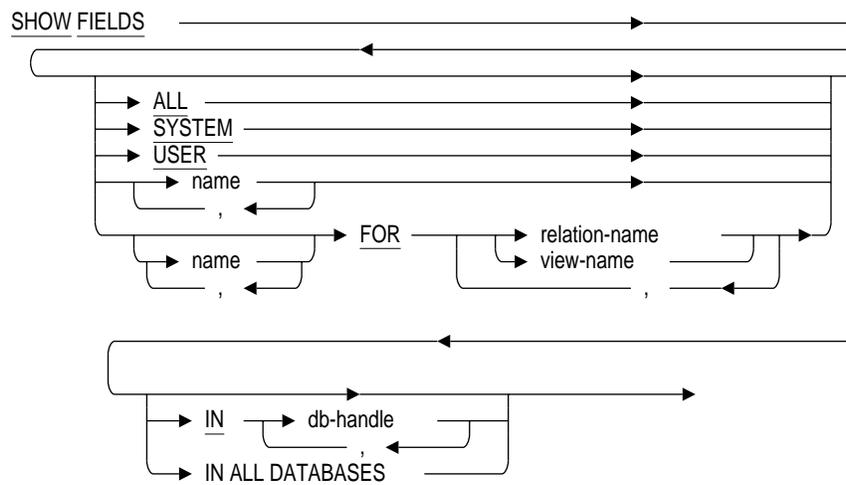
9.61.7 SHOW FIELDS Statement

Displays information about the system or user-defined fields:

- In the currently invoked database
- In all databases invoked
- For a specified relation

The SHOW FIELDS statement can also display information about a specified global field or about one or more fields in a specified relation.

Format



Arguments

relation-name

The name of the relation for which you want to list one or more fields.

view-name

The name of the view for which you want to list one or more fields.

SHOW FIELDS Statement

db-handle

A host language variable that you associate with the name of the database in the INVOKE DATABASE statement.

Usage Notes

You must have the Rdb/VMS READ privilege for a database to display information about a field with the SHOW FIELDS statement.

Examples

Example 1

The following example demonstrates several uses of the SHOW FIELDS statement:

```
RDO> INVOKE DATABASE FILENAME "WORK$DISK:PERSONNEL"
RDO> SHOW FIELDS SYSTEM
System Fields in Database with filename work$disk:personnel
  RDB$ACCESS_CONTROL          segmented string of type: TEXT
                              segment_length 512
  RDB$BASE_FIELD              text size is 31
  RDB$CARDINALITY             signed longword scale 0
  RDB$CDD_NAME                segmented string of type: TEXT
                              segment length 512
  RDB$CDD_PATH                text size is 256
  .
  .
  .
  RDBVMS$TRIGGER_NEW_CONTEXT  signed word scale 0
  RDBVMS$TRIGGER_OLD_CONTEXT  signed word scale 0
  RDBVMS$TRIGGER_SOURCE       segmented string of type: TEXT
                              segment length 512
  RDBVMS$TRIGGER_TYPE         signed word scale 0
  RDBVMS$USAGE                text size is 31
  RDBVMS$VERTICAL_PARTITION_INDEX signed word scale 0
RDO> !
RDO> SHOW FIELD RDBVMS$USAGE
RDBVMS$USAGE                  text size is 31
RDO> !
RDO> SHOW FIELD EMPLOYEE_ID, LAST_NAME FOR EMPLOYEES
Fields for relation EMPLOYEES
  EMPLOYEE_ID                 text size is 5
                              based on global field ID_NUMBER
Fields for relation EMPLOYEES
  LAST_NAME                   text size is 14
```

SHOW FIELDS Statement

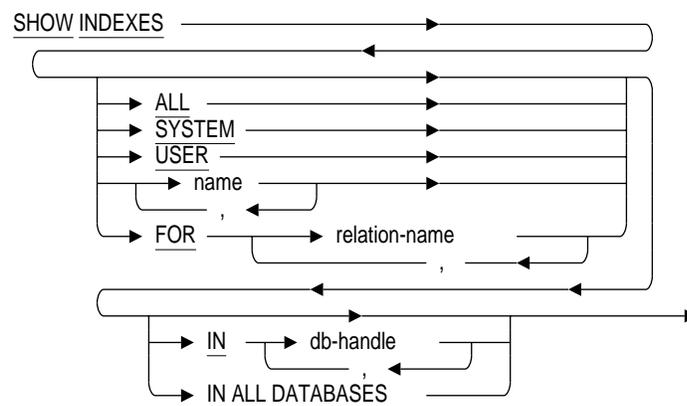
```
RDO> !
RDO> SHOW FIELDS FOR SALARY_HISTORY
Fields for relation SALARY_HISTORY
EMPLOYEE_ID          text size is 5
  based on global field ID_NUMBER
SALARY_AMOUNT        signed longword scale -2
  based on global field SALARY
SALARY_START         date
  based on global field STANDARD_DATE
SALARY_END           date
  based on global field STANDARD_DATE
RDO>
```

SHOW INDEXES Statement

9.61.8 SHOW INDEXES Statement

Displays index names and definitions for the invoked database, for a specified relation, or for all invoked databases. When you enter the SHOW INDEXES statement without any clauses, all the user-defined indexes in the currently invoked database are displayed. The Rdb/VMS system indexes are displayed when you specify the SYSTEM clause. Both the system and user-defined indexes are displayed when you specify the ALL clause.

Format



Arguments

relation-name

The name of the relation for which you want to list index definitions.

db-handle

A host language variable that you associate with the name of the database in the INVOKE DATABASE statement.

SHOW INDEXES Statement

Usage Notes

To use the SHOW INDEXES statement to display information about an index, you must have the Rdb/VMS READ privilege for the database.

The SHOW INDEXES statement is the same as the SHOW INDICES statement.

The values for an index's node size, initial fullness percentage, and description text are displayed *only* after you modify these values with a CHANGE INDEX statement. See Section 9.4 and its examples for details.

When you specify the index name in the SHOW statement, the text of the DESCRIPTION IS clause is displayed for the index. When you issue a SHOW INDEXES statement without any clauses, the text of the DESCRIPTION IS clause is *not* displayed.

When you issue a SHOW statement against a database defined using another interface, such as SQL, the output from the SHOW statement will contain statements from that interface rather than RDO statements.

Examples

Example 1

The following examples demonstrate several variations of the SHOW INDEXES statement:

```
RDO> SHOW INDEX SH_EMPLOYEE_ID

  Indexes for relation  SALARY_HISTORY
  in database with handle MF_PERSONNEL
SH_EMPLOYEE_ID          with field  EMPLOYEE_ID
                        Duplicates are allowed
                        Type is Sorted

RDO> ! Start a read/write transaction and change the SH_EMPLOYEE_ID index:
RDO> !
RDO> START_TRANSACTION READ_WRITE
RDO> !
RDO> CHANGE INDEX SH_EMPLOYEE_ID
cont> DESCRIPTION IS /* Test new parameters */
cont> NODE SIZE 600
cont> PERCENT FILL 85.
RDO> !
RDO> ! After changing the index, the node size, initial fullness percentage,
RDO> ! and description text are displayed with the SHOW INDEX statement:
RDO> SHOW INDEX SH_EMPLOYEE_ID
```

SHOW INDEXES Statement

```
Indexes for relation SALARY_HISTORY
in database with handle MF_PERSONNEL
SH_EMPLOYEE_ID          with field EMPLOYEE_ID
                        Duplicates are allowed
                        Type is Sorted
                        Node size 600
                        Percent fill 85
Description:           Test new parameters

RDO> ROLLBACK

RDO> ! Because no clauses are specified with the following statement, no
RDO> ! description clauses are displayed:
RDO> SHOW INDEXES
User Indexes in Database with filename MF_PERSONNEL

Indexes for relation COLLEGES
COLL_COLLEGE_CODE       with field COLLEGE_CODE
                        No duplicates allowed
                        Type is Sorted

Indexes for relation DEGREES
DEG_COLLEGE_CODE        with field COLLEGE_CODE
                        Duplicates are allowed
DEG_EMP_ID              with field EMPLOYEE_ID
                        Duplicates are allowed
                        Type is Sorted

Indexes for relation DEPARTMENTS
DEPARTMENTS_INDEX      with field DEPARTMENT_CODE
                        No duplicates allowed
                        Type is Sorted
Description:           sorted index for departments
Source:               WITHIN DEPARTMENTS

Indexes for relation EMPLOYEES
EMPLOYEES_HASH          with field EMPLOYEE_ID
                        No duplicates allowed
                        Type is Hashed
Description:           hash index for employees
Source:               USING EMPLOYEE_ID
                        WITHIN
                        EMPIDS_LOW WITH LIMIT OF "00200";
                        EMPIDS_MID WITH LIMIT OF "00400";
                        EMPIDS_OVER;
EMP_EMPLOYEE_ID         with field EMPLOYEE_ID
                        No duplicates allowed
                        Type is Sorted
```

SHOW INDEXES Statement

```
Indexes for relation JOB_HISTORY
JH_EMPLOYEE_ID          with field EMPLOYEE_ID
                        Duplicates are allowed
                        Type is Sorted
JOB_HISTORY_HASH        with field EMPLOYEE_ID
                        Duplicates are allowed
                        Type is Hashed
Description:            hash index for job_history
Source:                 USING EMPLOYEE_ID
                        WITHIN
                        EMPIDS_LOW WITH LIMIT OF "00200";
                        EMPIDS_MID WITH LIMIT OF "00400";
                        EMPIDS_OVER;

Indexes for relation SALARY_HISTORY
SH_EMPLOYEE_ID          with field EMPLOYEE_ID
                        Duplicates are allowed
                        Type is Sorted
```

SHOW LANGUAGE Statement

9.61.9 SHOW LANGUAGE Statement

Displays the language to be used for translation of month names and abbreviations in date and time input and display. The language name also determines the translation of other language-dependent text, such as the translation for the date literals YESTERDAY, TODAY, and TOMORROW.

Format

SHOW LANGUAGE

Usage Notes

You can use the SHOW LANGUAGE statement without invoking a database, so you do not need any special Rdb/VMS privileges to use the statement.

Examples

Example 1

The following example first shows setting the language to Swedish. It then shows the output of the SHOW LANGUAGE statement.

```
!  
! First, use the SET statement to specify a value that is not  
! the default:  
!  
RDO> SET LANGUAGE SWEDISH  
!  
! Now, use the SHOW statement to see that value:  
!  
RDO> SHOW LANGUAGE  
Language in use for date and time input/output formatting is SWEDISH  
!
```

SHOW PRIVILEGES Statement

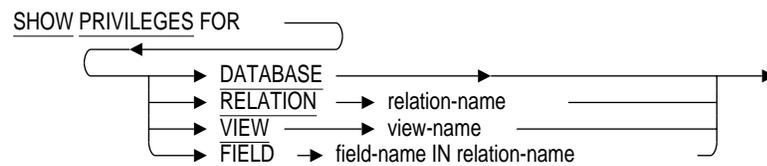
9.61.10 SHOW PRIVILEGES Statement

Displays your access control list (ACL) entry when Rdb/VMS matches your UIC with the identifier specified in the ACL entry (ACE). Although your UIC may match many ACL entries, Rdb/VMS grants you those access rights when it finds the first match between your identifier and an entry in the access control list.

You can display your particular privileges for the database, a relation, or a view. Note the difference between the SHOW PRIVILEGES and SHOW PROTECTION statements. The SHOW PROTECTION statement displays all ACL entries. The SHOW PRIVILEGES statement displays only your ACE or the privileges you have to a database object as a result of holding one or more of the VMS or Rdb/VMS overriding privileges.

For information about what privileges are needed to perform specific database tasks, see Table 9-6 in Section 9.17.

Format



Arguments

relation-name

The name of the Rdb/VMS relation for which you want to display your ACL entry.

view-name

The name of the Rdb/VMS view for which you want to display your ACL entry.

field-name IN relation-name

The name of the local field in a specified relation for which you want to display your ACL entry.

SHOW PRIVILEGES Statement

Usage Notes

You must have the Rdb/VMS READ privilege for a database to use the SHOW PRIVILEGES statement.

You must invoke a database before you issue the SHOW PRIVILEGES statement.

The display for the SHOW PRIVILEGES statement reflects information stored following a COMMIT or ROLLBACK statement and a detach from the database with a FINISH statement. Unlike the SHOW PROTECTION statement, SHOW PRIVILEGES does not reflect uncommitted changes. Any changes you make to your privileges or those of other users do not take effect until you detach from the database.

To perform certain operations you must have the correct access mode privilege (READ, WRITE, MODIFY, or ERASE) on both the database and the relation. Therefore, the SHOW PRIVILEGES display for a relation drops any access mode privileges that are not present for the database before displaying the privileges for the relation.

If you hold one or more of the VMS override privileges (SYSPRV, OPER, or SECURITY) or one or more of the Rdb/VMS role-oriented privileges (ADMINISTRATOR, OPERATOR, or SECURITY), you are implicitly granted privileges to database objects as a result of an ACL override. You operate as if you actually hold the privileges you are implicitly granted, even though these privileges are not stored in the ACL. The SHOW PRIVILEGES statement displays the privileges you have to a database object as a result of holding the VMS override privilege or Rdb/VMS role-oriented privilege. See Section 9.17 for more information on VMS override privileges and Rdb/VMS role-oriented privileges.

The SHOW PRIVILEGES statement displays only those privileges that are valid for the database object. For example, although you may hold the Rdb/VMS ADMINISTRATOR, OPERATOR, or SECURITY database privileges, these privileges are not displayed when you issue the SHOW PRIVILEGES statement for a relation because they are not relation privileges.

SHOW PRIVILEGES Statement

Examples

Example 1

The following example shows the difference between the SHOW PROTECTION and SHOW PRIVILEGES statements. The first statement displays all ACL entries for the relation EMPLOYEES. The second statement displays only the access rights for the current user with UIC [ADMIN,JONES].

```
RDO> INVOKE DATABASE FILENAME 'MF_PERSONNEL'
RDO> SHOW PROTECTION FOR RELATION EMPLOYEES
  (IDENTIFIER=[GROUP3,QUINN], ACCESS=READ+WRITE+MODIFY+ERASE+
    DEFINE+CONTROL+CHANGE+DELETE+REFERENCES+SECURITY+DISTRIBTRANS)
  (IDENTIFIER=[ADMIN,JONES], ACCESS=READ+WRITE+MODIFY+ERASE)
  (IDENTIFIER=[ADMIN,SMITH], ACCESS=READ+DELETE)
  (IDENTIFIER=[ADMIN,*], ACCESS=READ)

RDO> SHOW PRIVILEGES FOR RELATION EMPLOYEES
  (IDENTIFIER=[ADMIN,JONES], ACCESS=READ+WRITE+MODIFY+ERASE)
```

SHOW PROTECTION Statement

9.61.11 SHOW PROTECTION Statement

Displays all access control list (ACL) entries for the specified database, relation, or view. Note that you can use the VIEW keyword interchangeably with the RELATION keyword in the following protection statements:

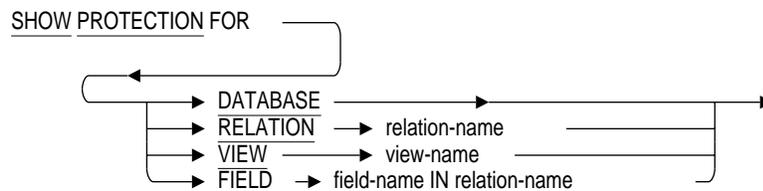
- SHOW PROTECTION
- DEFINE PROTECTION
- CHANGE PROTECTION
- DELETE PROTECTION

Thus, the following two statements produce identical results:

- SHOW PROTECTION FOR VIEW COLLEGES
- SHOW PROTECTION FOR RELATION COLLEGES

For information about what privileges are needed to perform specific database tasks, see Table 9-6 in the DEFINE PROTECTION statement section.

Format



Arguments

relation-name

The name of the Rdb/VMS relation for which you want to display all ACL entries.

view-name

The name of the Rdb/VMS view for which you want to display all ACL entries.

SHOW PROTECTION Statement

field-name IN relation-name

The name of the local field in a specified relation for which you want to display all ACL entries.

Usage Notes

To use the SHOW PROTECTION statement to display the protection for a database, relation, view, or field, you must have the Rdb/VMS READ privilege to the database.

The SHOW PROTECTION display reflects changes made prior to a COMMIT or ROLLBACK statement.

Examples

Example 1

The following example displays all ACL entries for the SALARY_HISTORY relation:

```
RDO> SHOW PROTECTION FOR RELATION SALARY_HISTORY
( IDENTIFIER=[ SCHWARTZ ] , ACCESS=READ+WRITE+MODIFY+ERASE+SHOW+DEFINE
+CHANGE+DELETE+CONTROL+OPERATOR+ADMINISTRATOR+REFERENCES+SECURITY
+DISTRIBTRANS )
( IDENTIFIER=[ * , * ] , ACCESS=NONE )
```

SHOW RADIX_POINT Statement

9.61.12 SHOW RADIX_POINT Statement

Displays the character that will be used as the radix point in output displays.

The radix point is the symbol that separates units from decimal fractions. For example, in the number 98.6, the period is the radix point.

Format

```
SHOW RADIX_POINT
```

Usage Notes

You can use the SHOW RADIX_POINT statement without invoking a database, so you do not need any special Rdb/VMS privileges to use the statement.

Examples

Example 1

The following example first shows setting the radix point to the comma. It then shows the output of the SHOW RADIX_POINT statement.

```
!  
! First, use the SET statement to specify a value other than  
! the default:  
!  
RDO> SET RADIX_POINT ,  
!  
! Now, use the SHOW RADIX_POINT statement to see that value:  
!  
RDO> SHOW RADIX_POINT  
Radix string in use is ,  
!
```

SHOW RELATIONS Statement

9.61.13 SHOW RELATIONS Statement

Displays the name of:

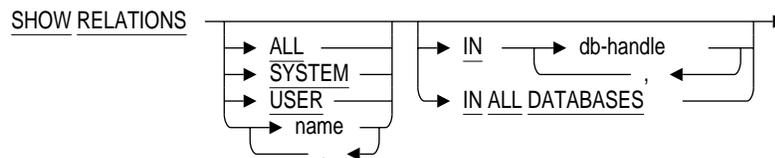
- All user-defined relations (and views)
- All system-defined relations
- The specified relation or view or both

By default, this information is returned for the relations and views in the currently invoked database. You can use the `IN ALL DATABASES` clause to return information about relations and views in all the invoked databases.

Views are included in the list of user-defined relations. The “A view.” comment appears in the display for views.

Use the `SHOW FIELDS FOR relation-name` statement when you want a list of the fields defined for a particular relation.

Format



Arguments

name

The name of the relation.

db-handle

A host language variable that you associate with the name of the database in the `INVOKE DATABASE` statement.

SHOW RELATIONS Statement

Usage Notes

You must have the Rdb/VMS READ privilege for the database to use the SHOW RELATIONS statement.

Examples

Example 1

The following example lists the names of the system-defined and user-defined relations in the PERSONNEL database.

```
RDO> INVOKE DATABASE FILENAME 'WORK$DISK:PERSONNEL'  
RDO> SHOW RELATIONS ALL  
All Relations in Database with filename work$disk:personnel  
RDB$CONSTRAINTS  
RDB$CONSTRAINT_RELATIONS  
RDB$DATABASE  
RDB$FIELDS  
RDB$FIELD_VERSIONS  
RDB$INDEX_SEGMENTS  
RDB$INDICES  
RDB$RELATIONS  
RDB$RELATION_FIELDS  
RDB$VIEW_RELATIONS  
RDBVMS$COLLATIONS  
RDBVMS$INTERRELATIONS  
RDBVMS$PRIVILEGES  
RDBVMS$RELATION_CONSTRAINTS  
RDBVMS$RELATION_CONSTRAINT_FLDS  
RDBVMS$STORAGE_MAPS  
RDBVMS$STORAGE_MAP_AREAS  
RDBVMS$TRIGGERS  
CANDIDATES  
COLLEGES  
CURRENT_INFO A view.  
CURRENT_JOB A view.  
CURRENT_SALARY A view.  
DEGREES  
DEPARTMENTS  
EMPLOYEES  
JOBS  
JOB_HISTORY  
RESUMES  
SALARY_HISTORY  
WORK_STATUS
```

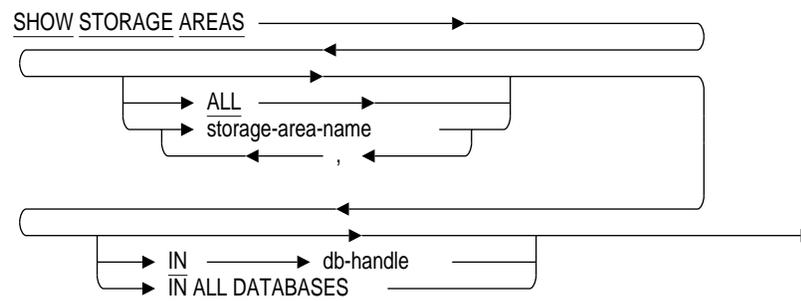
SHOW STORAGE AREAS Statement

9.61.14 SHOW STORAGE AREAS Statement

Displays the names of all storage areas for the currently invoked database, for a specific database, or for all invoked databases. If you specify a storage area name, the statement displays the following information about the storage area:

- Page format
- Identification of read-only areas
- Page size
- Storage file specification
- Storage file allocation, minimum extent, maximum extent, and percent growth
- Snapshot file specification
- Snapshot file allocation, minimum extent, maximum extent, and percent growth

Format



Arguments

storage-area-name

The name of the storage area for which you want to display the storage area characteristics.

SHOW STORAGE AREAS Statement

db-handle

A host language variable that you associate with the name of the database in the INVOKE DATABASE statement.

Usage Notes

You must have the Rdb/VMS READ privilege for a database to use the SHOW STORAGE AREAS statement.

Examples

Example 1

The following example displays information about the EMPIDS_LOW storage area:

```
RDO> SHOW STORAGE AREAS EMPIDS_LOW
EMPIDS_LOW
  Page format is mixed
  Page size is 2 blocks
  Storage file DDV12:[RICK.RDB]EMPIDS_LOW.RDA;1
  Allocation is 51 pages
  Minimum extent is 99 pages
  Maximum extent is 9999 pages
  Percent growth is 20%
  Snapshot file DDV12:[RICK.RDB]EMPIDS_LOW.SNP;1
  Allocation is 10 pages
  Minimum extent is 99 pages
  Maximum extent is 9999 pages
  Percent growth is 20%
```

Example 2

The following example displays information about all the storage areas in the MF_PERSONNEL database. The SHOW STORAGE AREAS statement display identifies any storage areas that were named in the SEGMENTED STRING STORAGE AREA clause of the DEFINE DATABASE statement:

```
RDO> SHOW STORAGE AREAS
All Storage Areas in Database with filename MF_PERSONNEL
RDB$SYSTEM
MF_PERS_SEGSTR           Default segmented string storage area
EMPIDS_LOW
EMPIDS_MID
```

SHOW STORAGE AREAS Statement

```
EMPIDS_OVER
DEPARTMENTS
SALARY_HISTORY
JOBS
EMP_INFO
```

Example 3

The following example displays information about the MF_PERS_SEGSTR storage area in the sample personnel database:

```
RDO> SHOW STORAGE AREAS MF_PERS_SEGSTR
MF_PERS_SEGSTR
  Default segmented string storage area
  Page format is uniform
  Page size is 2 blocks
  Storage file DDV12:[RICK.RDB]MF_PERS_SEGSTR.RDA;1
  Allocation is 51 pages
  Minimum extent is 99 pages
  Maximum extent is 9999 pages
  Percent growth is 20%
  Snapshot file DDV12:[RICK.RDB]MF_PERS_SEGSTR.SNP;1
  Allocation is 10 pages
  Minimum extent is 99 pages
  Maximum extent is 9999 pages
  Percent growth is 20%
```

Example 4

The CHANGE DATABASE statement in the following example changes the SALARY_HISTORY storage area to a read-only storage area, and the SHOW STORAGE AREAS statement displays that SALARY_HISTORY is a read-only storage area:

```
RDO> CHANGE DATABASE FILENAME 'MF_PERSONNEL'
cont> CHANGE STORAGE AREA SALARY_HISTORY
cont> READ_ONLY.
RDO> INVOKE DATABASE FILENAME 'MF_PERSONNEL'
```

SHOW STORAGE AREAS Statement

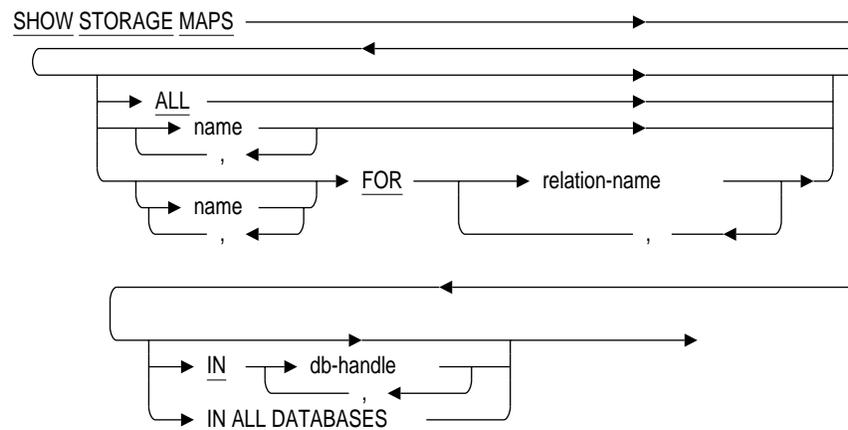
```
RDO> SHOW STORAGE AREAS SALARY_HISTORY
SALARY_HISTORY
    Page format is mixed
    Access is READ ONLY
    Page size is 2 blocks
    Storage file DDV12:[RICK.RDB]SALARY_HISTORY.RDA;1
Allocation is 126 pages
Minimum extent is 99 pages
Maximum extent is 9999 pages
Percent growth is 20%
Snapshot file DDV12:[RICK.RDB]SALARY_HISTORY.SNP;1
Allocation is 10 pages
Minimum extent is 99 pages
Maximum extent is 9999 pages
Percent growth is 20%
```

SHOW STORAGE MAPS Statement

9.61.15 SHOW STORAGE MAPS Statement

Displays the names of all storage maps for the currently invoked database, for a specific database, or for all invoked databases. You can also display storage map names for a specified relation. If you specify a storage map name, the statement displays information about the storage map.

Format



Arguments

name

The name of the storage map.

relation-name

The name of the relation for which you want to list one or more storage maps.

db-handle

A host language variable that you associate with the name of the database in the `INVOKE DATABASE` statement.

SHOW STORAGE MAPS Statement

Usage Notes

You must have the Rdb/VMS READ privilege for the database to display the relation's storage map with the SHOW STORAGE MAPS statement.

Examples

Example 1

The following example displays information about the storage map EMPLOYEES_MAP:

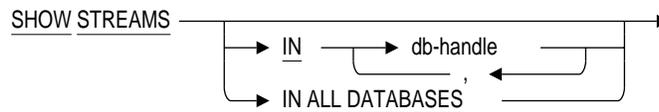
```
RDO> SHOW STORAGE MAP EMPLOYEES_MAP
EMPLOYEES_MAP
  Relation name is:    EMPLOYEES
  Source:             USING EMPLOYEE_ID
                    WITHIN
                    EMPIDS_LOW WITH LIMIT OF "00200";
                    EMPIDS_MID WITH LIMIT OF "00400";
                    EMPIDS_OVER
  Placement via:     EMPLOYEES_HASH
  Compression is:   Enabled
```

SHOW STREAMS Statement

9.61.16 SHOW STREAMS Statement

Identifies the streams that are currently open.

Format



Argument

db-handle

A host language variable that you associate with the name of the database in the INVOKE DATABASE statement.

Usage Notes

You must have the Rdb/VMS READ privilege to the database to use the SHOW STREAMS statement.

Examples

Example 1

The following example displays information about a stream called EMP_STREAM that the user creates with the START_STREAM statement:

```
RDO> INVOKE DATABASE FILENAME 'WORK$DISK:PERSONNEL'  
RDO> SHOW STREAMS  
No streams currently active  
RDO> !  
RDO> START_STREAM EMP_STREAM USING  
cont> E IN EMPLOYEES SORTED BY E.LAST_NAME  
RDO> !  
RDO> SHOW STREAMS  
Stream with name EMP_STREAM  
in Database with filename work$disk:personnel
```

SHOW TRANSACTION Statement

9.61.17 SHOW TRANSACTION Statement

Displays the following information about the current RDO transaction:

- **Transaction mode**
Rdb/VMS displays whether you started a read-only, read/write, or batch-update transaction.
- **Status of updates**
If you started a read/write transaction, Rdb/VMS displays whether updates have been applied to the database.
- **Full RUJ file specification**
Rdb/VMS displays the recovery-unit journal (RUJ) file specification if you have made an update to the database.
- **Transaction ID (TID) and transaction sequence number (TSN)**
The TID is a unique identifying number that Rdb/VMS assigns when you invoke a database. The TSN is a unique number that Rdb/VMS assigns when you start a transaction with the `START_TRANSACTION` statement.
- **Information regarding the snapshot file**

Format

SHOW TRANSACTION

Usage Notes

To use the `SHOW TRANSACTION` statement, you must have the Rdb/VMS `READ` privilege for a database.

SHOW TRANSACTION Statement

Examples

Example 1

The following example shows the information listed for an update transaction that has made changes to a database:

```
RDO> DATABASE FILENAME PERSONNEL
RDO> START_TRANS READ_WRITE RESERVING EMPLOYEES FOR EXCLUSIVE WRITE
RDO> STORE E IN EMPLOYEES USING E.EMPLOYEE_ID = "97532";
cont> E.LAST_NAME = "MINGOLD";
cont> END_STORE
RDO> SHOW TRANSACTION
All Transactions in Database with filename PERSONNEL
a read-write transaction is in progress
- updates have been performed
- transaction sequence number (TSN) is 185
- snapshot space for TSNs less than 185 can be reclaimed
- recovery unit journal filename is DISK$:[OPER]PERSONNEL.RUJ;5
- session ID number is 35
```

SHOW TRANSFER Statement (VAX Data Distributor)

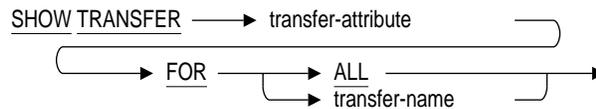
9.61.18 SHOW TRANSFER Statement (VAX Data Distributor)

Displays information about the transfer definition, schedule definition, the transfer status, or all these transfer attributes.

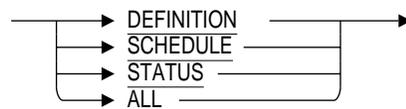
Note *To use this RDO statement, you must have VAX Data Distributor installed on your system. See your system manager or your Digital Equipment Corporation representative if you need information on Data Distributor.*

When you enter the SHOW TRANSFER statement, you can display information about one transfer or all of them, but not about a selected number of transfers.

Format



transfer-attribute =



Arguments

transfer-attribute

The type of information you want displayed about the selected transfer or all transfers. You can specify one or more of the following:

- **DEFINITION**

Displays transfer definition information for the specified transfer or for all transfers.

SHOW TRANSFER Statement (VAX Data Distributor)

- **SCHEDULE**
Displays schedule definition information for the specified transfer or for all transfers.
- **STATUS**
Displays the following status information for the specified transfer or for all transfers:
 - Transfer name
 - State of the current transfer (unscheduled, scheduled, active, waiting-to-retry, retrying, suspended)
 - Time the transfer entered its current transfer state
 - Last transfer status
 - Time the last successful transfer completed
 - Next scheduled transfer time (for transfers in the scheduled or waiting-to-retry state)
 - Number of the currently executing retry (for transfers in the retrying state)
 - Number of the next retry (for transfers in the waiting-to-retry state)
- **ALL**
Displays all information about the definition, schedule, and status for the specified transfer or for all transfers.

ALL

Specifies that you want Data Distributor to display information about all transfers associated with your UIC. For transfers belonging to other UICs, Data Distributor displays only the transfer name. To display information about transfers belonging to other UICs, you must have VMS BYPASS or READALL privilege.

transfer-name

The transfer about which you want Data Distributor to display information.

SHOW TRANSFER Statement (VAX Data Distributor)

Usage Notes

To use the SHOW TRANSFER statement, you must have the Rdb/VMS READ privilege for a database.

If you want to use the SHOW TRANSFER statement to display information about a particular transfer, the transfer definition must be associated with your UIC.

Examples

Example 1

This example displays information about the schedule for the NH_EMPLOYEES transfer. The transfer is scheduled to begin at 1:15 in the afternoon and to repeat every day. If the transfer fails, the transfer will retry three times with an hour between the retry attempts.

```
RDO> SHOW TRANSFER SCHEDULE FOR NH_EMPLOYEES

Transfer Schedule for NH_EMPLOYEES
  Start 14-MAY-1988 13:15:00.00
  Every  1 00:00:00.00
  Retry  3 Times
  Retry Every  0 01:00:00.00
```

Example 2

This example displays information about the definition, schedule, and status for the MA_EMPLOYEES transfer and shows the results of the statement.

```
RDO> SHOW TRANSFER ALL FOR MA_EMPLOYEES
-----
Transfer Definition for MA_EMPLOYEES
  Definer DBADMIN
  Extraction
  From DISK1:[DBADMIN.PERS]PERSONNEL.RDB
  To DISK2:[DBADMIN]MA_EMPLOYEES
  Move Relation e in employees with e.state = "MA"
  Select Fields E.LAST_NAME,
                E.FIRST_NAME,
                E.MIDDLE_INITIAL,
                E.STATE
  Log File is DISK1:[DBADMIN.PERS]MA_EMPLOYEES.LOG
  Noprologue
  Noepilogue
```

SHOW TRANSFER Statement (VAX Data Distributor)

```
-----  
Transfer Schedule for MA_EMPLOYEES  
  Start 28-MAR-1988 09:30:00.00  
  Every day at 06:00.00.00  
  Retry 7 times  
  Retry Every    0 00:30:00.00  
-----  
Transfer Status for MA_EMPLOYEES  
  Transfer State: SCHEDULED since 28-MAR-1988 14:31:28.34  
  Last transfer status: transfer was successful  
  Last successful transfer time    28-MAR-1988 14:30:00.00  
  Next transfer scheduled:         29-MAR-1988 06:00:00.00
```

Example 3

If you want information about all transfers associated with your UIC, use the **SHOW TRANSFER ALL FOR ALL** statement. Data Distributor displays output similar to the information in Example 2 for each transfer you have defined.

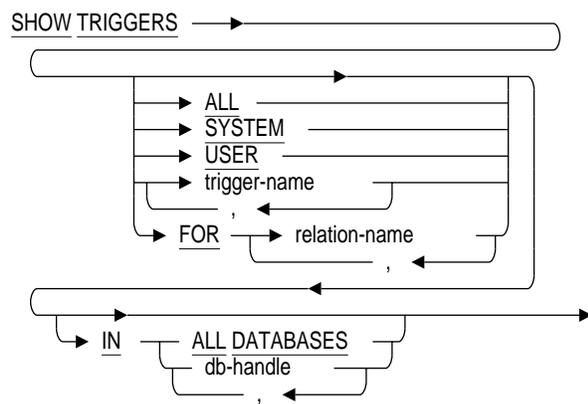
```
RDO> SHOW TRANSFER ALL FOR ALL  
.  
.  
.
```

SHOW TRIGGERS Statement

9.61.19 SHOW TRIGGERS Statement

Displays trigger names and definitions of the specified triggers for a specified database or all invoked databases.

Format



Arguments

trigger-name

The trigger name to be displayed.

relation-name

The name of the relation for which a listing of trigger definitions is desired.

Usage Notes

To display information about a trigger for a relation using the `SHOW TRIGGERS` command, you must have the Rdb/VMS `READ` privilege for the database.

SHOW TRIGGERS Statement

Examples

Example 1

The following example shows the triggers for the EMPLOYEES relation.

```
RDO> SHOW TRIGGERS FOR EMPLOYEES
Triggers for relation  EMPLOYEES
  EMPLOYEE_ID_CASCADE_DELETE
    Source:
        BEFORE ERASE
        FOR E IN EMPLOYEES EXECUTE
            FOR D IN DEGREES WITH
                D.EMPLOYEE_ID = E.EMPLOYEE_ID
            ERASE D
        END_FOR;
        FOR JH IN JOB_HISTORY WITH
            JH.EMPLOYEE_ID = E.EMPLOYEE_ID
        ERASE JH
        END_FOR;
        FOR R IN RESUMES WITH
            R.EMPLOYEE_ID = E.EMPLOYEE_ID
        ERASE R
        END_FOR;
        FOR JH IN JOB_HISTORY WITH
            JH.EMPLOYEE_ID = E.EMPLOYEE_ID
        ERASE JH
        END_FOR;
        FOR R IN RESUMES WITH
            R.EMPLOYEE_ID = E.EMPLOYEE_ID
        ERASE R
        END_FOR;
        FOR SH IN SALARY_HISTORY WITH
            SH.EMPLOYEE_ID = E.EMPLOYEE_ID
        ERASE SH
        END_FOR;
        ! Also, if an employee is terminated and that
        ! employee is the manager of a department, set
        ! the manager_id missing for that department.
        FOR D IN DEPARTMENTS WITH D.MANAGER_ID =
            E.EMPLOYEE_ID
            MODIFY D USING D.MANAGER_ID =
                RDB$MISSING (D.MANAGER_ID)  END_MODIFY
        END_FOR
    FOR EACH RECORD.
```

RDO>

SHOW TRIGGERS Statement

Example 2

The following example shows the triggers for the COLLEGES relation.

```
RDO> SHOW TRIGGERS FOR COLLEGES
Triggers for relation COLLEGES
COLLEGE_CODE_CASCADE_UPDATE
Source:
        BEFORE MODIFY OF COLLEGE_CODE NEW CONTEXT NEW_COL
        FOR OLD_COL IN COLLEGES EXECUTE
        FOR D IN DEGREES
            WITH D.COLLEGE_CODE = OLD_COL.COLLEGE_CODE
        MODIFY D USING
            D.COLLEGE_CODE = NEW_COL.COLLEGE_CODE
        END_MODIFY
        END_FOR
FOR EACH RECORD.
```

SHOW VERSIONS Statement

9.61.20 SHOW VERSIONS Statement

The SHOW VERSIONS statement displays all versions of the Rdb/VMS components you are running. If you invoke multiple databases, the SHOW VERSIONS statement displays component versions for each database you invoke.

If you do not invoke a database, the SHOW VERSIONS statement displays the version of RDO. If you invoke a database, the version of the shareable image is displayed as well as the RDO version.

Format

SHOW VERSIONS

Usage Notes

You can use the SHOW VERSIONS statement without invoking a database, so you do not need any special Rdb/VMS privileges to use the statement.

Examples

Example 1

The following example displays the version numbers of RDO that is part of the Rdb/VMS image and of the underlying shareable image.

```
RDO> INVOKE DATABASE FILENAME 'PERSONNEL'  
RDO> SHOW VERSIONS  
Current version of RDO is: Rdb/VMS V4.0-0  
Underlying versions are:  
Database with filename personnel  
    Rdb/VMS V4.0-0  
    Rdb/Disp V4.0-0
```

START_SEGMENTED_STRING Statement

9.62 START_SEGMENTED_STRING Statement

Starts a stream of segmented string segments within a record stream. Retrieving records with segmented string fields requires a segment stream within a record stream. This second stream is normally established by nesting a special segmented string FOR loop inside another FOR loop. However, in certain cases, you cannot or may not want to use FOR loops (as in Callable RDO programs).

The `START_SEGMENTED_STRING` statement is similar to the `START_STREAM` statement: it starts a stream of string segments. Unlike the `START_STREAM` statement, however, in a `START_SEGMENTED_STRING` statement, you do not use a `FETCH` statement to retrieve each segment. Instead, the `PRINT` or `GET` statement retrieves the segment and advances the pointer to the next segment. At the end of the stream, Rdb/VMS returns an end-of-stream message, `SEGSTR_EOF`.

To end the stream, use the `END_SEGMENTED_STRING` statement.

Format

```
START_SEGMENTED_STRING → ss-handle )
  USING → on-error )
  context-var → IN → ss-field
```

Arguments

ss-handle

A host language variable or name used to refer to the segmented string.

on-error

The ON ERROR clause. This clause specifies host language statements or Rdb/VMS data manipulation statements to be performed if an Rdb/VMS error occurs.

START_SEGMENTED_STRING Statement

context-var

A valid context variable. You use this context variable to qualify the segments in the data manipulation statements that follow.

ss-field

A qualified field name that refers to a field defined with the segmented string data type. Note that this field name must be qualified by its own context variable. This second context variable must match the variable declared in the START_STREAM statement.

Usage Note

If you have invoked a database, you have the necessary privileges to use the START_SEGMENTED_STRING statement.

Examples

Example 1

The following example shows how to use the START_SEGMENTED_STRING statement in a Callable RDO program:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. GETSEGSTR.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.

01 HOST_VARIABLE1 PIC X(30) VALUE SPACES.
01 HOST_VARIABLE2 PIC X(30) VALUE SPACES.
01 RDB_STRING PIC X(120) VALUE SPACES.
01 COND_VALUE PIC S9(9) COMP.
88 RDB_END VALUE IS EXTERNAL RDB$_STREAM_EOF.
01 SEG_COND_VALUE PIC S9(9) COMP.
88 SEG_RDB_END VALUE IS EXTERNAL RDB$_SEGSTR_EOF.

PROCEDURE DIVISION.

* Invoke the database.
*
DEFINE_SEGSTR_RELATION.
    DISPLAY 'Invoke.'.
    MOVE "INVOKE DATABASE FILENAME 'DOCDS:[RDMSDOC.TEST]PERSONNEL'"
    TO RDB_STRING.
    PERFORM RDB_INTERPRET.

    PERFORM PRINT_SEGSTRS.
    STOP RUN.
```

START_SEGMENTED_STRING Statement

```
* Print the segments of a segmented string field, RESUME.
*
PRINT_SEGSTRS.
    DISPLAY 'Print segmented strings'.
    MOVE 'START_STREAM STREAM USING R IN RESUMES' TO RDB_STRING.
    PERFORM RDB_INTERPRET.
    PERFORM FETCH_LOOP UNTIL RDB_END.
    MOVE 'END_STREAM STREAM' TO RDB_STRING.
    PERFORM RDB_INTERPRET.

* This loop uses the FETCH statement to retrieve each record
* in the record stream.
* For each record, you must start a segmented string.
*

FETCH_LOOP.
    MOVE 'FETCH STREAM' TO RDB_STRING.
    CALL "RDB$INTERPRET" USING BY DESCRIPTOR
        RDB_STRING
        GIVING COND_VALUE.
    EVALUATE TRUE
        WHEN COND_VALUE IS FAILURE AND NOT RDB_END
        PERFORM INVALID_COMMAND.

    IF NOT RDB_END
        DISPLAY 'Print segments'
        MOVE 'START_SEGMENTED_STRING STR USING T IN R.RESUME'
            TO RDB_STRING
        PERFORM RDB_INTERPRET

        PERFORM SEGMENT_LOOP UNTIL SEG_RDB_END
        MOVE 'END_SEGMENTED_STRING STR' TO RDB_STRING
        PERFORM RDB_INTERPRET
    END-IF.

* This loop uses the GET statement to retrieve each
* segment in the field.
*

SEGMENT_LOOP.
    MOVE 'GET !VAL = T.RDB$VALUE; !VAL = T.RDB$LENGTH END_GET'
        TO RDB_STRING.
    CALL "RDB$INTERPRET" USING BY DESCRIPTOR
        RDB_STRING,
        HOST_VARIABLE1,
        HOST_VARIABLE2
        GIVING SEG_COND_VALUE.
    EVALUATE TRUE
        WHEN SEG_COND_VALUE IS FAILURE AND NOT SEG_RDB_END
        MOVE SEG_COND_VALUE TO COND_VALUE
        PERFORM INVALID_COMMAND.
    DISPLAY HOST_VARIABLE1, HOST_VARIABLE2.

STOP RUN.
```

START_SEGMENTED_STRING Statement

```
RDB_INTERPRET.  
  CALL "RDB$INTERPRET" USING BY DESCRIPTOR  
    RDB_STRING  
    GIVING COND_VALUE.  
  
  EVALUATE TRUE  
    WHEN COND_VALUE IS FAILURE  
    PERFORM INVALID_COMMAND.  
  
  MOVE SPACE TO RDB_STRING.  
INVALID_COMMAND.  
  DISPLAY 'Invalid command: "' RDB_STRING '''.  
  CALL "LIB$SIGNAL" USING BY VALUE COND_VALUE.
```

The following Rdb/VMS statements, extracted from the COBOL code and listed in the order they execute, control the printing operation:

```
INVOKE DATABASE FILENAME 'DOCD$: [RDMSDOC.TEST] PERSONNEL'  
START_STREAM GET_RESUME USING R IN RESUMES  
  FETCH GET_RESUME  
    START_SEGMENTED_STRING STRING USING T IN R.RESUME  
      GET !VAL = T.RDB$VALUE;  
      !VAL = T.RDB$LENGTH  
    END_GET  
  END_SEGMENTED_STRING STRING  
END_STREAM GET_RESUME
```

This example:

- Starts a stream with the stream name called GET_RESUME. The END_STREAM GET_RESUME statement closes the block.
- Uses the FETCH statement to move the stream pointer to the first record in the relation, RESUMES.
- Uses the START_SEGMENTED_STRING statement to start a stream of segments from the RESUME field of RESUMES. Note that the context variable (R) on RESUME relates it to RESUMES.
- Uses the GET statement to retrieve each segment and its length. The second context variable relates each segment value to the segmented string field, RESUME. The COBOL program displays the results.
- Encloses both the FETCH and GET statements in loops. This outer and inner looping structure is necessary to retrieve each record and each segment within the RESUME field.

START_SEGMENTED_STRING Statement

Errors

Your program can check for and handle the following errors, using the name listed here in the form RDB\$_error-name. The following list includes two types of errors:

- E Expected errors. These are run-time errors. Your program should check for these errors and provide error handlers.
- U Unexpected errors. Preprocessors detect these errors when the program is preprocessed, so your program does not need to check for them. Callable RDO, however, detects these errors at run time. A Callable RDO program should check for them.

DEADLOCK (E)

request failed due to resource deadlock

LOCK_CONFLICT (E)

NO WAIT request failed because resource was locked

NO_PRIV (U)

privilege denied by database protection

SEGSTR_NO_TRANS (U)

attempt to use a segmented string after COMMIT/ROLLBACK

START_STREAM Statement, Declared

Examples

Example 1

The following program fragment shows how to use the DECLARE_STREAM statement with a declared START_STREAM statement. Note that although the START_STREAM, FETCH, and END_STREAM statements must come after the DECLARE_STREAM statement, they can be placed in any order within the program, as long as they are *executed* in the following order: START_STREAM, FETCH, END_STREAM.

```
DATA DIVISION.
WORKING-STORAGE SECTION.

.
.
.

&RDB&  INVOKE DATABASE FILENAME 'PERSONNEL'.

&RDB&  DECLARE_STREAM EMPL_STREAM USING E IN EMPLOYEES
-      SORTED BY E.LAST_NAME

      01 LAST_NAME  PIC X(14).
      01 FIRST_NAME PIC X(10).
      01 EMPLOYEE_ID PIC X(5).

PROCEDURE DIVISION.
INIT SECTION.
INIT-PARAGRAPH.

&RDB&  START_TRANSACTION READ_WRITE.

      PERFORM START-STREAM.

      PERFORM FETCH-STREAM.
      PERFORM GET-STREAM.
      DISPLAY LAST_NAME.
      DISPLAY FIRST_NAME.
      DISPLAY EMPLOYEE_ID.

      PERFORM FETCH-STREAM.
      PERFORM GET-STREAM.
      DISPLAY LAST_NAME.
      DISPLAY FIRST_NAME.
      DISPLAY EMPLOYEE_ID.

      PERFORM END_STREAM.

&RDB&  FINISH.

      GOTO END-PROGRAM.

END_STREAM.
```

START_STREAM Statement, Declared

```
&RDB&  END_STREAM EMPL_STREAM.  
GET-STREAM.  
&RDB&  GET  
-      LAST_NAME = E.LAST_NAME;  
-      FIRST_NAME = E.FIRST_NAME;  
-      EMPLOYEE_ID = E.EMPLOYEE_ID;  
-      END_GET.  
FETCH-STREAM.  
&RDB&  FETCH EMPL_STREAM.  
START-STREAM.  
&RDB&  START_STREAM EMPL_STREAM.  
END-PROGRAM.  
      STOP RUN.
```

START_STREAM Statement, Undeclared

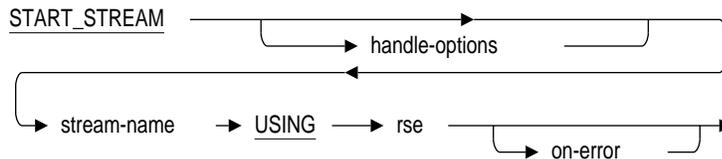
9.64 START_STREAM Statement, Undeclared

Declares and opens a record stream. The undeclared `START_STREAM` statement:

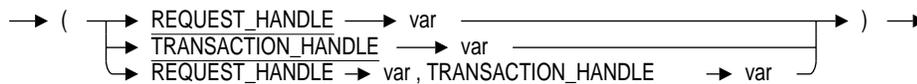
- Forms a record stream from one or more relations. The record selection expression determines the records in the record stream.
- Places a pointer for that stream just before the first record in this stream.

You must then use the `FETCH` statement to advance the pointer one record at a time through the stream. You can use other Rdb/VMS statements (`GET`, `MODIFY`, and `ERASE`) to manipulate each record.

Format



handle-options =



Arguments

REQUEST_HANDLE var

A keyword followed by a host language variable. A request handle points to the location of a compiled Rdb/VMS request. If you do not supply a request handle explicitly, Rdb/VMS associates a request handle with the compiled request. You must use a request handle to make an identical query to two different databases.

Use `!VAL` in Callable RDO as a substitution marker for host language variables.

START_STREAM Statement, Undeclared

See the *VAX Rdb/VMS Guide to Using RDO, RDBPRE, and RDML* for information on using request handles.

TRANSACTION_HANDLE var

A keyword followed by a host language variable. A transaction handle identifies each instance of a database attach. If you do not declare the transaction handle explicitly, Rdb/VMS attaches an internal identifier to the transaction.

Use !VAL in Callable RDO as a substitution marker for host language variables.

Note Normally, you do not need to use this argument. The ability to declare a transaction handle is provided for compatibility with other database products and future releases of Rdb/VMS.

Do not use this argument in interactive RDO.

stream-name

The name of the stream that you create. You refer to the stream name only when you want to move the stream pointer (FETCH) or terminate the stream (END_STREAM). Use a context variable for all other purposes. The stream-name following the START_STREAM keyword must match the stream-name following the END_STREAM keyword.

rse

A record selection expression. This RSE specifies the records included in the record stream. See Chapter 4 for a complete description of the record selection expression.

Any context variables that you define with the START_STREAM statement are valid for the life of that stream only.

Once you have defined a context variable in the RSE, you cannot redefine that context variable elsewhere until you have ended the stream.

on-error

The ON ERROR clause. This clause specifies the action to be taken if an Rdb/VMS error occurs while Rdb/VMS is compiling the RSE in the START_STREAM statement.

START_STREAM Statement, Undeclared

Usage Notes

You must have the Rdb/VMS READ privilege to the specified relations to use this statement.

Because the declared START_STREAM statement and the undeclared START_STREAM statement both begin with the keyword START_STREAM followed by a stream name, make sure you do not split the undeclared START_STREAM statement over two lines in such a way that Rdb/VMS will interpret it as a *declared* START_STREAM statement.

The following example ends the first line with the keyword USING to indicate to RDO that the statement is not complete:

```
RDO> START_STREAM EMP_STREAM USING  
cont> E IN EMPLOYEES SORTED BY E.LAST_NAME
```

The next example uses the continuation character to indicate to RDO that the undeclared START_STREAM statement is not complete:

```
RDO> START_STREAM EMP_STREAM -  
cont> USING E IN EMPLOYEES SORTED BY E.LAST_NAME
```

Use the START_STREAM statement instead of the FOR statement to establish a record stream in Callable RDO. You can also use the START_STREAM statement in a preprocessed program to control the processing of each record in a stream.

You can process streams only in the forward direction. If you want to move the stream pointer back to a record that you already processed, you must close the stream and reopen it.

The order of the stream is not predictable unless the RSE contains a SORTED BY clause.

Rdb/VMS examines the contents of any input host language variables when you use the START_STREAM statement. Rdb/VMS cannot evaluate the host language variables again until you close and reopen the stream. Therefore, you cannot change the value of a host language variable in the middle of a START_STREAM operation.

Once you have named the stream, you should refer to the stream name only when you want to do the following:

- Move the stream pointer with a FETCH statement
- Terminate the stream with the END_STREAM clause

START_STREAM Statement, Undeclared

For all other purposes, you should use a context variable.

Any context variables that you define with the `START_STREAM` statement are valid for the life of that stream only. Once you have defined a context variable in the RSE, you cannot redefine that context variable elsewhere inside the `START_STREAM . . . END_STREAM` block.

The statements following a `START_STREAM` statement must include at least one `FETCH` statement before you access any record in the stream.

Examples

Example 1

The following example creates a record stream in RDO:

```
RDO> START_TRANSACTION READ_ONLY
RDO>
RDO> START_STREAM EMP_STREAM USING
cont> E IN EMPLOYEES SORTED BY E.LAST_NAME
RDO>  FETCH EMP_STREAM
RDO>      PRINT E.LAST_NAME, E.ADDRESS_DATA_1,
cont>      E.ADDRESS_DATA_2, E.CITY, E.STATE, E.POSTAL_CODE
RDO> END_STREAM EMP_STREAM
RDO>
RDO> COMMIT
```

These statements:

- Create a record stream `EMP_STREAM`, defined by a record selection expression.
- Move the stream pointer to the first record. Because the `SORTED BY` clause is used, the records are in alphabetical order by last name.
- Retrieve the values of the fields `LAST_NAME`, `ADDRESS_DATA_1`, `ADDRESS_DATA_2`, `CITY`, `STATE`, and `POSTAL_CODE` for the first record in `EMP_STREAM`.

START_STREAM Statement, Undeclared

Example 2

The following example creates a record stream in a BASIC program using Callable RDO:

```
RDMS_STATUS = RDB$INTERPRET ( 'INVOKE DATABASE PATHNAME ' + &
                               ' "PERSONNEL" ' )

RDMS_STATUS = RDB$INTERPRET ( 'START_STREAM EMP USING ' + &
                               ' E IN EMPLOYEES ' )

RDMS_STATUS = RDB$INTERPRET ( 'FETCH EMP ' )

DML_STRING = 'GET ' + &
              '!VAL = E.EMPLOYEE_ID;' + &
              '!VAL = E.LAST_NAME;' + &
              '!VAL = E.FIRST_NAME' + &
              'END_GET '

RDMS_STATUS = RDB$INTERPRET (DML_STRING, EMP_ID, &
                              LST_NAM, FRST_NAM)
```

This BASIC program fragment shows how to display three field values from the EMPLOYEES relation in a Callable RDO program:

- The first three calls to RDB\$INTERPRET invoke the database, start a stream called EMP, and move the pointer to the first record in the stream.
- The assignment statement builds a command string to perform the GET operation.
- The final call sends the command string to Rdb/VMS, which assigns the database field values from the first record in the stream to the program's host language variables.

Errors

Your program can check for and handle the following errors, using the name listed here in the form RDB\$_error-name. The following list includes two types of errors:

- E Expected errors. These are run-time errors. Your program should check for these errors and provide error handlers.

START_STREAM Statement, Undeclared

U Unexpected errors. Preprocessors detect these errors when the program is preprocessed, so your program does not need to check for them. Callable RDO, however, detects these errors at run time. A Callable RDO program should check for them.

ARITH_EXCEPT (U)

truncation of a numeric value at run time

DEADLOCK (E)

request failed due to resource deadlock

LOCK_CONFLICT (E)

NO WAIT request failed because resource was locked

NO_PRIV (U)

privilege denied by database protection

WRONUMARG (U)

illegal number of arguments on call to Rdb system

START_TRANSACTION Statement

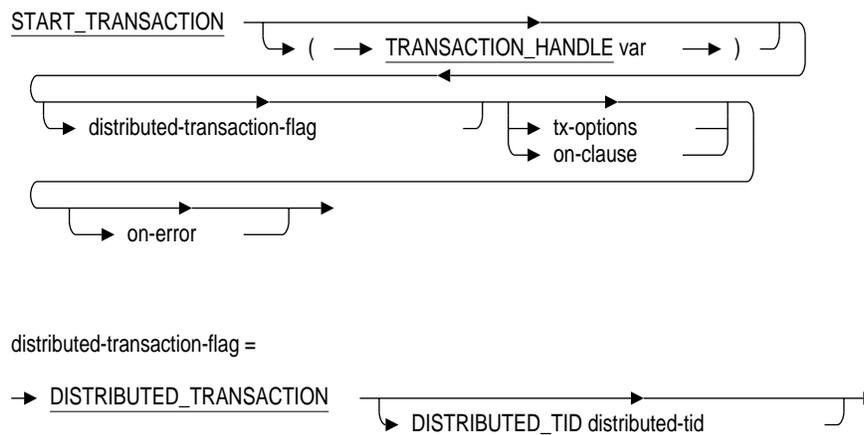
9.65 START_TRANSACTION Statement

Initiates a group of statements that Rdb/VMS executes as a unit. A transaction ends with a COMMIT or ROLLBACK statement. If you end the transaction with the COMMIT statement, all the statements within the transaction execute. If you end the transaction with the ROLLBACK statement, none of the statements takes effect.

If an application starts a distributed transaction by explicitly calling the DECdtm system service SYSSSTART_TRANS, it must complete the transaction by calling either the DECdtm system service SYSS\$END_TRANS or SYSS\$ABORT_TRANS.

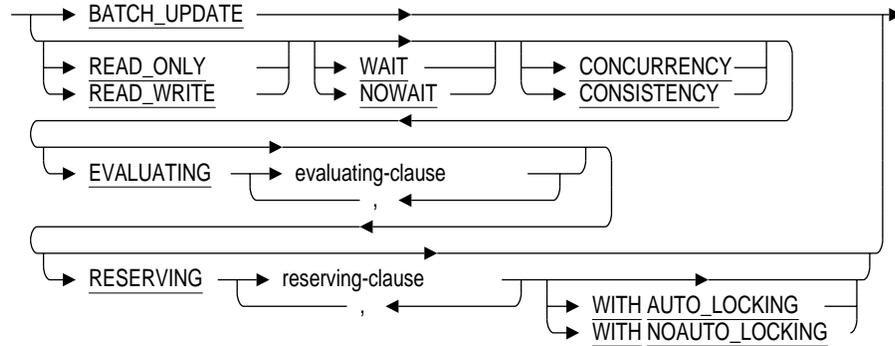
See the *VAX Rdb/VMS Guide to Distributed Transactions* for more information on using the SYSS\$END_TRANS or SYSS\$ABORT_TRANS system service call in distributed transactions.

Format

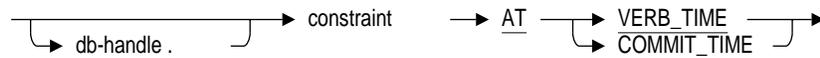


START_TRANSACTION Statement

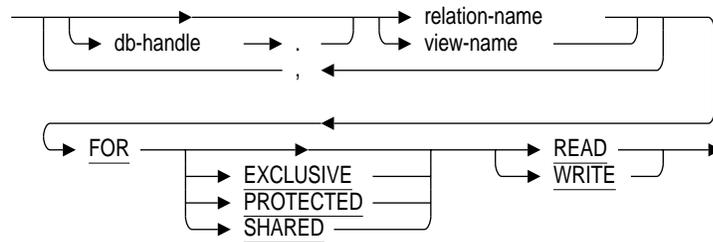
tx-options =



evaluating-clause =



reserving-clause =



on-clause =



START_TRANSACTION Statement

Defaults

There are several levels of defaults for the `START_TRANSACTION` statement. In general, it is recommended that you use explicit `START_TRANSACTION` statements, specifying the `READ_WRITE` option or `READ_ONLY` option, a list of relations in the `RESERVING` clause, and a share mode and lock type for each relation. If you wish to use defaults, Table 9–11 summarizes the defaults for each option and combination of options.

Table 9–11 Defaults for the `START_TRANSACTION` Statement

Option	Defaults
Transaction Mode	
<code>READ_ONLY</code> <code>READ_WRITE</code> <code>BATCH_UPDATE</code>	<p>If you omit the <code>START_TRANSACTION</code> statement, then the type of transaction started depends on the type of statement you issue:</p> <ul style="list-style-type: none">■ If your next statement is a data manipulation statement, then Rdb/VMS automatically starts a read-only transaction. Note that if the statement is a <code>STORE</code>, <code>MODIFY</code>, or <code>ERASE</code>, the result is an error, because you cannot update the database in a read-only transaction.■ If your next statement is a data definition statement, then Rdb/VMS automatically starts a read/write transaction. <p>If you use the <code>START_TRANSACTION</code> statement without a <code>READ_WRITE</code> or <code>READ_ONLY</code> clause, Rdb/VMS starts a read-only transaction.</p>

(continued on next page)

START_TRANSACTION Statement

Table 9-11 (Cont.) Defaults for the START_TRANSACTION Statement

Option	Defaults
Lock Specification	
RESERVING	<ul style="list-style-type: none"> ■ If you specify a read/write transaction and do not include a RESERVING clause, Rdb/VMS determines the lock specification for each relation when the relation is first accessed with a data manipulation statement. ■ If you specify a read/write transaction and include a RESERVING clause, the default share mode is SHARED. If you use the WITH AUTO_LOCKING option of the RESERVING clause, Rdb/VMS determines the lock specification for each relation accessed by a constraint or trigger when the relation is first accessed with a data manipulation statement from a constraint or trigger. ■ If you do not specify a transaction mode but do include a RESERVING clause, the share mode is SHARED. ■ If you specify a read-only transaction, the share mode is SHARED READ, whether or not you specify a RESERVING clause. A RESERVING clause can declare only READ locks, and the WITH AUTO_LOCKING option has no meaning (no updates can occur).
Share Mode	
SHARED PROTECTED EXCLUSIVE	The default is SHARED.

(continued on next page)

START_TRANSACTION Statement

Table 9-11 (Cont.) Defaults for the START_TRANSACTION Statement

Option	Defaults
Lock Type	
READ WRITE	<ul style="list-style-type: none">■ If you specify a read/write transaction mode, the default is WRITE.■ If you specify a read-only transaction mode, the default, and only allowed lock type, is READ.
Concurrency Option	
CONSISTENCY CONCURRENCY	CONSISTENCY is the default (and, for Rdb/VMS, the only option).
Wait Mode	
WAIT NOWAIT	WAIT is the default.
Evaluating Clause	
VERB_TIME COMMIT_TIME	By default, Rdb/VMS evaluates each constraint at the time specified in its DEFINE CONSTRAINT definition. If the constraint definition does not specify when the constraint should be checked, the definition default is CHECK ON UPDATE.

Arguments

TRANSACTION_HANDLE var

A keyword followed by a host language variable. A transaction handle identifies each instance of a database attach. If you do not declare the transaction handle explicitly, Rdb/VMS attaches an internal identifier to the transaction.

START_TRANSACTION Statement

If you specify a transaction handle on a START_TRANSACTION statement, you must also specify the transaction handle on any COMMIT, FOR, ROLLBACK, START_STREAM, and STORE statements that relate to the transaction.

You can specify this clause only once within a single START_TRANSACTION statement.

In Callable RDO, use !VAL as a substitution marker for host language variables. See the *VAX Rdb/VMS Guide to Using RDO, RDBPRE, and RDML* for information about the use of !VAL.

Note *Normally, you do not need to use this argument. The ability to declare a transaction handle is provided for compatibility with other database products and future releases of Rdb/VMS.*

Do not use this argument in interactive RDO.

DISTRIBUTED_TRANSACTION

A distributed transaction is a transaction that uses more than one database handle. Examples of a distributed transaction include:

- A transaction that attaches more than once to a single Rdb/VMS database
- A transaction that attaches to two or more Rdb/VMS databases
- A transaction that attaches to more than one database management system (an Rdb/VMS database and a VAX DBMS database, for example)

Distributed transactions are managed by DECdtm services. DECdtm services assigns a unique distributed transaction identifier (distributed TID) to each distributed transaction when the distributed transaction is started.

Use the DISTRIBUTED_TRANSACTION keyword by itself (without the DISTRIBUTED_TID distributed-tid clause) when you want to start a distributed transaction.

For complete information on distributed transactions, see the *VAX Rdb/VMS Guide to Distributed Transactions*.

DISTRIBUTED_TID distributed-tid

A keyword followed by a host language variable in application programs. When you want the transaction you are starting to join a distributed transaction, use this clause to specify the distributed TID of the distributed transaction that you want your transaction to join.

START_TRANSACTION Statement

The distributed-tid is a host language variable. You use the distributed-tid variable to hold the value of the distributed TID that DECdtm services generates and returns to the application. DECdtm services uses the distributed TID to distinguish the databases involved in a particular distributed transaction.

Note that if you want to start a distributed transaction, you should use only the DISTRIBUTED_TRANSACTION clause. However, if you want your application to start a transaction that will join a distributed transaction, then you must use both the DISTRIBUTED_TRANSACTION clause and the DISTRIBUTED_TID distributed-tid clause.

Your application must explicitly call the SYS\$START_TRANS system service and you must specify the DISTRIBUTED_TID distributed-tid clause if you want the transaction you are starting to join a distributed transaction. The distributed-tid variable is an octaword (16 bytes) that you should declare and initialize to zero at the beginning of your application.

Do not use the DISTRIBUTED_TID clause in interactive RDO. It is valid only in RDBPRE and Callable RDO.

For complete information on distributed transactions, see the *VAX Rdb/VMS Guide to Distributed Transactions*.

BATCH_UPDATE

READ_ONLY

READ_WRITE

The Rdb/VMS transaction mode.

BATCH_UPDATE

Before you begin a batch-update transaction in your programs, you should create a backup copy of the database.

You can reduce overhead in large, initial load operations by using the BATCH_UPDATE option. To speed update operations, Rdb/VMS does not write to the process' recovery-unit journal (RUJ) file in the batch-update transaction. Therefore, you cannot explicitly roll back a batch-update transaction with a ROLLBACK statement. If Rdb/VMS attempts to perform an automatic rollback due to any error (for example, a constraint condition is violated) that you do not trap in your program, the transaction fails and your database is permanently corrupted (because no RUJ file exists). You must then re-create the database from the backup copy you created prior to starting the batch-update transaction. After you have corrected the error condition, you can restart the program from the beginning.

START_TRANSACTION Statement

If you do attempt to explicitly roll back a batch-update transaction, the ROLLBACK statement generates an error message and the transaction will not be rolled back, as shown in the following session:

First, back up the database files before starting the batch-update transaction:

```
$ RMU/BACKUP PERSONNEL BACK_DISK:PERS_BACKUP.RBF
```

Then invoke RDO:

```
$ RDO
RDO> INVOKE DATABASE FILENAME "PERSONNEL"
RDO> START_TRANSACTION BATCH_UPDATE
RDO> STORE E IN EMPLOYEES USING
cont> E.EMPLOYEE_ID = "15399";
cont> E.LAST_NAME = "SMITH";
cont> E.FIRST_NAME = "JOHN"
cont> END_STORE
RDO> !
RDO> ! At this point, assume the user - who does not understand that
RDO> ! rolling back a batch-update transaction would corrupt the
RDO> ! database - enters ROLLBACK:
RDO> !
RDO> ROLLBACK
%RDB-E-NO_ROLLBACK, no rollback is allowed with the recovery mechanism
disabled
```

At this point, the user's batch-update transaction is still active:

```
RDO> SHOW TRANSACTION
All Transactions in Database with filename PERSONNEL
a read-write transaction is in progress
- updates have been performed
- transaction sequence number (TSN) is 152
- snapshot space for TSNs less than 152 can be reclaimed
- session ID number is 55
```

If you receive the RDB\$_NO_ROLLBACK error during a batch-update transaction, you have two choices:

- 1 Manually undo any changes you made (or fix the problem you were having) and then commit the transaction. For example, if you stored a record, erase that record and then issue a COMMIT statement:

```
RDO> FOR E IN EMPLOYEES WITH E.EMPLOYEE_ID = "15231"
cont> ERASE E
cont> END_FOR
RDO> COMMIT
RDO> FINISH
RDO> EXIT
```

- 2 Exit the program or RDO session, which *will* corrupt the database.

START_TRANSACTION Statement

The second option assumes the user heeded the documentation warning to create a backup copy of the database prior to starting the batch-update transaction. After restoring the database files from the backup file, the user can attempt to correct the situation that led up to the error and then re-enter the update program or RDO session.

You cannot qualify the BATCH_UPDATE option with the arguments available to the READ_ONLY and READ_WRITE options (lock specification, share mode, etc.).

See Table 9-13 for information on how the BATCH_UPDATE transaction mode compares with the three update share modes (SHARED WRITE, PROTECTED WRITE, and EXCLUSIVE WRITE).

READ_ONLY

If you start a read-only transaction, you can retrieve a snapshot of the database at the moment the transaction started. Other users can update records in the relation you are using, but your transaction retrieves the records as they existed at the time the transaction started. Any changes that other users make and commit during the transaction are invisible to you. Using the READ_ONLY option lets you read data without incurring the overhead of record locking. You cannot modify, store, or erase records or execute data definition statements in a read-only transaction.

At the time the START_TRANSACTION READ_ONLY statement is entered, if snapshots are ENABLED DEFERRED (an option on the DEFINE DATABASE, CHANGE DATABASE, and IMPORT statements), read-only users can access the database only when at least one read-only user had invoked the database before a currently *active* read/write user.

That is, if only read/write transactions are active in the database, subsequent read-only users who attempt to start their transactions must *wait* until all the active read/write transactions enter a COMMIT or ROLLBACK statement. In this case, note that the read-only transactions must wait even if the NOWAIT option was specified on the START_TRANSACTION statement.

Note that specifying READ_ONLY as a transaction mode automatically sets SHARED READ locking, regardless of what you specify in the RESERVING clause.

READ_WRITE

Signals that you want to use the locking mechanisms of Rdb/VMS to get

START_TRANSACTION Statement

consistency in data retrieval and update. Use the READ_WRITE option when you need to:

- Store, modify, or erase data
- Retrieve data that is guaranteed to be correct at the moment of retrieval
- Use Rdb/VMS data definition statements

When you are reading a record in READ_WRITE mode, no other user can update the record. Note that under some circumstances, Rdb/VMS may lock records that you are not explicitly reading:

- If your query is scanning a relation without using an index, Rdb/VMS locks all the records in the record stream.
- If your query uses indexes, Rdb/VMS may lock part of an index.

WAIT [Default]

NOWAIT

Determines what your transaction does when it encounters a locked record:

- If you specify WAIT, the transaction waits for others to complete and then proceeds.
- If you specify NOWAIT, your transaction returns an error message when it encounters a locked record.

An exception could occur for read-only transactions that use the NOWAIT option when snapshots are ENABLED DEFERRED. In this one circumstance, read-only users must wait even when they use the NOWAIT option. At the time the START_TRANSACTION READ_ONLY statement is entered, if snapshots are ENABLED DEFERRED (an option on the DEFINE DATABASE, CHANGE DATABASE, and IMPORT statements) read-only users can access the database only when at least one read-only user had invoked the database before a currently *active* read/write user.

That is, if only read/write transactions are active in the database, subsequent read-only users who attempt to start their transactions must *wait* until all the active read/write transactions enter a COMMIT or ROLLBACK statement. In this case, the read-only transactions must wait even if the NOWAIT option was specified on the START_TRANSACTION statement.

START_TRANSACTION Statement

See the *VAX Rdb/VMS Guide to Database Maintenance and Performance* for details on the deferred snapshot feature.

Consider another WAIT example. Assume that snapshots are ENABLED IMMEDIATE (the default). If an update transaction (called User-1 here) reserves a relation for EXCLUSIVE WRITE, and a subsequent read-only transaction (User-2) attempts to access that relation and uses the WAIT option (the default), the read-only transaction will wait until the active READ_WRITE . . . EXCLUSIVE WRITE transaction commits or rolls back and then will receive an error message.

However, after receiving the error message, User-2's read-only transaction will still be active.

For example, assume that User-1 already has reserved the EMPLOYEES relation for EXCLUSIVE WRITE. User-2 then enters:

```
RDO> START_TRANSACTION READ_ONLY WAIT
RDO> FOR E IN EMPLOYEES
RDO> PRINT E.*
RDO END_FOR
```

[waits until User-2's EXCLUSIVE WRITE transaction completes]

.
.
.

[User-2 enters COMMIT or ROLLBACK...]

```
%RDB-E-LOCK_CONFLICT, request failed due to locked resource; no-wait
parameter specified for transaction
-RDMS-F-CANTSNAP, can't ready storage area for snapshots
RDO> !
RDO> SHOW TRANSACTION
All Transactions in Database with filename personnel
a snapshot transaction is in progress
```

.
.
.

Again, the previous example assumed that snapshots were ENABLED and IMMEDIATE. If snapshots had been DEFERRED, the read-only user would have waited for this (and any other) active read/write transactions to complete before getting past the START_TRANSACTION step.

START_TRANSACTION Statement

CONCURRENCY

CONSISTENCY [Default]

The extent to which the database protects the consistency of your data. In Rdb/VMS, this distinction is not meaningful. Rdb/VMS always guarantees degree 3 consistency. Degree 3 consistency means that the database system guarantees that data you have read will not be changed by another user before you issue a COMMIT statement.

In other relational systems that you might access using the remote feature of Rdb/VMS, this option specifies the degree to which you want to control the consistency of the database. In such systems, the CONCURRENCY option sacrifices some consistency protection for improved performance with many users.

Note *This feature is provided for compatibility with other Digital Equipment Corporation relational database products. If you use the CONCURRENCY option, you may be able to transport your programs to another system that takes advantage of that option and achieve improved performance.*

evaluating-clause

The point at which the named constraints are evaluated. If you specify VERB_TIME, they are evaluated when the data manipulation statement is issued. If you specify COMMIT_TIME, they are evaluated when the COMMIT statement executes. The evaluating clause is allowed syntactically, but is ignored, with read-only transactions.

db-handle

A host language variable used to refer to the database.

constraint

The name of an Rdb/VMS constraint.

reserving-clause

The list of relations to be locked during the transaction. In general, include all the relations your transaction will access. If you use the WITH AUTO_LOCKING option, constraints and triggers defined on the reserved relations will be able to access additional relations that do not appear in the list of reserved relations.

Note that if you use the RESERVING clause with the WITH NOAUTO_LOCKING option, you can access only those relations that you have explicitly reserved. If you access multiple databases with a single START_TRANSACTION statement and use the RESERVING clause for one or more

START_TRANSACTION Statement

databases, you can access only the reserved relations in a database for which you reserved relations. See the example in the Examples section.

EXCLUSIVE
PROTECTED
SHARED [Default]

The Rdb/VMS share modes. The keyword you choose determines what operations you allow others to perform on the relations you are reserving. For read-only transactions, the EXCLUSIVE and PROTECTED keywords are syntactically allowed, but are ignored.

Table 9–12 summarizes the share modes.

Table 9–12 VAX Rdb/VMS Share Modes

Option	Constraints on Access
SHARED [Default]	Other users can work with the same relations as you. Depending on the option they choose, they can have read-only or read and write access to the relations.
PROTECTED	Other users can read the relations you are using. They cannot have write access.
EXCLUSIVE	Other users cannot read records from the relations included in your transaction. If another user refers to the same relation in a START_TRANSACTION statement, Rdb/VMS denies access to that user.

Table 9–13 summarizes the effect the access modes (including the batch-update transaction) have on database functions.

Table 9–13 Comparison of Share Modes for Updates

Type of Access —>	SHARED WRITE	PROTECTED WRITE	EXCLUSIVE WRITE	BATCH UPDATE
Locks —>	Specific relations	Specific relations	Specific relations	Entire database
Writes to RUJ	Yes	Yes	Yes	No

(continued on next page)

START_TRANSACTION Statement

Table 9-13 (Cont.) Comparison of Share Modes for Updates

Type of Access —>	SHARED WRITE	PROTECTED WRITE	EXCLUSIVE WRITE	BATCH UPDATE
Writes to SNP	Yes	Yes	No	No
Recovery	Yes	Yes	Yes	No
Multi-User Access	Yes	Yes	No	No

READ

WRITE

The Rdb/VMS lock type. This keyword declares what you intend to do with the relations you are reserving.

READ

The default for the READ_ONLY option. You will only read data from the relations.

WRITE

The default for the READ_WRITE option. You will store, modify, or erase data in the relations. This option is not available for read-only transactions.

WITH AUTO_LOCKING

WITH NOAUTO_LOCKING

An optional clause that can be specified with the RESERVING clause. When you specify the WITH AUTO_LOCKING clause, Rdb/VMS automatically locks any relations referenced from constraints and triggers defined on the reserved relations when the referenced relations are accessed from the constraints or triggers. If one of these referenced relations is also a reserved relation, the explicitly specified lock mode must not conflict with the lock mode required by the constraint or trigger that references the relation.

The default is WITH AUTO_LOCKING. Use the WITH NOAUTO_LOCKING option if you do not want to use the auto-locking option.

on-error

The ON ERROR clause. This clause specifies a host language statement or statements to be performed if an Rdb/VMS error occurs during the START_TRANSACTION operation. See Section 9.54 for a syntax diagram and details of the ON ERROR clause.

START_TRANSACTION Statement

Usage Notes

If you have invoked a database, you have the necessary privileges to use the `START_TRANSACTION` statement.

If you issue a data manipulation language statement without issuing a `START_TRANSACTION` statement first, Rdb/VMS automatically starts a read-only transaction on all attached databases for you. Thus, you can perform simple data retrieval without starting a transaction explicitly. If you issue a data definition language statement without issuing a `START_TRANSACTION` statement first, Rdb/VMS automatically starts a read/write transaction for you. (See the Defaults section for important details about these statements.)

If you issue a data manipulation statement, such as `GET` or `PRINT`, and then try to use the `START_TRANSACTION` statement, you may get an error message warning you that a transaction is already in progress. To avoid this problem, you issue an explicit `START_TRANSACTION` statement when you plan to perform more than one operation.

You can specify different locking or transaction modes for multiple databases within the same transaction. For example, your application can access any relation in one database using the `READ_ONLY` option to check certain data values, while it updates relations in another database using the `READ_WRITE` option.

Rdb/VMS takes out `PROTECTED` record level locks even when you start your transaction in `SHARED` mode. When an update statement is issued, Rdb/VMS attempts to get an `EXCLUSIVE WRITE` lock on the record specified. If Rdb/VMS cannot get an `EXCLUSIVE WRITE` lock on the record specified, Rdb/VMS either waits for the lock to be available or issues a lock conflict message, depending upon whether `WAIT` or `NOWAIT` mode was specified for the transaction. (The default is `WAIT`.)

Referring to more than one Rdb/VMS database for actual update operations within one transaction is *not* recommended. That is, attempting to *update* multiple databases using a single `START_TRANSACTION` statement is not advisable; for instance, the `COMMIT` may succeed for the first database in which you performed updates, but could then fail if the disk device holding the second database (in which you also performed updates) is damaged by a system failure. This could leave your application in an inconsistent state.

START_TRANSACTION Statement

Examples

Example 1

The following example readies a relation for read/write access. Otherwise, it uses the defaults:

```
START_TRANSACTION READ_WRITE
```

This statement allows access to all the relations in the current database so that all users can modify records. It is equivalent to readying all the relations for SHARED WRITE access.

Example 2

The following example starts a read-only transaction:

```
START_TRANSACTION READ_ONLY
```

This statement lets you read data from the database but not store or modify data. When you retrieve data, you see the state of the records as they existed at the time of the START_TRANSACTION statement. You do not see any updates to the database made after that time.

Example 3

The following statement lets you specify the intended action for each relation in the transaction:

```
START_TRANSACTION READ_WRITE RESERVING  
    EMPLOYEES FOR PROTECTED WRITE,  
    JOBS, SALARY_HISTORY FOR SHARED READ
```

Assume that this transaction updates the EMPLOYEES relation based on values found in two other relations, JOBS and SALARY_HISTORY:

- The transaction must update the EMPLOYEES relation, so EMPLOYEES is readied for PROTECTED WRITE access.
- The program will only read values from the JOBS and SALARY_HISTORY relations, so there is no need for PROTECTED or WRITE access.

Example 4

You can access multiple databases from within the same transaction. The example in this section explains how you can benefit from this feature.

START_TRANSACTION Statement

Before you can use the multiple database feature of the START_TRANSACTION statement, you must issue an INVOKE statement for each database you intend to access. The INVOKE statement must include a database handle. For example, the following INVOKE statements identify two databases required by an update application:

```
&RDB&  INVOKE DATABASE DB1 = FILENAME 'PERSONAL$DISK:PERSONNEL'  
&RDB&  INVOKE DATABASE DB2 = FILENAME 'PERSONAL$DISK:BENEFITS'
```

Because the program only needs to read the EMPLOYEES relation of the PERSONNEL database (DB1), but needs to change values in two relations, TUITION and STATUS of the BENEFITS database (DB2), the program might contain the following START_TRANSACTION statement:

```
&RDB&  START_TRANSACTION  
&RDB&  ON DB1 USING (READ_WRITE  
&RDB&  RESERVING  
&RDB&  EMPLOYEES FOR SHARED READ) AND  
&RDB&  ON DB2 USING (READ_WRITE  
&RDB&  RESERVING  
&RDB&  TUITION FOR SHARED WRITE  
&RDB&  STATUS  FOR SHARED WRITE)
```

Note Referring to more than one Rdb/VMS database for actual update operations within one transaction is not recommended. That is, attempting to update multiple databases using a single START_TRANSACTION statement is not advisable; for instance, the transaction COMMIT may succeed for the first database in which you performed updates, but could then fail if the disk device holding the second database (in which you also performed updates) is damaged by a system failure. This could leave your application in an inconsistent state.

Example 5

In the following example, only the EMPLOYEES relation in the BENEFITS database (DB2) can be accessed during the transaction because it is the only relation reserved in the database. Any of the relations in the PERSONNEL database (DB1) can be accessed because the RESERVING clause was not specified for the PERSONNEL database:

```
&RDB&  INVOKE DATABASE DB1 = FILENAME 'PERSONAL$DISK:PERSONNEL'  
&RDB&  INVOKE DATABASE DB2 = FILENAME 'PERSONAL$DISK:BENEFITS'
```

START_TRANSACTION Statement

```
&RDB& START_TRANSACTION
&RDB& ON DB1 USING (READ_WRITE) AND
&RDB& ON DB2 USING (READ_WRITE
&RDB& RESERVING
&RDB& EMPLOYEES FOR EXCLUSIVE WRITE)
```

Example 6

In the following example that uses the sample personnel database, the ERASE statement in the first READ_WRITE transaction fails because the relations referred to by the EMPLOYEE_ID_CASCADE_DELETE trigger are not reserved. When the WITH AUTO_LOCKING clause is specified in the second START_TRANSACTION statement, all the relations referenced by triggers and constraints involving the EMPLOYEES relation are reserved automatically.

```
INVOKE DATABASE FILENAME 'PERSONNEL'
!
START_TRANSACTION READ_ONLY
!
! Show the EMPLOYEE_ID_CASCADE_DELETE trigger:
SHOW TRIGGER EMPLOYEE_ID_CASCADE_DELETE
    EMPLOYEE_ID_CASCADE_DELETE

        BEFORE ERASE
        FOR E IN EMPLOYEES EXECUTE
            FOR D IN DEGREES WITH
                D.EMPLOYEE_ID = E.EMPLOYEE_ID
                ERASE D
            END_FOR;
        FOR JH IN JOB_HISTORY WITH
            JH.EMPLOYEE_ID = E.EMPLOYEE_ID
            ERASE JH
        END_FOR;
        FOR R IN RESUMES WITH
            R.EMPLOYEE_ID = E.EMPLOYEE_ID
            ERASE R
        END_FOR;
        FOR SH IN SALARY_HISTORY WITH
            SH.EMPLOYEE_ID = E.EMPLOYEE_ID
            ERASE SH
        END_FOR;
! Also, if an employee is terminated and that employee
! is the manager of a department, set the manager_id
! missing for that department.
        FOR D IN DEPARTMENTS WITH D.MANAGER_ID = E.EMPLOYEE_ID
            MODIFY D USING D.MANAGER_ID =
                RDB$MISSING (D.MANAGER_ID) END_MODIFY
        END_FOR
    FOR EACH RECORD.
```


START_TRANSACTION Statement

```
!
! The following statement fails because the EMPLOYEE_ID_CASCADE_DELETE
! trigger cannot delete the rows with the value of 00165 in the EMPLOYEE_ID
! fields of the DEGREES, JOB_HISTORY, and SALARY_HISTORY relations. The
! statement fails because the relations referred to by the
! EMPLOYEE_ID_CASCADE_DELETE trigger on EMPLOYEES were not explicitly
! reserved before the transaction.
START_TRANSACTION READ_WRITE RESERVING EMPLOYEES FOR EXCLUSIVE WRITE;
FOR E IN EMPLOYEES
    WITH E.EMPLOYEE_ID = "00165"
    ERASE E
END_FOR
%RDB-E-UNRES_REL, relation DEGREES in specified request is not a
relation reserved in specified transaction
ROLLBACK
!
! Try the same statement again, this time specifying the WITH AUTO_LOCKING
! option. Specifying the WITH AUTO_LOCKING option automatically reserves the
! relations referred to by the EMPLOYEE_ID_CASCADE_DELETE trigger on the
! EMPLOYEES relation, allowing the trigger to execute.
START_TRANSACTION READ_WRITE RESERVING EMPLOYEES
FOR EXCLUSIVE WRITE WITH AUTO_LOCKING;
    FOR E IN EMPLOYEES
        WITH E.EMPLOYEE_ID = "00165"
        ERASE E
    END_FOR
COMMIT
!
! The row in the EMPLOYEES relation that had an EMPLOYEE_ID
! field with a value of 00165 has been deleted:
START_TRANSACTION READ_ONLY
FOR E IN EMPLOYEES
    WITH E.EMPLOYEE_ID = "00165"
    PRINT E.EMPLOYEE_ID
END_FOR
!
! The row in the DEGREES relation that had an EMPLOYEE_ID
! field with a value of 00165 has been deleted:
FOR D IN DEGREES
    WITH D.EMPLOYEE_ID = "00165"
    PRINT D.EMPLOYEE_ID
END_FOR
!
! The rows in the JOB_HISTORY relation that had an EMPLOYEE_ID
! field with a value of 00165 have been deleted:
FOR JH IN JOB_HISTORY
    WITH JH.EMPLOYEE_ID = "00165"
    PRINT JH.EMPLOYEE_ID
END_FOR
```

START_TRANSACTION Statement

```
!  
! The rows in the SALARY_HISTORY relation that had an EMPLOYEE_ID  
! field with a value of 00165 have been deleted:  
FOR SH IN SALARY_HISTORY  
  WITH SH.EMPLOYEE_ID = "00165"  
  PRINT SH.EMPLOYEE_ID  
END_FOR  
COMMIT  
FINISH  
EXIT
```

Errors

To prevent one database user from corrupting another user's picture of the database, Rdb/VMS:

- Delays an operation if the operation needs a record that is locked by another process, or returns an error if the user specified NOWAIT
- Rejects an operation if deadlocks occur (where two processes have locked records that each process needs)

No part of a transaction that modifies a database is complete until the entire transaction has been committed successfully. In particular, a deadlock may occur at any time during the transaction until it has been successfully committed. In programs, except for transactions started in READ_ONLY or EXCLUSIVE mode, you should check for RDBS_DEADLOCK after each database operation.

Generally, the best way to recover from a deadlock is to use a ROLLBACK statement and restart the transaction.

When you write or modify data in shared mode, Rdb/VMS locks any index records for that relation. This feature ensures that Rdb/VMS will be able to update those index records for the new data. This process frequently causes deadlocks.

Your program can check for and handle the following errors, using the name listed here in the form RDBS_error-name. The following list includes two types of errors:

- E Expected errors. These are run-time errors. Your program should check for these errors and provide error handlers.

START_TRANSACTION Statement

U Unexpected errors. Preprocessors detect these errors when the program is preprocessed, so your program does not need to check for them. Callable RDO, however, detects these errors at run time. A Callable RDO program should check for them.

BAD_DB_HANDLE (U)
invalid handle for database

BAD_TRANS_HANDLE (U)
invalid handle for transaction

DEADLOCK (E)
request failed due to resource deadlock

EXCESS_TRANS (U)
limit of *n* transactions per process exceeded

LOCK_CONFLICT (E)
NO WAIT request failed because resource was locked

NO_PRIV (U)
privilege denied by database protection

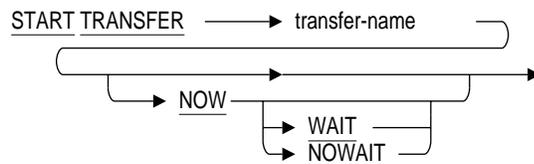
START TRANSFER Statement (VAX Data Distributor)

9.66 START TRANSFER Statement (VAX Data Distributor)

Executes a transfer on demand, or changes the state of a transfer from the suspended state to the scheduled, unscheduled, or active state.

Note To use this RDO statement, you must have VAX Data Distributor installed on your system. See your system manager or your Digital Equipment Corporation representative if you need information on Data Distributor.

Format



Arguments

transfer-name

The transfer to be started. The `transfer-name` parameter is required.

NOW

Changes the transfer state to active and begins execution. If the transfer has a schedule defined for it, subsequent transfers occur as specified by that schedule. The `NOW` argument is optional.

WAIT

NOWAIT

Specifies whether Data Distributor returns control to RDO immediately after the transfer starts or waits until the transfer completes. You must use the `NOW` argument if you specify `WAIT` or `NOWAIT`. The default, `NOWAIT`, causes Data Distributor to return control immediately to the RDO command mode while the transfer executes. With `NOWAIT` in effect, you can enter other RDO statements while the copy process completes the transfer of database records to the target database.

START TRANSFER Statement (VAX Data Distributor)

The WAIT option ensures that the transfer completes before control is returned to RDO.

Usage Notes

You can use the START TRANSFER statement without any qualifiers to change the state of a suspended transfer. Issuing the START TRANSFER statement places the transfer in the scheduled state if a schedule definition exists for the transfer. Execution then occurs at the next scheduled time. If a schedule definition does not exist, the transfer is placed in the unscheduled state and will execute only when you issue a START TRANSFER statement using the NOW qualifier or you define a schedule for the transfer.

You cannot enter the START TRANSFER statement when a transaction is outstanding. You must terminate any outstanding transactions before issuing the START TRANSFER statement.

You can use the START TRANSFER statement to initiate a transfer *on demand* by including the NOW option. The NOW option immediately places the transfer in the active state. This execute-on-demand feature is useful in batch processing environments when you want to initiate the transfer after another job has completed successfully, regardless of the transfer's schedule.

If you want to start a transfer, the transfer definition must be associated with your UIC.

Examples

Example 1

This example starts a transfer called ENROLL_SPANISH1. If this transfer has a schedule defined for execution one hour from now but the transfer is suspended, issuing the START TRANSFER statement only changes the transfer's state to scheduled. Execution of the transfer will not occur for another hour.

```
RDO> START TRANSFER ENROLL_SPANISH1
```

START TRANSFER Statement (VAX Data Distributor)

Example 2

In this example, transfer ENROLL_SPANISH1 has been defined but has no schedule. You issue this START TRANSFER NOW statement because you want to execute the transfer immediately. Control returns to RDO as soon as the transfer begins.

```
RDO> START TRANSFER ENROLL_SPANISH1 NOW
```

STOP TRANSFER Statement (VAX Data Distributor)

9.67 STOP TRANSFER Statement (VAX Data Distributor)

Places a transfer in the suspended state. Data Distributor does not attempt to execute the transfer until you remove it from the suspended state using the START TRANSFER statement. If the transfer is in the active state, the STOP TRANSFER statement also stops the copy process associated with that transfer.

Note To use this RDO statement, you must have VAX Data Distributor installed on your system. See your system manager or your Digital Equipment Corporation representative if you need information on Data Distributor.

Format

`STOP TRANSFER` → `transfer-name` →

Argument

transfer-name

The transfer to be suspended. The transfer-name parameter is required.

Usage Notes

You must terminate any outstanding transactions before you issue a STOP TRANSFER statement.

If you want to stop a particular transfer, the transfer definition must be associated with your UIC.

Examples

Example 1

The following example places the EUROPE_PERS transfer in the suspended state:

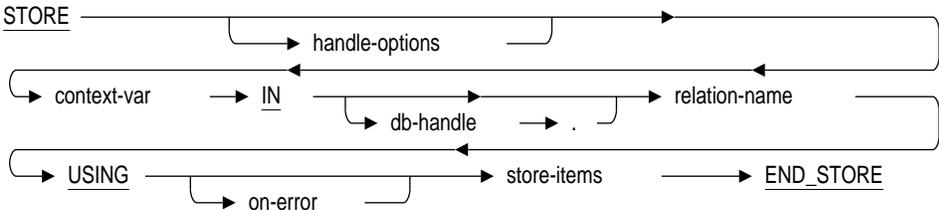
```
RDO> STOP TRANSFER EUROPE_PERS
```

9.68 STORE Statement

Inserts a record into an existing relation. Within a single STORE statement, you can refer to only one relation.

Note Storing data in a segmented string field requires special syntax. See Section 9.69 for more information on the STORE Statement with Segmented Strings.

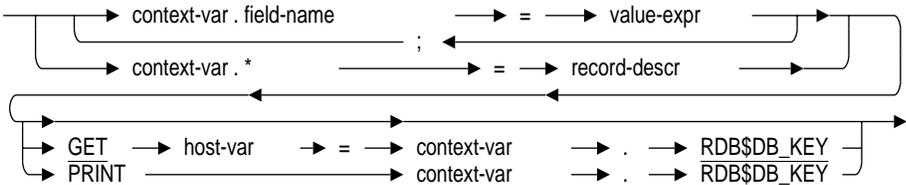
Format



handle-options =



store-items =



STORE Statement

Arguments

REQUEST_HANDLE var

A keyword followed by a host language variable. A request handle points to the location of a compiled Rdb/VMS request. If you do not supply a request handle explicitly, Rdb/VMS associates a request handle with the compiled request. You must use a request handle to make an identical query to two different databases.

In Callable RDO, use !VAL as a substitution marker for host language variables. You can put parentheses around the host language variable name.

See the *VAX Rdb/VMS Guide to Using RDO, RDBPRE, and RDML* for information on using request handles.

TRANSACTION_HANDLE var

A keyword followed by a host language variable. A transaction handle identifies each instance of a database attach. If you do not declare the transaction handle explicitly, Rdb/VMS attaches an internal identifier to the transaction.

In Callable RDO, use !VAL as a substitution marker for host language variables.

You can put parentheses around the host language variable name.

Note *Normally, you do not need to use this argument. The ability to declare a transaction handle is provided for compatibility with other database products and future releases of Rdb/VMS.*

Do not use this argument in interactive RDO.

context-var

A valid context variable. See Chapter 4 for more information.

db-handle

A host language variable used to refer to the database.

relation-name

The name of the relation into which the value is stored.

on-error

The ON ERROR clause. This clause specifies a host language statement or Rdb/VMS data manipulation statement to be performed if an Rdb/VMS error occurs.

STORE Statement

field-name

The name of the field in the relation where the value is stored.

value-expr

A valid Rdb/VMS value expression that specifies the value to be stored.

If you do not supply a value for a field or if the value you supply is the same as the field's missing value, Rdb/VMS marks the field as missing. For more information about making field values null, see Chapter 3, Value Expressions, and the *VAX Rdb/VMS Guide to Using RDO, RDBPRE, and RDML*.

record-descr

A valid data dictionary record descriptor that matches all the fields of the relation. You can use a host language statement to include the relation definition in your program. Each field of the record descriptor must match exactly the field names and data types of the fields in the Rdb/VMS relation referred to by the context variable.

host-var

A user-defined host language variable, into which you may store the dbkey of the record just stored. Use the GET . . . RDB\$DB_KEY construct available only in RDBPRE (BASIC, COBOL, and FORTRAN) programs to assign the value of the dbkey to the host language variable. See Section 9.48 and Section 9.52 for related information on the DBKEY SCOPE clause.

Usage Notes

You need the Rdb/VMS WRITE privilege for the relation in which rows are stored, and the Rdb/VMS READ and WRITE privileges for the database to use the STORE statement.

You cannot store records into a view that is defined using one or more of these clauses:

WITH
CROSS
REDUCED

If a query is active and you store a record that would satisfy that query, you will get unpredictable results when you try to display the results of that query.

STORE Statement

Examples

Example 1

The following example shows how to store a record in RDO:

```
RDO> START_TRANSACTION READ_WRITE
RDO> !
RDO> STORE D IN DEPARTMENTS USING
cont> D.DEPARTMENT_CODE = "RECR";
cont> D.DEPARTMENT_NAME = "Recreation";
cont> D.MANAGER_ID = "00175";
cont> D.BUDGET_PROJECTED = 240000;
cont> D.BUDGET_ACTUAL = 128776;
cont> END_STORE
RDO> !
RDO> COMMIT
```

This RDO statement explicitly assigns a literal value to each field in the DEPARTMENTS relation.

Example 2

The following example shows how to store a record in COBOL:

```
STORE-JOB-HISTORY.

    DISPLAY "Enter employee ID:           " WITH NO ADVANCING.
    ACCEPT EMPL-ID.
    DISPLAY "Enter job code:             " WITH NO ADVANCING.
    ACCEPT JOB-CODE.
    DISPLAY "Enter starting date:        " WITH NO ADVANCING.
    ACCEPT START-DATE.
    DISPLAY "Enter ending date:          " WITH NO ADVANCING.
    ACCEPT END-DATE.
    DISPLAY "Enter department code:      " WITH NO ADVANCING.
    ACCEPT DEPT-CODE.
    DISPLAY "Enter supervisor's ID:      " WITH NO ADVANCING.
    ACCEPT SUPER.

&RDB& START_TRANSACTION READ_WRITE
&RDB& RESERVING JOB_HISTORY,
&RDB& FOR PROTECTED WRITE,
&RDB& JOBS, EMPLOYEES
&RDB& FOR SHARED READ
&RDB& STORE J IN JOB_HISTORY USING
&RDB& ON ERROR
&RDB& ROLLBACK
&RDB& DISPLAY "An error has occurred. Try again."
&RDB& GO TO STORE-JOB-HISTORY
```

STORE Statement

```
&RDB&      END_ERROR
&RDB&      J.EMPLOYEE_ID = EMPL-ID;
&RDB&      J.JOB_CODE = JOB-CODE;
&RDB&      J.JOB_START = START-DATE;
&RDB&      J.JOB_END = END-DATE;
&RDB&      J.DEPARTMENT_CODE = DEPT-CODE;
&RDB&      J.SUPERVISOR_ID = SUPER;
&RDB&      END_STORE

&RDB&      COMMIT
```

This sequence stores a new record in the `JOB_HISTORY` relation. The COBOL program does the following:

- Prompts for the field values.
- Starts a read/write transaction. Because you are updating the `JOB_HISTORY` relation, you do not want to conflict with other users who may be reading data from this relation. Therefore, you use the `PROTECTED WRITE` reserving option.
Constraints on the database ensure that the employee and the job code being stored actually exist in other relations. Because the constraints check these other relations, you must reserve those relations also.
- Stores the record by assigning the host language variables to database field values.
- Includes an `ON ERROR` clause to check for errors.
- Uses the `COMMIT` statement to make the update permanent.

A more extensive example appears in Section 9.8.

Example 3

The following FORTRAN program fragment shows how the `dbkey` of a record just stored can be retrieved into a host language variable, using `GET . . . RDB$DB_KEY`, in a `STORE . . . END_STORE` block. For this complete program, see the Examples section in Section 9.52.

```
&RDB&      DATABASE FILENAME 'PERSONNEL' DBKEY SCOPE IS FINISH
&RDB&      START_TRANSACTION READ_WRITE

&RDB&      STORE P IN EMPLOYEES USING P.EMPLOYEE_ID =15231;
&RDB&          P.LAST_NAME = "Santoli";
&RDB&          GET
&RDB&          MY_DB_KEY = P.RDB$DB_KEY;
&RDB&          END_GET
&RDB&      END_STORE
```

STORE Statement

```
&RDB&    COMMIT
&RDB&    START_TRANSACTION READ_WRITE
&RDB&    FOR J IN EMPLOYEES WITH J.RDB$DB_KEY = MY_DB_KEY
&RDB&    GET
1          JC = J.EMPLOYEE_ID;
2          JT = J.LAST_NAME;
3          END_GET
4    END_FOR
          WRITE (6,60020) JC, JT
.
.
.
```

Errors

Your program can check for and handle the following errors, using the name listed here in the form `RDB$_error-name`. The following list includes two types of errors:

- E Expected errors. These are run-time errors. Your program should check for these errors and provide error handlers.
- U Unexpected errors. Preprocessors detect these errors when the program is preprocessed, so your program does not need to check for them. Callable RDO, however, detects these errors at run time. A Callable RDO program should check for them.

DEADLOCK (E)

request failed due to resource deadlock

INTEG_FAIL (E)

constraint *name* failed

LOCK_CONFLICT (E)

NO WAIT request failed because resource was locked

NO_DUP (E)

update would cause duplicates on unique index *name*

NO_PRIV (U)

privilege denied by database protection

NO_SEGSTR_CLOSE (U)

segmented string must be closed before being stored

STORE Statement

NOT_VALID (U)

validation failure on field *name*

OBSOLETE_METADATA (U)

request references undefined fields or relations

READ_ONLY_REL (U)

relation *name* was reserved for READ, updates disallowed

READ_ONLY_TRANS (U)

attempt to update from a READ_ONLY transaction

READ_ONLY_VIEW (U)

view *name* can not be updated

REQ_NO_TRANS (U)

attempt to continue request after COMMIT/ROLLBACK

REQ_WRONG_DB (U)

request database is not in current transaction

SEGSTR_NO_WRITE (U)

segmented string is open for read, write prohibited

UNRES_REL (U)

relation *name* not in the reserving list

WRONUMARG (U)

illegal number of arguments on call to Rdb system

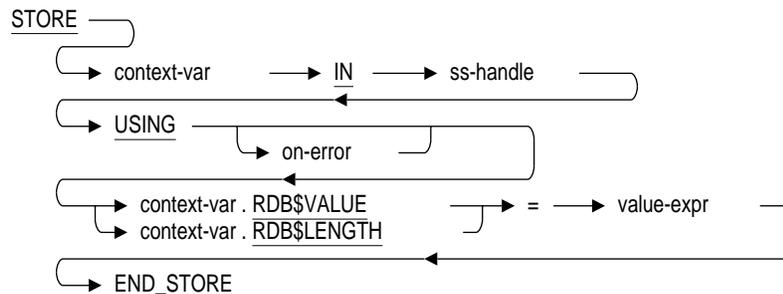
STORE Statement with Segmented Strings

9.69 STORE Statement with Segmented Strings

Inserts a segment into a segmented string. Storing a segmented string requires four steps:

- 1 Create a segmented string with the `CREATE_SEGMENTED_STRING` statement. See Section 9.9 for more information on the `CREATE_SEGMENTED_STRING` statement.
- 2 Store each segment using the syntax shown in the following Format section. You must use the special Rdb/VMS value expression `RDB$VALUE` or `RDB$LENGTH` as the segment name in the `USING` clause.
- 3 Store the entire segmented string using the usual syntax for storing records in relations. Use the segmented string handle, declared in the `CREATE_SEGMENTED_STRING` statement, as the value expression in this `USING` clause.
- 4 Close the segmented string.

Format



Arguments

context-var

A valid context variable. See Section 4.3 for more information.

STORE Statement with Segmented Strings

ss-handle

A host language variable or name used to refer to the segmented string. This handle must match the one declared in the CREATE_SEGMENTED_STRING statement.

on-error

The ON ERROR clause. This clause specifies host language or Rdb/VMS data manipulation statements to be performed if an Rdb/VMS error occurs.

value-expr

A valid Rdb/VMS value expression that specifies the value to be stored.

Usage Notes

You need the Rdb/VMS WRITE privilege for the relation in which rows are stored, and the Rdb/VMS READ and WRITE privileges to the database to use the STORE statement.

Rdb/VMS defines a special name to refer to the segments of a segmented string. This value expression is equivalent to a field name; it names the fields or segments of the string. Furthermore, because segments can vary in length, Rdb/VMS also defines a name for the length of a segment. These names are:

RDB\$VALUE

The value stored in a segment of a segmented string

RDB\$LENGTH

The length in bytes of a segment

When using the RDML and RDBPRE precompilers, be sure to define a sufficiently large value for the RDMS\$BIND_SEGMENTED_STRING_BUFFER logical name. An adequate buffer size is needed to store large segmented strings (using segmented string storage maps) in storage areas other than the default RDB\$SYSTEM storage area. The minimum acceptable value for the RDMS\$BIND_SEGMENTED_STRING_BUFFER logical name must equal the sum of the length of the segments of the segmented string. For example, if you know that the sum of the length of the segments is one megabyte, then 1,048,576 bytes is an acceptable value for this logical name.

You must specify the logical name value because when RDML and RDBPRE precompilers store segmented strings, Rdb/VMS does not know which table contains the string until after the entire string is stored. Rdb/VMS buffers

STORE Statement with Segmented Strings

the entire segmented string, if possible, and does not store it until the STORE statement executes.

If the segmented string remains buffered, it is stored in the appropriate storage area. If the string is not buffered (because it is larger than the defined value for the logical name or the default value of 10,000 bytes), it is not stored in the default storage area and the following exception message is displayed:

```
%RDB-F-IMP_EXC, facility-specific limit exceeded  
-RDMS-E-SEGSTR_AREA_INC, segmented string was stored incorrectly
```

To avoid this error, set the value of the RDMS\$BIND_SEGMENTED_STRING_BUFFER logical name to a sufficiently large value. Note that a value of up to 500 MB can be specified for this logical name. See Section 9.14 for more information on defining storage areas.

Examples

See Section 9.9.

Errors

Your program can check for and handle the following errors, using the name listed here in the form RDB\$_error-name. The following list includes two types of errors:

- E Expected errors. These are run-time errors. Your program should check for these errors and provide error handlers.
- U Unexpected errors. Preprocessors detect these errors when the program is preprocessed, so your program does not need to check for them. Callable RDO, however, detects these errors at run time. A Callable RDO program should check for them.

DEADLOCK (E)
request failed due to resource deadlock

INTEG_FAIL (E)
constraint *name* failed

LOCK_CONFLICT (E)
NO WAIT request failed because resource was locked

NOT_VALID (E)
validation failure on field *name*

STORE Statement with Segmented Strings

NO_DUP (E)

update would cause duplicates on unique index *name*

NO_PRIV (U)

privilege denied by database protection

NO_SEGSTR_CLOSE (U)

segmented string must be closed before being stored

OBSOLETE_METADATA (U)

request references undefined fields or relations

READ_ONLY_REL (U)

relation *name* was reserved for READ, updates disallowed

READ_ONLY_TRANS (U)

attempt to update from a READ_ONLY transaction

READ_ONLY_VIEW (U)

view *name* can not be updated

REQ_NO_TRANS (U)

attempt to continue request after COMMIT/ROLLBACK

REQ_WRONG_DB (U)

request database is not in current transaction

SEGSTR_NO_WRITE (U)

segmented string is open for read, write prohibited

A

Rdb/VMS Reserved Words

This appendix lists the words reserved for use by the Rdb/VMS language and also keywords for RDO commands. Digital Equipment Corporation recommends that you do not use the words in this appendix as names of entities in definition statements. If you do, Rdb/VMS returns an error message in many cases. For example:

```
RDO> DEFINE FIELD AVERAGE DATATYPE TEXT SIZE 9.  
%RDO-F-RESERVED_WORD, AVERAGE is a keyword; it cannot be used as a name  
RDO>
```

```
~  
;  
=  
,  
(  
)  
.  
>  
<  
-  
+  
|  
*  
/  
:  
<>  
>=  
<=  
ABM  
ACCESS  
ACL  
ADJUSTABLE
```

AFTER
AIP_ENTRIES
ALENGTH
ALL
ALLOCATION
ALLOWED
ANALYZE
ANAME
AND
ANY
AREA
AREAS
ASC
ASCII
ASCENDING
AT
ATTACH
AUDIT
AUTO_LOCKING
AUTOMATIC
AVERAGE
BASED
BATCH_UPDATE
BEFORE
BETWEEN
BLR
BUFFER
BUFFERS
BY
BYTE
CARDINALITY
CDD_LINKS
CHANGE
CHECK
CHECKSUM
CLOSE
CLUSTER
COLLATING_SEQUENCE
COMMAND
COMMIT
COMMIT_TIME
COMPILETIME
COMPRESSION
COMPUTED
CONCURRENCY

CONSISTENCY
CONSISTENT
CONSTRAINT
CONSTRAINTS
CONT
CONTAINING
CONVERT
COPY
COPY_DATABASE
CORRUPT
COUNT
CREATE_SEGMENTED_STRING
CROSS
DATA
DATABASE
DATABASES
DATATYPE
DATE
DATE_FORMAT
DAY
DB_HANDLE
DBKEY
DECIMAL
DECLARE_STREAM
DEFAULT
DEFAULTS
DEFAULT_VALUE
DEFERRED
DEFINE
DEFINITION
DELETE
DELETION
DEPOSIT
DESC
DESCENDING
DESCRIPTION
DETACH
DICTIONARY
DISABLE
DISABLED
DISPLAY
DUPLICATES
EACH
EDIT_STRING
ENABLE

ENABLED
END
END_ERROR
END_FETCH
END_FOR
END_GET
END_MODIFY
END_PLACE
END_SEGMENTED_STRING
END_STORE
END_STREAM
ENTRY
EPILOGUE
EQ
ERASE
ERROR
EVALUATING
EVERY
EXCLUSIVE
EXECUTE
EXIT
EXPORT
EXTENSIONS
EXTENT
EXTERNAL
EXTRACTION
FETCH
F_FLOATING
FIELD
FIELDS
FILE
FILENAME
FILL
FINISH
FIRST
FOR
FOREIGN
FORMAT
FREE_SPACE
FRIDAY
FROM
GE
GET
G_FLOATING
GLOBAL

GLOBAL_TRANSACTION
GRANULARITY
GROWTH
GT
HASHED
HEADER
HELP
HEXADECIMAL
IDENTIFIER
IF
IMMEDIATE
IMPORT
IN
INCONSISTENT
INDEX
INDEXES
INDICES
INTEGRATE
INTERVAL
INTO
INUSE
INVOKE
IS
JOURNAL
KEEP
KEY
LABEL
LANGUAGE
LE
LENGTH
LIMIT
LINE
LOCAL
LOCK
LOCKED_FREE_SPACE
LOCKS
LOG
LOGICAL_AREA
LONGWORD
LT
MAKE_CONSISTENT
MANUAL
MAP
MATCHING
MAX

MAXIMUM
MIN
MINIMUM
MISSING
MISSING_VALUE
MIXED
MODIFY
MONDAY
MONITOR
MONTH
MOVE
MOVE_AREA
NE
NEW
NEXT_AIP
NO
NOACL
NOAUTO_LOCKING
NOCDD_LINKS
NODE
NODES
NOEPILOGUE
NOEXECUTE
NOEXTENSIONS
NOJOURNAL
NOKEEP
NOLOG
NOOPEN
NOOUTPUT
NOPROLOGUE
NOT
NOTRACE
NOVERIFY
NOW
NOWAIT
NUMBER
OF
OFFSET
OLD
ON
OPEN
OR
OUTPUT
OVER
PAGE

PAGES
PATHNAME
PERCENT
PHYSICAL_AREA
PLACE
PLACEMENT
POSITION
PREPARE
PRIMARY
PRIVILEGES
PROLOGUE
PROTECTED
PROTECTION
PSECT
PURGE
QUADWORD
QUERY_HEADER
QUERY_NAME
RADIX
RADIX_POINT
RDB\$DB_KEY
RDB\$LENGTH
RDB\$MISSING
RDB\$VALUE
READ
READ_ONLY
READ_WRITE
READY
REAL
RECORD
RECORD_LENGTH
RECORD_TYPE
RECOVER
RECOVER/RESOLVE
RECOVERY
REDUCED
REFERENCES
REFRESH
REINITIALIZE
RELATION
RELATIONS
REORGANIZE
REPLICATION
REQUEST_HANDLE
REQUIRE

REQUIRED
RESERVING
RESOLVE
RETRY
ROLLBACK
ROLLUP
ROOT
RUNTIME
SATURDAY
SCALE
SCHEDULE
SCOPE
SEGMENTED
SEGMENT_LENGTH
SEGMENTS
SELECT
SET
SHARED
SHOW
SIGNED
SIZE
SNAPSHOT
SNAPSHOT_FILENAME
SORTED
SPACE
SPECIFICATION
START
STARTING
START_SEGMENTED_STRING
START_STREAM
START_TRANSACTION
STATISTICS
STATUS
STORAGE
STORAGE_AREA
STORE
STREAMS
STRING
SUB_TYPE
SUNDAY
SYSTEM
TEXT
THRESHOLDS
THURSDAY
TIME_STAMP

TO
TODAY
TOMORROW
TOTAL
TRACE
TRANSACTION
TRANSACTION_HANDLE
TRANSACTIONS
TRANSFER
TRIGGER
TRIGGERS
TUESDAY
TYPE
UNCORRUPT
UNIFORM
UNIQUE
UNTIL
UPDATE
USED
USER
USERS
USING
VALID
VARYING
VAXCLUSTER
VERB_TIME
VERIFY
VERSION
VERSIONS
VIA
VIEW
VIEWS
WAIT
WEDNESDAY
WEEK
WITH
WITHIN
WORD
WRITE
YESTERDAY

B

Rdb/VMS Error Message Explanation Files

Documentation for RDB, RDO, RDMS, and DDAL facility messages is provided in online files:

- Common Rdb/VMS and Rdb/ELN (RDB facility) messages:
SYSS\$COMMON:[SYSHLP]RDB_MSG.DOC
- RDO messages:
SYSS\$COMMON:[SYSHLP]RDO_MSG.DOC
- Rdb/VMS specific (RDMS facility) messages:
SYSS\$COMMON:[SYSHLP]RDMS_MSG.DOC
- VAX Data Distributor (DDAL facility) messages:
SYSS\$HELP:DDAL\$MSG.DOC

Note that the RDO, RDMS, and DDAL error message explanations are also included in the RDO Help file under the Errors module.

The RDB facility messages are no longer included in the RDO Help file because a subsequent installation of VIDA or Rdb/ELN upgrades the RDB_MSG.DOC file whenever the RDB message image, RDBMSG\$.EXE, includes new or revised error messages. Thus, the source for the latest, most up-to-date RDB facility messages is always the RDB_MSG.DOC file in SYSS\$HELP.

The message documentation for all the facilities follows the same format, with messages alphabetized by message name. After the message name and text, the documentation includes an explanation and suggested user action.

You can print the online message documentation files for reference. In

addition, you can use the SEARCH command to see only the message information you need. For example:

```
$ RUN GET_EMPS
%RDB-F-WRONG_ODS, database filename uses wrong version of on disk structure
-RDMS-F-ROOTMAJVER, database format is not compatible with software version
%TRACE-F-TRACEBACK, symbolic stack dump follows
module name      routine name      line      rel PC      abs PC
RDML_SIGNAL_ERR RDML$SIGNAL_ERROR      73      00000015    00000C16
GET_EMPS         main                688      00000046    000008A6

$ SEARCH/WINDOW=(0,10) SYS$HELP:RDMS_MSG.DOC rootmajver
ROOTMAJVER,      database format is not compatible with software version

Explanation:     Your database was created with an incompatible version
                  of the software.

User Action:     Your database cannot be used with the version of the
                  software you have installed on your machine.

                  .
                  .
                  .
```

C

Components of Run-Time Only License

The VAX Rdb/VMS run-time only (RTO) software is designed to let users run existing, executable DML programs that access existing Rdb/VMS databases. Program development using the Rdb/VMS language preprocessors and data definition are not supported by the RTO kit. The RDBPRE.EXE and RDML.EXE preprocessors are not available.

However, a subset of Callable RDO is available in the RTO kit. Thus, you can process VAX host language programs that contain RDB\$INTERPRET calls and only use the same subset of DML statements available in the RTO version of the interactive RDO utility, as shown in Table C-1.

Table C-1 identifies which RDO statements can be executed using the RTO version of Rdb/VMS.

Table C-1 RDO Statements Available in RTO License

Statement or Clause	Available in RTO?
ANALYZE	Yes
AT END	¹
CHANGE DATABASE	Yes
CHANGE FIELD	No
CHANGE INDEX	No
CHANGE PROTECTION	Yes
CHANGE RELATION	No

¹Only available in DML programs already generated using the Rdb/VMS full development kit.

(continued on next page)

Table C-1 (Cont.) RDO Statements Available in RTO License

Statement or Clause	Available in RTO?
CHANGE STORAGE MAP	No
COMMIT	Yes
CREATE_SEGMENTED_STRING	Yes
DCL Invoke (\$)	Yes
DECLARE_STREAM	Yes
DEFINE COLLATING_SEQUENCE	Yes
DEFINE CONSTRAINT	No
DEFINE DATABASE	No
DEFINE FIELD	No
DEFINE INDEX	No
DEFINE PROTECTION	Yes
DEFINE RELATION	No
DEFINE STORAGE MAP	No
DEFINE TRIGGER	No
DEFINE VIEW	No
DELETE COLLATING_SEQUENCE	No
DELETE CONSTRAINT	No
DELETE DATABASE	No
DELETE FIELD	No
DELETE INDEX	No
DELETE PATHNAME	No
DELETE PROTECTION	Yes
DELETE RELATION	No
DELETE STORAGE MAP	No
DELETE TRIGGER	No
DELETE VIEW	No
EDIT	Yes
END_SEGMENTED_STRING	Yes
END_STREAM	Yes
ERASE	Yes
Execute (@)	Yes

(continued on next page)

Table C-1 (Cont.) RDO Statements Available in RTO License

Statement or Clause	Available in RTO?
EXIT	Yes
EXPORT	Yes
FETCH	Yes
FINISH	Yes
FOR	Yes
FOR statement with segmented strings	Yes
GET	Never used in RDO. Use PRINT instead.
HELP	Yes
IMPORT	Yes
INTEGRATE DATABASE	Yes
INVOKE DATABASE	Yes
MODIFY	Yes
Monitor statements (all)	Yes
ON ERROR	¹
PLACE	Yes
PRINT	Yes
ROLLBACK	Yes
SET statements (all)	Yes
SHOW statements (all)	Yes
START_SEGMENTED_STRING	Yes
START_STREAM, declared	Yes
START_STREAM, undeclared	Yes
START_TRANSACTION	Yes
STORE	Yes
STORE statement with segmented strings	Yes

¹Only available in DML programs already generated using the Rdb/VMS full development kit.

D

Default, Minimum, and Maximum Values for Database Parameters

Table D-1 presents the default values for database-wide parameters. It also indicates the minimum and maximum acceptable values for those parameters.

Table D-1 Rdb/VMS Database-Wide Parameter Default Values and Minimum and Maximum Values

Database-Wide Parameters	Default Values (Minimum/Maximum)
Path name	CDD\$DEFAULT
Description	None
Number of users	50 (1/2032)
Number of buffers	20 (2/32768)
Maximum number of VAXcluster nodes	16 (1/64)
Number of recovery buffers	20 (2/32768)
Buffer size	3 times the maximum area page size
Lock granularity	Enabled
Enable/disable snapshot file	Enabled
Snapshot immediate/deferred	Immediate
Database opening	Open is automatic
Specify after-image journal	AIJ is disabled until file name specified

(continued on next page)

Table D-1 (Cont.) Rdb/VMS Database-Wide Parameter Default Values and Minimum and Maximum Values

Database-Wide Parameters	Default Values (Minimum/Maximum)
After-image journal allocation	0 blocks (0/disk device size)
After-image journal extent	512 blocks (no extent options) (0/disk device size)
Requiring a dictionary	Dictionary is not required
Using a dictionary	Dictionary is used

E

Storage Area Parameter Default, Minimum, and Maximum Values

Table E-1 presents the default values for storage area parameters. It also indicates the minimum and maximum values for those parameters.

Table E-1 Rdb/VMS Multifile Storage Area Parameter Default Values and Minimum and Maximum Values

Storage Area Parameters	Default Values (Minimum/Maximum)
Storage area name	RDB\$SYSTEM
Storage area file name	Must be specified (RDA)
Allocation	400 pages (1/disk device size)
Page size	2 blocks (1 block/32 blocks)
Page format	Uniform
SPAM thresholds	70%, 85%, 95% (Mixed only)
SPAM interval	256 pages (Mixed only) ⁽¹⁾
Storage area extent pages	100 pages (0/disk device size)
Storage area extent options	Minimum 99 pages, maximum 9,999 pages, growth 20 percent
Snapshot file name	Same as storage area file (SNP)
Snapshot file allocation	100 pages (0/disk device size)

¹(Minimum: 256/Maximum: ((Blocks-per-page * 512) - 22) * 4)

(continued on next page)

Table E-1 (Cont.) Rdb/VMS Multifile Storage Area Parameter Default Values and Minimum and Maximum Values

Storage Area Parameters	Default Values (Minimum/Maximum)
Snapshot file extent pages	100 pages (0/disk device size)
Snapshot file extent options	Minimum 99 pages, maximum 9,999 pages, growth 20 percent

RDO Statements Not Supported in Rdb/VMS Versions 3.1 and 4.0

This appendix describes RDO statements that are no longer supported or no longer functional. The description of each statement notes the release of Rdb/VMS during which this change of status occurred.

F.1 BACKUP Statement

Note The **BACKUP** statement is no longer supported in Rdb/VMS V3.1. The **EXPORT** statement replaces the **BACKUP** statement. The **BACKUP** statement is maintained for compatibility with applications that used previous versions of Rdb/VMS. However, Digital Equipment Corporation recommends the use of the **EXPORT** statement rather than the **BACKUP** statement.

The **BACKUP** statement makes a copy of a database in an intermediate, compressed form. The **BACKUP** statement lets you move a database from Rdb/VMS to Rdb/ELN or from Rdb/ELN to Rdb/VMS. The **BACKUP** statement creates a special type of RMS sequential file. You can use the Rdb/VMS **RESTORE** statement on this backup file to rebuild the Rdb/VMS database. You can also use the same backup file with Rdb/ELN to build an Rdb/ELN database identical to the Rdb/VMS database. The corresponding Rdb/ELN utility is called EBRP.

BACKUP → db-file-spec → backup-file-spec

db-file-spec

The VMS file specification for the database you want to back up. Use either a full or partial file specification or a logical name. If you use a simple file name, Rdb/VMS looks for the database in the current default directory. If you do not specify a file type, Rdb/VMS uses RDB.

backup-file-spec

The VMS file specification for a file in which BACKUP places a compressed version of the database. Use either a full or partial file specification or a logical name. If you use a simple file name, Rdb/VMS places the backup file in the current default directory. The default file type is RBR.

This parameter can also refer to a magnetic tape volume. If you are backing up the database to tape, use the device name as part of the file specification. For example:

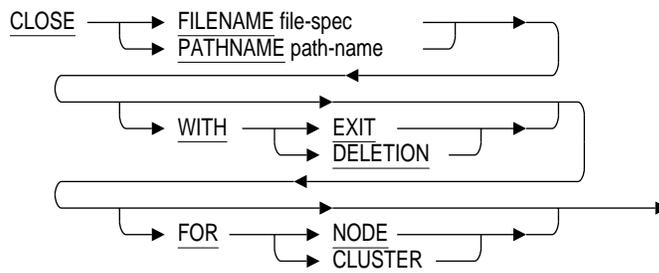
```
RDO> BACKUP 'DISK2:[DEPT3]PERSONNEL.RDB'  
cont> 'MTA0:BACKPERS.RBR'
```

F.2 CLOSE Statement

Note *The CLOSE statement is no longer functional in Rdb/VMS V4.0. The RMU/CLOSE command replaces the CLOSE statement.*

The CLOSE statement controls the process of eliminating active users in a *specific* database. The CLOSE options control whether access to the database is curtailed for users on a single node or for users on all nodes in a VAXcluster environment. Users with sufficient privilege (either ADMINISTRATOR privilege, VMS SYSPRV, or BYPASS) can still enter an explicit OPEN statement to open the database.

If you do not specify the WITH EXIT or WITH DELETION option, Rdb/VMS waits for active users to terminate their sessions with an explicit or implicit FINISH statement, then the database is closed.



FILENAME file-spec

The VMS file specification of the database you want to close.

PATHNAME path-name

The full or relative data dictionary path name in which the definitions reside for the database you want to close.

WITH EXIT

All active users will be immediately forced off the specified database. A user whose process is swapped out is not considered an active user.

WITH DELETION

The processes of all active users for the specified database will be immediately deleted. Because the WITH EXIT option does not force off users whose processes are swapped out, the WITH DELETION option is sometimes necessary to remove all active users. If the FOR CLUSTER option is also specified, the processes of all active users throughout the VAXcluster will be immediately deleted.

FOR NODE

The CLOSE statement affects active users of the specified database on this VAX node only. This is the default.

FOR CLUSTER

The CLOSE statement affects active users of the specified database on all nodes in the VAXcluster.

F.3 CONVERT Statement

Note *The CONVERT statement is no longer functional in Rdb/VMS V3.1. The RMU/CONVERT command replaces the CONVERT statement.*

The CONVERT statement converts an existing database file (RDB) to a format compatible with a new version of Rdb/VMS. Refer to the most recent *VAX Rdb/VMS Installation Guide* for information on whether a conversion is required for the latest release. If a conversion is necessary, back up your existing database(s) with the VMS BACKUP utility and the RDO BACKUP statement before installing the latest version of Rdb/VMS and using the CONVERT statement. In most cases, the database file structure is changed to improve performance.

CONVERT DATABASE FILENAME → db-filename →

db-filename

The file specification for the database you want to convert. Use either a full or partial file specification, or a logical name. If you use a simple file name, Rdb/VMS looks for the database in the current default directory. If you do not specify a file type, Rdb/VMS uses RDB.

The CONVERT statement prompts you with the following query:

Are you satisfied with the backup of <db-file-spec>?

Answer Y or press the RETURN key if you have backup copies of your database (using the VMS Backup utility and the RDO BACKUP statement), or if you are comfortable without the backup copies. While Rdb/VMS does not require that you perform a backup before the conversion, it is strongly recommended that backup copies exist (using the VMS Backup utility and the RDO BACKUP statement). If you answer N, the conversion is not started and the RDO prompt returns.

During a conversion, an informational message is displayed. If the conversion succeeds, a second message indicates that the database has been converted. Also, the version number of the Rdb/VMS software used with the database before and after the conversion is identified.

F.4 OPEN Statement

Note *The OPEN statement is no longer functional in Rdb/VMS V4.0. The RMU/OPEN command replaces the OPEN statement.*

The OPEN statement optimizes database attach operations for all users of a specific database by mapping the database root file for each process. Overhead normally charged to your process is absorbed by Rdb/VMS when you use the OPEN statement. You can use the OPEN statement in conjunction with the CHANGE DATABASE statement to control access to the database. See Section 9.2 for details.



FILENAME file-spec

The VMS file specification for the database root (RDB) file.

PATHNAME path-name

The full or relative data dictionary path name for the database data dictionary entity that refers to the database root (RDB) file.

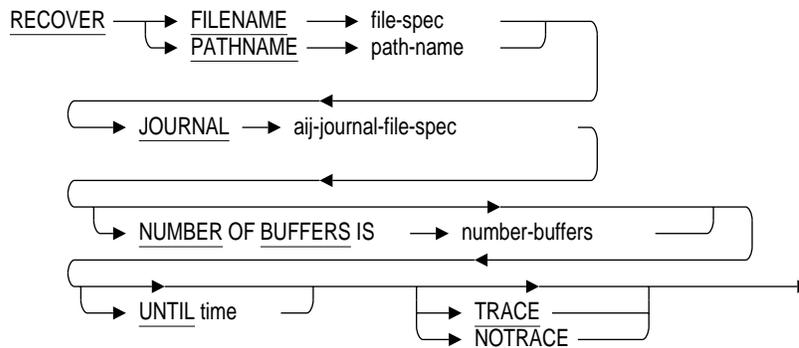
F.5 RECOVER Statement

Note *The RECOVER statement is no longer functional in Rdb/VMS V4.0. The RMU/RECOVER command replaces the RECOVER statement.*

The RECOVER statement reconstructs a database by reentering past transactions from the after-image journal (AIJ) file in a database restored from a VMS Backup utility (BACKUP) copy. After-image journaling is especially useful for repairing a database after a disk failure. See the *VAX Rdb/VMS Guide to Database Maintenance and Performance* for complete information on database recovery.

The RECOVER statement rolls forward completed (committed) transactions from the time the AIJ file is opened until the AIJ file is closed. For example, your installation may keep an AIJ file for each day's transactions. In this case, you probably maintain archived AIJ files on tape. When you have restored a copy of the database from a known good point prior to the system failure, you can apply one or more AIJ files to that copy of the database.

Incomplete transactions roll back automatically when a failure occurs. These transactions must be reentered in the usual way after the RECOVER statement has executed.



FILENAME file-spec

The VMS file specification for the database to be recovered. If you use a partial file specification, Rdb/VMS uses it in conjunction with the fully qualified database file specification that is recorded in the specified after-image journal file. It is recommended that you use a fully qualified file specification instead of a partial file specification.

PATHNAME path-name

The path name that refers to the data dictionary entity for the database. Put the name in quotation marks. You can specify a full or relative path name. When you specify **PATHNAME**, the fully qualified file specification that is recorded in the data dictionary is passed as the file specification for the database to be recovered.

JOURNAL aij-journal-file-spec

The file specification for the AIJ file to be used as the source of the transactions that the **RECOVER** statement reenters in the database. The *aij-journal-file-spec* must be a fully qualified file specification or a systemwide logical name that translates to a fully qualified file specification. This file specification should match the AIJ file you specified when you enabled journaling with the **CHANGE DATABASE** statement.

NUMBER BUFFERS IS number-buffers

The number of database buffers used during the roll-forward process. Specifying a large number of buffers results in a faster recovery time. The default is 20 buffers.

time

The time that represents the limit to the recovery operation. If you use the **UNTIL** qualifier, all the operations in the AIJ file will be applied to the target database from the time the AIJ file was created to the time specified by the *time* parameter. If you do not specify the **UNTIL** qualifier, Rdb/VMS applies the entire AIJ file.

The time must be in the standard VMS absolute time format:

dd-mmm-yyyy hh:mm:ss.cc

You can omit any of the trailing fields in the date or time. You can omit any of the fields in the middle of the format as long as you specify the punctuation marks. For more information on the format of absolute date and time, see the information on **\$BINTIM** system service in the VMS documentation set.

TRACE

NOTRACE [Default]

Determines whether details of the roll-forward process are displayed at your terminal.

F.6 REFRESH MONITOR LOG Statement

Note *The REFRESH MONITOR LOG statement is no longer supported in Rdb/VMS V4.0. The RMU/MONITOR REOPEN_LOG command replaces the REFRESH MONITOR LOG statement. The REFRESH MONITOR LOG statement is maintained for compatibility with applications that used previous versions of Rdb/VMS. However, Digital Equipment Corporation recommends the use of the RMU/MONITOR REOPEN_LOG command rather than the REFRESH MONITOR LOG statement.*

The REFRESH MONITOR LOG statement closes the current Rdb/VMS monitor log file (SYSSYSTEM:RDMMON.LOG by default) and opens another log file without stopping the monitor. You or your database administrator should use this statement if the monitor log file is getting too large.

REFRESH MONITOR LOG

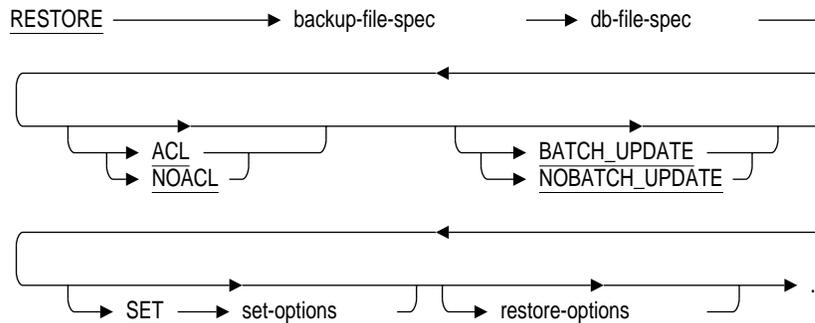
F.7 RESTORE Statement

Note *The RESTORE statement is no longer supported in Rdb/VMS V3.1. The IMPORT statement replaces the RESTORE statement. The RESTORE statement is maintained for compatibility with applications that used previous versions of Rdb/VMS. However, Digital Equipment Corporation recommends the use of the IMPORT statement rather than the RESTORE statement.*

The RESTORE statement re-creates a database that has been backed up with an Rdb/VMS BACKUP statement. Use the RESTORE statement to transfer a database from one Rdb database management system to another. You can use the RESTORE statement to transfer a database to an Rdb/VMS system, whether the backup file was created by Rdb/VMS or by Rdb/ELN.

A BACKUP statement translates the definitions and data in a database into an intermediate form in a special type of RMS sequential file. The RESTORE statement reads the records in this file and uses them to create an Rdb/VMS database identical to the one that the BACKUP statement used.

The Rdb/VMS BACKUP and RESTORE statements are intended for migrating a database from one database system to another. For regular backups of the database, use the RMU/BACKUP and RMU/RESTORE commands.



db-file-spec

The VMS file specification for the new database you want to create from the backup copy. Use either a full or partial file specification or a logical name. If you use a partial file specification, Rdb/VMS creates the database in the current default directory. If you do not specify a file type, Rdb/VMS uses the default file type RDB.

ACL [Default]***NOACL***

ACL, the default, causes the restoration of the access control list (ACL) from the backed-up database. NOACL overrides the default. Use the NOACL option if the ACL from the backed-up database prevents you from accessing the restored database.

BATCH_UPDATE [Default]***NOBATCH_UPDATE***

In the BATCH_UPDATE transaction (the default), before-image journaling is not performed. The NOBATCH_UPDATE option overrides the default and causes the RESTORE operation to run in EXCLUSIVE share mode. If an error occurs during the RESTORE operation, this provides journaling so Rdb/VMS can roll back the database to the last successfully restored database entity.

set-options

For each optional SET clause you enter (you can include multiple SET clauses, one for each index and/or relation whose characteristics you wish to change or preserve), the set option arguments are index-name to DISABLE/ENABLE COMPRESSION in the following list:

index-name

The name of the index whose NODE SIZE, PERCENT FILL, or USAGE clause value you want to change or preserve in the newly restored database.

NODE SIZE number-bytes

The size of each index node. The number and level of the resulting index nodes depend on this value, the number and size of the index keys, and the value of the PERCENT FILL clause. For each user-defined index that does not include a SET INDEX . . . NODE SIZE clause, Rdb/VMS uses default values. These default values are:

- 430 bytes if the total index key size is 120 bytes or less
- 960 bytes if the total index key size is more than 120 bytes

PERCENT FILL number

Sets the *initial* fullness percentage for each node in the index specified in the SET INDEX clause. The valid range is 1 percent to 100 percent. The default PERCENT FILL of each index is 70 percent in the newly restored database if you omit this clause, or if you use the SET INDEX clause for an index but omit the PERCENT FILL argument. If the PERCENT FILL and USAGE clauses are both specified in the same SET INDEX clause, the USAGE value is used.

USAGE UPDATE

The default. There is no need to specify this clause in the RESTORE . . . SET clause. The USAGE UPDATE clause sets the PERCENT FILL value to 70 percent.

USAGE QUERY

Sets the PERCENT FILL value at 100 percent. Specify this clause for each index whose PERCENT FILL value you want to set at 100 percent (or use PERCENT FILL 100). Supplying PERCENT FILL and USAGE QUERY options is allowed in the syntax; however, the USAGE option takes precedence over an explicit PERCENT FILL value. The existing value of the NODE SIZE parameter is unchanged if you do not specify a new NODE SIZE value along with the USAGE UPDATE clause.

relation-name

The name of the relation for which you want to preserve disabled data compression.

DISABLE COMPRESSION**ENABLE COMPRESSION [Default]**

Specifies whether records in the relation will be compressed or uncompressed when stored. ENABLE COMPRESSION is the default.

path-name

The dictionary path name for the directory where the database definition will be stored.

Specify either:

- A full dictionary path name such as DISK1:[DICTIONARY]CORP.EMPS
- A relative dictionary path name such as EMPS

If you use a relative path name, CDD\$DEFAULT must be defined to include all the path name segments that precede the relative path name. If you do not specify a dictionary path name for the database, Rdb/VMS appends the database file name to the current default dictionary directory. This new path name becomes the name of the dictionary directory for the database definitions.

ALLOCATION IS number-pages

The number of database pages allocated to the database initially. Rdb/VMS automatically extends the allocation to handle the loading of data and subsequent expansion. The default is 400 pages. If you are loading a large database, a large allocation prevents the file from having to be extended many times.

PAGE SIZE IS page-blocks

The size in blocks of each database page. Page size is allocated in 512-byte blocks. The default is 2 blocks (1024 bytes). If your largest record is larger than approximately 950 bytes, allocate more blocks per page to prevent records from being fragmented.

NUMBER VAXCLUSTER NODES n

Sets the upper limit on the maximum number of VAXcluster nodes from which users can access the shared database. The default is 16 nodes. The range is 1 node to 64 nodes. The actual maximum limit is the current VMS VAXcluster limit.

NUMBER BUFFERS IS number-buffers

The number of buffers Rdb/VMS allocates per process using this database. Specify an unsigned integer greater than zero. The default is 20 buffers.

NUMBER USERS IS number-users

The maximum number of users allowed to access the database at one time. The default is 50 users. After the maximum number is reached, the next user who tries to invoke the database receives an error message and must wait. The largest number of users you can specify is 508 and the fewest number of users is 1. With VAX ACMS, each attached server can support multiple users. Thus, the actual number of terminal users can be greater than 508 if you are using VAX ACMS.

Note that the number of users is defined as the number of active attaches to the database. Thus, if a single process is running one program, but that program performs 12 INVOKE . . . FINISH operations, to Rdb/VMS there are 12 active users.

BUFFER SIZE IS buffer-blocks

The number of blocks Rdb/VMS allocates per buffer. Specify an unsigned integer greater than zero. If you do not specify this parameter, Rdb/VMS uses a buffer size that is three times the PAGE SIZE value.

SNAPSHOT IS ENABLED IMMEDIATE [Default]

SNAPSHOT IS ENABLED DEFERRED

The SNAPSHOT IS ENABLED IMMEDIATE option specifies that read/write transactions write copies of records to the snapshot file before those records are modified, regardless of whether a read-only transaction is active.

The **SNAPSHOT IS ENABLED DEFERRED** option specifies that read/write transactions *not* write copies of records they modify to the snapshot file unless a read-only transaction is active. Read-only transactions that attempt to start after an active read/write transaction begins must wait for all active read/write users to complete their transactions.

The default is **SNAPSHOT IS ENABLED IMMEDIATE**.

SNAPSHOT IS DISABLED

Disables snapshot writing.

Do not delete snapshot files unless you also delete the database itself. If you do not want snapshots enabled, disable them with the **SNAPSHOT IS DISABLED** clause.

SNAPSHOT ALLOCATION IS snp-pages

The number of pages allocated for the snapshot file. The default is 100 pages.

EXTENT IS extent-pages

SNAPSHOT EXTENT IS extent-pages

The number of pages of each extent. Use this parameter for simple control over the extent. For greater control, and particularly for multivolume databases, use the **MIN**, **MAX**, and **PERCENT** parameters instead. The default is 100 pages.

min-pages

The minimum number of pages of each extent. The default is 100 pages.

max-pages

The maximum number of pages of each extent. The default is 10,000 pages.

growth

The percent growth of each extent. The default is 20 percent growth.

F.8 SHOW MONITOR Statement

Note The SHOW MONITOR statement is no longer supported in Rdb/VMS V4.0. The RMU/SHOW SYSTEM command replaces the SHOW MONITOR statement. The SHOW MONITOR statement is maintained for compatibility with applications that used previous versions of Rdb/VMS. However, Digital Equipment Corporation recommends the use of the RMU/SHOW SYSTEM command rather than the SHOW MONITOR statement.

The SHOW MONITOR statement displays information about the Rdb/VMS monitor and all users attached to a database. It displays the following information:

- Product version, node, time
- Database name
- Process identification (PID), process name, user name, and status of the user
- Image name

SHOW MONITOR

F.9 SHOW USERS Statement

Note The *SHOW USERS* statement is no longer supported in Rdb/VMS V4.0. The *RMU/SHOW USERS* command replaces the *SHOW USERS* statement. The *SHOW USERS* statement is maintained for compatibility with applications that used previous versions of Rdb/VMS. However, Digital Equipment Corporation recommends the use of the *RMU/SHOW USERS* command rather than the *SHOW USERS* statement.

The *SHOW USERS* statement displays information about active database users on a VAXcluster node. A user is any process that is attached to an Rdb/VMS database. A user can be an RDO user, an application program, a remote user, or a journaling or recovery process. The *SHOW USERS* statement displays the following information:

- Product version, node, time
- Database name
- PID, process name, user name, and status of the user
- Image name

To display information about users on *all* nodes in a VAXcluster, use the *RMU/DUMP/USERS* command described in Chapter 6.



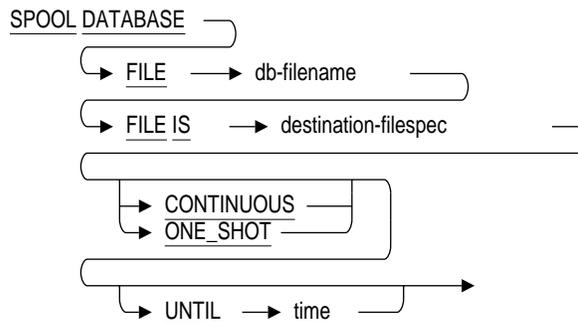
db-handle

A host language variable or name that you associate with the database. If you specify a database handle, the *SHOW USERS* statement displays information about the users attached to that database.

F.10 SPOOL Statement

Note *The SPOOL statement is no longer functional in Rdb/VMS V3.1. The RMU/BACKUP/AFTER_JOURNAL command replaces the SPOOL statement.*

The SPOOL statement directs Rdb/VMS to copy information from the after-image journal (AIJ) file to a file on a tape drive, or an alternate disk drive. When you issue the SPOOL statement, Rdb/VMS copies AIJ entries to the specified device either as a continual process or as a single event. As the AIJ entries are copied to the destination device, they are simultaneously deleted from the AIJ. The SPOOL statement, therefore, allows you to manage primary disk storage for after-image journaling efficiently, and provides you with a permanent copy of the journal on magnetic tape.



db-filename

The name of the database file with file type RDB. To avoid ambiguity, place quotation marks around the file specification.

destination-filespec

The full file specification of the destination file, including the device name. You can specify either a tape drive or another disk as the destination device. To avoid ambiguity, place quotation marks around the file specification.

CONTINUOUS

Directs Rdb/VMS to copy journal entries to the destination file as a non-stop operation.

ONE_SHOT

Directs Rdb/VMS to copy journal entries to the destination file as a single, attached task.

UNTIL time

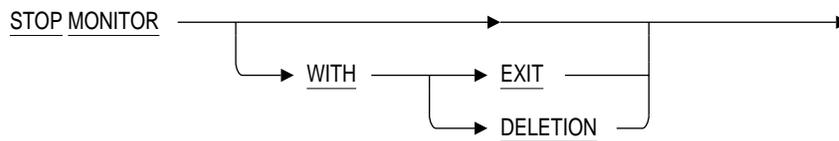
The point at which the spooling task terminates. Use the standard VMS time specification.

F.11 STOP MONITOR Statement

Note *The STOP MONITOR statement is no longer supported in Rdb/VMS V4.0. The RMU/MONITOR STOP command replaces the STOP MONITOR statement. The STOP MONITOR statement is maintained for compatibility with applications that used previous versions of Rdb/VMS. However, Digital Equipment Corporation recommends the use of the RMU/MONITOR STOP command rather than the STOP MONITOR statement.*

The STOP MONITOR statement stops the VAX Rdb/VMS monitor.

If you enter the STOP MONITOR statement without any options, the default is to let active users complete their database activity up to the FINISH statement. An active user is defined as any user who is attached to the database and has not yet entered a FINISH or (in RDO) EXIT statement.



WITH EXIT

Immediately forces all active database users off any Rdb/VMS databases on the VAX node by forcing their images to exit.

WITH DELETION

Deletes all processes that were actively using Rdb/VMS databases on the VAX node.

Index

\$ (dollar sign)

See Dollar sign statement (\$)

A

Absolute time, 3–10, 9–166

DAY, 9–166

in dates, 3–10

MONTH, 9–167

WEEK, 9–166

weekday, 9–167

Access control entry (ACE), 9–47, 9–141

Access control lists (ACLs), 9–43,
9–132, 9–187

auditing changes to, 6–132

Accessing multiple databases

using the READY statement, 9–353

Accessing records

START_TRANSACTION statement,
9–443

Accessing the database

DEFINE PROTECTION statement,
9–132

user identification code (UIC), 9–44,
9–134

Access modes

updates

effect on database functions,
9–443t

Access privilege sets (APSS)

auditing changes to, 6–132

Access rights

See also Privileges, Protection

assigning, 9–45, 9–132

displaying, 9–392

displaying all ACL entries

SHOW PROTECTION statement,
9–395

field level, 9–132

ACE

See Access control entry (ACE)

ACLs

See Access control lists (ACLs)

Adding

database records, 9–457

storage areas, 9–17

ADMINISTRATOR privilege, xxi,

9–142, 9–143t

using to override database privileges,
9–142

Advancing in a stream

FETCH statement, 9–276

After-image journal (AIJ)

backup, 6–37

copying, 6–37

disabling, 9–16

displaying output, 6–59

- After-image journal (AIJ) (Cont.)
 - displaying unresolved transactions, 6–60
 - enabling, 9–16
 - using to recover a database, 6–91
- Aggregate expressions
 - See* Statistical expressions
- AIJ
 - See* After-image journal (AIJ)
- Alarms
 - security, 6–127
- Allocating snapshot file size, 9–15, 9–105, 9–189
- Allocating storage area pages, 9–15, 9–103
- ALL option
 - MOVE clause, 9–192
 - SHOW TRANSFER statement, 9–410
- Alphabetic characters, 3–6
- Alphanumeric identifiers, 9–134
- ANALYZE statement, 9–2
 - See also* RMU/ANALYZE command
 - arguments, 9–2
 - description, 9–2
- Analyzing
 - cardinality of tables and indexes, 6–9
 - databases, 6–5, 6–16
 - indexes, 6–13
 - record placement relative to indexes, 6–16
- AND logical operator, 3–47
- ANY clause, 3–39t
- AREA . . . PAGE command (RdbALTER), 7–3
- Area qualifiers
 - See* Parameter qualifiers
- Arithmetic expressions, 3–21
 - format, 3–3
 - order of evaluation, 3–22
- ASCENDING
 - sort-clause, 9–192
- Ascending index, 9–122
- Assigning access rights, 9–132
- Assistance in RDO
 - HELP statement, 9–297
- AT END clause
 - error handling, 9–276
- @ (at sign)
 - See* Execute statement (@)
- At sign command (@)
 - See* Execute statement (@)
- ATTACH command
 - releasing, 7–17
- ATTACH command (RdbALTER), 7–5
- Attaching RdbALTER to a database, 7–5
- Attaching to a database, 9–329
 - privilege required, 9–334
- Attributes
 - global, 5–1
 - local, 5–2
- Audit event classes, 6–128
- Audit events
 - disabling DACCESS, 6–128
 - enabling DACCESS, 6–128
- Auditing
 - ACL changes, 6–132
 - APS changes, 6–132
 - disabling, 6–128
 - enabling, 6–127, 6–128
 - specific objects, 6–130
 - specific object types, 6–129
 - starting, 6–133
 - stopping, 6–133
 - use of RMU commands, 6–132
- Audit journal, 6–127
- Audit journal records
 - defining database table for storing, 6–71
 - defining relation for storing, 6–71
- AUTO_LOCKING option
 - See* WITH AUTO_LOCKING clause
- AVERAGE statistical expression, 3–14
- AVERAGE statistical function, 3–11t

B

- Backing up a database
 - by area
 - RMU/BACKUP/EXCLUDE command, 6-24
 - RMU/BACKUP/INCLUDE command, 6-26
 - EXPORT statement, 9-271
- Backing up AIJ file
 - SPOOL statement, F-18
- Backup
 - after-image journal file, 6-37
 - by area, 6-24
 - purpose of, 6-24
 - database, 6-20, 6-107
 - online, 6-28
 - to tape
 - truncation of backup file name, 6-33, 6-119
- Backup file
 - displaying output, 6-62
 - truncation of name, 6-33, 6-119
- Backup operations
 - by area
 - purpose of, 6-24
 - to tape
 - truncation of backup file name, 6-33, 6-119
- BACKUP statement
 - See also* EXPORT statement
 - copying a database, F-1
- BATCH_UPDATE transaction mode, 9-437
- BETWEEN clause, 3-39t
- Binding to a database
 - See* Attaching to a database
- Blocked transactions
 - See* Unresolved transactions
- Boolean expressions
 - See* Conditional expressions

- B-tree index
 - See* Sorted index
- Buffer blocks
 - allocation, 9-101, 9-188
- Buffers
 - allocation, 9-188
- Buffer size
 - specifying, 9-100
- By area backup
 - purpose of, 6-24
- By area restore
 - purpose of, 6-110, 6-112
- BYPASS privilege, 9-142

C

- Cardinality
 - of tables and indexes
 - RMU/ANALYZE/CARDINALITY command, 6-9
- Cascading delete
 - specifying through
 - DEFINE TRIGGER statement, 9-204, 9-213e
- Cascading update
 - specifying through
 - DEFINE TRIGGER statement, 9-204, 9-213e
- CDD\$DATABASE dictionary entity
 - copying data definitions from, 9-325
- CHANGE DATABASE statement, 9-9
 - after-image journaling process, 9-16
 - database shutdown process, 9-23
 - examples, 9-23
 - modifying database parameters, 9-9
- OPEN IS clause
 - AUTOMATIC option, 9-12
 - examples, 9-23
 - MANUAL option, 9-12
 - examples, 9-23
- required privileges, 9-22

CHANGE DATABASE statement (Cont.)

 SNAPSHOT IS clause

 ENABLED DEFERRED option,
 9-14

 ENABLED IMMEDIATE option,
 9-14

CHANGE FIELD statement, 9-27

CHANGE INDEX statement, 9-36

CHANGE PROTECTION statement,
9-43

 modifying access rights, 9-43

CHANGE RELATION statement, 9-48

 modifying local field attributes, 9-53

 options and privileges, 9-57t

CHANGE STORAGE MAP statement

 examples, 9-67

 NO PLACEMENT VIA INDEX clause,
 9-64

 PLACEMENT VIA INDEX clause,
 9-63

 REORGANIZE clause, 9-64

Changing

 access rights, 9-45

 databases, 9-9

 default storage area, 9-21

 fields, 9-27

 indexes, 9-36

 protection, 9-43

 RDO parameters, 9-362

 record values, 9-338

 relations, 9-48

 storage map definitions, 9-61

Characteristics

 security auditing, 6-143

Character string literals, 3-6

Checking

 tape labels

 with RMU, 6-34, 6-66, 6-120

Checking internal structures, 6-161

Classes

 of audit events, 6-128

CLOSE statement

See also RMU/CLOSE command

 closing a database, F-3

Closing

See also RMU/CLOSE command
 a database, 6-42

 CLOSE statement, F-3

 FINISH statement, 9-280

 monitor log file, 6-79

 open streams, 9-261

 segmented strings, 9-260

Clumplets

 used in RDBVMS\$TRIGGER_
 ACTIONS, 8-36

Collating sequence clause, 9-31, 9-100,
9-115

See also CHANGE FIELD statement

See also DEFINE DATABASE
 statement

See also DEFINE FIELD statement

See also IMPORT statement

Collating sequences

 ASCII, 9-85, 9-100, 9-307

 changing, 9-34, 9-34e

 DEFINE COLLATING_SEQUENCE
 statement, 9-85

 DELETE COLLATING_SEQUENCE
 statement, 9-225, 9-226e

 deleting, 9-225

 displaying information about, 9-376

 NCS, 9-85, 9-100, 9-307

 predefined, 9-86

 restrictions for changing, 9-31

 specifying for a field, 5-19

 user-defined, 9-86, 9-100, 9-307

 viewing at DCL level, 9-100

COLLATING_SEQUENCE clause

 DEFINE DATABASE statement,
 9-100

 DEFINE FIELD statement, 9-116

 IMPORT statement, 9-307

Column headers

 PRINT statement, 9-350

- Command procedures
 - logical symbol names for, 9–198, 9–199
- COMMIT command (RdbALTER), 7–7
- COMMIT statement
 - writing changes to a database, 9–69
- Committing changes
 - with RdbALTER, 7–7
- Compressed index, 9–127
 - defining, 9–123
- COMPUTED_BY clause (Data Distributor)
 - transferring fields, 9–192
- COMPUTED_BY fields (Data Distributor)
 - transferring views, 9–193
- Concatenated expressions, 3–25
- Concealed logical names, 9–108, 9–317
- Conditional-expr (Data Distributor)
 - RSE, 9–191
 - with-clause, 9–191
- Conditional expressions
 - defining, 3–1
 - definition, 3–36
- Connecting to a database, 9–329
- Constants
 - date, 3–9
- Constraints
 - DEFINE CONSTRAINT statement, 9–88
 - defining, 9–88
 - defining on extraction databases, 9–185
 - defining on extraction rollup databases, 9–186
 - DELETE CONSTRAINT statement, 9–228
 - displaying information, 9–377
 - evaluation of, 9–91
 - naming in
 - CHANGE RELATION statement, 9–52
 - DEFINE RELATION statement, 9–157
- Constraints (Cont.)
 - not deleted by
 - DELETE TRIGGER statement, 9–253
 - showing in
 - SHOW TRIGGERS statement, 9–413
- CONTAINING clause, 3–39t
- Context variables, 4–6
 - format, 3–3
- Context-variables (Data Distributor)
 - first-clause, 9–191
- Continuation prompt (cont>), 1–3
- Control statements
 - summary, 2–8t
- Conventions
 - used in manual, xviii
- Conversion of data types
 - by RMU/LOAD operations, 6–72
- Converting databases
 - CONVERT statement, F–5
 - RMU/CONVERT command, 6–46
 - RMU/RESTORE command, 6–107
- CONVERT statement
 - See also* RMU/CONVERT command
 - converting a database, F–5
- Copying
 - a database, 6–50
 - after-image journal file, 6–37
- Copying a database
 - BACKUP statement, F–1
- Copy process, 9–195
- Copy process log files, 9–198
 - logical symbol names for, 9–198
- Corrupted database
 - recovery of, 6–100
- COUNT statistical expression, 3–14
- COUNT statistical function, 3–11t
- CREATE_SEGMENTED_STRING statement, 9–74
- Creating
 - a duplicate database
 - RMU/COPY_DATABASE command, 6–50

Creating (Cont.)

- databases, 9–93
 - restricting users from, 9–107
 - field definitions, 9–112
 - index definitions, 9–119
 - relational constraints, 9–155
 - relation definitions, 9–149
 - storage map definitions, 9–171
 - view definitions, 9–219
- CROSS clause (of RSE), 4–12
- Cross product, 4–12
- CTRL/Z
- leaving RDO, 1–3

D

- DACCESS audit events
- enabling or disabling, 6–128
 - enabling or disabling for specific objects, 6–129
- DACCESS privileges for database objects, 6–130
- Data
- definition statements, 2–2
 - manipulation statements, 2–5
- Database
- adding records
 - STORE statement, 9–457
 - analyzing
 - RMU/ANALYZE command, 6–5
 - attaching
 - INVOKE DATABASE statement, 9–329
 - backup
 - RMU/BACKUP command, 6–20
 - changing root file specification, 7–16
 - changing to read-only access, 9–19
 - changing to read/write access, 9–19
 - closing
 - RMU/CLOSE command, 6–42
 - creating a duplicate of
 - RMU/COPY_DATABASE command, 6–50

definitions

- CHANGE DATABASE statement, 9–9
 - DEFINE DATABASE statement, 9–93
 - DELETE DATABASE statement, 9–232
- detaching
- FINISH statement, 9–280
- displaying fields, 7–18
- displaying information about, 9–379
- displaying security auditing characteristics, 6–143
- duplicating metadata, 9–273, 9–299, 9–314
- erasing records
- ERASE statement, 9–263
- exporting without the data
- See also* EXPORT statement, xxii
- full backup, 6–107
- full restore, 6–108
- handle, 3–4, 9–99
- importing without the data, xxii
- See also* IMPORT statement
- incremental backup, 6–107
- incremental restore, 6–108
- information
- SHOW MONITOR statement, F–16
 - SHOW USERS statement, F–17
- key, 3–32
- setting scope, 9–331
- maintenance statements
- summary, 2–7t
- maintenance statements summary, 2–7
- managing AIJ files, 6–37
- monitor
- starting, 6–80
 - stopping, 6–82
- monitor log
- refreshing, 6–79
- moving, 7–1
- moving data, 7–33

Database (Cont.)

- moving root file, 6–85
- moving storage areas, 6–85
- multifile, 9–97
- online backup, 6–20
- opening
 - RMU/OPEN command, 6–89
- pages
 - analyzing usage, 9–2
 - checksum verification of, 6–162
- parameters
 - default values, D–1
 - maximum values, D–1
 - minimum values, D–1
- patching, 7–1
- physical design
 - default values
 - database-wide, D–1t
 - storage area parameter, E–1t
- protection
 - CHANGE PROTECTION statement, 9–43
 - DEFINE PROTECTION statement, 9–132
 - DELETE PROTECTION statement, 9–241
- recovery, 6–91, 6–100
 - single-file databases, 6–95
- resolving an unresolved transaction, 6–100, 6–103
- restoring, 6–107
- restructuring, 9–9, 9–64, 9–271, 9–299
- security
 - CHANGE PROTECTION statement, 9–43
 - DEFINE PROTECTION statement, 9–132
 - DELETE PROTECTION statement, 9–241
- statistics, 6–148
 - ANALYZE statement, 9–2
 - RMU/ANALYZE/CARDINALITY command, 6–9

Database

- statistics (Cont.)
 - RMU/ANALYZE command, 6–5
 - RMU/ANALYZE/INDEXES command, 6–13
 - RMU/ANALYZE/PLACEMENT command, 6–16
- system relations, 8–1
- uncorrupting, 7–39
- unloading tables or views, 6–158
- verifying integrity, 6–161
- Database contents
 - displaying output
 - RMU/DUMP command, 6–54
- Database definition
 - allowing users, 9–136
 - restricting users from, 9–107
 - storing in dictionary, 9–102, 9–310
- Database handle, 9–74, 9–99
 - scope of, 9–329
- Database information
 - SHOW ALL statement, 9–375
 - SHOW COLLATING_SEQUENCE statement, 9–376
 - SHOW CONSTRAINT statement, 9–377
 - SHOW DATABASES statement, 9–379
 - SHOW DATE_FORMAT statement, 9–381
 - SHOW DICTIONARY statement, 9–383
 - SHOW FIELDS statement, 9–384
 - SHOW INDEXES statement, 9–387
 - SHOW LANGUAGE statement, 9–391
 - SHOW PRIVILEGES statement, 9–392
 - SHOW PROTECTION statement, 9–395
 - SHOW RADIX_POINT statement, 9–397
 - SHOW RELATIONS statement, 9–398

- Database information (Cont.)
 - SHOW STORAGE AREAS statement, 9-400
 - SHOW STORAGE MAPS statement, 9-404
 - SHOW STREAMS statement, 9-406
 - SHOW TRANSACTION statement, 9-407
 - SHOW TRANSFER statement, 9-409
 - SHOW TRIGGERS statement, 9-413
 - SHOW VERSIONS statement, 9-416
- Database keys (dbkeys), 3-32
 - See also* Dbkeys
- Database migration
 - converting to higher version
 - RMU/CONVERT command, 6-46
- Database names
 - CDD/Plus restrictions, 9-97
- Database pages
 - allocation, 9-187
 - block size, 9-187
 - checksum verification of, 6-162
- Database parameters
 - default values, D-1
 - maximum values, D-1
 - minimum values, D-1
- Databases
 - accessing multiple, 9-436
 - closing, 6-42
 - opening, 6-89
- DATABASE statement
 - See* INVOKE DATABASE statement
- Database table
 - for storing security audit journal records, 6-71
- Database users
 - eliminating
 - RMU/CLOSE command, 6-42
- Data definition
 - statements
 - summary, 2-2
- Data definition statements
 - summary, 2-2t
- Data dictionary, 1-7
- Data Distributor error messages
 - explanation files, B-1
- Data Distributor statements
 - See also* VAX Data Distributor
 - summary, 2-10t
- Data manipulation statements
 - summary, 2-5, 2-5t
- DATATRIEVE support clauses
 - access characteristics, 5-16
 - DEFAULT_VALUE clause, 5-17
 - EDIT_STRING clause, 5-16, 5-17
 - QUERY_HEADER clause, 5-16, 5-17
 - QUERY_NAME clause, 5-16, 5-17
- DATATYPE clause, 5-5
- Data types, 5-7t
 - conversion of by RMU/LOAD operations, 6-72
 - DATE, 5-6
 - for Rdb/VMS, 5-7
 - F_FLOATING, 5-7t
 - G_FLOATING, 5-7t
 - REAL, 5-6
 - SEGMENTED STRING, 5-8
 - SIGNED BYTE, 5-7t
 - SIGNED LONGWORD, 5-7t
 - SIGNED QUADWORD, 5-7t
 - SIGNED WORD, 5-7t
 - TEXT, 5-7t
 - VARYING STRING, 5-8t
- Date
 - in absolute time format, 3-10
- DATE data type, 5-6
 - format of, 3-10
 - references from literals, 3-9
- Date formats
 - displaying, 9-381
 - displaying language for, 9-391
 - language for, 9-365
 - SET DATE_FORMAT statement, 9-363

- Date formats (Cont.)
 - SHOW DATE_FORMAT statement, 9-381
- Date literals, 3-9
- DATE_FORMAT clause
 - of SET statement, 9-363
 - of SHOW statement, 9-381
- Db-handle (Data Distributor)
 - relation-clause, 9-191
- Dbkeys
 - STORE statement
 - retrieving value of record just stored, 9-457
- DBKEY SCOPE clause
 - using with INVOKE DATABASE, 9-331
- DCL Invoke statement (\$), 9-81
 - accessing DCL, 9-81
- DECdtm services
 - coordinating distributed transactions with, 9-436
 - defined, 9-436
- Declared record streams
 - START_STREAM statement, 9-422
- Declared streams, 9-82
- DECLARE_STREAM statement, 9-82
 - record stream, 9-82
- DETrace for VMS software
 - RMU/MONITOR STOP command, 6-83
- DEFAULT identifier
 - DEFINE PROTECTION command, 9-144
- Default protection
 - modifying, 9-144
 - specifying, xxi
- Defaults
 - database parameters, D-1
 - DEFINE DATABASE statement, 9-96, 9-98
 - global, 9-21, 9-98, 9-105, 9-107, 9-310, 9-316
 - local, 9-21, 9-107, 9-310, 9-316
 - storage area parameters, E-1
- Default storage area, 9-21, 9-62, 9-105, 9-173
- Default value
 - defined with SQL, 3-29
 - effects of changing, 3-29
- DEFAULT_VALUE FOR DTR clause, 5-17
- Deferred snapshots
 - changing with CHANGE DATABASE, 9-14
- DEFINE COLLATING_SEQUENCE statement, 9-34e, 9-85
- DEFINE CONSTRAINT statement, 9-88
 - restricting values, 9-88
- DEFINE DATABASE statement, 9-93
 - creating a database, 9-93
 - examples, 9-108
- DEFINE FIELD statement, 9-112
 - creating field definitions, 9-112
- DEFINE INDEX statement, 9-119
 - creating index definitions, 9-119
- DEFINE PROTECTION command
 - DEFAULT identifier, 9-144
- DEFINE PROTECTION statement, 9-132
 - accessing the database, 9-132
- DEFINE RELATION statement
 - creating relation definitions, 9-149
- DEFINE SCHEDULE statement, 9-164
 - changing parameters in, 9-168
 - defining schedules, 9-164
 - restrictions, 9-168
 - RETRY clause, 9-168
 - time formats, 9-165
 - UIC required with, 9-168
 - with suspended transfers, 9-168
 - with unscheduled transfers, 9-168
- DEFINE STORAGE MAP statement, 9-171
 - defining storage maps, 9-171
 - PLACEMENT VIA INDEX clause, 9-174
- DEFINE TRANSFER statement, 9-180

DEFINE TRANSFER statement (Cont.)
 defaults, 9-186
 file specification, 9-186
 logical symbol names in, 9-198
 move-relations-clause, 9-189, 9-197
 move-relations-rollup-clause, 9-194, 9-197
 move-views-clause, 9-193
 SYSS\$LOGIN, 9-186
 transfer-file-options-clause, 9-194
DEFINE TRIGGER statement, 9-204
DEFINE VIEW statement, 9-219
 creating view definitions, 9-219
Defining
 collating sequences, 9-85
 constraints, 9-88
 databases, 9-93
 restricting users from, 9-107
 data definition statements, 2-2
 extent size, 9-189
 fields, 9-112
 indexes, 9-119
 protection, 9-132
 relations, 9-149
 schedules, 9-164
 snapshot file size, 9-189
 storage area options, 9-310
 storage areas, 9-93
 storage maps, 9-171
 transfers, 9-180
 triggers, 9-204
 views, 9-219
Defining a database
 restricting users from, 9-107, 9-136
DELETE COLLATING_SEQUENCE statement, 9-225
DELETE CONSTRAINT statement, 9-228
DELETE DATABASE statement, 9-232
DELETE FIELD statement, 9-234
DELETE INDEX statement, 9-237
DELETE PATHNAME statement, 9-240
DELETE PROTECTION statement, 9-241
DELETE RELATION statement, 9-244
DELETE SCHEDULE statement, 9-247
DELETE STORAGE MAP statement, 9-249
DELETE TRANSFER statement, 9-251
DELETE TRIGGER statement, 9-253
DELETE VIEW statement, 9-255
Deleting
 collating sequences, 9-225
 constraints, 9-228
 database definitions, 9-232
 field definitions, 9-234
 indexes, 9-237
 protection, 9-241
 records from a database, 9-263
 relations, 9-244
 schedules, 9-247
 storage areas, 9-20
 storage maps, 9-249
 transfers, 9-251
 triggers, 9-253
 views, 9-255
Deletion
 cascading with DEFINE TRIGGER statement, 9-204, 9-213e
Delta time, 9-166
DEPOSIT command (RdbALTER), 7-8
DEPOSIT FILE command (RdbALTER), 7-14
DEPOSIT ROOT command (RdbALTER), 7-16
DESCENDING
 sort-clause, 9-192
Descending index, 9-123
DETACH command (RdbALTER), 7-17
Detaching RdbALTER from a database, 7-17
Dictionary
 disabling use of, 9-102, 9-310
 enforcing use of, 9-16, 9-102, 9-309
 restructuring, 9-318

Dictionary (Cont.)

- setting default, 9-364
- Dictionary definitions
 - copying, 9-325
 - deleting, 9-240
 - re-creating, 9-325
 - shareable fields, 9-112
 - shareable relations, 9-149
- Dictionary path names, 1-8
 - displaying
 - SHOW DICTIONARY statement, 9-383
- Disabling after-image journaling, 9-16
- Disabling DACCESS audit events, 6-128
- Disabling security alarms, 6-128
- Disabling snapshot files, 9-188
 - for a multifile database, 9-14, 9-102
- DISPLAY command (RdbALTER), 7-18
- DISPLAY FILE command (RdbALTER), 7-25
- Displaying
 - access rights, 9-392
 - all ACL entries, 9-395
 - all database information, 9-375
 - collating sequence, 9-376
 - current dictionary default directory, 9-383
 - database information
 - using RMU, 6-1
 - date format, 9-381
 - field names, 9-384
 - index names, 9-387
 - information about a database
 - SHOW statements, 9-372t
 - information about databases
 - RMU/SHOW SYSTEM command, 6-154
 - information about transactions, 9-407
 - information about transfers, 9-409
 - information about users, 6-155
 - language for date format, 9-391
 - language for time format, 9-391

Displaying (Cont.)

- names of relations, 9-398
- names of streams, 9-406
- privileges, 9-392
- radix point character, 9-397
- Rdb/VMS version number, 6-157
- relation definitions, 9-398
- statistics, 6-148
- storage area names, 9-400
- storage map names, 9-404
- time format, 9-381
- transfer state, 9-410
- version number, 9-416
- Displaying output
 - after-image journal
 - RMU/DUMP/AFTER_JOURNAL command, 6-59
 - backup file
 - RMU/DUMP/BACKUP_FILE command, 6-62
 - database
 - RMU/DUMP command, 6-54
 - recovery-unit journal
 - RMU/DUMP/RECOVERY_JOURNAL command, 6-68
- DISPLAY ROOT command (RdbALTER), 7-27
- Distributed TID
 - defined, 9-436
 - specifying value of, 9-436
- Distributed-tid variable
 - defined, 9-437
 - initializing, 9-437
 - specifying values of, 9-437
- Distributed transaction
 - coordinating, 9-436
 - defined, 9-436
 - joining, 9-436
 - resolving, 6-103
 - specifying, xxi
 - starting, 9-436
 - START_TRANSACTION statement, 9-431

- Distributed transaction identifier
 - See* Distributed TID
 - See* Distributed-tid variable
- DISTRIBUTED_TRANSACTION clause
 - of START_TRANSACTION statement, 9-436
- Distributing databases
 - DEFINE SCHEDULE statement, 9-164
 - DEFINE TRANSFER statement, 9-180
 - DELETE SCHEDULE statement, 9-247
 - DELETE TRANSFER statement, 9-251
 - REINITIALIZE TRANSFER statement, 9-357
 - SHOW TRANSFER statement, 9-409
 - START TRANSFER statement, 9-453
 - STOP TRANSFER statement, 9-456
 - summary of statements, 2-10t
 - using RDO statements, 2-9
- Dollar sign statement (\$)
 - invoking DCL, 9-81
- E**
- Editing buffer
 - controlling size, 9-364
- EDIT statement
 - editing command lines, 9-257
 - EDT editor, 9-257
 - VAXTPU editor, 9-257
 - RDO\$EDIT logical, 9-257
- EDIT_STRING FOR DTR clause, 5-16, 5-17
- EDT editor
 - using EDIT statement, 9-257
- Eliminating active users
 - RMU/CLOSE command, 6-42
- Enabling after-image journaling, 9-16
- Enabling DACCESS audit events, 6-128
- Enabling security alarms, 6-128
- Enabling security auditing, 6-127, 6-128
- Enabling snapshot files, 9-188
 - for a multifile database, 9-14, 9-98, 9-102
- Ending an RDO session
 - EXIT statement, 9-270
- Ending transactions
 - COMMIT statement, 9-69
 - ROLLBACK statement, 9-359
- End-of-stream
 - AT END clause, 9-276
- END_SEGMENTED_STRING statement
 - closing a segmented string, 9-260
- END_STREAM statement, 9-261
 - closing an open stream, 9-261
- Enforcing use of dictionary, 9-16, 9-102, 9-309
- Epilogue-file-spec
 - transfer-file-options-clause, 9-194
- EQ relational operator, 3-39t
- ERASE statement
 - deleting records from a database, 9-263
 - specifying through
 - DEFINE TRIGGER statement, 9-207
- Error handling
 - AT END clause, 9-276
 - ON ERROR clause, 9-343
 - online message documentation, B-1
- Error messages
 - location of explanation files, B-1
- Evaluating constraints, 9-91
- EXCLUSIVE UPDATE lock (RdbALTER)
 - , 7-5, 7-17
- Execute statement (@)
 - running RDO command files, 9-267
- EXIT command (RdbALTER), 7-28
- Exiting from RDO
 - EXIT statement, 9-270
- EXIT statement, 1-3, 9-270
- Exporting a database, 9-271

- Exporting a database (Cont.)
 - without the data, xxii
 - See also* EXPORT statement
- EXPORT statement
 - backing up a database, 9-271
- Expression
 - arithmetic, 3-21
 - concatenated, 3-25
- Expressions, 1-6, 3-1
 - arithmetic
 - order of evaluation, 3-22
- Extending dbkey scope, 9-331
- Extent
 - after-image journal file, 9-17
 - database file, 9-312
 - snapshot file, 9-15, 9-104, 9-189, 9-312
- Extraction rollup
 - defining, 9-185
- Extractions
 - defining, 9-185
- F**
- FETCH statement
 - advancing in a stream, 9-276
- Field attributes, 5-1
 - data types, 5-5
 - for VAX Rdb/VMS, 5-5
 - missing value, 5-13
 - validity, 5-11
- Field definitions
 - CHANGE FIELD statement, 9-27
 - CHANGE RELATION statement, 9-52
 - DEFINE FIELD statement, 9-112
 - DELETE FIELD statement, 9-234
 - displaying
 - SHOW FIELDS statement, 9-384
 - shareable, 9-112
- Field names
 - displaying
 - SHOW FIELDS statement, 9-384
 - format, 3-3
- Fields
 - specifying access rights, 9-133
- FILACCERR
 - in RMU, 6-33
- File
 - directing output to, 9-365
- File access conflicts
 - in RMU, 6-33
- File name
 - truncation of during RMU/BACKUP command, 6-33, 6-119
- File qualifiers
 - See* Parameter qualifiers
- File specifications, 1-8
 - changing with RdbALTER, 7-14, 7-16
 - DEPOSIT FILE command, 7-14
 - duplicates, 9-187
- File type
 - RDO, 7-5
- FINISH statement
 - closing a database, 9-280
- First-clause (Data Distributor)
 - context-variable, 9-191
 - db-handle, 9-191
 - RSE, 9-191
- FIRST clause (of RSE), 4-3
- FIRST FROM expression, 3-3, 3-31
- FOREIGN KEY constraints
 - naming in
 - CHANGE RELATION statement, 9-52
 - DEFINE RELATION statement, 9-157
- FOR statement
 - loops, 9-282
 - segmented strings, 9-287
 - transaction handle restriction, 9-283
- Function
 - global aggregate, 3-16
- F_FLOATING data type, 5-7t

G

Gathering statistics

- ANALYZE statement, 9-2
- RMU/ANALYZE/CARDINALITY command, 6-9
- RMU/ANALYZE command, 6-5
- RMU/ANALYZE/INDEXES command, 6-13
- RMU/ANALYZE/PLACEMENT command, 6-16

General identifiers, 9-134

GE relational operator, 3-39t

GET statement

- retrieving records from a stream, 9-291

Global aggregate function, 3-16

Global attributes, 5-1

Global defaults, 9-21, 9-98, 9-105, 9-107, 9-310, 9-316

Granting privileges

- CHANGE PROTECTION statement, 9-43
- DEFINE PROTECTION statement, 9-132

GT relational operator, 3-39t

G_FLOATING data type, 5-7t

H

Handle

- database, 3-4, 9-74, 9-99
 - scope of, 9-329
- request, 3-5, 9-82, 9-282, 9-425, 9-458
 - setting scope, 9-332
- segmented string, 9-74
- transaction, 3-5, 9-83, 9-283, 9-426, 9-458

Handling errors

- ON ERROR clause, 9-343
- online message documentation, B-1

Hashed index, 9-40, 9-122, 9-127

HELP command (RdbALTER), 7-29

HELP statement, 1-2

- assistance on RDO topics, 9-297
- ### Host language variables, 3-4
- format, 3-2

I

Identifier clause, 9-44, 9-134

Identifiers, 9-44, 9-134

- general, 9-45, 9-134
- system-defined, 9-45, 9-134
- UIC, 9-44, 9-134

Importing a database

- without the data, xxii

See also IMPORT statement

IMPORT statement, 9-299

Index definitions

- CHANGE INDEX statement, 9-36
- DEFINE INDEX statement, 9-119
- DELETE INDEX statement, 9-237
- displaying
 - SHOW INDEXES statement, 9-387

Indexes

- analyzing
 - RMU/ANALYZE/INDEXES command, 6-13
 - analyzing usage, 9-3
 - ascending, 9-122
 - changing, 9-36
 - compressed, 9-123, 9-127
 - defining, 9-119
 - defining on extraction databases, 9-185
 - defining on extraction rollup databases, 9-186
 - defining storage areas for, 9-38
 - deleting, 9-237
 - descending, 9-123
 - displaying, 9-387
 - hashed, 9-40, 9-122, 9-127
 - showing statistics
 - RMU/ANALYZE/INDEXES command, 6-13

- Indexes (Cont.)
 - sorted, 9–40, 9–122
 - statistics, 6–148
 - storing records by means of, 9–63, 9–174
- Initialization file, 1–2
- Input formats
 - for dates and times
 - defining, 9–363
 - displaying, 9–381
- INTEGRATE DATABASE statement, 9–325
- Interactive control statements
 - summary, 2–8, 2–8t
- Internationalization features
 - changing field collating sequences, 9–31
 - deleting collating sequences, 9–225
 - displaying database collating sequences, 9–376
 - displaying date format, 9–381
 - displaying field collating sequences, 9–376
 - displaying language for date formats, 9–391
 - displaying language for time formats, 9–391
 - displaying radix point character, 9–397
 - displaying time format, 9–381
 - logical names used for, 9–366
 - specifying database collating sequences, 9–85, 9–100, 9–307
 - specifying date formats, 9–363
 - specifying field collating sequences, 9–86, 9–116
 - specifying language for date formats, 9–365
 - specifying language for time formats, 9–365
 - specifying radix point character, 9–365
 - specifying time formats, 9–363
- INVOKE DATABASE statement, 9–329
 - INVOKE DATABASE statement (Cont.)
 - setting scope of dbkey, 9–331
 - setting scope of request handle, 9–332
 - transaction handle restriction, 9–332, 9–333
 - Invoking a database
 - privileges required, 9–334
 - Invoking RdbALTER utility, 6–4
 - Invoking RDO, 1–1
- J**
 - Join operation
 - See* CROSS clause
 - Journal
 - security audit, 6–127
 - Journaling
 - after-image
 - RMU/RECOVER command, 6–91
 - disabling, 9–16
 - enabling, 9–16
- K**
 - Keywords, 1–4, A–1
 - optional, 1–5
 - required, 1–4
- L**
 - Labels
 - checking on tapes, 6–34, 6–66, 6–120
 - Language for date format
 - displaying, 9–391
 - SET LANGUAGE statement, 9–365
 - SHOW LANGUAGE statement, 9–391
 - specifying, 9–365
 - LE relational operator, 3–39t
 - Literals, 3–6
 - character string, 3–6
 - compile-time translation of, 3–8
 - date, 3–9
 - nonnumeric, 1–6
 - numeric, 1–5, 3–8

- Loading relations, 6–70
- Loading tables, 6–70
 - from security audit journal, 6–70
- Local attributes, 5–2
- Local defaults, 9–21, 9–107, 9–310, 9–316
- Locking relations
 - during a transaction, 9–442
- Lock timeout
 - for quiet point
 - specifying the duration of, 6–32
- LOG command (RdbALTER), 7–30
- Log files
 - closing, 6–79
 - copy process, 9–195
- Logical names
 - concealed, 9–108, 9–317
- Logical operators, 3–47, 3–47t
- Loops
 - FOR statement, 9–282
- LT relational operator, 3–39t

M

- MAKE_CONSISTENT command (RdbALTER), 7–31
- MAX statistical function, 3–11t, 3–15
- Metadata
 - system relations, 8–1
- MIN statistical function, 3–11t
- Missing value, 3–26
 - displaying with RDO, 3–28, 3–29
 - displaying with SQL, 3–29
 - effects of changing, 3–29
 - SQL interpretation of, 3–29
- MISSING_VALUE clause, 3–39t, 5–13
 - format, 3–3
 - retrieving, 3–26
- Modifying
 - access rights, 9–45
 - database parameters, 9–9
 - default storage area, 9–21
 - field definitions, 9–27
 - index definitions, 9–36

Modifying (Cont.)

- local field attributes, 9–53
- privileges
 - CHANGE PROTECTION statement, 9–43
- protection, 9–43
- relations, 9–48
- storage map definitions, 9–61
- MODIFY statement, 9–338
 - specifying through DEFINE TRIGGER statement, 9–207
- Monitor
 - closing log file, 6–79
 - refreshing log file, 6–79, F–9
 - starting, 6–80
 - stopping, 6–82
- MOVE clause
 - ALL option, 9–192
- MOVE command (RdbALTER), 7–33
- Move-relations-clause, 9–197
 - defining a transfer, 9–189
 - MOVE RELATIONS ALL option, 9–190
 - RSE, 9–190
 - select field name option, 9–192
- Move-relations-rollup-clause, 9–197
 - defining a transfer, 9–194
- Move-views-clause
 - defining a transfer, 9–193
 - view-name, 9–193
- Moving
 - a database, 7–1
 - root file, 6–85
 - storage areas, 6–85
- Multifile database, 9–97
 - adding storage areas, 9–17
 - creating from single-file database, 9–299
 - defining, 9–93
 - deleting storage areas, 9–20
 - disabling snapshot files, 9–14, 9–102
 - enabling snapshot files, 9–14, 9–98, 9–102

Multifile database (Cont.)
 storage area options, 9–17, 9–20, 9–98

Multiple storage areas
 for segmented strings, 9–173

N

Names
 user-supplied, 1–5
 using uppercase letters, 9–97

NCS\$LIBRARY
 viewing collating sequences in, 9–86, 9–100, 9–307

NCS/LIST command
 viewing collating sequences at DCL level, 9–100

NCS utility, 2–2t, 9–85
 default library, 9–86
 specifying collating sequences defined with, 2–3, 9–85, 9–100, 9–307

NE relational operator, 3–39t

New features
 of Rdb/VMS, xxi
 of RMU, xxii

NO COLLATING_SEQUENCE clause
 CHANGE FIELD statement, 9–31
 DEFINE FIELD statement, 9–116

NODE SIZE clause
 CHANGE INDEX statement, 9–37
 DEFINE INDEX statement, 9–125
 displaying value, 9–387

NOLOG command (RdbALTER), 7–35

Nonnumeric literals, 1–6

Non-supported statements
See Obsolete statements

NO PLACEMENT VIA INDEX clause
 CHANGE STORAGE MAP statement, 9–64

NOT logical operator, 3–47

Null flag
 defined, 3–26

Numeric literals, 1–5, 3–8

Numeric string
 format, 3–3

O

Objects
 auditing, 6–129, 6–130
 auditing for specific privileges, 6–130

Object types
 auditing, 6–129
 auditing for specific privileges, 6–130

Obsolete statements, F–1
 BACKUP statement, F–1
 CLOSE statement, F–3
 CONVERT statement, F–5
 OPEN statement, F–6
 RECOVER statement, F–7
 REFRESH MONITOR LOG statement, F–9
 RESTORE statement, F–10
 SHOW MONITOR statement, F–16
 SHOW USERS statement, F–17
 SPOOL statement, F–18
 STOP MONITOR statement, F–20

ON ERROR clause
 handling an error, 9–343

Opening
 after-image journal file, 9–16
 databases, 6–89, 9–353
 OPEN statement, F–6
 READY statement, 9–353
 RMU/OPEN command, 6–89
 record streams, 9–422, 9–425
 root files, 6–33

OPEN statement
 opening a database, F–6

OPERATOR privilege, xxi, 9–142, 9–143t
 using to override database privileges, 9–142

Optional condition clause
 specifying through
 DEFINE TRIGGER statement, 9–206

- Optional keywords, 1–5
- Optional root file parameter
 - RMU/ALTER command, 7–1
- OR logical operator, 3–47
- Output
 - directing to a file or terminal screen, 9–365
- Override privileges
 - See* Role-oriented privileges
- Overriding global defaults, 9–98, 9–310, 9–316

P

- PAGE command (RdbALTER), 7–36, 7–41e
- Parameter
 - default values for database, D–1t
 - maximum values for database, D–1t
 - minimum values for database, D–1t
 - qualifiers
 - See* Parameter qualifiers
- Parameter qualifiers
 - defined, 6–3
 - global use of, 6–3
 - local use of, 6–3
 - positional semantics of, 6–3
- Parameters
 - for RMU commands, 6–2
- Partitioning data, 9–62, 9–171
- Patching databases, 6–4, 7–1
- Path names
 - dictionary, 1–8
 - displaying default directory, 9–383
 - relative, 1–8
- PERCENT FILL clause
 - CHANGE INDEX statement, 9–38
 - DEFINE INDEX statement, 9–125
- PLACEMENT VIA INDEX clause
 - CHANGE STORAGE MAP statement, 9–63
 - DEFINE STORAGE MAP statement, 9–174

- PLACE statement, 9–346
- Positional qualifiers
 - See* Parameter qualifiers
- PRIMARY KEY constraints
 - naming in
 - CHANGE RELATION statement, 9–52
 - DEFINE RELATION statement, 9–157
- PRINT statement, 9–349
- Privileges
 - changing
 - CHANGE PROTECTION statement, 9–43
 - defining
 - CHANGE PROTECTION statement, 9–43
 - DEFINE PROTECTION statement, 9–132
 - denying
 - CHANGE PROTECTION statement, 9–43
 - displaying, 9–392
 - SHOW PROTECTION statement, 9–395
 - granting
 - CHANGE PROTECTION statement, 9–43
 - DEFINE PROTECTION statement, 9–132
 - modifying
 - CHANGE PROTECTION statement, 9–43
 - removing
 - CHANGE PROTECTION statement, 9–43
 - revoking
 - CHANGE PROTECTION statement, 9–43
 - role-oriented, xxi, 9–142
- Process
 - statistics, 6–148
- Project operation
 - See* REDUCED TO clause (of RSE)

Prologue-file-spec
transfer-file-options-clause, 9-194

Prompts, 1-3

Protection

changing, 9-43
defining, 9-132
extraction rollup databases, 9-186
modifying default, 9-144
on extraction databases, 9-185
specifying default, xxi

Protection definition

CHANGE PROTECTION statement,
9-43
DEFINE PROTECTION statement,
9-132

Proxy accounts, 9-187

Q

Qualifiers

for RMU commands, 6-2

RMU

area, 6-3
file, 6-3
parameter, 6-3
positional, 6-3

QUERY_HEADER FOR DTR clause,
5-16, 5-17

QUERY_NAME FOR DTR clause, 5-16,
5-17

Quiet point lock timeout

specifying the duration of, 6-32

Quotation marks

in character strings, 3-7

Quoted strings, 3-6

format, 3-3

R

RADIX command (RdbALTER), 7-37

Radix point

SET RADIX_POINT statement,
9-365
SHOW RADIX_POINT statement,
9-397

RDB\$CONSTRAINTS system relation,
8-4

RDB\$CONSTRAINT_RELATIONS
system relation, 8-5

RDB\$DATABASE system relation, 8-6

RDB\$DB_KEY value expression
used within STORE . . . END_STORE
block, 3-33, 9-291, 9-292, 9-294,
9-331, 9-336, 9-349, 9-352,
9-457, 9-461

RDB\$FIELDS system relation, 8-11

RDB\$FIELD_VERSIONS system
relation, 8-9

RDB\$INDEX_SEGMENTS system
relation, 8-15

RDB\$INDICES system relation, 8-16

RDB\$LENGTH segmented string
expression, 3-34, 5-10, 9-465

RDB\$MISSING expression, 3-26

RDB\$RELATIONS system relation,
8-20

RDB\$RELATION_FIELDS system
relation, 8-17

RDB\$SYSTEM storage area, 9-21,
9-62, 9-98, 9-105, 9-173

changing to read-only, 9-15

changing to read/write, 9-15

RDB\$VALUE segmented string
expression, 3-34, 5-10, 9-465

RDB\$VIEW_RELATIONS system
relation, 8-23

RdbALTER

altering data fields, 7-8

attaching to a database, 7-5

changing area and snapshot file
specifications, 7-14

changing root file specification, 7-16

changing root file specifications, 7-16

commands, 7-1

AREA . . . PAGE, 7-3

ATTACH, 7-5

COMMIT, 7-7

DEPOSIT, 7-8

DEPOSIT FILE, 7-14

RdbALTER

commands (Cont.)

- DEPOSIT ROOT, 7-16
 - DETACH, 7-17
 - DISPLAY, 7-18
 - DISPLAY FILE, 7-25
 - DISPLAY ROOT, 7-27
 - EXIT, 7-28
 - HELP, 7-29
 - LOG, 7-30
 - MAKE_CONSISTENT, 7-31
 - MOVE, 7-33
 - NOLOG, 7-35
 - PAGE, 7-36
 - RADIX, 7-37
 - ROLLBACK, 7-38
 - UNCORRUPT, 7-39
 - VERIFY, 7-41
 - committing changes, 7-7
 - detaching from a database, 7-17
 - displaying area or snapshot file specifications, 7-25
 - displaying data fields, 7-18
 - displaying root file specifications, 7-27
 - ending the session, 7-28
 - exclusive update lock, 7-5, 7-17
 - fetching pages, 7-36
 - getting information on, 7-29
 - invoking, 6-4
 - keeping an audit trail, 7-30
 - moving data, 7-33
 - patching fields, 7-8
 - resetting an inconsistent flag, 7-31
 - resetting the corruption flag, 7-39
 - setting default radix, 7-37
 - specifying area, 7-3
 - specifying page, 7-3
 - static verification, 7-41
 - stopping logging, 7-35
 - undoing changes, 7-38
 - verifying a page, 7-41
- RDB error messages
explanation files, B-1

Rdb/VMS

- new features, xxi
- security auditing
 - enabling, 6-127
- RDBVMSSCOLLATIONS system
 - relation, 8-24
- RDBVMSSCONSTRAINT_TYPE field
 - values for, 8-30
- RDBVMSSCREATE_DB identifier
 - allowing database creation, 9-136
 - controlling database creation, 9-107
- RDBVMSSCREATE_DB logical name
 - allowing database creation, 9-136
 - controlling database creation, 9-107
- RDBVMSSINTERRELATIONS system
 - relation, 8-25
- RDBVMSSPRIVILEGES system
 - relation, 8-26
- RDBVMSSRELATION_CONSTRAINTS system relation, 8-27
- RDBVMSSRELATION_CONSTRAINT_FLDS system relation, 8-31
- RDBVMSSSTORAGE_MAPS system
 - relation, 8-31
- RDBVMSSSTORAGE_MAP_AREAS system relation, 8-32
- RDBVMSSTRIGGERS system relation, 8-33
- Rdb/VMS Management Utility (RMU), 6-1
 - See also* RMU/(command name)
- RDMS error messages
 - explanation files, B-1
- RDO
 - command lines
 - EDIT statement, 9-257
 - database maintenance statements, 2-7
 - data definition statements, 2-2
 - data manipulation statements, 2-5
 - distributing database statements, 2-9
 - exiting, 1-3, 9-270
 - HELP statement, 9-297

RDO (Cont.)

- initialization file, 1–2
- interactive control statements, 2–8
- invoking, 1–1
- language elements
 - summary, 1–1 to 1–10
- obsolete statements
 - See* Obsolete statements
- prompt, 1–3
- RTO license
 - availability of RDO statements, 9–1, C–1
- statements
 - See also* each statement's entry
 - components of, 1–3
 - unsupported statements, F–1
- RDO\$EDIT logical
 - using with EDIT statement, 9–257
- RDO error messages
 - explanation files, B–1
- RDOINI.RDO
 - RDO initialization file, 1–2
- RDO statements
 - components of, 1–3
- READALL privilege, 9–143
- Read-only databases
 - creating, 9–20
- Read-only storage areas
 - changing to read/write access, 9–20
- Read/write storage areas
 - changing to read-only access, 9–20
- READY statement
 - described, 9–353
 - example, 9–354, 9–355, 9–356
 - opening a database, 9–353
 - to access multiple databases, 9–353
- REAL data type, 5–6
- Record data
 - availability of during trigger updates, 9–211
- Record selection expression
 - See also* RSE
 - restriction for START_STREAM statement, 9–422
- Record streams, 4–1
 - advancing
 - FETCH statement, 9–276
 - closing, 9–261
 - DECLARE_STREAM statement, 9–82
 - FOR statement, 9–282
 - START_STREAM statement, 9–425
- Record values
 - modifying, 9–338
 - retrieving
 - GET statement, 9–291
 - PRINT statement, 9–349
- RECOVER statement
 - See also* RMU/RECOVER command
 - re-entering lost transactions, F–7
- Recovery of databases, 6–91, 6–93, 6–96e
 - areas, 6–92
 - logging recovery, 6–93
 - number of buffers used, 6–92
 - specifying new location for root file, 6–94
 - specifying time limits, 6–93
 - tracing recovery, 6–93
- Recovery of single-file databases, 6–95
- Recovery-unit journal (RUJ)
 - displaying output, 6–68
- Re-creating a database
 - RESTORE statement, F–10
- Re-creating dictionary definitions, 9–325
- Reduce-clause (Data Distributor)
 - RSE, 9–191
- REDUCED TO clause (of RSE), 4–11
- Re-entering lost transactions
 - RECOVER statement, F–7
- Referential integrity
 - relation-specific constraints, 9–157
 - triggers, 9–204
 - using constraints to maintain, 9–157
- REFRESH MONITOR LOG statement
 - refreshing the monitor log, F–9

- REINITIALIZE TRANSFER statement, 9-357
- Relation
 - for storing security audit journal records, 6-71
- Relational constraints
 - DEFINE RELATION statement, 9-155
- Relational Database Operator utility
 - See* RDO
- Relational joins
 - See* CROSS clause
- Relational operators, 3-39, 3-39t
- Relation clause, 4-6
 - RSE (Data Distributor), 9-191
- Relation definitions
 - CHANGE RELATION statement, 9-48
 - DEFINE RELATION statement, 9-149
 - DELETE RELATION statement, 9-244
 - shareable, 9-149
 - SHOW RELATIONS statement, 9-398
 - specifying default protection for, 9-144
- Relations
 - changing, 9-48
 - default protection for
 - modifying, 9-144
 - defining, 9-149
 - deleting, 9-244
 - inserting records into, 9-457
 - loading, 6-70
 - locking during a transaction, 9-442
 - storing in multiple areas, 9-176
 - system, 8-1
 - unloading from the database, 6-158
- Relative path name, 1-8
- Remote access
 - access control lists, 9-187
 - proxy accounts, 9-187
- Remote databases
 - specifying, 1-9
- REORGANIZE clause
 - effects of not specifying, 9-67
 - effects of using, 9-66
 - of CHANGE STORAGE MAP statement, 9-64
 - using by areas, 9-66
 - using by pages, 9-66
- Reorganizing a database, 9-64
- Replacing field definitions
 - CHANGE FIELD statement, 9-27
- Replacing index definitions
 - CHANGE INDEX statement, 9-36
- Replication databases
 - special-purpose relations, 9-185
- Replications
 - defining, 9-185
 - updates, 9-185
 - using REINITIALIZE TRANSFER statement, 9-357
- Request handle, 3-5, 9-82, 9-282, 9-425, 9-458
 - availability of, 3-5
 - restriction for FOR statement, 9-283
 - restriction for INVOKE DATABASE statement, 9-332, 9-333
 - restriction for START_STREAM statement, 9-422
 - setting scope, 9-332
 - zeroing out, 3-5
- Required keywords, 1-4
- Reserved words
 - See* Keywords
- RESERVING clause
 - defined, 9-442
 - START_TRANSACTION statement, 9-442
- Resolving
 - unresolved database transactions, 6-100, 6-103
- Restore operations
 - by area
 - purpose of, 6-110, 6-112

RESTORE statement
See also RMU/RESTORE command
 re-creating a database, F-10

Restoring
 database, 6-107
 IMPORT statement, 9-299
 from backup, 6-107
 from by area backup, 6-110, 6-112
 full restore, 6-108
 incremental restore, 6-108

Restricting values
 DEFINE CONSTRAINT statement,
 9-88

Restructuring
 a database, 9-9, 9-64, 9-271, 9-299
 a dictionary, 9-318

Retrieving records from a stream
 GET statement, 9-291
 PRINT statement, 9-349

Retry count, 9-167

Retrying transfers, 9-186

RMU
 area qualifiers
 See Parameter qualifiers
 command parameters, 6-2
 command qualifiers, 6-2
 FILACCERR, 6-33
 file access conflicts, 6-33
 file qualifiers
 See Parameter qualifiers
 new features, xxii
 parameter qualifiers
 defined, 6-3
 positional semantics of, 6-3
 positional qualifiers
 See Parameter qualifiers

RMU/ALTER command, 6-4
See also RdbALTER
 invoking the RdbALTER utility, 6-4
 patching databases, 6-4

RMU/ANALYZE/CARDINALITY
 command, 6-9
 /CONFIRM, 6-10

RMU/ANALYZE/CARDINALITY
 command (Cont.)
 description, 6-9
 /OUTPUT, 6-11
 storing cardinality of tables and
 indexes, 6-9
 /UPDATE, 6-11

RMU/ANALYZE command, 6-5
 /AREAS, 6-5
 /BINARY_OUTPUT, 6-6
 description, 6-5
 /END, 6-7
 gathering statistics, 6-5
 /LAREAS, 6-7
 /OPTION, 6-7
 /OUTPUT, 6-7
 /START, 6-7

RMU/ANALYZE/INDEXES command,
 6-13
 analyzing indexes, 6-13
 /BINARY_OUTPUT, 6-13
 description, 6-13
 gathering statistics, 6-13
 /OPTION, 6-14
 /OUTPUT, 6-13

RMU/ANALYZE/PLACEMENT
 command, 6-16
 /AREAS, 6-17
 /BINARY_OUTPUT, 6-17
 description, 6-16
 gathering statistics, 6-16
 /OPTION, 6-18
 /OUTPUT, 6-18

RMU/BACKUP/AFTER_JOURNAL
 command, 6-37
 /CONTINUOUS, 6-39
 example, 6-41
 /INTERVAL, 6-39
 /LOG, 6-40
 /THRESHOLD, 6-40
 /UNTIL, 6-40

RMU/BACKUP command, 6-20
 /ACTIVE_IO, 6-30
 backing up databases, 6-20

RMU/BACKUP command (Cont.)

- /BLOCK_SIZE, 6-29
- by area
 - purpose of, 6-24
- /CHECKSUM_VERIFICATION, 6-31
- /CRC=AUTODIN_II, 6-30
- /CRC=CHECKSUM, 6-30
- /DENSITY, 6-29
- examples, 6-33
- /EXCLUDE, 6-24
- format, 6-20
- /GROUP_SIZE, 6-30
- /INCLUDE, 6-26
- /INCREMENTAL, 6-23
- /LABEL, 6-29
- /LOCK_TIMEOUT=seconds, 6-32
- /LOG, 6-28
- /NOCRC, 6-31
- /ONLINE, 6-28
- /OWNER_ID=user-id, 6-31
- /PROTECTION=vms-file-protection, 6-32
- /REWIND, 6-29
- tape label checking, 6-34
- /TAPE_EXPIRATION=date-time, 6-32
- truncating file names, 6-33, 6-119

RMU/CLOSE command, 6-42

- /ABORT, 6-43
- /CLUSTER, 6-43
- eliminating active users, 6-42
 - /ABORT=DELPRC option, 6-43
 - /ABORT=FORCEX option, 6-43
 - /CLUSTER option, 6-43
 - /PATH option, 6-44
 - /WAIT option, 6-44
- example, 6-45
- format, 6-42
- /PATH, 6-44
- /WAIT, 6-44

RMU commands

- auditing the use of, 6-132
- new, xxii

RMU/CONVERT command, 6-46

RMU/CONVERT command (Cont.)

- called by RMU/RESTORE, 6-107
- /COMMIT, 6-47
- committing a database conversion
 - /COMMIT option, 6-47
- /CONFIRM, 6-47
- enabling user input during conversion
 - /CONFIRM option, 6-47
- function, 6-46
- /PATH, 6-47
- /ROLLBACK, 6-47
- rolling back a converted database
 - /ROLLBACK option, 6-47
- specifying a path name
 - /PATH option, 6-47

RMU/COPY_DATABASE command, 6-50

- /AFTER_JOURNAL, 6-51
- /BLOCKS_PER_PAGE, 6-52
- /CHECKSUM_VERIFICATION, 6-51
- /DIRECTORY, 6-51
- /FILE, 6-52
- /LOG, 6-51
- /NODES_MAX, 6-51
- /ONLINE, 6-52
- /OPTION, 6-52
- /PAGE_BUFFERS, 6-52
- /ROOT, 6-52
- /SNAPSHOTS, 6-52
- /THRESHOLDS, 6-53
- /USERS_MAX, 6-52

RMU/DUMP/AFTER_JOURNAL command, 6-59

- /DATA, 6-60
- /END, 6-60
- /OUTPUT, 6-60
- /START, 6-60
- /STATE=PREPARED, 6-60

RMU/DUMP/BACKUP_FILE command, 6-62

- /ACTIVE_IO, 6-63
- /BLOCK_SIZE, 6-63
- /LABEL, 6-65
- /OPTIONS, 6-63

RMU/DUMP/BACKUP_FILE command
 (Cont.)
 /OUTPUT, 6-63
 /PROCESS, 6-65
 /REWIND, 6-63
 /SKIP, 6-64
 RMU/DUMP command, 6-54
 /AREAS, 6-56
 /END, 6-57
 examples, 6-58
 /HEADER, 6-55
 /LAREAS, 6-57
 /OPTION, 6-55
 /OUTPUT, 6-55
 /SNAPSHOTS, 6-56
 /START, 6-57
 /STATE=BLOCKED, 6-57
 /USERS, 6-57
 RMU/DUMP/RECOVERY_JOURNAL
 command, 6-68
 /DATA, 6-69
 /OUTPUT, 6-68
 RMU/LOAD command, 6-70
 /AUDIT, 6-73
 /BUFFERS, 6-74
 /COMMIT_EVERY, 6-74
 examples, 6-77
 /FIELDS, 6-74
 /PLACE, 6-74
 /RMS_RECORD_DEF, 6-75
 /SKIP, 6-75
 /TRANSACTION_TYPE=share-mode,
 6-76
 /TRIGGER_RELATIONS, 6-76
 RMU/MONITOR REOPEN_LOG
 command, 6-79
 RMU/MONITOR START command,
 6-80
 /OUTPUT, 6-81
 /PRIORITY, 6-80
 /SWAP, 6-81
 RMU/MONITOR STOP command, 6-82
 /ABORT, 6-82
 examples, 6-84
 RMU/MONITOR STOP command (Cont.)
 using with DECtrace for VMS, 6-83
 /WAIT, 6-83
 RMU/MOVE_AREA command, 6-85
 /AFTER_JOURNAL, 6-86
 /AREA, 6-86
 /BLOCKS_PER_PAGE, 6-87
 /CHECKSUM_VERIFICATION, 6-86
 /DIRECTORY, 6-86
 /FILE, 6-87
 /LOG, 6-86
 /NODES_MAX, 6-86
 /OPTION, 6-87
 /PAGE_BUFFERS, 6-87
 /ROOT, 6-87
 /SNAPSHOTS, 6-87
 /THRESHOLDS, 6-87
 /USERS_MAX, 6-87
 RMU/OPEN command, 6-89
 example, 6-90
 /PATH, 6-90
 RMU/RECOVER command, 6-91, 6-93,
 6-96e
 /AIJ_BUFFERS, 6-92
 /AREAS, 6-92
 /LOG, 6-93
 reentering lost transactions, 6-91
 /RESOLVE, 6-95
 /ROOT, 6-94
 /TRACE, 6-93
 /UNTIL, 6-93
 RMU/RECOVER/RESOLVE command,
 6-100
 /CONFIRM, 6-101
 /STATE=option, 6-101
 RMU/RESOLVE command, 6-103
 /CONFIRM, 6-104
 /LOG, 6-104
 /PARENT_NODE, 6-104
 /PROCESS, 6-105
 /STATE, 6-105
 /TSN, 6-105
 RMU/RESTORE command, 6-107
 /ACTIVE_IO, 6-116

RMU/RESTORE command (Cont.)

- /AFTER_JOURNAL, 6-114**
- /AREA, 6-112**
- /BLOCKS_PER_PAGE, 6-117**
- by area
 - purpose of, 6-110, 6-112
- calling RMU/CONVERT, 6-107
- /CDD_INTEGRATE, 6-114**
- /CONFIRM, 6-113**
- /DIRECTORY, 6-115**
- examples, 6-121
- /FILE, 6-118**
- /INCREMENTAL, 6-111**
- /LABEL, 6-117**
- /LOG, 6-115**
- /NEW_VERSION, 6-115**
- /NODES_MAX, 6-115**
- /OPTIONS, 6-117**
- /PAGE_BUFFERS, 6-117**
- /PATH, 6-114**
- /REWIND, 6-116**
- /ROOT, 6-116**
- /SNAPSHOT, 6-118**
- /THRESHOLDS, 6-118**
- /USERS_MAX, 6-116**

RMU/SET AUDIT command, 6-127

- /DISABLE, 6-128**
- /ENABLE, 6-128**
- /EVERY, 6-132**
- /FIRST, 6-132**
- /FLUSH, 6-132**
- format, 6-127
- /START, 6-133**
- /STOP, 6-133**
- /TYPE=ALARM, 6-133**
- /TYPE=AUDIT, 6-133**

RMU/SHOW AUDIT command, 6-143

- /ALL, 6-143**
- /DACCESS, 6-144**
- /EVERY, 6-144**
- /FLUSH, 6-144**
- format, 6-143
- /IDENT, 6-144**
- /OUTPUT, 6-144**

RMU/SHOW AUDIT command (Cont.)

- /PROTECTION, 6-144**
- /RMU, 6-144**
- /TYPE=ALARM, 6-145**
- /TYPE=AUDIT, 6-145**

RMU/SHOW command

- See also* RMU/SHOW AUDIT command
- See also* RMU/SHOW STATISTICS command
- See also* RMU/SHOW SYSTEM command
- See also* RMU/SHOW USERS command
- See also* RMU/SHOW VERSION command

description of options, 6-142

RMU/SHOW STATISTICS command, 6-148

- /INPUT=file-name, 6-150**
- /INTERACTIVE, 6-151**
- /LOG, 6-151**
- /OUTPUT, 6-150**
- /TIME, 6-152**
- /UNTIL, 6-151**

RMU/SHOW SYSTEM command, 6-154

- /OUTPUT, 6-154**

RMU/SHOW USERS command, 6-155

- example, 6-156
- /OUTPUT, 6-155**

RMU/SHOW VERSION command, 6-157

- example, 6-157
- /OUTPUT, 6-157**

RMU/UNLOAD command, 6-158

- /BUFFERS, 6-159**
- examples, 6-160
- /FIELDS, 6-159**
- /RMS_RECORD_DEF, 6-160**

RMU/VERIFY command, 6-161

- /ALL, 6-164**
- /AREAS, 6-165**
- /CHECKSUM_ONLY, 6-162**
- /CONSTRAINTS, 6-165**

RMU/VERIFY command (Cont.)

- /DATA, 6-164
 - /END, 6-166
 - examples, 6-167
 - format, 6-161
 - /INCREMENTAL, 6-162
 - /INDEXES, 6-164
 - /LAREAS, 6-165
 - /LOG, 6-164
 - /OUTPUT, 6-165
 - /ROOT, 6-163
 - /SNAPSHOTS, 6-165
 - /START, 6-166
 - /TRANSACTION_TYPE, 6-166
- Role-oriented privileges, 9-142
- ROLLBACK command (RdbALTER), 7-38
- ROLLBACK statement, 9-359
 - undoing changes to a database, 9-359
- Root file
 - errors opening, 6-33
 - moving, 6-85
- RSE
 - conditional-expr (Data Distributor), 9-191
 - context-variable (Data Distributor), 9-191
 - context variables, 4-6
 - CROSS clause, 4-12
 - database handles in (Data Distributor), 9-191
 - defined, 4-1
 - FIRST clause, 4-3
 - first-clause (Data Distributor), 9-191
 - in views, 4-15
 - move-relations-clause (Data Distributor), 9-190
 - reduce-clause (Data Distributor), 9-191
 - REDUCED TO clause, 4-11
 - relation-clause (Data Distributor), 9-191
 - sort-clause (Data Distributor), 9-191

RSE (Cont.)

- SORTED BY clause, 4-9
 - syntax, 4-2
 - value-expr (Data Distributor), 9-192
 - WITH clause, 4-8
 - with-clause (Data Distributor), 9-191
- RTO license
 - availability of RDO statements, 9-1, C-1
- RUJ
 - See* Recovery-unit journal (RUJ)
- Running command files
 - Execute statement (@), 9-267
- Run-time only (RTO) license
 - See* RTO license

S

Schedules

- defining, 9-164
- defining transfer start time, 9-165
- deleting, 9-247
- RETRY clause, 9-168
- retry count, 9-167

Scope of dbkeys

- extending, 9-331

Security

- See* Protection

Security alarms, 6-127

- disabling, 6-128
- enabling, 6-128

Security auditing

- disabling, 6-128
- displaying characteristics, 6-143
- enabling, 6-127, 6-128
- starting, 6-133
- stopping, 6-133

Security audit journal, 6-127

- loading into database, 6-70

Security audit journal records

- defining database table for storing, 6-71
- defining relation for storing, 6-71

SECURITY privilege, 9-142, 9-143t

SECURITY privilege (Cont.)
 using to override database privileges, 9-142

SEGMENTED STRING data type, 5-8

Segmented strings
 creating, 9-74
 defined, 3-4
 displaying information about, 9-3
 END_SEGMENTED_STRING statement, 9-260
 expressions, 3-34
 handle, 9-74
 multiple storage area restrictions, 9-176
 retrieving, 9-417
 retrieving values
 FOR statement, 9-287
 storing
 CREATE_SEGMENTED_STRING statement, 9-74
 STORE statement, 9-464
 storing in multiple storage areas, xxii, 9-173
 example, 9-178

SET DATE_FORMAT statement, 9-363

SET LANGUAGE statement, 9-365

SET statement, 9-362
 changing RDO parameters, 9-362
 DATE_FORMAT clause, 9-363
 logical names used in, 9-363, 9-366
 setting display formats for date and time values, 9-364
 setting language for date and time input and displays, 9-365
 setting radix point character in displays, 9-365
 setting RDO parameters, 9-362

Setting scope of database keys, 9-331

Setting scope of request handle, 9-332

Shareable definitions
 fields, 9-112
 relations, 9-149

Share modes, 9-443t

SHOW ALL statement, 9-375

SHOW ALL statement (Cont.)
 displaying all database information, 9-375

SHOW COLLATING_SEQUENCE statement, 9-376

SHOW CONSTRAINTS statement, 9-377

SHOW DATABASES statement, 9-379

SHOW DATE_FORMAT statement, 9-381

SHOW DICTIONARY statement, 9-383

SHOW FIELDS statement, 9-384

SHOW INDEXES statement, 9-387

SHOW LANGUAGE statement, 9-391

SHOW MONITOR statement
See also RMU/SHOW SYSTEM command
 displaying monitor information, F-16

SHOW PRIVILEGES statement, 9-392
 displaying access rights, 9-392

SHOW PROTECTION statement, 9-395
 displaying all ACL entries, 9-395

SHOW RADIX_POINT statement, 9-397

SHOW RELATIONS statement, 9-398

SHOW statements
 displaying information about a database, 9-372t

SHOW STORAGE AREAS statement, 9-400

SHOW STORAGE MAPS statement, 9-404
 displaying storage map information, 9-404

SHOW STREAMS statement, 9-406
 displaying names of streams, 9-406

SHOW TRANSACTION statement, 9-407

SHOW TRANSFER statement, 9-409
 ALL option, 9-410
 DEFINITION option, 9-409
 displaying transfer information, 9-409

SHOW TRANSFER statement (Cont.)
 displaying transfer state, 9-410
 distributing databases, 9-409
 SCHEDULE option, 9-410
 STATUS option, 9-410
 SHOW TRIGGERS statement, 9-413
 SHOW USERS statement
 displaying information about users,
 F-17
 SHOW VERSIONS statement, 9-416
 SIGNED BYTE data type, 5-7t
 SIGNED LONGWORD data type, 5-7t
 SIGNED QUADWORD data type, 5-7t
 SIGNED WORD data type, 5-7t
 Single-file database
 recovery of, 6-95
 Snapshot file
 adjusting size, 9-15
 defining size, 9-189
 disabling, 9-188
 enabling deferred, 9-188
 enabling immediate, 9-188
 extension, 9-189
 Snapshot pages
 checksum verification of, 6-165
 sort-clause
 ASCENDING, 9-192
 DESCENDING, 9-192
 RSE, 9-191
 SORTED BY clause (of RSE), 4-9
 Sorted index, 9-40, 9-122
 Sort key, 4-9
 Source databases
 inconsistency with target database,
 9-185
 logical symbol names for, 9-198
 Source nodes
 failure of, 9-186
 Space area management pages (SPAMs),
 9-18
 defining number of data pages
 between, 9-18
 output from RMU/VERIFY, 6-161
 Special characters, 3-6
 Specifying access rights, 9-132
 Specifying files, 1-8
 SPOOL statement
 See also RMU/BACKUP/AFTER_
 JOURNAL command
 backing up AIJ file, F-18
 Starting
 monitor, 6-80
 RDO, 1-1
 record streams, declared, 9-422
 record streams, undeclared, 9-425
 Starting security auditing, 6-133
 STARTING WITH clause, 3-39t
 START TRANSFER statement, 9-453
 distributing databases, 9-453
 NOWAIT option, 9-453
 NOW option, 9-453, 9-454
 transfer state, 9-453
 WAIT option, 9-453
 START_SEGMENTED_STRING
 statement, 9-417
 retrieving segmented strings, 9-417
 START_STREAM statement, 9-422,
 9-425
 declared record stream, 9-422
 displaying name of current stream
 SHOW STREAM statement,
 9-406
 examples, 9-423, 9-428
 restrictions for RSE and handles,
 9-422
 transaction handle restriction, 9-426
 undeclared record stream, 9-425
 START_TRANSACTION statement,
 9-431
 accessing records, 9-431, 9-452
 arguments, 9-435
 defaults, 9-432, 9-433t
 distributed transactions, 9-431
 DISTRIBUTED_TRANSACTION
 clause, 9-436
 format, 9-431
 RESERVING clause, 9-442
 share modes, 9-443t

- START_TRANSACTION statement
 - (Cont.)
 - WITH AUTO_LOCKING clause, 9-444
 - with two-phase commit syntax, 9-431
- Statements
 - database maintenance, 2-7
 - data definition, 2-2
 - data manipulation, 2-5
 - distributing databases, 2-9
 - interactive control, 2-8
 - no longer supported in RDO
 - See* Obsolete statements
 - VAX Data Distributor, 2-9
- Statements in RDO
 - components of, 1-3
- Statement terminator
 - entering a complete statement, 1-9
- Statistical expressions, 3-11
- Statistics
 - database, 6-148
 - gathering
 - ANALYZE statement, 9-2
 - RMU/ANALYZE/CARDINALITY command, 6-9
 - RMU/ANALYZE command, 6-5
 - RMU/ANALYZE/INDEXES command, 6-13
 - RMU/ANALYZE/PLACEMENT command, 6-16
 - process, 6-148
- STOP MONITOR statement
 - See also* RMU/STOP MONITOR command, F-20
 - stopping the database monitor, F-20
- Stopping
 - monitor, 6-82
 - transfers, 9-456
- Stopping security auditing, 6-133
- STOP TRANSFER statement, 9-456
 - distributing databases, 9-456
 - stopping transfers, 9-456
- Storage area options
 - defining, 9-310
- Storage area pages
 - allocation of, 9-15, 9-103
- Storage area parameters
 - default values, E-1
 - maximum values, E-1
 - minimum values, E-1
- Storage areas
 - changing to read-only access, 9-19
 - changing to read/write access, 9-19
 - default (RDB\$SYSTEM), 9-21, 9-62, 9-98, 9-105, 9-107, 9-173
 - default parameters, 9-96
 - defining, 9-17, 9-20, 9-93, 9-98
 - deleting, 9-20
 - for segmented strings, 9-173
 - moving, 6-85
 - options, 9-17, 9-20, 9-98
 - partitioning data, 9-62, 9-171
 - SHOW STORAGE AREAS statement, 9-400
 - specifying for segmented strings, xxii
 - specifying with RdbALTER utility, 7-3
 - storing relations, 9-176
- Storage maps
 - changing, 9-61
 - defining, 9-171
 - deleting, 9-249
 - SHOW STORAGE MAPS statement, 9-404
- STORE statement, 9-457
 - adding database records, 9-457
 - retrieving dbkey value, 9-457
 - specifying through
 - DEFINE TRIGGER statement, 9-207
 - storing segmented strings, 9-464
 - transaction handle restriction, 9-458
- Storing records, 9-62, 9-173
 - by means of indexes, 9-63, 9-174
- Streams
 - closing, 9-261

- Streams (Cont.)
 - declaring, 9-82
 - displaying name of current, 9-406
 - starting, 9-422, 9-425
- String literals
 - translation of, 9-391
- Substitution variable
 - See* Host language variables
- Summation update
 - specifying through
 - DEFINE TRIGGER statement, 9-204, 9-217e
- Syntax diagrams
 - reading, xvii
- System administration
 - summary of maintenance statements, 2-7t
- System-defined identifiers, 9-134
- System relations, 8-1
 - changing to read-only, 9-15
 - changing to read/write, 9-15
 - detailed, 8-1
 - list of, 8-3
 - RDB\$CONSTRAINTS, 8-4
 - RDB\$CONSTRAINT_RELATIONS, 8-5
 - RDB\$DATABASE, 8-6
 - RDB\$FIELDS, 8-11
 - RDB\$FIELD_VERSIONS, 8-9
 - RDB\$INDEX_SEGMENTS, 8-15
 - RDB\$INDICES, 8-16
 - RDB\$RELATIONS, 8-20
 - RDB\$RELATION_FIELDS, 8-17
 - RDB\$VIEW_RELATIONS, 8-23
 - RDBVMS\$COLLATIONS, 8-24
 - RDBVMS\$INTERRELATIONS, 8-25
 - RDBVMS\$PRIVILEGES, 8-26
 - RDBVMS\$RELATION_
 - CONSTRAINTS, 8-27
 - RDBVMS\$RELATION_
 - CONSTRAINT_FLDS, 8-31
 - RDBVMS\$STORAGE_MAPS, 8-31

- System relations (Cont.)
 - RDBVMS\$STORAGE_MAP_AREAS, 8-32
 - RDBVMS\$TRIGGERS, 8-33

T

- Tables
 - loading, 6-70
 - unloading from the database, 6-158
- Tapes
 - checking labels, 6-34, 6-66, 6-120
- Target databases
 - inconsistency with source database, 9-185
 - logical symbol names for, 9-198
- Target page number
 - determining before storing records, 9-346
- Terminal
 - directing output to, 9-365
 - displaying command procedure output on, 9-366
- Terminal screen
 - directing output to, 9-365
 - displaying command procedure output on, 9-366
- Terminating a statement, 1-9
- TEXT data type, 5-7t
- Time, in dates, 3-10
- Time formats
 - displaying, 9-381
 - SET DATE_FORMAT statement, 9-363
 - setting, 9-364
 - SHOW DATE_FORMAT statement, 9-381
- Time specifications
 - absolute time, 9-166
 - delta time, 9-166
 - displaying, 9-381
 - setting, 9-363, 9-364
- TODAY string literal
 - translation of, 3-8, 9-365, 9-391

TOMORROW string literal
translation of, 3-8, 9-365, 9-391

TOTAL statistical function, 3-11t, 3-13

Transaction
distributed
See Distributed transaction

Transaction handle, 3-5, 9-83, 9-283, 9-426, 9-458
restriction for START_STREAM statement, 9-422, 9-426

Transactions
See also Distributed transaction
See also Unresolved transactions
blocked
See Unresolved transactions
COMMIT statement, 9-69
displaying information about SHOW TRANSACTION statement, 9-407
distributed, 9-437
ROLLBACK statement, 9-359
START_TRANSACTION statement, 9-431
unresolved
See Unresolved transactions
WITH AUTO_LOCKING clause, 9-444
with multiple database handles, 9-437

Transfer-file-options-clause
DEFINE TRANSFER statement, 9-194
epilogue-file-spec, 9-194
prologue-file-spec, 9-194

Transferring data from Rdb/ELN, 9-299

Transferring views, 9-196

Transfers
defining, 9-180, 9-184
defining extraction, 9-185
defining extraction rollup, 9-185
defining replications, 9-185
deleting, 9-251
displaying, 9-409
starting, 9-453

Transfers (Cont.)
states, 9-410
stopping, 9-456

Transfer states
active, 9-195
affected by START_TRANSFER, 9-453
scheduled, 9-195
unscheduled, 9-195

Transfer time, 9-197

Triggers
defining, 9-204
definition of, 9-204
deleting, 9-253
displaying, 9-413
specifying cascading deletes with, 9-204
specifying cascading updates with, 9-204
specifying summation updates with, 9-204
using for referential integrity, 9-204

Truncation
of file name during RMU/BACKUP command, 6-33, 6-119

Two-phase commit syntax
START_TRANSACTION statement, 9-431

U

UIC
accessing databases, 9-44
identifiers, 9-47, 9-134

Unary minus, 3-4

UNCORRUPT command (RdbALTER), 7-39

Undoing changes to a database
ROLLBACK statement, 9-359

UNIQUE clause, 3-39t

UNIQUE constraints
naming in
CHANGE_RELATION statement, 9-52

UNIQUE constraints
naming in (Cont.)
 DEFINE RELATION statement,
 9-157

Unloading
relations, 6-158
tables, 6-158
views, 6-158

Unresolved transactions
aborting in the AIJ file, 6-100
committing in the AIJ file, 6-100
displaying a list of, 6-57
resolving, xxiii, 6-103
resolving for a database, 6-100

Unsupported statements
See Obsolete statements

Update
cascading
 with DEFINE TRIGGER
 statement, 9-204, 9-213e

Update access modes
effect on database functions, 9-443t

Upgrading software version
RMU/CONVERT command, 6-46

USAGE clause
DEFINE INDEX statement, 9-125

USAGE QUERY clause, 9-38, 9-125
displaying value, 9-387

USAGE UPDATE clause, 9-38, 9-125
displaying value, 9-387

User identification code
See UIC

Users
displaying information about
 RMU/SHOW USERS command,
 6-155
 SHOW USERS statement, F-17
number allowed, 9-188

User-supplied names, 1-5

V

VALID IF clause, 5-11

Validity
field attribute, 5-11

Value-expr (Data Distributor)
RSE, 9-192
sort-clause, 9-192

Value expressions, 3-1, 9-192
arithmetic expressions, 3-3, 3-21
concatenated expressions, 3-4, 3-25
context variable, 3-3
database key, 3-4, 3-32
db field, 3-2
definition, 3-1
field name, 3-3
FIRST FROM expression, 3-3, 3-31
host language variable, 3-2
literals, 3-6
missing value, 3-3
numeric string, 3-3
quoted string, 3-3
RDB\$LENGTH, 3-34
RDB\$MISSING, 3-26
RDB\$VALUE, 3-34
segmented string, 3-4, 3-34
statistical expression, 3-3
unary minus, 3-4

Variables

host language, 3-4
VARYING STRING data type, 5-8t

VAX CDD/Plus
See Data dictionary

VAX Data Distributor
DEFINE SCHEDULE statement,
 9-164

DEFINE TRANSFER statement,
 9-180

DELETE SCHEDULE statement,
 9-247

DELETE TRANSFER statement,
 9-251

error messages explanation files, B-1
REINITIALIZE TRANSFER

statement, 9-357

SHOW TRANSFER statement, 9-409

- VAX Data Distributor (Cont.)
 - START TRANSFER statement, 9-453
 - STOP TRANSFER statement, 9-456
 - summary of statements, 2-2t, 2-9
- VAXTPU editor
 - using EDIT statement, 9-257
- Verb statistics, 6-148
- VERIFY command (RdbALTER), 7-41
- Verifying
 - database integrity, 6-161
- Version number
 - displaying information about
 - RMU/SHOW VERSION command, 6-157
 - SHOW VERSIONS statement, 9-416
- View-name
 - move-views-clause, 9-193
- Views
 - default protection for
 - modifying, 9-144
 - definitions
 - DEFINE VIEW statement, 9-219
 - DELETE VIEW statement, 9-255
 - record selection expressions, 4-15
 - specifying access rights, 9-133
 - specifying default protection for, 9-144
 - transferring, 9-196
 - unloading from the database, 6-158
- VMS BYPASS privilege, 9-143t
 - using to override database privileges, 9-142, 9-143
- VMS DEFAULT identifier
 - DEFINE PROTECTION command, 9-144
- VMS National Character Set
 - See* NCS utility
- VMS OPER privilege
 - using to override database privileges, 9-142
- VMS privileges
 - BYPASS
 - using to override database privileges, 9-142, 9-143
 - OPER
 - using to override database privileges, 9-142
 - READALL
 - using to override database privileges, 9-143
 - SECURITY
 - using to override database privileges, 9-142
 - SYSPRV
 - using to override database privileges, 9-142, 9-143
- VMS READALL privilege, 9-143t
 - using to override database privileges, 9-143
- VMS security audit journal
 - loading into database, 6-70
- VMS SECURITY privilege
 - using to override database privileges, 9-142
- VMS SYSPRV privilege, 9-143t
 - using to override database privileges, 9-142, 9-143

W

- WITH AUTO_LOCKING clause
 - defined, 9-444
 - example, 9-448
 - START_TRANSACTION statement, 9-444
- With-clause (Data Distributor)
 - conditional-expr, 9-191
 - RSE, 9-191
- WITH clause (RSE), 4-8
- Writing changes to a database
 - COMMIT statement, 9-69

Y

YESTERDAY string literal
translation of, 3–8, 9–365, 9–391