Replication Option for Rdb

Handbook

Release 7.0

Part No. A49000-1

ORACLE®

Replication Option for Rdb Handbook

Release 7.0

Part No. A49000-1

Copyright © 1993, 1996, Oracle Corporation. **All rights reserved.**

This software contains proprietary information of Oracle Corporation; it is provided under a license agreement containing restrictions on use and disclosure and is also protected by copyright law. Reverse engineering of the software is prohibited.

The information contained in this document is subject to change without notice. If you find any problems in the documentation, please report them to us in writing. Oracle Corporation does not warrant that this document is error free.

Restricted Rights Legend Programs delivered subject to the DOD FAR Supplement are 'commercial computer software' and use, duplication and disclosure of the programs shall be subject to the licensing restrictions set forth in the applicable Oracle license agreement. Otherwise, programs delivered subject to the Federal Acquisition Regulations are 'restricted computer software' and use, duplication and disclosure of the programs shall be subject to the restrictions in FAR 52.227-14, Rights in Data—General, including Alternate III (June 1987). Oracle Corporation, 500 Oracle Parkway, Redwood City, CA 94065.

The programs are not intended for use in any nuclear, aviation, mass transit, medical, or other inherently dangerous applications. It shall be the licensee's responsibility to take all appropriate fail-safe, back up, redundancy and other measures to ensure the safe use of such applications if the programs are used for such purposes, and Oracle disclaims liability for any damages caused by such use of the programs.

Oracle and SQL*Plus are registered trademarks of Oracle Corporation, Redwood City, California. Hot Standby, Oracle CODASYL DBMS, Oracle Rdb, Oracle RMU, and Rdb7 are trademarks of Oracle Corporation, Redwood City, California.

All other company or product names are used for identification purposes only and may be trademarks of their respective owners.

Contents

Send Us Your Comments xix

Preface xxi

Technical Changes and New Features xxix

1 Introduction

1.1 Replication Option Major Capabilities 1–1
1.2 Data Management and Data Access Products 1–3
1.3 User and Application Interfaces to Replication Option 1–6
1.4 Using Centralized Data Versus Replicated Data 1–7
1.4.1 Central Data Management 1–7
1.4.2 Replicated Data Management 1–9

2 Overview

2.1 Replication Option Concepts 2–1
2.1.1 Transfers 2–1
2.1.2 Local and Remote Databases 2–2
2.1.3 Data Distribution Scenarios 2–2
2.1.3.1 Distributing Data from One Source to Many Targets 2–3
2.1.3.2 Distributing Data from Many Sources to One Target 2–3
2.1.3.3 Exchanging Data Among Peer Databases 2–4
2.1.3.4 Archiving Data 2–4
2.1.4 Selecting Subsets of Data 2–5
2.1.5 Methods of Transfer 2–9
2.1.5.1 Extraction and Extraction Rollup Transfers 2–9
2.1.5.2 Replication Transfers 2–14
2.2 Choosing a Method of Transfer 2–18
2.2.1 Classifying Your Database Tasks 2–18

2.2.2 Transfer Choices and Considerations 2–20
2.2.2.1 Types of Source Database Used 2–21
2.2.2.2 Location of Replication Option 2–22
2.2.2.3 Data Currency and Volume of Data Transferred 2–22
2.2.2.4 Target Disk Storage Usage 2–22
2.2.2.5 Source Disk Storage Usage 2–24
2.2.2.6 Source Database Transaction Rate 2–24
2.2.2.7 Source Database Availability 2–24

3 Components of Replication Option

3.1 Overview 3–1
3.2 Using the SQL Interfaces 3–3
3.2.1 Using Oracle SQL/Services to Query a Target Database 3–4
3.2.2 Using SQL Applications to Issue Replication Option Statements 3–4
3.3 How Rdb or Distributed Option Software Options Affect Use of Replication Option 3–5
3.4 Source Database 3–6
3.5 Target Database 3–6
3.6 Transfer Monitor 3–7
3.7 Transfer Database 3–8
3.8 Transfer and Schedule Definitions 3–9
3.9 Copy Process 3–10
3.10 Special Replication Option System Tables for Replications 3–11

4 Preparations for Distributing Data

4.1 Determining Product Compatibility 4–2
4.2 Using Replication Option in a Multiversion Rdb Environment 4–3
4.3 Requirements for Locating Replication Option at Various Sites 4–4
4.4 Moving Applications to Target Sites 4–5
4.4.1 Using a Data Dictionary with Replication Option 4–5
4.4.2 Using Transfer Definitions to Provide Remote Copies of Database Definitions 4–6
4.5 Preparations at the Transfer Definition Site 4–7
4.6 Preparations at the Source Sites 4–7
4.7 Preparations at the Target Site 4–9
4.8 Security and Environment Considerations for Remote Access 4–10
4.8.1 Remote Server Security and Environment Considerations 4–11
4.8.2 Remote Access Using an Explicit User Account and Password 4–12
4.8.3 Remote Access Using a DECnet Proxy Account 4–15
4.8.3.1 Setting Up DECnet Proxy Accounts on Remote OpenVMS Systems 4–16

4.8.4 Remote Access Using an Rdb Configuration File 4–18
4.8.5 Remote Access Using an Account in a Replication Option File 4–19
4.8.6 Remote Access Using Implicit Authorization 4–21
4.9 Preparing Database Access Using the RMU Open Command 4–22
4.10 DECnet Network Tuning Parameters 4–23
4.11 Performing Database Maintenance, Backup, Recovery, and Conversion 4–24
4.11.1 Maintenance, Backup, Recovery, and Conversion of Source Databases 4–25
4.11.1.1 Backing Up and Recovering a Source Database 4–25
4.11.1.2 Converting Source Databases 4–26
4.11.2 Maintenance, Backup, Recovery, and Conversion of Target Databases 4–26
4.11.2.1 Backing Up and Recovering a Target Database 4–27
4.11.2.2 Converting Target Databases 4–28
4.11.3 Maintenance, Backup, Recovery, and Conversion of Transfer Databases 4–28
4.11.3.1 Backing Up and Recovering a Transfer Database 4–28
4.11.3.2 Converting Transfer Databases 4–29
4.11.4 Changing Data Definitions and Restructuring Databases 4–29
4.11.4.1 Changing Definitions in and Restructuring Source Databases . . . 4–29
4.11.4.2 Changing Definitions in and Restructuring Target Databases . . . 4–32

5 Creating Database Transfers

5.1 Extraction and Extraction Rollup Transfers 5–2
5.1.1 Extraction and Extraction Rollup Transfer Definition Elements 5–3
5.1.1.1 Identifying Source Databases 5–4
5.1.1.2 Target Nodes and Directories 5–5
5.1.1.3 Target Database Parameters 5–6
5.1.1.4 Using the Wildcard Character 5–7
5.1.1.5 Database Aliases 5–8
5.1.1.6 Location of Copy Process Log Files 5–8
5.1.1.7 Using Command Procedures 5–8
5.1.2 Selecting Subsets of Rows to Transfer 5–8
5.1.3 Forming Select Expressions 5–10
5.1.4 Transferring Views 5–10
5.1.5 Creating and Starting an Extraction or an Extraction Rollup Transfer 5–11
5.2 Replication Transfers 5–13

5.2.1 Replication Transfer Definition Elements 5–13
5.2.1.1 Identifying the Source Database 5–14
5.2.1.2 Target Nodes and Directories 5–15
5.2.1.3 Target Database Parameters 5–16
5.2.1.4 Wildcard Character 5–17
5.2.1.5 Database Aliases 5–17
5.2.1.6 Location of Copy Process Log Files 5–17
5.2.1.7 Using Command Procedures 5–17
5.2.2 Identifying Rows 5–17
5.2.2.1 Effect of Inserting, Updating, and Deleting Rows 5–19
5.2.3 Forming Select Expressions 5–19
5.2.4 Transferring Views 5–20
5.2.5 Creating and Starting a Replication Transfer 5–21
5.3 Character Set Considerations with Database Transfers 5–23
5.3.1 Terminal Device Characteristics Restrict Use of Character Sets 5–23
5.3.2 Effects of Transferring Database Objects with Different Character Sets 5–23
5.3.3 Using Different Versions of Rdb for Source and Target Databases . . . 5–23
5.4 Using Multischema Databases as Source Databases 5–24
5.5 Transfers to Distributed Option for Rdb and Rdb Transparent Gateway Databases 5–25
5.5.1 Manually Create the DDAL$TRANSFER_INFO Table 5–26
5.5.2 Manually Create the DDAL$TRANSFER_IDX1 Index 5–27
5.5.3 Add Access Privileges to Target Tables 5–27
5.5.4 Create Transfer Definitions 5–27
5.5.4.1 Limit to Number of Tables Per Transfer 5–28
5.5.4.2 Data Type Limitations 5–28
5.5.5 Create the Configuration File 5–29
5.5.6 Create a DDAL.RC File 5–29
5.5.7 Execute the Transfer 5–29
5.5.8 Modify the Configuration File 5–30
5.5.9 Manually Create Indexes on Target Tables 5–31
5.5.10 Manually Drop Indexes After Transfer Reinitialization 5–33
5.5.11 Multiple Transfers and Uniqueness of Object Names 5–33
5.5.12 Source Definition Changes 5–34

6 Working with Transfers

6.1 Creating a Schedule for Transfers 6–1
6.2 Transfer States 6–4
6.3 Executing the Transfer 6–5
6.4 Modifying Your LOGIN.COM File for the Copy Process 6–8
6.5 Stopping Transfers 6–9
6.6 Changing and Dropping Transfers 6–11
6.6.1 Changing Transfer Definitions Using the SQL ALTER TRANSFER Statement 6–11
6.6.2 Dropping Transfer Definitions 6–12
6.7 Changing and Dropping Schedules 6–15
6.7.1 Changing Schedules 6–16
6.7.2 Dropping Schedules 6–16
6.8 Changing Data Definitions After a Transfer Is Defined 6–17
6.9 Displaying Transfer Information 6–18
6.9.1 Showing a Transfer Definition 6–18
6.9.2 Examining Target Database Information 6–19
6.9.3 Showing a Transfer Schedule 6–20
6.9.4 Showing Transfer Status 6–21
6.9.5 Showing All Transfer Information 6–21
6.10 Diagnostic Tools 6–22
6.10.1 Using Copy Process Log Files 6–22
6.10.2 Copy Process Option for Logging Individual Row Keys 6–28
6.10.3 Using Transfer Monitor Log Files 6–29
6.10.4 Using Event Logs 6–37
6.11 Using Checkpoints for Quicker Recovery from Some Transfer Failures 6–37
6.12 User-Added Data Definitions in a Target Database 6–39
6.12.1 Effects of Transfer Re-Execution on User-Added Definitions 6–40
6.12.1.1 Extraction and Extraction Rollup Transfers 6–40
6.12.1.2 Replication Transfers 6–41
6.12.2 Adding Indexes to Target Tables 6–41
6.12.3 Adding Constraints for Target Tables 6–42
6.12.4 Adding Triggers for Target Tables 6–42
6.12.5 Adding Multischema Attributes to Target Tables 6–43
6.12.6 Adding Multifile Storage Areas to a Target Database 6–43
6.12.7 Modifying Protection for Target Databases and Tables 6–43
6.12.8 Adding Tables to Target Databases 6–44
6.12.9 Adding Views to Target Databases 6–44
6.13 User-Added Data in a Target Database 6–44
6.14 Special Data Definitions Supporting Replications 6–45

6.14.1 Storing Updates to the Source Database for Replication Transfers 6–45
6.14.2 Tracking Data Between Target and Source Databases 6–46
6.15 Application Requirements for Transfers 6–47
6.15.1 Storing Database Definitions in the Data Dictionary 6–47
6.15.2 Identifying the Location of the Database 6–49
6.15.3 Determining the State of a Target Database 6–50
6.15.4 Modifying Row Descriptions in Application Programs 6–50
6.15.5 Modifying Transaction Share Modes and Lock Types for Extraction or Extraction Rollup Databases 6–51
6.15.6 Modifying Transaction Share Modes and Lock Types for Replication Databases 6–52
6.15.7 Evaluating Application Schedules for Extraction or Extraction Rollup Databases 6–52
6.15.8 Evaluating Application Schedules for Replication Databases 6–53
6.16 Performance Options for Replication and Extraction Transfers 6–53
6.16.1 Specifying How Many Data Rows Are Read and Written at a Time 6–54
6.16.2 Combining the Deleting and Writing of Data into One Transaction 6–54
6.16.3 Transferring Data Starting with Particular Tables 6–55
6.16.4 Limiting Concurrent Transfers on a CPU 6–55
6.17 Performance Options for Replication Transfers 6–57
6.17.1 Specifying When RDB$CHANGES Is Purged 6–57
6.17.2 Executing Concurrent Transfers from the Same Source Database . . . 6–58
6.17.3 Protecting Against Concurrent Writers During a Transfer 6–60
6.17.4 Adding Indexes to the RDB$CHANGES Table 6–60
6.17.4.1 Performance Considerations 6–61
6.17.4.2 Optional Indexing 6–62
6.17.4.2.1 Using the DDAL$CHANGES_INDEX_transfername Environment Variable 6–62
6.17.4.2.2 Using the DDAL$CHANGES_INDEX Environment Variable 6–63
6.17.4.3 Creating the Indexes 6–63
6.17.4.3.1 Index Creation for Replication Initialization and Reinitialization 6–63
6.17.4.3.2 Index Creation for Replication Updates 6–64
6.17.4.4 Dropping the Indexes 6–65
6.17.4.5 Creating Indexes for Replication Transfers 6–65
6.17.5 Avoiding Concurrent Transfers into PC Databases 6–66
6.18 Maximum Size of Source Database Table 6–66

7 Using Command Procedures with Transfers

7.1 Defining and Executing Command Procedures 7–1
7.1.1 Executing Command Procedures 7–3
7.1.1.1 Executing Remotely or Locally Stored Command Procedures on a Local Node 7–3
7.1.1.2 Executing Remotely Stored Command Procedures on a Remote Node 7–4
7.1.2 Completing the Transfer with DDAL$EPILOGUE.COM 7–4
7.1.3 Executing Multiple Command Procedures 7–6
7.1.4 Passing Parameters to Command Procedures 7–7
7.1.5 Using DDAL$TRANSFER_NAME to Identify a Transfer 7–8
7.1.6 Executing a Command Procedure on a Remote Node 7–8
7.1.7 Finding Command Procedure Information in the Copy Process Log File 7–9
7.2 Determining Transfer Status 7–11
7.2.1 Using DDAL$CP_CONTINUE to Affect Transfer Execution 7–11
7.2.1.1 Determining Prologue Success Status with Prologue Command Procedures 7–12
7.2.1.2 Determining Transfer Success Status with Epilogue Command Procedures 7–14
7.2.1.3 Determining Epilogue Success Status with Epilogue Command Procedures 7–15
7.2.2 Using the SHOW TRANSFER STATUS Statement 7–15
7.2.3 Using the Copy Process Log for Status Information 7–16
7.3 Typical Uses of Command Procedures 7–17
7.3.1 Creating Target Indexes and Other Database Attributes 7–17
7.3.2 Granting Target Database Access Rights 7–18
7.3.3 Defining Transfer and Schedule Definition Protection Using Access Control Lists 7–21
7.3.4 Bringing Up the Network at Transfer Time 7–21
7.3.5 Storing Data Definitions in the Data Dictionary at a Target Site . . . 7–22
7.3.5.1 Using the SUBMIT/REMOTE Command to Run a Remote Command Procedure 7–23
7.3.5.2 Using the TYPE Command to Run a Remote Command Procedure 7–25

8 Transferring Data to Existing Databases

8.1 Overview 8–1
8.1.1 Merging Reference Data with Existing Data 8–2
8.1.2 Rolling Up Multiple Sources to One Target Database 8–2
8.1.3 Sharing Data Among Multiple Databases 8–5
8.1.4 Archive Databases 8–6
8.2 Using Transfers to Existing Databases 8–7
8.2.1 Extracting Reference Tables 8–7
8.2.1.1 Using an Epilogue Procedure to Add Indexes, Constraints, and New Views 8–7
8.2.1.2 Updating the Target Data Dictionary 8–8
8.2.1.3 Using Prologue Procedures for Transfer Re-Execution 8–8
8.2.1.4 Changing Table Definitions 8–8
8.2.1.5 Changing Domain Definitions 8–9
8.2.1.6 Changing View Definitions 8–9
8.2.1.7 Dropping Indexes and Constraints 8–10
8.2.1.8 Database Access Conflicts 8–11
8.2.2 Replicating Reference Tables 8–11
8.2.3 Archiving Production Data 8–11
8.2.4 Rollup Transfers 8–15
8.2.5 Rules for Multiple Writers to the Same Tables 8–16
8.2.5.1 Transferring Data from Multiple Source Databases into a Common Target Table 8–23
8.2.5.2 Transferring Data from Multiple Databases to Each Other 8–26
8.2.6 Rollup of RMS Files into a Single Database 8–29
8.2.7 Writing to Target Databases 8–30
8.2.7.1 New Databases 8–30
8.2.7.2 Existing Databases 8–31
8.2.8 Transfer Failure and Storage Usage 8–31
8.2.9 Database Consistency 8–31
8.3 Restrictions on Using Transfers to Existing Databases 8–32

9 SQL Statements for Replication Option

9.1 ALTER SCHEDULE Statement 9–3
9.2 ALTER TRANSFER Statement 9–10
9.3 CREATE SCHEDULE Statement 9–19
9.4 CREATE TRANSFER Statement 9–28
9.5 DROP SCHEDULE Statement 9–72
9.6 DROP TRANSFER Statement 9–74
9.7 GRANT Statement 9–76
9.8 HELP Statement 9–85

9.9 REINITIALIZE TRANSFER Statement 9–87
9.10 REVOKE Statement 9–89
9.11 SET LOGICAL_NAME TRANSLATION FOR TRANSFER Statement . . . 9–96
9.12 SHOW TRANSFER Statement 9–99
9.13 START TRANSFER Statement 9–104
9.14 STOP TRANSFER Statement 9–108

A Transfer Database and Special-Purpose System Tables

A.1 Replication Option Transfer Database Tables A–1
A.1.1 Change in Transfer Database Tables A–1
A.1.2 Storage of Large Data Objects A–2
A.1.3 DDAL_INVOKE_TIME Transfer Database Table A–3
A.1.4 DDAL_TRANSFERS Transfer Database Table A–3
A.1.5 DDAL_SOURCE_DBS Transfer Database Table A–5
A.1.6 DDAL_TRANSFER_DB Transfer Database Table A–5
A.1.7 DDAL_TNSF_COLUMNS Transfer Database Table A–6
A.1.8 DDAL_TNSF_TABLES Transfer Database Table A–6
A.1.9 DDAL_TNSF_SCHEDULE Transfer Database Table A–7
A.1.10 DDAL_TNSF_STATUS Transfer Database Table A–8
A.1.11 DDAL_TNSF_VIEWS Transfer Database Table A–9
A.1.12 DDAL_COMMENT Transfer Database Table A–9
A.1.13 DDAL_DEST_DPB Transfer Database Table A–10
A.1.14 DDAL_MOVE_CLAUSES Transfer Database Table A–10
A.1.15 DDAL_SCHED_OFFSETS Transfer Database Table A–10
A.1.16 DDAL_TNSF_PRIVS Transfer Database Table A–11
A.1.17 DDAL_TNSF_SEL_EXPR Transfer Database Table A–11
A.2 Replication Option System Tables in Source Databases A–12
A.2.1 RDB$TRANSFERS System Table A–12
A.2.2 RDB$TRANSFER_RELATIONS System Table A–13
A.2.3 RDB$CHANGES System Table A–13
A.2.4 RDB$CHANGES_MAX_TSER System Table A–14
A.3 Replication Option System Tables in Target Databases A–14
A.3.1 RDB$VINTAGE and DDAL$TRANSFER_INFO Tables A–15
A.3.2 RDBVMS$TRANSFER_OWNER and DDAL$TRANSFER_OWNR Table A–17
A.3.3 DDAL$INDEXES Table A–18

B Replication Option Environment Variables

B.1 Customizing Replication Option Using Environment Variables B–1
B.1.1 Environment Variables for Replication Option B–2
B.1.2 General Usage Guidelines for Replication Option Environment Variables B–3
B.2 DDAL$CHANGES_BUFFER_SIZE Environment Variable B–8
B.3 DDAL$CHANGES_INDEX Environment Variable B–9
B.4 DDAL$CHECK_TARGET_DEFINITIONS Environment Variable B–10
B.5 DDAL$CHECKPOINT_DELAY_IN_MINUTES and DDAL$CHECKPOINT_MAX_RETRIES Environment Variables B–11
B.6 DDAL$CP_CONTINUE Environment Variable B–13
B.7 DDAL$CP_LOG_OPTIONS Environment Variable B–15
B.8 DDAL$CP_READ_CHANGES Environment Variable B–16
B.9 DDAL$EXTRACT_DATA Environment Variable B–17
B.10 DDAL$LOG_TRMON_CLIENT_EVENTS Environment Variable B–18
B.11 DDAL$MAX_COPY_PROCESS Environment Variable B–19
B.12 DDAL_NUMBER_OF_DATABASES, DDAL_DATABASE_n, DDAL_USERNAME_n, and DDAL_PASSWORD_n Environment Variables B–20
B.13 DDAL$PURGE Environment Variable B–23
B.14 DDAL$RW_ROW_COUNT Environment Variable B–24
B.15 DDAL$SHARED_WRITE Environment Variable B–25
B.16 DDAL$SINGLE_DATA_COMMIT Environment Variable B–25
B.17 DDAL$START_WITH_SOURCE_TABLE Environment Variable B–26
B.18 DDAL$STORE_DEF_SEPARATE Environment Variable B–27
B.19 DDAL$TARGET_INDEX_OPTIONS Environment Variable B–28
B.20 Rules for Setting Up a Configuration File B–29

C Replication Option Operational Information

C.1 Operational Procedures C–1
C.1.1 Creating the Transfer Database on OpenVMS C–1
C.1.1.1 Using Rdb as the Transfer Database C–2
C.1.1.2 Using Oracle as the Transfer Database C–3
C.1.2 Starting the Transfer Monitor on OpenVMS C–3
C.1.2.1 Additional Files Created When Transfer Monitor Is Started C–8
C.2 Stopping the Transfer Monitor on OpenVMS C–8
C.3 Transfer States and Transfer Status Codes C–8
C.4 Source Database Tables Used for Replication Option C–11
C.5 Target Database Tables and Indexes Used for Replication Option C–12

C.6 Access Rights for Transfer Database and Special-Purpose System Tables C–14
C.7 Transaction Activity During Transfer Execution C–15
C.7.1 Transfer Retries After Recoverable Errors C–16
C.7.2 Common Operations C–17
C.7.3 Extraction or Extraction Rollup Transfer C–18
C.7.3.1 Transfer to a New Target Database C–18
C.7.3.2 Transfer to an Existing Target Database C–20
C.7.4 Initial or Reinitialized Replication Transfer C–24
C.7.4.1 Transfer to a New Target Database C–24
C.7.4.2 Transfer to an Existing Target Database C–28
C.7.5 Replication Update Transfer C–34

D Examples

D.1 Using an SQL Application to Create and Control the Execution of Transfers D–1
D.1.1 Compiling and Linking the Test Program D–8
D.1.2 Checking Transfer Execution Status D–8
D.2 Using Rdb Transparent Gateway to RMS with Replication Option D–9

E Replication Option Error Messages and Online Help

E.1 Replication Option Error Messages E–1
E.2 Online Help Files for OpenVMS E–1

Glossary

Index

Examples

2–1 Selecting a Subset of Tables for the Target Database 2–6
2–2 Selecting a Subset of Columns for the Target Database 2–7
2–3 Selecting a Subset of Rows for the Target Database 2–8
4–1 Explicitly Named Remote Account Using Syntax for DECnet 4–12
4–2 Explicitly Named Remote Account Using Syntax for TCP/IP 4–13
4–3 Explicitly Named Remote Database Account with Local Transparent Gateway 4–14

4–4	Explicitly Named Remote Accounts with Remote Database and Transparent Gateway	4–14
4–5	Explicitly Named Remote Database Account with Local Transparent Gateway	4–15
4–6	Remote Access Using a DECnet Proxy	4–16
4–7	DDAL.RC Configuration File with Two User Accounts and Databases	4–20
4–8	Implicitly Named Remote Server Account	4–22
5–1	Extraction Transfer Definition on OpenVMS	5–12
5–2	Replication Transfer Definition	5–21
5–3	Sample Target DB2 Attach String	5–30
5–4	Sample Configuration File for Nonshared-Write Replication Transfer	5–30
5–5	Sample Configuration File for Shared-Write Replication Transfer	5–31
6–1	Schedule Definition	6–2
6–2	Schedule Definition Specifying Absolute Time	6–4
6–3	Transfer in the SCHEDULED State	6–5
6–4	Customizing LOGIN.COM File to Skip Portions of Its Code	6–9
6–5	Dropping a Transfer Definition	6–14
6–6	Displaying a Transfer Definition on OpenVMS	6–19
6–7	Displaying a Schedule Definition	6–21
6–8	Displaying the Status of a Transfer	6–21
6–9	Sample Copy Process Log File on OpenVMS	6–23
6–10	Sample Summary Statistics with Overflow Values Recorded	6–27
6–11	Transfer Monitor Log File	6–30
7–1	Prologue and Epilogue Files Specified in CREATE TRANSFER	7–2
7–2	Prologue Information Within a Copy Process Log File	7–9
7–3	DDAL$EPILOGUE.COM Output in Copy Process Log File	7–16
7–4	Epilogue File Named in a Transfer Definition	7–23
7–5	Epilogue Command Procedure Using SUBMIT	7–24
7–6	Remote Command Procedure Executed by SUBMIT	7–25
7–7	Epilogue Command Procedure Using TYPE	7–26
7–8	Remote Command Procedure Executed by TYPE	7–27
8–1	Creating an Archive Database Using Replication with NO DELETE	8–14
B–1	Sample User Configuration File	B–30

Figures

1–1 Replication Option Architecture 1–3
1–2 Network Containing a Central Database 1–9
2–1 Subset of Tables Transferred to a Target Database 2–5
2–2 Subset of Columns Transferred to a Target Database 2–6
2–3 Selecting Rows for Transfer to a Target Database 2–8
2–4 Remote Nodes Requiring Database Subsets 2–11
2–5 Extracted Data Subsets at Remote Nodes 2–12
2–6 Extraction Databases 2–13
2–7 Extraction Rollup 2–14
3–1 Data Transfer with Replication Option Components 3–2
4–1 DECnet Proxy Access for Remote OpenVMS Nodes 4–17
5–1 Identifying Rows for an Extraction 5–9

Tables

1–1 Supported Data Sources and Database Targets 1–4
1–2 Supported Data Sources and Targets and Associated Oracle Products 1–4
2–1 Data Location Requirements 2–19
2–2 Application Type and Transfer Method 2–20
2–3 Source Database Types 2–21
2–4 Target Database Types 2–21
2–5 Types of Transfer Available 2–22
2–6 Storage Requirements for Transfers that Create a Target Database 2–23
2–7 Storage Requirements for Transfers to an Existing Target Database 2–23
2–8 Storage Requirements for Source Databases 2–24
3–1 Main Components of Replication Option 3–2
4–1 Correspondence Between Replication Option and Rdb Versions 4–3
6–1 Tables and Columns Useful for Target Users and Applications 6–20
6–2 Replication Option Columns in Target Tables 6–47
6–3 Indexes for the RDB$CHANGES Table 6–65
6–4 Equivalence Names for Index Variables 6–66
7–1 Status Corresponding to DDAL$CP_CONTINUE 7–5
7–2 Epilogue Failure Status Conditions for DDAL$CP_CONTINUE 7–6

7–3	Transfer Status Conditions	7–13
7–4	Prologue and Epilogue Command Procedures Status Messages and Their Meanings	7–15
8–1	Modes of Data Sharing in Replication Option	8–2
9–1	Replication Option Privileges	9–78
A–1	Replication Option Transfer Database: DDAL_INVOKE_TIME Table	A–3
A–2	Replication Option Transfer Database: DDAL_TRANSFERS Table	A–3
A–3	Replication Option Transfer Database: DDAL_SOURCE_DBS Table	A–5
A–4	Replication Option Transfer Database: DDAL_TRANSFER_DB Table	A–6
A–5	Replication Option Transfer Database: DDAL_TNSF_COLUMNS Table	A–6
A–6	Replication Option Transfer Database: DDAL_TNSF_TABLES Table	A–6
A–7	Replication Option Transfer Database: DDAL_TNSF_SCHEDULE Table	A–7
A–8	Replication Option Transfer Database: DDAL_TNSF_STATUS Table	A–8
A–9	Replication Option Transfer Database: DDAL_TNSF_VIEWS Table	A–9
A–10	Replication Option Transfer Database: DDAL_COMMENT Table	A–9
A–11	Replication Option Transfer Database: DDAL_DEST_DPB Table	A–10
A–12	Replication Option Transfer Database: DDAL_MOVE_CLAUSES Table	A–10
A–13	Replication Option Transfer Database: DDAL_SCHED_OFFSETS Table	A–11
A–14	Replication Option Transfer Database: DDAL_TNSF_PRIVS Table	A–11
A–15	Replication Option Transfer Database: DDAL_TNSF_SEL_EXPR Table	A–11
A–16	Replication Option Source Database: RDB$TRANSFERS System Table	A–12
A–17	Replication Option Source Database: RDB$TRANSFER_RELATIONS System Table	A–13

A–18	Replication Option Source Database: RDB$CHANGES System Table	A–14
A–19	Replication Option Source Database: RDB$CHANGES_MAX_TSER System Table	A–14
A–20	Replication Option Tables in a Target Database	A–15
A–21	Replication Option Target Database: RDB$VINTAGE or DDAL$TRANSFER_INFO System Table	A–16
A–22	Replication Option Target Database: RDBVMS$TRANSFER_OWNER or DDAL$TRANSFER_OWNR Table	A–18
A–23	Replication Option Target Database: DDAL$INDEXES Table	A–18
B–1	Functions and Formats for Replication Option Environment Variables	B–5
B–2	Environment Variables with Specific Formats	B–7
B–3	Definitions for DDAL$CHANGES_INDEX	B–10
B–4	Definitions for DDAL$CP_CONTINUE	B–14
B–5	Definitions for DDAL$CP_LOG_OPTIONS	B–16
B–6	Definitions for DDAL$CP_READ_CHANGES	B–17
C–1	Transfer States	C–9
C–2	Events Affecting Transfer States	C–9
C–3	Status Codes for the DDAL$LAST_COMPLETION_STATUS Column	C–11
C–4	Special-Purpose System Tables Access Rights	C–14
C–5	Access Rights for Transfer Database Tables	C–15
C–6	Replication Option Recoverable Errors	C–16

Send Us Your Comments

Oracle Corporation welcomes your comments and suggestions on the quality and usefulness of this publication. Your input is an important part of the information used for revision.

You can send comments to us in the following ways:

- Electronic mail — nedc_doc@us.oracle.com
- FAX — 603-881-0120 Attn: Rdb Documentation
- Postal service

  Oracle Corporation
  Oracle Rdb Documentation
  110 Spit Brook Road, ZKO2-1/O19
  Nashua, NH 03062-2698
  USA

If you like, you can use the following questionnaire to give us feedback. (Edit the online release notes file, extract a copy of this questionnaire, and send it to us.)

Name _____ Title _____

Company _____ Department _____

Mailing Address _____ Telephone Number _____

Book Title _____ Version Number _____

- Did you find any errors?
- Is the information clearly presented?
- Do you need more information? If so, where?

- Are the examples correct? Do you need more examples?
- What features did you like most about this manual?

If you find any errors or have any other suggestions for improvement, please indicate the chapter, section, and page number (if available).

Preface

This manual provides information on how to use Replication Option for Rdb to transfer data.

Prior versions of this product were called VAX Data Distributor and DEC Data Distributor.

Intended Audience

Read this manual if you are involved in distributing data from source databases to target databases on local or remote nodes. To gain the most from this manual, you should be familiar with basic data processing procedures, database management concepts and terminology, the OpenVMS operating system, either Rdb, Oracle, or Sybase software, and software associated with any of the data management products you intend to use.

Operating System Information

Information about the versions of the operating system and related software that are compatible with this version of Replication Option for Rdb is included in the installation guide, which accompanies the Replication Option for Rdb media kit.

Structure

This manual has the following chapters and appendixes, and a glossary:

Chapter 1	Introduces Replication Option for Rdb and describes the advantages of central and distributed databases.
Chapter 2	Provides an overview of Replication Option for Rdb features. Contains guidelines for choosing the transfer method suited to particular database applications.

Chapter 3	Describes the components of Replication Option for Rdb.
Chapter 4	Provides descriptions of general tasks you can perform to prepare your application environment for data transfers.
Chapter 5	Describes how to distribute data from a single source database using extraction or replication transfers, or from one or more source databases using extraction rollup transfers.
Chapter 6	Describes how to work with transfers once they are defined.
Chapter 7	Describes how to use pre- and postprocessing command procedures with data transfers on OpenVMS.
Chapter 8	Provides information required to define transfers that use existing, user-created databases as target databases.
Chapter 9	Provides descriptions and syntax diagrams for the Replication Option for Rdb statements used with SQL.
Appendix A	Contains information about Replication Option for Rdb tables in a transfer database, a source database, and a target database.
Appendix B	Describes the environment variables used by Replication Option for Rdb software that you can define.
Appendix C	Provides information for programmers and system managers about how Replication Option for Rdb works.
Appendix D	Provides examples and information on using dynamic SQL, SQL module language, and the Rdb Transparent Gateway to RMS.
Appendix E	Provides information on error messages and online help.
Glossary	Defines terminology used with Replication Option for Rdb.

Related Documents

The other documents in the Replication Option for Rdb documentation set are:

- *Replication Option for Rdb Installation Guide*—describes how to install Replication Option for Rdb on the OpenVMS operating systems.
- *Replication Option for Rdb Release Notes*—describes problems corrected in the current release, and known problems and restrictions. They are provided in online format only, in the following file, once Replication Option is installed:

  ```
  SYS$HELP:DDAL070.RELEASE_NOTES
  ```

For more information on SQL, see the Oracle Rdb SQL Reference Manual for the version of Oracle Rdb you are using.

For more information about the other database products with which Replication Option works, see the documentation sets for Oracle Rdb, Distributed Option for Rdb, Rdb Transparent Gateway to Custom Drivers, Rdb Transparent Gateway to DB2, Rdb Transparent Gateway to Oracle CODASYL DBMS, Rdb Transparent Gateway to Oracle, Rdb Transparent Gateway to PC Data, Rdb Transparent Gateway to RMS, and Rdb Transparent Gateway to Sybase.

For related information on creating proxy accounts on OpenVMS systems using DECnet, refer to the OpenVMS documentation about system management, networking, and system security.

Syntax Diagrams

This manual presents the syntax of Replication Option for Rdb statements using diagrams. These syntax diagrams graphically portray optional, required, and repeated characteristics of any Replication Option for Rdb statement. You can learn the syntax of a 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 column where column-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 using 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 after the diagram.

For complete details about reading and using syntax diagrams, refer to the Oracle Rdb SQL Reference Manual for the version of Oracle Rdb you are using.

Conventions

This manual uses the following conventions:

WORD	An underlined uppercase word in a syntax diagram means that you must include the word in the clause or statement.
WORD	An uppercase word without underlining in a syntax diagram means that you can omit the word without changing the meaning of the clause or statement.
word	A lowercase word in a syntax diagram indicates a syntax element that you supply.
xxx-xxx-clause	Word strings joined with hyphens and ending in the word "clause" throughout this manual refer to elements of the syntax diagrams that appear in Chapter 9 of this manual.
$	The dollar sign represents Digital Equipment Corporation's Command Language prompt on the OpenVMS operating system. This symbol indicates that the DCL interpreter is ready for input.
⋮	Vertical ellipsis in an example mean that information not directly related to the example has been omitted.
. . .	Horizontal ellipsis in commands mean that parts of the 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.
environment variable	This term is used in this manual to refer to logical names and to variable names that can be entered in configuration files. See Appendix B for information on valid formats for environment variables.

In this manual, OpenVMS means both the OpenVMS Alpha operating system and the OpenVMS VAX operating system. The following icons are used to identify operating-system-specific information:

OpenVMS Alpha

The Alpha icon denotes information specific to the OpenVMS Alpha operating system.

OpenVMS VAX

The VAX icon denotes information specific to the OpenVMS VAX operating system.

The diamond symbol (♦) denotes the end of information specific to an operating system.

References to Products

Product Name Changes

As a convenience for users familiar with previous product versions, the following table lists old and new names of several products.

Old Name(s)	New Name(s)
DBI or DB Integrator	Distributed Option for Rdb and Parallel Query Option for Rdb and DataBase Integrator
DDD or Data Distributor	Replication Option for Rdb
DB Integrator Gateway for xxx	Rdb Transparent Gateway to xxx

The documentation for Distributed Option and Replication Option often refers to the following products by their abbreviated names:

- The term Distributed Option is often used to refer collectively to the following options and products:
  - Distributed Option for Rdb
  - DataBase Integrator (like the preceding item, but for heterogeneous data integration)
  - Parallel Query Option for Rdb
- Distributed Option for Rdb is referred to as Distributed Option.
- Parallel Query Option for Rdb is referred to as Parallel Query Option.

- Oracle Rdb is referred to as Rdb.

  Note

  The use of the term *Oracle* by itself — when applied to a database format, database management software, or transparent gateway — usually refers to the current major release of the Oracle product (such as Oracle7), as opposed to the Oracle Rdb (or simply Rdb) product.

- The CDD/Repository for OpenVMS Alpha or VAX product is referred to as CDD/Repository, the data dictionary, dictionary, or, more commonly, repository.
- The Replication Option for Rdb for OpenVMS Alpha or VAX product is referred to as Replication Option.
- The DEC DATATRIEVE for OpenVMS Alpha or VAX product is referred to as DATATRIEVE.
- The Rdb Transparent Gateway to DB2, Rdb Transparent Gateway to Oracle, Rdb Transparent Gateway to PC Data, and Rdb Transparent Gateway to Sybase products are referred to as the relational gateways.
- The Rdb Transparent Gateway to Custom Drivers, Rdb Transparent Gateway to Oracle CODASYL DBMS, and Rdb Transparent Gateway to RMS products are referred to as the nonrelational gateways.
- The Rdb Transparent Gateway family of data access products is often referred to as Rdb Transparent Gateways.

  The Rdb Transparent Gateway family is made up of the following products. Each product is often referred to as *the gateway to [identifying product name]*. For example, the Rdb Transparent Gateway to Custom Drivers is often referred to as the gateway to Custom Drivers.

  - Rdb Transparent Gateway to Custom Drivers[1], [2]
  - Rdb Transparent Gateway to DB2 Client[1], [2]
  - Rdb Transparent Gateway to DB2 Server[3]
  - Rdb Transparent Gateway to Oracle CODASYL DBMS[1]

[1] This product is available on the OpenVMS Alpha and OpenVMS VAX systems.
[2] This product is available on the Digital UNIX Alpha system.
[3] This product is only available on the IBM system.

  - Rdb Transparent Gateway to Oracle[1, 2]
  - Rdb Transparent Gateway to PC Data[1, 2]
  - Rdb Transparent Gateway to RMS[1]
  - Rdb Transparent Gateway to Sybase[1, 2]
- The Digital gateways to SNA networks are referred to as the DEC SNA gateway products. These products include:
  - OpenVMS SNA
  - DECnet SNA Gateway for Channel Transport
  - DECnet SNA Gateway for Synchronous Transport
  - DEC SNA Domain Gateway
  - DEC SNA Peer Server
- The DECnet/SNA Data Transfer Facility product is referred to as DTF.
- DECnet is the software facility from Digital Equipment Corporation that implements the Digital Network Architecture, allowing users to access information on a remote computer through telecommunications lines. DECnet software is referred to as DECnet.
- DECnet/OSI for OpenVMS VAX or Alpha and DECnet/OSI for Digital UNIX Alpha software are referred to as DECnet.
- DEC TCP/IP Services for OpenVMS VAX or Alpha is referred to as TCP/IP Services.
- DEC TCP/IP for Digital UNIX Alpha is referred to as TCP/IP.
- The Oracle Trace for OpenVMS Alpha or VAX product is referred to as Oracle Trace or simply Trace.
- The Oracle Expert for OpenVMS Alpha or VAX product is referred to as Oracle Expert or simply Expert.
- The Oracle Rdb for OpenVMS Alpha or VAX product is referred to as Rdb.
- The OpenVMS RMS (Record Management Services) product is referred to as RMS.
- SQL refers to the SQL interface to Oracle Rdb and to the Distributed Option family of products. SQL is an implementation of the SQL standard ANSI X3.135-1992, ISO 9075:1992, commonly referred to as ANSI/ISO SQL.

The term **DDAL** appears in Replication Option documentation. For example, in the *Replication Option for Rdb Installation Guide*, DDAL appears as part of the installation name for Replication Option. The term also appears in the following contexts:

- Column descriptions displayed when you issue a SHOW COLUMNS statement for a target database, and for table and column names in the transfer database
- Message texts
- File names that are part of the Replication Option software
- Environment variables and logical names

Technical Changes and New Features

This section identifies the new and updated features in Replication Option Version 7.0.

Each of these features is described in this handbook. The *Replication Option for Rdb Release Notes* describe current limitations and restrictions. They are located on line in the following file, once Replication Option is installed:

```
SYS$HELP:DDAL070.RELEASE_NOTES
```

The rest of this section contains brief descriptions of new and updated features, grouped by topics.

New Features in Gateway Support

There are some new features affecting gateway support:

- Support for the Rdb Transparent Gateway to PC Data has been added. It is supported as both data sources and targets. See Chapter 2 and Section 5.5 through Section 5.5.12 for more information on this gateway product.

  See the *Replication Option for Rdb Installation Guide* for information on installing this gateway product for use with Replication Option.

- Support for default read-only gateways. If a new gateway product appears, that Replication Option does not recognize, that gateway product will be treated as a read-only database and can be used for sources of data transfers.

New Features Supporting TCP/IP

TCP/IP is now supported as a network transport (see Section 4.8.2 for more information). When TCP/IP is used, there is an option that permits access to multiple databases:

- You can specify multiple accounts in the DDAL.RC configuration file. The environment variable used to do this, DDAL_NUMBER_OF_ DATABASES, and other related variables are described in Section B.12.

New Feature Supporting Digital UNIX

- You can now transfer data between Rdb on OpenVMS and Rdb on Digital UNIX Alpha systems. See Section 4.8.5 for information about this type of transfer.

New Feature Supporting Oracle Transfer Database

Replication Option now gives you a choice of products you can use to manage the transfer database tables. You can use either Oracle Rdb, as in the past, or you can use Oracle RDBMS.

- For information on how to create transfer database tables in an Oracle database, refer to Section C.1.1.2.
- The command procedure used to start the transfer monitor, DDAL$START_TR_MON.COM, has been changed to support use of an Oracle database as the transfer database. See Section C.1.2 for more information.

New Features That Help Improve Performance

There are some new options that help improve transfer performance:

- You can specify how many data rows are read and written at a time by Replication Option during an extraction, extraction rollup, or initial replication transfer. The environment variable that enables this option, DDAL$RW_ROW_COUNT, is described in Section B.14.
- You can direct Replication Option to delete and write data that is transferred into the target tables all in one transaction. The environment variable used to enable this option, DDAL$SINGLE_ DATA_COMMIT, is described in Section B.16.
- For concurrently executing extraction or initial replication transfers, you can instruct a copy process to transfer data starting with a particular table. The environment variable used to enable this option, DDAL$START_WITH_SOURCE_TABLE is described in Section B.17.

- You can define whether replication transfers execute one after the other or concurrently. The environment variable used to do this, DDAL$STORE_DEF_SEPARATE, is described in Section B.18.
- To enhance performance of data lookup in the RDB$TRANSFER_ RELATIONS table, an index is now used. The index, RDB$TRAN_ RELS_REL_NAME_NDX, is a sorted index on the RDB$RELATION_ NAME column with duplicate values allowed. This is beneficial when you replicate many tables and have numerous replication transfers on a single source database.
- Performance can be improved by moving the RDB$CHANGES table to its own storage area on a separate disk. See Section 3.10 for more information.

There are also new options that improve performance by making Replication Option more tolerant of errors.

- You can alter the size of the virtual memory buffer so that during a replication update, the copy process can store all the data for the current transaction in virtual memory. In the event of a target data access failure this feature enables the copy process to restart the target transaction without also having to restart the source transaction. The environment variable that lets you do this, DDAL$CHANGES_ BUFFER_SIZE, is described in Section B.2.
- You can now make use of the the checkpoint restart feature in deadlock or lock-conflict situations that last more than 10 minutes. See Section B.5 for information on the two environment variables that affect this option, DDAL$CHECKPOINT_DELAY_IN_MINUTES and DDAL$CHECKPOINT_MAX_RETRIES.

New Features That Permit Sharing of Data

There are some new features that allow data to be shared in ways which were not possible before:

- You can have several transfers writing to the same target table. With this capability, you can perform replication rollup transfers, for example. The environment variable that enables this shared-write capability, DDAL$SHARED_WRITE, is described in Section B.15.
- Customer applications and replication transfers can write into the same tables, under limited circumstances. The environment variable that enables this option, DDAL$TARGET_INDEX_OPTIONS, is described in Section B.19.

New Features That Improve Usability of Replication Option

There are some new options that make Replication Option easier to use:

- You can suppress comparison of source and target table definitions any time an extraction transfer is re-executed or a replication transfer is reinitialized and re-executed. See Section B.4 for a description of the environment variable, DDAL$CHECK_TARGET_DEFINITIONS, that enables this option.
- You can disable extraction of the data itself and transfer only the data definitions. See Section B.9 for a description of the environment variable, DDAL$EXTRACT_DATA, used to enable this option.
- As an alternative to defining logical names, you can use a configuration file, DDAL.RC, to define environment variables.

  See Section B.1 for more information on configuration files and Section B.20 for rules on using these configuration files.
- Replication Option messages that appear in the transfer monitor and copy process log files have changed. The messages now use SQL terminology instead of the RDO terminology specific to Rdb.

New Features Improving Support of Multiple Rdb Versions

- The DDAL$CREATE_TR_DB.COM procedure, which creates an Rdb transfer database, has been changed to accept a version number parameter. This means that you can use a version of Rdb that is not the standard version on the system. Previous releases of Replication Option ran only against the system version of Rdb. If you wanted to run it against a different version of Rdb, then that version had to be made the system version.

  See Section C.1.1 for more information on the new format of the DDAL$CREATE_TR_DB.COM procedure.
- The DDAL$START_TR_MON.COM command procedure also supports using a version of Rdb that is not the standard version on the system. See Section C.1.2 for more information.

New Feature That Increases Diagnostic Capabilities

The following new option provides data you can use to diagnose problems encountered while using Replication Option:

- You can log events used to help diagnose communication problems between the transfer monitor and the user interface (for example, interactive SQL). See Section B.10 for a description of the environment

variable, DDAL$LOG_TRMON_CLIENT_EVENTS, used to enable this option.

Important Changes

The following lists important changes that might require you to adapt your operational procedures in order to continue working correctly with Replication Option:

- The command procedure for shutting down Replication Option, DDAL$STOP_TR_MON.COM, no longer automatically shuts down Replication Option on all nodes in a cluster. See Section C.2 for more information on the DDAL$STOP_TR_MON.COM procedure.
- Changes to the transfer database schema. Refer to Appendix A for a complete description of the new transfer database schema.

  Refer to the *Replication Option for Rdb Installation Guide* for information on converting existing databases to the new schema. If you have applications that directly access the transfer database, they will need to be redesigned.
- In target databases, the size of the RDBVMS$VINTAGE_TRANSFER_NODE column in the RDB$VINTAGE table is now created as CHAR(255) instead of CHAR(20). Similarly, the DDAL$TRANSFER_NODE column in the DDAL$TRANSFER_INFO table is created as CHAR(255). This change was made to accommodate transfers using TCP/IP.

  Conversion of existing CHAR(20) columns to the larger size is not performed automatically. If you have a target database in which one of those tables was created by an earlier version of Replication Option, you should alter the column size yourself. You can do so using the SQL ALTER statement.

1
Introduction

This chapter briefly lists the major features of Replication Option, associated data management and data access products, and associated interactive and application interfaces to Replication Option. It also contrasts two approaches to data management:

- Where data is maintained in a central database
- Where data is replicated at several sites

1.1 Replication Option Major Capabilities

Replication Option makes data available to users and applications at one or more sites in a network. Some noteworthy features of Replication Option are described in the following list:

- Replication Option runs on the OpenVMS VAX and OpenVMS Alpha operating systems and can transfer data using either DECnet or TCP/IP networking services.

  In addition to accessing data sources and targets on systems running OpenVMS, Replication Option can access data sources and targets on remote systems running other operating systems such as MVS, UNIX, and Microsoft Windows.
- You can choose from a variety of data management products to be the sources and targets of the data transfers (see Section 1.2).
- Data can be transferred from one source to one target database or it can be rolled up from several sources into one target.
- You can copy all tables from a data source or select which tables to copy.
- You can use all columns for a particular table or you can choose only selected columns.
- You can transfer all data from a given table or specify criteria that limit data rows transferred to those you selected.

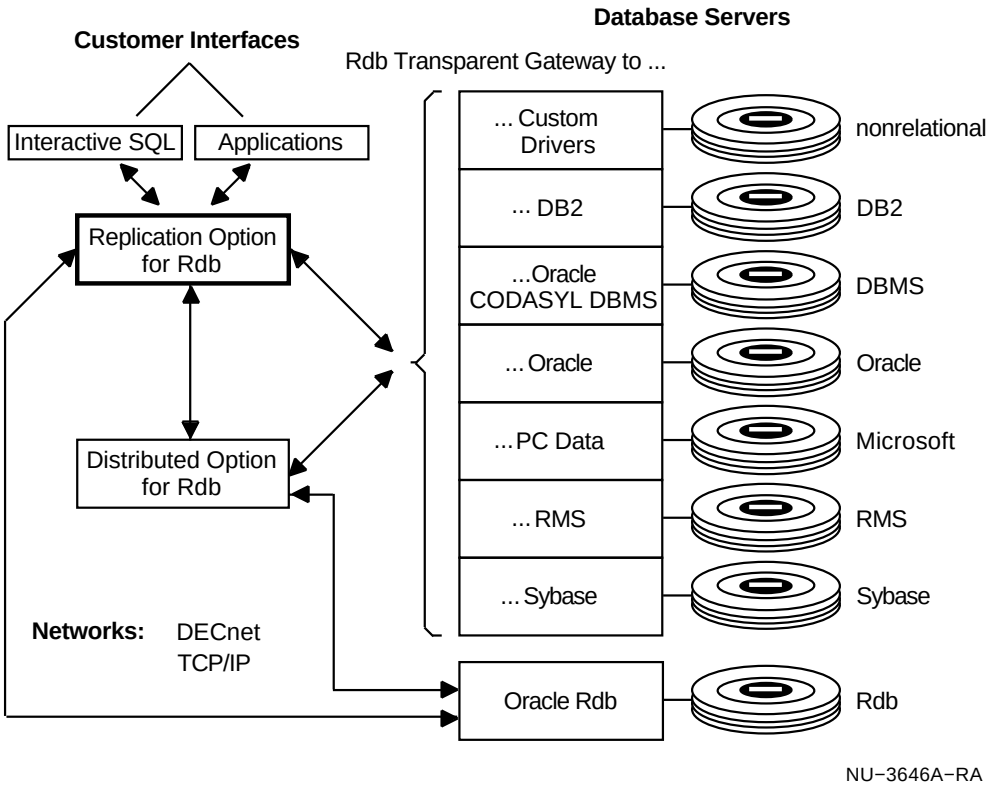
- Source and target databases can be located on the same machine (processor) on which Replication Option is executing or they can be accessed remotely in other parts of the network.
- Replication Option can create the target database, and will copy table and view definitions as needed for those products that support these operations.
- Index, trigger and constraint definitions, and access rights to database objects can be created as part of the data transfer process through the use of customer-written prologue and epilogue procedures.
- Replication Option can maintain a target database as an archive of data records that might since have been deleted from the production, source database for those products that support those operations.
- Data transfers can be executed on demand or on an automatic, scheduled basis.

Distributing copies of data using Replication Option provides these benefits:

- You can separate the data entry function of a production (enterprise) database from the data querying and reporting function of an MIS database.
- Congestion and competition for access to your production database is reduced when queries are performed on copies of the data rather than on the production data directly.
- Users on processors that are physically distant from the source database can have more direct, efficient, and convenient access to the data when they have their own copy of that data.

Figure 1–1 shows the customer interfaces to Replication Option, the networks that Replication Option runs on, and the various database servers that work with Replication Option.

Figure 1–1 Replication Option Architecture



1.2 Data Management and Data Access Products

Some of the Rdb Transparent Gateways provide access to more than one database management product. When these are used as data sources, Replication Option supports access to the full range of products. However, when used as targets, Replication Option supports only specific database managers. These exceptions are marked with footnotes in Table 1–1 and Table 1–2.

Table 1–1 shows the data managers supported by Replication Option.

Table 1–1 Supported Data Sources and Database Targets

Data Sources	Data Targets
Custom Drivers	
DB2	DB2
Distributed Option for Rdb	Distributed Option for Rdb[1]
Oracle CODASYL DBMS	
Oracle	Oracle[1]
Oracle Rdb	Oracle Rdb
PC Data	PC Data[1]
RMS	
Sybase	Sybase[1]

[1]For information about restrictions when using the gateway to this product, refer to the appropriate gateway description, later in this chapter.

Table 1–2 shows which Oracle database product allows you to access these data managers.

Table 1–2 Supported Data Sources and Targets and Associated Oracle Products

Data Source or Target	Oracle's Data Access Product
Custom Drivers	Rdb Transparent Gateway to Custom Drivers
DB2	Rdb Transparent Gateway to DB2
Distributed Option for Rdb	Distributed Option for Rdb
Oracle CODASYL DBMS	Rdb Transparent Gateway to Oracle CODASYL DBMS
Oracle	Rdb Transparent Gateway to Oracle[1]
Oracle Rdb	Oracle Rdb
PC Data	Rdb Transparent Gateway to PC Data[1]

[1]For information about restrictions when using the gateway to this product, refer to the appropriate gateway description, later in this chapter.

(continued on next page)

Table 1–2 (Cont.) Supported Data Sources and Targets and Associated Oracle Products

Data Source or Target	Oracle's Data Access Product
RMS	Rdb Transparent Gateway to RMS
Sybase	Rdb Transparent Gateway to Sybase[1]

[1]For information about restrictions when using the gateway to this product, refer to the appropriate gateway description, later in this chapter.

The following list describes each of the Oracle data access products.

- Rdb Transparent Gateway to Custom Drivers

  The Oracle database interconnect software product that provides access to third-party nonrelational data sources not explicitly supported by Rdb Transparent Gateway to RMS.

- Rdb Transparent Gateway to DB2

  The Oracle database interconnect software product that provides read/write access to IBM DB2 is known as Rdb Transparent Gateway to DB2.

- Rdb Transparent Gateway to Oracle CODASYL DBMS

  The Oracle database interconnect software product that allows you to update and read data from Oracle CODASYL DBMS databases is known as Rdb Transparent Gateway to Oracle CODASYL DBMS.

- Rdb Transparent Gateway to Oracle

  The Oracle database interconnect software product that provides read/write access to data stored in the Oracle relational database management system and, through SQL*Net, to many other databases.

  When this gateway is used as a data source, Replication Option supports access to any of the databases available. When this gateway is used as a data target, Replication Option supports only Oracle databases.

- Rdb Transparent Gateway to PC Data

  The Rdb Transparent Gateway to PC Data enables OpenVMS or Digital UNIX users to have access to data stored in data sources associated with PC Data servers. Programmers and end users can use one of many Oracle or third-party tools with the gateway to have access to the data. The gateway to PC Data client works with a gateway to PC Data server and associated data sources to provide access to data.

When this gateway is used as a data source, Replication Option supports access to any of the databases available. When this gateway is used as a data target, Replication Option supports only Microsoft Access or Microsoft SQL Server databases.

- Rdb Transparent Gateway to RMS

  The Oracle database interconnect software product that provides access to VMS RMS data files.

- Rdb Transparent Gateway to Sybase

  The Oracle database interconnect software product that provides access to data stored at all Sybase open servers, and to many other databases accessible to Sybase open servers.

  When this gateway is used as a data source, Replication Option supports access to any of the databases available. When this gateway is used as a data target, Replication Option supports only Sybase databases.

- Distributed Option for Rdb

  The Distributed Option family of products allows you to use the SQL interface to store, retrieve, and update data from data sources or metadata sources (repositories) of your choice in many different types of databases, such as Rdb, Distributed Option, and Rdb Transparent Gateway products on many different types of computer systems.

  When Distributed Option is used as a data source, Replication Option supports access to any of the underlying databases available. When Distributed Option is used as a data target, Replication Option supports writing to only those underlying targets shown in Table 1–1 and Table 1–2.

- Rdb

  An Rdb database contains data residing on a VAX or Alpha processor running the OpenVMS or Digital UNIX operating systems.

1.3 User and Application Interfaces to Replication Option

Statements to direct the operation of Replication Option can be entered by a user interactively or by an application program.

- Interactive interfaces

  Replication Option statements are entered interactively through the Rdb SQL interface. SQL is a standard interface of the database community and is recommended by Oracle as the interface of choice.

Note

To access data tables in the databases to which Replication Option transfers data, you can use either the interactive SQL interface native to the database system or you can use Rdb SQL. When communicating with the Replication Option product (to monitor the progress of a data transfer, for instance), you can only use interactive Rdb SQL.

- Application interfaces

  Application programs can control the execution of Replication Option through statements written in the SQL module language or dynamic SQL.

1.4 Using Centralized Data Versus Replicated Data

Many businesses have users and applications spread out geographically and require the use of multiple computer systems tied together by means of communication networks. In a loose sense, there are two approaches to data management:

- You can maintain a single, central database that requires remote users and applications to use the network for access to the data.
- You can make copies of the data available at the remote sites giving users and applications local copies.

The following sections discuss the advantages and disadvantages of each approach.

1.4.1 Central Data Management

This section describes the advantages and disadvantages of using a single, central database for both local and remote access.

Maintaining a central database in a network can provide the following benefits:

- Data and database definitions residing in a central database are under the control of a single, specific site.
- Data in the central database is always current. Updates are made to the only existing copy of every data element; access to the database always yields current data values.
- All authorized users are explicitly or implicitly named in a single database environment. The central database is a secure data storage environment.

However, when users and applications at sites physically removed from the database need access to a central database for routine information retrieval, several problems can result:

- Database access contention and poor performance

  Data stored in a central database is likely to be in great demand. If the central database must satisfy an increased load of remote requests, it might not be able to provide adequate response time for queries. Queries execute faster when fewer applications compete for the desired data. That is, response time is better when contention for the data is low. For example, to generate a report, one site might require access to only a few selected columns from two or three tables in a distant, central database. This application must contend with all other users attempting to access the database, just to retrieve a few data items.

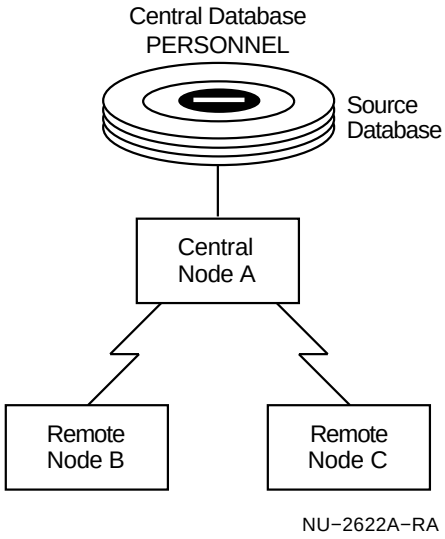
- Inflexibility to site application requirements

  Applications at different sites have different requirements. A performance tuning consideration for the central database might make sense for one site, but not for another. One site, for example, might wish to define indexes. If that site had access to a local copy of the database, it could use as many indexes as needed without affecting applications from other sites.

- Communications dependency

  Remote access requires a continuously available network link from the remote nodes to the node containing the central database. Whether users on the remote nodes access the central database for read-only tasks or for update tasks, managing both database and network resources and resolving access conflicts in the central database can be costly. Figure 1–2 shows three nodes (A, B, and C) in a network. The central database PERSONNEL resides on Node A. Nodes B and C access the database remotely.

Figure 1–2 Network Containing a Central Database



Access to data stored at a remote site is costlier than access to data stored locally. If your system uses remote access to the central database, then your system incurs these increased costs.

Data might not be available if communications lines are down. Even when communications links do perform properly, access can be slow depending on the speed of the line. Also, communications costs can inhibit or discourage full exploration of the available data.

1.4.2 Replicated Data Management

The benefits of creating and maintaining copies of the central database on local nodes include:

- Providing required relational tables at local sites
- Using local resources rather than requiring all applications to compete for resources on a single central database
- Allowing local creation and control of database entities, such as views and indexes
- Allowing the local node to determine strategies to achieve the best performance possible
- Defining local access rights

The local database administrator can specify and manage all authorized users of the local data.

- Reducing communications cost

  Network overhead occurs only when the data is being transferred to local nodes. Applications incur less communications cost and perform more efficiently when they access local databases.

- Increasing local control of data

  The local site can participate in determining when data is transferred from the source database.

Replicating data in this fashion can have its disadvantages too.

- Because of propagation delays and intermittent network failures, the data replicas can be out-of-date with respect to the central, production data.
- Management of Replication Option transfers is an additional burden one has to assume.

2
Overview

This chapter is in two parts. The first part defines the main concepts of Replication Option supporting the copying of data and data definitions from one data store to another and the various methods available for accomplishing the transfers. The second part leads you through the process of choosing a method of transfer that is appropriate to your application.

2.1 Replication Option Concepts

This section introduces the terminology of data transfers used throughout this handbook. It describes the different data distribution scenarios supported by Replication Option, and the various methods by which data distribution can be accomplished.

2.1.1 Transfers

The concept of a **transfer** is central to the operation of Replication Option. During a transfer, Replication Option duplicates the data definitions and data contained in one or more databases by copying those definitions and data to a second database. A database that contains definitions and data to be transferred is called a **source database**; a database that receives the definitions and data is called a **target database**.

Each transfer has an associated **transfer definition**. The transfer definition identifies the source database(s), the target database, the data definitions and data to be transferred, and the method of transfer execution to be used. These are the key parts of a transfer definition. Other parts of a transfer definition will be introduced later in this handbook. In a transfer definition where you specify what is to be transferred, you have the following options when transferring specified data:

- All the tables contained in a source database or a subset of those tables
- All the data rows for a given table or a subset of those rows
- All the data columns for a given table or a subset of those columns

Refer to Section 2.1.4 for information on selecting subsets of data.

Within a transfer definition, you must specify whether or not Replication Option will create the target database files before creating the table definitions and copying the data. The option of creating a new target database or of transferring to an existing target database is available only for Rdb target databases. For transfers to other targets, the database files must exist prior to executing the transfers.

The early chapters in this handbook explain how to implement transfers that create the target database files as part of transfer execution. Although that option is not available for most target systems, you can disregard that aspect of the discussions and pay attention to the rest. Chapter 8 describes transfers to existing databases but the earlier chapters are prerequisite reading.

Methods of transfer execution are explained in a later section of this chapter.

A transfer can have a **transfer schedule**, which specifies when a transfer is executed and how often the transfer is repeated. Transfers that do not have a transfer schedule are started on demand.

Each installation of Replication Option can create and support any number of transfers. Each transfer is defined, scheduled, and managed independently of the other transfers. Practical considerations, such as CPU load and size, limit the actual number of transfers that can execute concurrently, but there is no inherent Replication Option limitation that restricts the number of transfers that can be defined.

2.1.2 Local and Remote Databases

Transfers are defined on a CPU running Replication Option. In many implementations, the source database is local to the CPU on which transfers are defined and executed, and the target database is located on a remote CPU. However, except where explicitly noted elsewhere in this handbook, you can also use Replication Option in the reverse manner. That is, the target database can be local to the CPU on which transfers are defined and executed, and the source database can be located on a remote CPU. Additionally, you can create transfers where both the source and target databases are remote from where Replication Option is installed and executed.

2.1.3 Data Distribution Scenarios

You can define transfers to accomplish the following types of data distribution among databases:

- One source to many targets
- Many sources to one target
- Source and target data exchange

- Simple data archiving

The remainder of this section describes each of these data distribution scenarios.

2.1.3.1 Distributing Data from One Source to Many Targets

Forwarding reference data from a central, source database to multiple, remote target databases is one common use of Replication Option. You define a separate transfer for each target database and use the same source database in each transfer definition. You can define each transfer to copy the same data to all the target databases, or you can tailor the transfer definitions so that each target database receives data that accommodates site-specific needs.

If you define multiple transfers to copy the same data from a single source database, be aware of the following potential problem. After some of the transfers execute, it is possible for someone to make changes to the source database. If this happens, the changes will be included in subsequent transfers but not in transfers executed prior to the changes. If the transfers must be identical for all sites, first transfer the data to an intermediate database that cannot be modified. Then define the transfers to the remote sites using this intermediate database as the source.

Intermediate databases are also useful when there are a large number of target databases. You may find it impractical to transfer data from the source directly to all of the targets. When this is the case, consider using intermediate databases to effect a multistage, or **cascaded** transfer. With a cascaded transfer, one set of transfers executes and copies data from a single source to several targets. The second set of transfers executes and, using the targets from the first set of transfers as sources, creates a new set of targets. A cascaded transfer can be pictured as a pyramid, with the single source database at the top and the final target databases at the bottom.

2.1.3.2 Distributing Data from Many Sources to One Target

You can define a transfer that copies data from multiple source databases into a single target database. This many-to-one type of transfer is called a **rollup** transfer.

Rollup transfers can also use the cascaded transfer method described in Section 2.1.3.1. However, instead of copying data from a single source database to successively more target databases, a cascaded rollup transfer copies data from many source databases to successively fewer target databases. Finally, the last set of transfers copies data to a single target database. A cascaded rollup transfer can be pictured as an inverted pyramid, with the multiple source databases at the top and the single target database at the bottom.

2.1.3.3 Exchanging Data Among Peer Databases

You can define transfers that enable two or more existing databases to exchange data. In this type of transfer, known as a **two-way** transfer, Replication Option allows data collected in database A to be copied to database B. The definition for this transfer uses database A as the source, and database B as the target. A second definition uses database B as the source and database A as the target so that data collected in database B is copied to database A. The two databases, A and B, are considered peers. By defining multiple transfers, you can enable many existing databases to share data.

Transfers are not permitted in both directions between the same two tables unless you enable the shared-write option. See Section 8.2.5 for information on the shared-write option.

If you do not have shared writing enabled, the following restriction exists. You cannot create a transfer that copies data from a table of one database into that same table in the other database. To illustrate this point, consider two databases, each with a CUSTOMER table. Instead, you must transfer into a new table, such as, CUSTOMER2, to be created by Replication Option during transfer execution. Each database will have two tables: its original CUSTOMER table plus a new CUSTOMER2 table containing data transferred from the other database.

2.1.3.4 Archiving Data

A production database contains operational data for managing your current business, such as inventory on hand, work in progress, or recent customer invoices. To keep the size of the production database within manageable limits and to keep database performance from degrading, you need to delete old transactions. For historical reference, you must keep such old transactions available, some in alternate, online storage and older-still data captured on offline, reloadable media.

Replication Option provides a mechanism for performing a simplified type of data archiving. In this scenario, the target database holds a copy of each data row in the production database. This is the so-called *live* data. The target database also holds any data that has since been deleted from the production, source database. The target database accumulates more and more data over time. To keep the size of the archive database within reason, you must implement some mechanism to move data from the online, target, archive database to some offline storage, such as magnetic tape or optical disk.

Because of the transfer method required to manage the archiving and because that method is limited in its use, the source production database must be an Rdb database.

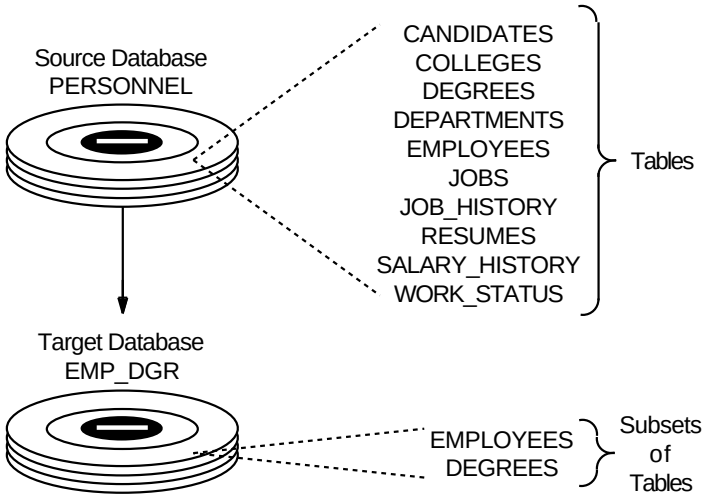
2.1.4 Selecting Subsets of Data

You can define a subset of a source database by using any or all of the following methods:

- Transferring only certain tables (see Figure 2–1)
- Transferring only certain columns (see Figure 2–2)
- Transferring only those rows that meet particular selection criteria (see Figure 2–3)

Figure 2–1 shows a subset of several tables in a PERSONNEL source database transferred to the target database. The source database contains 10 tables. The transfer definition selects two tables, EMPLOYEES and DEGREES, from the source database for transfer to the target database.

Figure 2–1 Subset of Tables Transferred to a Target Database



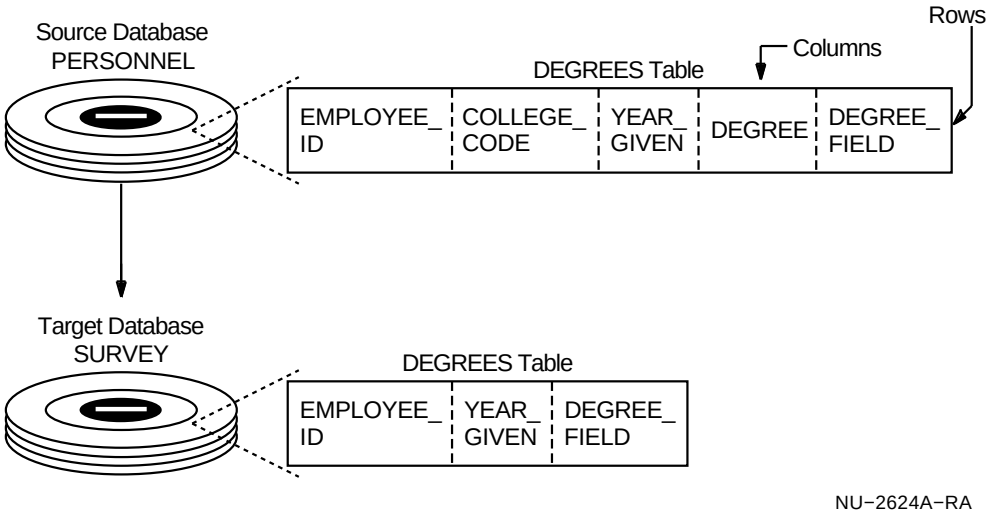
Example 2–1 shows a portion of a transfer definition that selects the tables illustrated in Figure 2–1 for transfer to the EMP_DGR target database.

Example 2–1 Selecting a Subset of Tables for the Target Database

```
MOVE TABLES
     EMPLOYEES,
     DEGREES
TO NEW FILENAME EMP_DGR
```

Similarly, you can transfer a subset of columns. Figure 2–2 shows a subset of three columns in the DEGREES table transferred to the target database. The DEGREES table contains five columns: EMPLOYEE_ID, COLLEGE_CODE, YEAR_GIVEN, DEGREE, and DEGREE_FIELD. The transfer definition selects the EMPLOYEE_ID, YEAR_GIVEN, and the DEGREE_FIELD columns to be transferred to the target database.

Figure 2–2 Subset of Columns Transferred to a Target Database



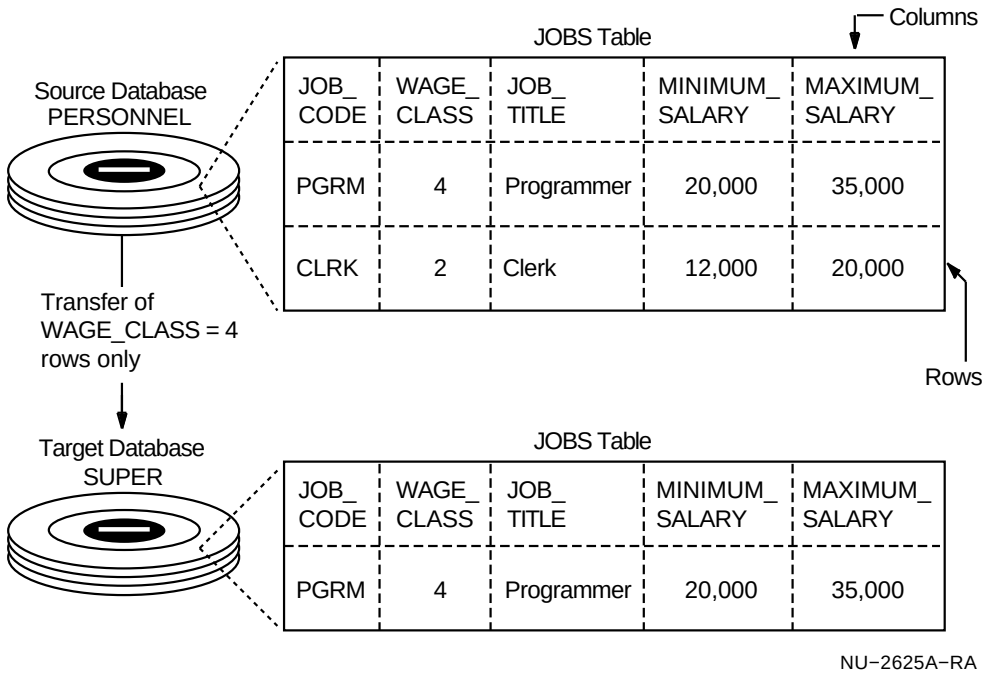
Example 2–2 shows a portion of a transfer definition that selects the tables illustrated in Figure 2–2 for transfer to the SURVEY target database.

Example 2–2 Selecting a Subset of Columns for the Target Database

```
MOVE TABLES
     SELECT
        EMPLOYEE_ID,
        YEAR_GIVEN,
        DEGREE_FIELD
      FROM DEGREES
TO EXISTING FILENAME SURVEY;
```

You can also select specific rows for transfer to a target database as shown in Figure 2–3.

Figure 2–3 Selecting Rows for Transfer to a Target Database



Example 2–3 shows a portion of a transfer definition that selects the rows illustrated in Figure 2–3 for transfer to the SUPER target database.

Example 2–3 Selecting a Subset of Rows for the Target Database

```
MOVE TABLES
     SELECT * FROM JOBS
     WHERE WAGE_CLASS = 4
TO NEW FILENAME SUPER;
```

When you form subsets in the transfer definition, you can use one of the types of subsets—tables, columns, or rows—or any combination of them.

2.1.5 Methods of Transfer

The process of duplicating data by transfers is informally called data replication. However, in Replication Option terminology, replication has a narrower meaning; it is one of the three methods of transfer:

- Extraction
- Extraction rollup
- Replication

With some exceptions noted elsewhere in this handbook, these methods achieve the same result: the replication of data from one or more data stores to others. The difference is in how the transfers are accomplished.

Although there is no formal transfer method known as "replication rollup", the rollup of data using multiple replication transfers is possible. This is discussed in the following sections.

Target databases are sometimes referred to as **extraction databases**, **extraction rollup databases**, and **replication databases** to highlight the method of transfer used to move the data.

2.1.5.1 Extraction and Extraction Rollup Transfers

With the extraction and the extraction rollup methods, Replication Option creates a new copy of the target database with each transfer only if the TO NEW FILENAME clause is specified in the transfer definition.

If you use Replication Option to create the target database files within OpenVMS, then a new set of database files is created without any existing ones being deleted with each transfer execution.

The Digital UNIX operating system does not support the creation of multiple versions of a file with the same path name. Therefore, if you use Replication Option to create the target database files, you must be prepared to delete such files if they already exist. This situation would occur when you re-execute a transfer.

The extraction method uses a single source database and transfers data into a single target database. With the extraction rollup method, data from multiple source databases can be rolled up into a single new target database. Once the transfer is complete, all connections with the source database are severed. Users can query or update the transferred tables in an extraction or extraction rollup database; changes they make affect the target database only and are not transferred back to the source database.

Extraction and extraction rollup transfers offer several benefits in common. In the following list, the term extraction will be used to signify either transfer method:

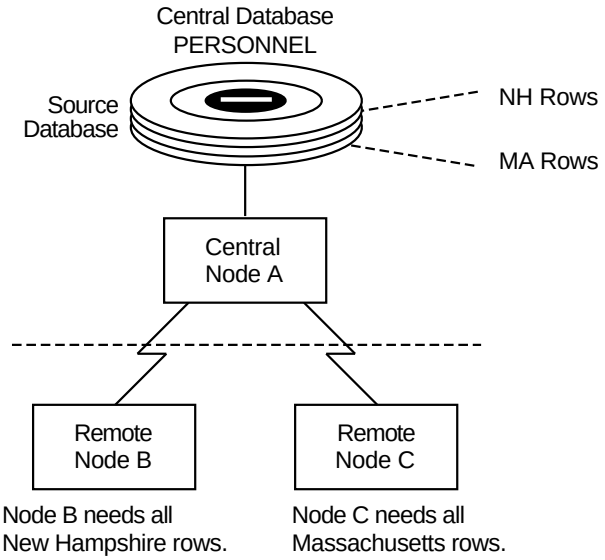
- Extraction databases let users manipulate data at target sites, thus eliminating both the response time penalties resulting from remote access and the competition for database resources created by many users accessing a central database.
- Extraction databases can be altered in ways that make data retrieval more efficient than it might be in a production database. A production database is designed primarily for efficient capture of data, not necessarily for data retrieval. Once you have an extraction database separate from the production source, you are free to add indexes that will better the performance of those applications used specifically with the target database.
- Data contained in extraction databases can be changed at target sites without affecting the source database. For example, you can experiment with data values by using statistical or other mathematical functions on them to produce sets of forecast data. You then can return the data to its original state or commit the changes. In either case, the changes are restricted to the local extraction and affect only those users accessing the data in the target database. Updates you make to a target database are not automatically applied to the source database by Replication Option.
- The extraction method lets you transfer the same subset of data to multiple targets, after which users of each target database may change the data to suit their individual needs. For example, you might modify a corporate price list to incorporate items local to each store. The price list for each target database then will reflect the local items. These changes will not be automatically applied to the source database.
- Data can be transferred from several source databases to a single target database using either a single extraction rollup transfer, or using one extraction transfer per source database. For example, with extraction rollup, you can select the New Hampshire employees from one database and the Massachusetts employees from a second database. Then you can roll up the two sets of employees into a single database called NORTHEAST PERSONNEL. Extraction rollup allows you to merge data from several like tables into one target table (or keep them separate if you so desire). The extraction rollup method can be used with any type of source database, but only with Rdb target databases at present. Alternatively, you can create separate transfers for each source database, using the same target database in each case. When you choose to use

independently executed transfers in this way, each must write data into different target tables unless you have the shared-write feature enabled. See Section 8.2.5 for more information on the shared-write feature.

- Execution of an extraction transfer can be a single, discrete event or an event that is automatically repeated at regular, scheduled intervals. How up-to-date the data in the extraction database is depends entirely on the frequency of the data transfer.

Figure 2–4 shows the PERSONNEL database again. Node B has identified a group of rows, all employees who live in New Hampshire, that are useful to applications it runs locally. Node C has selected a different set of rows from the central database, the employees who live in Massachusetts, that its applications routinely access. When you request that copies of these rows be placed in local databases of the respective nodes, those nodes do not need to rely on a continuous communications link with the central node in the network to retrieve the required data.

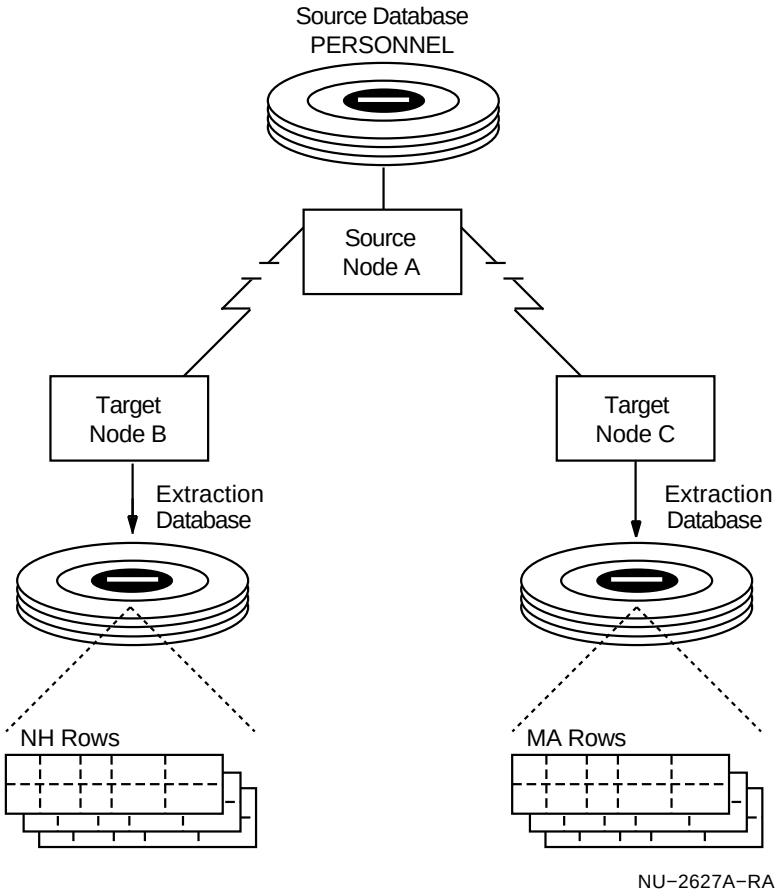
Figure 2–4 Remote Nodes Requiring Database Subsets



Two extraction databases result, one on Node B containing New Hampshire rows and the other on Node C containing Massachusetts rows. With these extractions, users can access the data relevant to their applications locally without accessing the source database.

Figure 2–5 shows the extraction databases that have been created at the remote nodes by the preceding transfer.

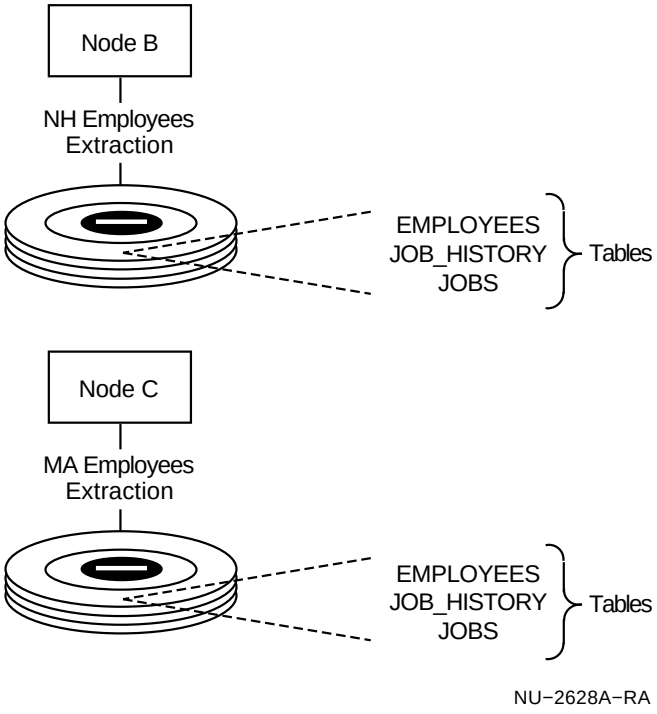
Figure 2–5 Extracted Data Subsets at Remote Nodes



In this example, each node receives an independent, locally owned database containing the same columns as the source database, but each containing a different subset of rows from the other. The network link between the remote nodes and the central node is now unnecessary. The database

applications on Node B and Node C no longer require the link once the transfer has successfully completed. The transfers of these two database subsets are independently defined. Each can have its own schedule for execution. Figure 2–6 shows the tables that make up the two independent target databases.

Figure 2–6 Extraction Databases

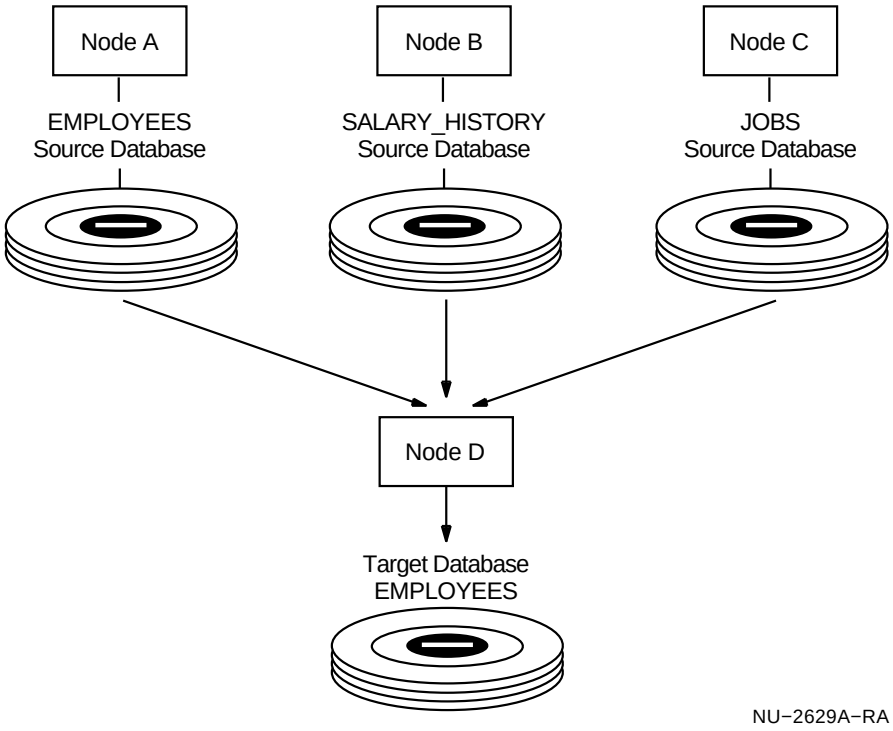


Once an extraction transfer has executed, the target tables can be used as the source data for additional transfers. For example, the location with the Massachusetts employee rows could divide those rows between two regional sites, one handling employees in the eastern part of the state and one handling employees in the western part. To achieve these results, either of the following options could be used:

- Create each subset as an extraction transfer from the original source.
- Create an intermediate extraction database that includes both subsets at the target. Then create additional extraction transfers using that target database as a data source.

One or more databases can function as source databases for an extraction rollup target. Figure 2–7 shows an extraction rollup database. In this example, three extraction databases, EMPLOYEES, SALARY_HISTORY, and JOBS are rolled up into a single extraction rollup target, EMPLOYEES.

Figure 2–7 Extraction Rollup



2.1.5.2 Replication Transfers

The first time a replication transfer is executed, the transfer behaves the same as an extraction transfer, except for minor details. The replication transfer method differs from the extraction method in one important respect. Each time an extraction transfer is executed, the entire set of data originally selected is sent again. This set reflects any changes that might have occurred to the data in the interim between transfers but also includes data that has not changed. By contrast, each time a replication transfer is re-executed only changes to the source data (inserts, updates and deletions) are forwarded to the target database. The implication is that replication transfers take less time to execute and consequently can be repeated more frequently than extractions.

Keeping track of what has changed in the source database since the previous execution of a transfer is done primarily by Rdb. The changes are recorded in system tables created in the source database by Replication Option. Rdb uses an internal triggering mechanism that not only captures the data changes (inserts, updates, deletions) but also records the transaction boundaries within which the changes take place. When replicating these changes to the target database, Replication Option does so in the same order as transactions were committed in the source database, thus preserving data integrity in the target database as each transaction is applied there. Because the recording of source changes is done by the database management system, it is available only in Rdb. The target of replication transfers, however, can be any of the databases supported by Replication Option as targets as shown in Table 1–1.

Replication Option supports two types of replication transfers:

- Standard replication transfers

  A standard replication transfer will result in an exact match between the data in the source database and the data transferred to the target database. All database inserts, updates, and deletions made in the source database are forwarded to the target database.

- Replication transfers with the NO DELETE attribute

  A replication transfer with the NO DELETE attribute will result in a target database that is an archive of all the data that is or was stored in the source database. All database inserts and updates made in the source database are forwarded to the target database. However, data deleted in the source database will not be deleted from the target. Instead, those deleted rows are retained in the target database and marked in a way to show they have been deleted.

  Replication transfers with the NO DELETE attribute are a special type of replication transfer; for information on creating and using a replication transfer with the NO DELETE attribute, see Section 8.2.3.

The rest of this section discusses only standard replication transfers.

Replication transfers use the following modes of operation:

- **Initialization**

  Initialization includes all the events that occur the first time a replication transfer runs, including any retries necessary to achieve success. The initial execution of a replication transfer is similar to an extraction transfer, with respect to the resulting target database; a replication transfer creates some indexes specifically for Replication Option that an extraction transfer does not.

- **Update**

  Replication transfers subsequent to the first execution (the replication initialization transfer) are called replication update transfers. A replication update modifies tables in the target database that are maintained by Replication Option to reflect any changes made to the source database since the last transfer.

- **Reinitialization**

  Reinitialization forces the replication transfer to perform its initialization steps rather than execute an update. Operationally, a reinitialization of a replication transfer to a new target is the same as executing an extraction transfer a second time. All the tables defined in the replication transfer definition are re-created in the target database from the current source database tables. Chapter 8 discusses what happens when a replication transfer to an existing database is reinitialized.

It is recommended that you use the replicated data for data retrieval only. If the target is a pre-existing database, one not created by Replication Option during the replication initialization phase, you may have local tables of your own that you can manage as you need. Your local tables can be written to; the Replication Option tables should be treated as read-only unless you enable shared writing. Also, if the transfer specifies the target database as new, rather than existing, it is recommended that you not create other tables of your own in the target database. The reasons are explained in the information that follows.

During a replication transfer if a mismatch is detected between what appears in the source database and what is found (or not found) in the target database, the transfer fails. To recover from this failure, you must reinitialize the transfer. Reinitialization causes the transfer to create a new set of files if the transfer created the original target database. Any tables that you added to the original target will not automatically appear in the re-created target. However, if the transfer is to a pre-existing database, reinitialization does not create a new set of target database files. In such a case, your locally created tables remain untouched. However, the data in the transferred tables is erased and replaced with a complete new set from the source database. Anything you might have written to the transferred tables will thereby be deleted.

You can choose an optional transfer schedule to determine when Replication Option forwards the changed rows to the replication database. By using several transfer definitions, data changes in a single source database can be copied to several target databases.

The benefits of extraction transfers listed in the preceding section also apply to replication transfers except where noted in the following list:

- The target database offers users locally accessible data with better response to data retrieval queries.
- Indexes can be added to transferred target tables to suit local data usage patterns.
- It is recommended that you do not modify the replicated data.
- Replicated target databases can be used as the sources of further transfers.
- Though there is no replication transfer method comparable to extraction rollup, as with simple extractions you can create several replication transfers, each referencing its own source database and all referencing the same target database, though different target tables unless you have shared writing enabled. If you enable shared writing, several replication transfers can target the same tables in the same target database. See Section 8.2.5 for more information on using the shared-write feature.

Replication transfers have these additional benefits:

- Forwarding of only inserted, updated, or deleted rows to the target

  After the replication tables are first created in the target database and the rows have been copied, subsequent replication transfers forward data only for rows that have been inserted, updated, or deleted in the source database, not the entire set of rows.
- Automatic updates to the target database

  The replication database maintains an association with the source database so that the replication can receive automatic updates at scheduled intervals. The currency of a replication database depends on the frequency with which updates to the source database are made and the frequency with which these updates are transferred to the replication database.
- Simplified database file management in target directories

  For replication transfers in which Replication Option is asked to create the database files in an OpenVMS system, Replication Option maintains one version of those files in the target directory. Unlike database extractions where a new version of the target database is created with each transfer operation, subsequent replication transfers continue to use the original target database files to store updates.
- Consistency between source database and replication database

If a failure occurs during an update transfer of changes to a replication database, all changes forwarded and committed up to the point of the failure are valid. Either the next time a retry is attempted or at the next regularly scheduled update to the replication, all remaining source changes will be forwarded.

2.2 Choosing a Method of Transfer

This section provides information on selecting the transfer method, extraction, extraction rollup, or replication, that best suits your database requirements. To select a transfer method, examine the following two aspects of your requirements:

- Business considerations

  Examine the needs of your business or, more specifically, the nature of your database tasks.

- Technical considerations

  Consider the requirements and impact that each method of transfer will have on your environment.

2.2.1 Classifying Your Database Tasks

The first step in considering how to use Replication Option is to examine your database tasks. If you have a central database with existing applications, consider the following benefits of duplicating tables from the central database to one or more target databases:

- Applications that once referenced the central database remotely, over a network link, could now reference their own local copy of the tables.
- You could move applications currently executing at the central database so that they execute on their own local copies of the data.

To determine which of your applications may benefit from such a change, it is useful to classify them in the following manner:

- Data origination tasks

  Applications that store and modify the original data must do so in the central source database.

- Database maintenance tasks

  Applications that maintain the characteristics of the data, the data definitions, and the physical structure and access methods used on the central database must remain in the source database. This is also true of applications that perform tasks such as database backup and recovery.

- Decision support tasks

  Applications that make only temporary changes to the database, or no changes at all, such as data analysis and decision support, can do so on copies of the original tables rather than on the original tables themselves.

- Application prototype tasks

  For purposes of prototypes and developing and testing new applications, having copies of the data tables in a separate database just for these purposes makes good sense. The data copies can also be used for ongoing applications maintenance and for training application designers and users.

- Query and report generation tasks

  Applications that only read data, such as those that perform queries and that generate reports, are good candidates for moving to target databases to avoid direct use of the central database.

Table 2–1 lists each database application type and indicates whether the application type could be used with data copied to a target database or whether the application must use data stored in the source database.

Table 2–1 Data Location Requirements

Application Type	Must Access Source?	Can Access Target?
Data origination	Yes	No
Database maintenance	Yes	No
Decision support	No	Yes
Prototype	No	Yes
Query	No	Yes
Report generation	No	Yes

Table 2–2 shows the application types and indicates which transfer methods you can use to copy data into a target database. The key factors in selecting the transfer method is whether or not the application type writes or modifies data in the transferred tables, or simply reads data.

Table 2–2 Application Type and Transfer Method

Application Type	Read/Write	Extraction	Extraction Rollup	Replication
Decision support	Write	Yes	Yes	Maybe[1]
	Read	Yes	Yes	Yes
Prototype	Write	Yes	Yes	Maybe[1]
	Read	Yes	Yes	Yes
Query	Read	Yes	Yes	Yes
Report generation	Read	Yes	Yes	Yes

[1]Yes, if shared writing is enabled. Otherwise, No.

Although Table 2–2 indicates that you can write to extraction and extraction rollup target tables and maybe write to replicated tables, you should consider the consequences of writing to Replication Option-created target tables. If you create target tables by performing an extraction transfer (or extraction rollup transfer) only once, then you can perform whatever operations you want on these tables. However, if you execute a transfer periodically and you do not have shared writing enabled, each execution of the transfer re-creates the database tables[1] and does not include any changes you might have written to the target tables. If you have shared writing enabled, only the transfer that wrote the data into the target tables can alter or delete that data when you execute extraction transfers after the first time. Data in the target tables that is owned by other transfers or that was inserted by customer applications will be unaffected.

2.2.2 Transfer Choices and Considerations

In addition to classifying your applications, you must take into account the characteristics of the various transfer methods. Each method has its own requirements and each can affect your operation in different ways.

[1] In the case of a replication transfer, the tables are recreated only after a transfer is reinitialized, not during regular update transfers.

2.2.2.1 Types of Source Database Used

Replication Option can use different types of databases as source databases. Table 2–3 shows which types of source database can be used for a particular transfer method.

Table 2–3 Source Database Types

Source Database Server	Extraction	Extraction Rollup	Replication
Rdb Transparent Gateway to Custom Drivers	Yes	Yes	No
Rdb Transparent Gateway to DB2	Yes	Yes	No
Rdb Transparent Gateway to Oracle CODASYL DBMS	Yes	Yes	No
Rdb Transparent Gateway to Oracle	Yes	Yes	No
Rdb Transparent Gateway to PC Data	Yes	Yes	No
Rdb Transparent Gateway to RMS	Yes	Yes	No
Rdb Transparent Gateway to Sybase	Yes	Yes	No
Distributed Option for Rdb[1]	Yes	Yes	No
Oracle Rdb	Yes	Yes	Yes

[1]Distributed Option for Rdb supports these transfers if the underlying gateway supports them.

Note that all source database types can participate in the same extraction rollup transfer.

Table 2–4 shows which types of target databases can be used for a particular transfer method.

Table 2–4 Target Database Types

Target Database Server	Extraction	Extraction Rollup	Replication
Rdb Transparent Gateway to DB2	Yes	No	Yes
Rdb Transparent Gateway to Oracle	Yes	No	Yes
Rdb Transparent Gateway to PC Data	Yes	No	Yes
Rdb Transparent Gateway to Sybase	Yes	No	Yes
Distributed Option	Yes	No	Yes
Oracle Rdb	Yes	Yes	Yes

Although the extraction rollup method of transfer can be used only if the target database is an Rdb database, you can perform an equivalent type of transfer to other kinds of databases using multiple extraction transfers. If you intend

for similar tables to roll up into a single target table, enable the shared writing option.

2.2.2.2 Location of Replication Option

Whether or not you have Replication Option installed at the site of the source database is a factor in the type of transfer you can select. Refer to Table 2–5.

Table 2–5 Types of Transfer Available

Replication Option Installed at Source Site?	Extraction	Extraction Rollup	Replication
Yes	Yes	Yes	Yes
No	Yes	Yes	No

2.2.2.3 Data Currency and Volume of Data Transferred

You should consider how current the data in the target database must be. If a transfer is re-executed frequently to maintain up-to-date data, generally you incur far less data traffic by using a replication transfer to copy only changed data from the source to the target. This reduced overhead applies only to updates to the target database, not for the initial replication transfer. Extracting all the data each time a transfer executes, as in extraction and extraction rollup transfers, inhibits fast transfer repetition rates.

2.2.2.4 Target Disk Storage Usage

The following section applies generally to all target database types. Though the table entries have some specifics geared to Rdb concepts, there are counterparts in the other target database systems. You should consider disk storage space limits at the target site. How much storage you will need depends on how much data you want to transfer and the type of transfer you use.

On OpenVMS, the disk and user account quotas limit available space.

Tables 2–6 and 2–7 show how different transfer types affect disk space at the target site.

Table 2–6 Storage Requirements for Transfers that Create a Target Database

Target Disk Storage Space Used	Extraction	Extraction Rollup	Initial Replication	Replication Update
Root file created	Yes	Yes	Yes	No
Snapshot file created	Yes	Yes	Yes	No
New files each transfer	Yes	Yes	Yes	No
Data definitions written	Yes	Yes	Yes	No
Data written	Yes	Yes	Yes	Yes
Before-image journal written	No	No	No	Yes
Snapshot file written	No	No	No	Yes
After-image journal written	No	No	No	Yes

Table 2–6 shows that for each execution of an extraction, extraction rollup, and replication initialization transfer, Replication Option creates a new set of target database files but does not write to a before-image journal file, an after-image journal file (.AIJ file), or a snapshot file. Subsequent executions of replication transfers, called updates, do the opposite. Replication Option does not create target databases enabling after-image journaling. However, you can enable journaling after the transfer completes.

Table 2–7 Storage Requirements for Transfers to an Existing Target Database

Target Disk Storage Space Used	Extraction	Extraction Rollup	Initial Replication	Replication Update
Root file created	No	No	No	No
Snapshot file created	No	No	No	No
New files each transfer	No	No	No	No
Data definitions written	Yes	Yes	Yes	No
Data written	Yes	Yes	Yes	Yes
Before-image journal written	Yes	Yes	Yes	Yes
Snapshot file written	Yes	Yes	Yes	Yes
After-image journal written	Yes	Yes	Yes	Yes

Although transfers to existing databases do not create the database files, they do write data to storage areas, to a before-image file, to a snapshot file, and to an .AIJ file (if enabled). Though many of these concepts are specific to Rdb databases, the issues involved have counterparts in other database systems. Extraction and initial replication transfers copy data for an entire table in one transaction. If the volume of data is large, the space needed in the before-image journal, the snapshot file, and the after-image journal file can be extensive. When extraction, extraction rollup, and replication initialization transfers are re-executed, Replication Option checks data definitions to see if

they need to be modified and if so, will modify them only if the target database and gateway products support modifying data definitions.

2.2.2.5 Source Disk Storage Usage

You should consider disk storage space limits on the system on which you have the source database. How much additional disk space you will need depends on what transfer method is used, what you choose to transfer, how often changes are made, and how often the transfer is re-executed.

Table 2–8 gives a broad indication of space usage requirements on the source database disk. Replication transfers can significantly increase the size of a source database if the frequency of data changes is high and the transfer repetition rate is low.

Table 2–8 Storage Requirements for Source Databases

Source Disk Storage Space Used	Extraction	Extraction Rollup	Replication
Changes to rows in log table	No	No	Yes

2.2.2.6 Source Database Transaction Rate

You should consider the transaction rate your source database must sustain. Can you absorb a decrease in this rate in order to support transfers? Replication transfers decrease the rate at which data changes can be applied to a source database because they require data rows to be written twice (once to the source database and once to the logging table, RDB$CHANGES, that captures these changes to apply them to the target in the next transfer). By contrast, extraction and extraction rollup transfers use the source database for reading only, which puts less of a load on available disk and CPU cycles.

2.2.2.7 Source Database Availability

You should consider the availability requirements of your source database. Do you require your source database to be available 24 hours per day? Continuously operating database servers may be useful in such an environment. A 24-hour-per-day availability requirement can conflict with the operation of replication transfers. When a replication transfer first executes (replication initialization), Replication Option requires exclusive access to the source database to perform the following tasks:

- Add special system tables
- Record portions of the transfer definition

Extraction and extraction rollup transfers do not require exclusive access and do not interfere with servers because they read information only from the source database.

3

Components of Replication Option

Replication Option consists of several components. This chapter describes each component in detail and explains how Replication Option uses each component to distribute data in a network.

3.1 Overview

To transfer data from a source database to a target database using Replication Option, the following steps must take place:

1. A user or SQL application issues a transfer and (optionally) a schedule definition using SQL.
2. The **transfer monitor** processes the definitions by storing or manipulating information in the **transfer database**.
3. The transfer monitor starts a **copy process** to execute the transfer when either of the following events occurs:
   - A user or application issues a START TRANSFER NOW statement
   - A user or application creates a schedule definition and the time specified in the definition arrives
4. The copy process retrieves information about the transfer from the transfer database and copies data definitions and data from the **source database** to the **target database**.

Figure 3–1 illustrates the steps in this process.

Figure 3–1 Data Transfer with Replication Option Components

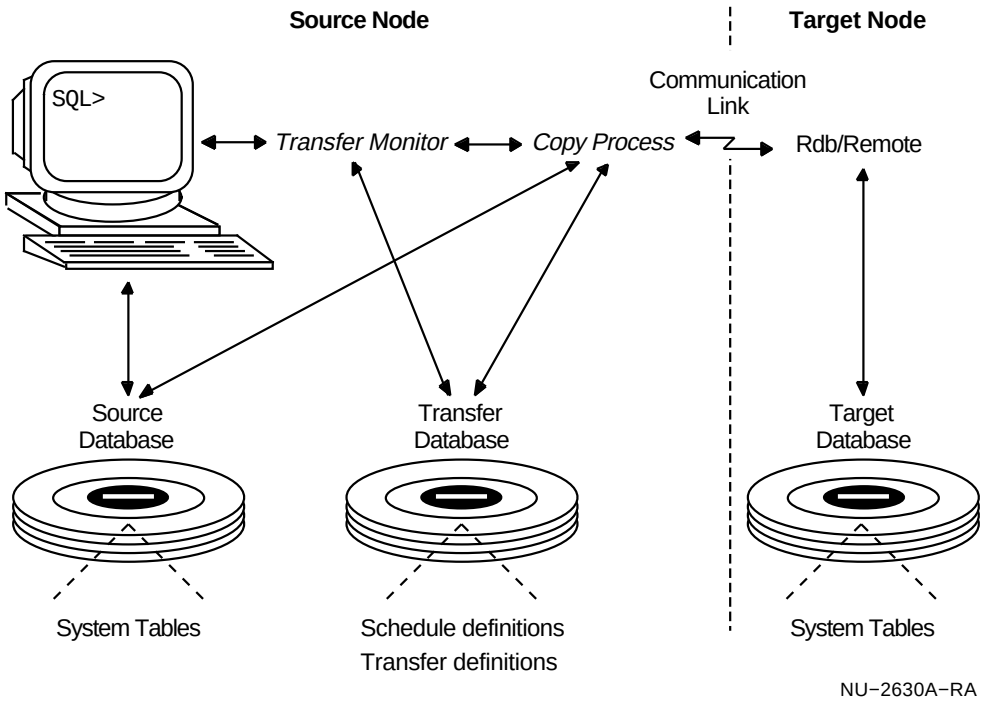


Table 3–1 briefly describes each component. The remaining sections in this chapter describe the components in greater detail.

Table 3–1 Main Components of Replication Option

Main Component	Description
SQL	SQL, packaged and shipped with Rdb and/or the gateway products, is the user interface to Replication Option. Users enter interactive SQL statements or write SQL applications to perform Replication Option functions.
Source database	One or more databases that provide the data definitions and data values Replication Option retrieves and transfers to target databases during extraction, extraction rollup, or replication transfers.

(continued on next page)

Table 3–1 (Cont.) Main Components of Replication Option

Main Component	Description
Target database	A database that receives the table definitions and data from a source database or databases. Replication Option can create the target database during the transfer or copy data into an existing database.
Transfer monitor	The control program that oversees all operations provided by Replication Option on a particular CPU. It starts copy processes, answers user requests, gathers and stores information in the transfer database, and enforces schedules.
Transfer database	A Replication Option-created Rdb database or the Replication Option-created tables for Oracle databases that store transfer and schedule definitions and other information needed for data distribution operations in a network. In the case of Oracle databases, Replication Option does not create the database files; it only creates the tables. A single transfer database is used when Replication Option is installed on a standalone system or in a VMScluster system.
Transfer definition	A definition created with the CREATE TRANSFER statement of SQL. A transfer definition specifies the source and target databases and the data to be transferred.
Schedule definition	A definition entered with the SQL CREATE SCHEDULE statement. The schedule definition specifies when a particular transfer is executed.
Copy process	A program, run by the transfer monitor, that transfers data from source to target database.
System tables	Special-purpose tables created by Replication Option in a source or target database. Tables in source databases are used to record and manage data changes that will be replicated to target databases. Tables in target databases are used to track the progress of transfers and to record data ownership.

3.2 Using the SQL Interfaces

To create and manipulate transfers, you use the interactive SQL interface provided with Rdb and with Distributed Option for Rdb and the Rdb Transparent Gateway products. Or you can write an SQL application using SQL module language, precompiled SQL, or dynamic SQL. SQL recognizes all Replication Option statements including those used to:

- Create, modify, and delete transfers and their schedules (CREATE TRANSFER, CREATE SCHEDULE, ALTER TRANSFER, ALTER

SCHEDULE, DROP TRANSFER, DROP SCHEDULE, and SET LOGICAL_ NAME TRANSLATION FOR TRANSFER).

- Manage the operation of transfers (START TRANSFER, STOP TRANSFER, and REINITIALIZE TRANSFER).
- Display information about transfers (SHOW TRANSFER) from the interactive SQL interface. Note that the SHOW TRANSFER statement cannot be issued within an application.
- Grant and revoke privileges to alter, drop, or execute a transfer (GRANT and REVOKE).

Most examples in this manual use interactive SQL. See Appendix D for an example using dynamic SQL and SQL module language programs.

3.2.1 Using Oracle SQL/Services to Query a Target Database

Replication Option for Rdb can transfer data to certain database management systems on PC and workstation platforms. In such case, the database is local to the PC or workstation itself; and it can be accessed directly by customer applications or by tools native to the local database system.

In other cases, the data must be transferred to a database system on a server machine, and then accessed from the PC or workstation using Oracle SQL/Services. Oracle SQL/Services is a client/server system. For example, an application program executing on a Microsoft Windows 95 platform can issue SQL statements through Oracle SQL/Services to access data that has been transferred into an Rdb target database on a Digital Equipment Corporation Alpha machine by a Replication Option transfer. For further information about Oracle SQL/Services, refer to the Oracle Rdb documentation set.

3.2.2 Using SQL Applications to Issue Replication Option Statements

Replication Option SQL statements can be embedded in the SQL module language or the precompiled SQL programming interfaces. Both of these programming interfaces only accept Replication Option statements in dynamic form (requiring run-time compilation).

An example of using dynamic SQL and SQL module language to create and control the execution of transfers is provided in Appendix D.

3.3 How Rdb or Distributed Option Software Options Affect Use of Replication Option

Note

The information in this section regarding software options is only applicable if you are using Oracle Rdb Version 6.1.

You can execute all Replication Option functions at a site that has installed the Rdb or Distributed Option full-development kit, or an Rdb or Distributed Option interactive kit, or an Rdb or Distributed Option run-time only kit.

You can use Replication Option to create a database at a target location that has installed the Rdb or Distributed Option full development, interactive, or run-time only (RTO) software.

An RTO environment has restrictions that can affect how you create a target database. A typical configuration in which you might use Replication Option with the RTO option is a cluster with the following characteristics:

- A central production computer with Replication Option and a full development Rdb kit installed
- Satellite department processors with an Rdb RTO kit installed on each one

In this configuration, extractions and replications transfer data from a central database to the satellite processors. However, in creating the target database, Replication Option does not transfer indexes or constraints. Furthermore, the RDO utility available with the Rdb run-time-only license on the target does not support the creation of indexes and constraints.

If you require indexes or constraints created by RDO on your RTO target databases, you should select one of the following options:

- Install Rdb full development or interactive kits on the satellite processors. An Rdb full development or interactive kit enables you to use RDO procedures to define indexes and constraints in the target databases.
- Develop programs using the Rdb preprocessors on the central system with an Rdb full development kit. Then, copy these programs to the run-time only target locations and execute them there to create indexes and constraints.
- Include as part of your transfer definition an epilogue command procedure that defines indexes or constraints.

Chapter 7 contains information on using epilogue command procedures to define indexes on OpenVMS.

Using SQL, you can create indexes and constraints in the full development, interactive, or run-time only environments.

3.4 Source Database

The source database can consist of a single central database. For an extraction rollup transfer, there can be one or more databases that are used as source databases. The source database provides the data definitions and data values Replication Option retrieves and transfers to target databases. Target databases can be located on the same system as the source database or on hosts/nodes different from the source database.

The host/node on which the source database resides is called the **source node**. All database updates in your information management system are performed on the source database. See Table 1–1 for a list of source databases.

You can have more than one source database on each host/node in a network. Using different transfer definitions, you can transfer data from a single source database to many target databases. The same source database can provide data to both extraction and replication databases, and several source databases can participate in an extraction rollup transfer. In fact, both extraction and replication databases can serve as a source database for further transfers.

3.5 Target Database

A target database is a database in which Replication Option creates table definitions and into which Replication Option copies data; definitions and data are derived from source databases. You can use Replication Option to create the target database files as part of the transfer process, or to reference a target database that pre-exists. Target databases are called extraction, extraction rollup, or replication databases depending on the method used for the transfer. Replication Option can use the databases listed in Table 1–1 for its target databases.

The target database can be located on the same node as the source database or on a remote node. Several target databases can receive data from the same source database. In an extraction rollup transfer, several source databases are read to create one target database.

There are a number of database objects or attributes that are not explicitly transferable by Replication Option. In an SQL CREATE TRANSFER statement, you can name tables and views to be copied by Replication Option. The CREATE TRANSFER statement has no explicit provision for

identifying objects such as indexes, constraints, triggers, query outlines, or stored procedures. Nor does Replication Option duplicate access rights to the database objects. However, transfer definitions may name external command procedures. These command procedures are executed before and after the Replication Option transfer proper is done. These procedures can be used to add target database objects not explicitly transferred by Replication Option.

See Chapter 7 for more information on writing command procedures on OpenVMS.

The following is specific to Rdb source and target databases:

- Multifile databases

  You can use a multifile database as a source database for replication, extraction, and extraction rollup transfers. However, you cannot specify storage areas in a CREATE TRANSFER statement as you can in a CREATE DATABASE statement. The resulting target database will be a single-file database. You can convert the target database to a multifile database by using the SQL IMPORT and EXPORT statements.

- Multischema databases

  You can use a multischema database as a source for Replication Option transfers. The tables that are transferred by Replication Option become part of the default catalog and default schema. However, catalog and schema information are not preserved during the transfer. When Replication Option creates the target database, it creates a non-multischema database. You can alter the target database to make it a multischema database.

3.6 Transfer Monitor

The transfer monitor oversees all functions provided by Replication Option:

- Coordinating all operations on a particular host/node
- Starting copy processes on demand or by schedule
- Responding to user requests
- Recording the status of transfers in the transfer database
- Writing entries in a transfer monitor log file about such activities as copy process events and transfer failures

The transfer monitor adheres to the following rules for single-node systems and for systems in a cluster configuration:

- Single-node system

Single-node systems need one transfer monitor. A single transfer monitor on each node controls all Replication Option operations concerned with transfer definitions stored at that node.

- VMScluster system

  When Replication Option is used on more than one node in a cluster environment, each node requires a transfer monitor. If a transfer is being executed on a particular node in a cluster and that node fails, the transfer can be re-executed by a transfer monitor on another node in the cluster. In this event, you need to restart your transfer using the START TRANSFER statement. Active transfer monitors remaining in the cluster assume control of all current Replication Option activities occurring on the cluster. The transfer monitor log provides a record of changes in a cluster configuration.

When a transfer finishes, the transfer monitor updates status information stored in the transfer database. If a transfer is successful, the transfer monitor also updates the latest completion time of the transfer. If a schedule is defined for the transfer, the transfer monitor calculates the time needed to execute the transfer again.

3.7 Transfer Database

The transfer database can be either an Rdb or an Oracle database created by Replication Option to store information necessary to carry out data distribution operations in the network. See Section C.1.1 for information on the command procedures used to create transfer databases.

If the transfer database is not an Rdb database, the use of a cluster lock file is required when you start the transfer monitor. See Section C.1.2 for more information on lock files.

Among items stored in the transfer database are:

- Transfer definitions, including the name of the definer, the type of transfer, the names of the source and target databases, and the selection criteria in the form of select expressions
- Schedule definitions, including the next computed time to transfer and the number of times that Replication Option should retry a transfer that fails
- Current status of a transfer, the time of its last successful execution, and the current state of the transfer
- Security information in the form of the access control entries (ACEs) and user identification codes (UICs) associated with each transfer definition

Appendix A lists the names and descriptions of the tables and columns defined in the transfer database.

In Replication Option, it makes a difference on which node you define transfers—source node or target node:

- Extraction or extraction rollup transfers can be defined at the source node, the destination node, or a third node, one from which both the source and target databases are remote.
- Replications can be defined only at the source node.

You define the location of the transfer database when Replication Option is installed.

Replication Option startup procedures create the system logical name DDAL_TRANSFER_DATABASE which points to the location of the transfer database. In addition, a DDAL$TRANSFER_DATABASE logical name is created that points to a lock file.

3.8 Transfer and Schedule Definitions

A **transfer definition** includes the source database names, selection expressions, the identification of the target database that will receive the data, optional prologue or epilogue command procedures, and physical attributes of the target database. A **schedule definition** specifies the times and intervals when transfers automatically execute. Creating transfer definitions is a required task in Replication Option; creating schedules is optional. Replication Option stores transfer definitions and schedule definitions in the transfer database.

Users of target databases can request copies of a subset of the rows or columns in the source database. Each may request a different subset of the source database. You can negotiate the selection criteria with target sites to determine what rows in which tables they require. By assembling sets of conditions into individual selection expressions that identify subsets of the source database, you can then define transfers that move copies of those rows to target databases.

You can create several transfer definitions referencing the same source database. Each transfer definition uses its own schedule for transferring the selected rows and columns and its own target database specification for storing them.

Although you are the owner of any transfer and schedule definition you create, you can allow others access to your transfer and schedule definitions by granting them privileges to alter, drop, or display your definitions. Likewise, you can also revoke these privileges at a later time should you desire.

3.9 Copy Process

The copy process retrieves information from the transfer database in order to initiate and execute transfers. Starting a transfer either on demand or by schedule sets off a series of events. By spawning a detached process, the transfer monitor starts a copy process to initiate and execute transfers. The copy process:

1. Executes the login command file (LOGIN.COM) associated with the OpenVMS account of the person who created the transfer. This establishes any logical name definitions that are private to the person's account and that might be referenced within the transfer definition.
2. Executes a prologue command procedure (optional). Chapter 7 contains information on prologue command procedures.
3. Obtains from the transfer monitor the transfer database name, the transfer name, and the transfer method.
4. Retrieves the transfer definition from the transfer database.
5. Attaches to the source database or databases. If these are on remote systems, the remote Rdb server account provides the connection mechanism for Replication Option. Information about accessing remote databases appears in Section 4.8. This same information is applicable for access to a remote target database. For initial replication transfers, the copy process creates special system tables in the source database (Appendix A describes these tables).
6. Reads information from each source database in a single read-only transaction to ensure data consistency for an extraction, extraction rollup, or initial replication transfer.
7. Attaches to or creates the target database, as directed in the transfer definition.
8. Creates the tables named in the transfer definition and copies all data identified by the select expression specified for those tables. A separate transaction is started for the creation of each table in the target database.

9. Retrieves stored changes to the source database from a special-purpose system table and transfers them to the target database for replication updates. After update rows stored in that special-purpose system table are no longer needed by any transfer, they are automatically purged from that table by the copy process.
10. Throughout the transfer operation, records its operations and status messages in the copy process log file specified in the transfer definition.
11. Executes an epilogue command procedure (optional). Chapter 7 describes epilogue command procedures.

The copy process can execute in a nonclustered configuration or in a VMScluster system.

- If you define a transfer from one host/node to another in a nonclustered configuration, the copy process executes on the host/node on which you define the transfer and on which the transfer monitor is running.
- If the transfer is triggered by a schedule you define to execute in a cluster configuration, the copy process can execute on any host/node in the cluster on which a transfer monitor is running. Transfer monitors automatically handle the selection of a host/node on which the copy process will execute. A single transfer database is shared by all hosts/nodes in a cluster.

3.10 Special Replication Option System Tables for Replications

The first time a copy process starts a replication transfer, Replication Option creates a set of special-purpose system tables in the source database. These tables are:

RDB$CHANGES

RDB$TRANSFERS

RDB$TRANSFER_RELATIONS

RDB$CHANGES_MAX_TSER

Inserts, updates, and deletion of rows in source tables involved in replication transfers are stored in these system tables. Whenever an insert, update, or deletion of a row occurs in a source table that matches a replication select expression, Rdb stores the information in the RDB$CHANGES table. This table is managed by both Rdb, which writes into this table, and Replication Option, which reads and deletes from this table. The copy process uses this

table to copy data rows to target databases or to determine which target rows to delete (or in the case of a replication transfer with the NO DELETE attribute, which target rows to flag as having been deleted from the source database).

By default, these special system tables are created in the RDB$SYSTEM storage area. You can improve performance by moving RDB$CHANGES and any or all of the other three special system tables to their own storage areas on a separate physical disk.

Take the following steps to move the RDB$CHANGES table to its own area. Similar steps can be taken for the other three system tables.

1. If necessary, alter the source database to reserve space for additional storage areas and commit.
2. Alter the source database to add a storage area, such as RDB_CHANGES_ AREA, and commit.
3. Create a storage map for RDB$CHANGES to map it to the RDB$SYSTEM storage area and commit. This is a required step. Although RDB$CHANGES, when first created resides in RDB$SYSTEM, it does not have an explicit storage map until this step is performed.
4. Alter the RDB$CHANGES storage map to map the table to the new storage area, RDB_CHANGES_AREA. Commit.

Moving Replication Option system tables to their own storage area also has the following benefits:

- The RDB$CHANGES table can be managed on its own without interference from other system tables and I/O operations.
- You can use a large page size (4, 8, or larger) to reduce the amount of I/O for active transactions. This means that fewer RDB$CHANGES rows are written when there are many (or large) rows changed during a transaction.
- It is easier to use tools such as ALTER STORAGE MAP ... REORGANIZE and RMU Repair when only the RDB$CHANGES table needs to be managed.
- The storage map allows the use of a THRESHOLDS clause, and the DBA can choose area attributes such as MIXED or UNIFORM to improve page selection.

The map is restricted in many areas. For example, there is no partitioning and the indexes cannot be mapped to other areas. See the Rdb Release Notes for more complete information about moving system tables.

Note

Rdb databases do not normally contain any of the special-purpose system tables required by Replication Option for replications. All special system tables are created in the Rdb source database only after the copy process performs a replication transfer for the first time. Appendix A contains information on special-purpose system tables.

4
Preparations for Distributing Data

Before you use Replication Option to create and execute transfers, you must be sure that your operating environment is appropriate both for Replication Option and your applications. Also, you must check the compatibility of software products you have installed at local and remote sites.

Because Replication Option creates copies of source tables in target databases, you need to consider how to use existing applications with those copies or how to develop new applications for them. One consideration is how to make data definitions available for developing or modifying the applications. Either a data dictionary or a local database copy may be appropriate.

The definition and execution of Replication Option transfers generally involve multiple sites. You must have accounts with proper privileges and, for OpenVMS systems, resource quotas, as well as privileges to access the various databases on CPUs at these sites. Establishing the accounts and privileges may involve negotiating with system managers and database administrators of other systems.

Using Replication Option requires additional steps in your normal database operation and maintenance procedures. For example, in some situations, you may need to suspend Replication Option transfers while you perform database maintenance. In other situations, you may need to change transfer definitions to coordinate them with data definition changes in your databases.

This chapter provides information on the following topics:

- Product version compatibility among independently managed sites
- Using Replication Option in a multiversion Rdb environment
- Requirements for locating Replication Option at various sites
- Moving applications to target sites, with emphasis on data definition considerations
- Preparations at the transfer definition site, such as tailoring your login procedure and defining environment variables

- Preparations at the source sites, including what database access privileges you need; what to transfer, when, and how often; proxy accounts; and privileges and quotas
- Preparations at the target sites, including identifying the database administrator, establishing proxy accounts, and determining privileges and quotas
- Preparations for database access at each node in a cluster
- Steps for tuning network parameters that can alleviate transfer failure due to heavy network traffic
- Changes to your normal operations and maintenance procedures, including database backup, recovery, data definition changes, and database structure changes

4.1 Determining Product Compatibility

Before you attempt to define transfers, it is important to determine that the versions of Replication Option and related products installed at your various sites are compatible.

On a local system, whether it is a standalone node or a cluster, the system manager ensures that products function together at the time of installation. Thereafter, you generally do not have to worry about product compatibility.

Replication Option, however, works with more than one system in a network environment. These systems may be independently managed. Consequently, the versions of Replication Option and related products on your system might differ from the versions installed on other systems. Even if the same people manage all the systems, it often is not possible to convert the systems to the latest software at the same time. Therefore, the question of product compatibility among multiple sites must be considered.

Consider the following two cases where you have a product from a previous release and one from a release before that:

- Generally, if another system has the previously released version of a related product, such as Rdb, Replication Option should be able to perform transfers to and from such a system. If restrictions prevent such transfers, those restrictions will be listed in the Replication Option for Rdb release notes.
- If the product versions vary by more than a single release, Replication Option may not succeed in performing data transfers. Before you define transfers you should know what product versions exist at the remote sites.

You should also know what set of a product's capabilities exist at a given site. If a site does not have the product's full capabilities, the use of Replication Option may be restricted. Such a situation exists when Replication Option is used with the Rdb or Distributed Option for Rdb run-time only option (version 6.1 or earlier) and the data dictionary at a target site. Refer to Chapters 5 and 6 for information on this configuration with extraction, extraction rollup, and replication databases.

4.2 Using Replication Option in a Multiversion Rdb Environment

Note

This section only applies if you are using Rdb and Replication Option on the same system, and Rdb manages the transfer database.

Rdb Version 6.0 and higher may coexist with other versions of Rdb on the same processor. This is referred to as the multiversion capability of Rdb. This was developed to allow users to migrate applications and databases gradually from one version of Rdb to the next.

Replication Option does not support multiple versions of Replication Option on the same processor or on the same cluster. However, it is possible to use a version of Replication Option in an environment that has more than one version of Rdb.

In order to do so, you must set the version of Rdb to correspond to the version of Replication Option you are using. Table 4–1 shows the versions of Replication Option and the versions of Rdb with which it can operate.

Table 4–1 Correspondence Between Replication Option and Rdb Versions

Replication Option	Rdb
6.0	6.0 or higher
7.0	6.1 or higher

There are two ways in which the Rdb version can be established for Replication Option. The method used depends on how the Replication Option monitor is started. For details about starting the Replication Option monitor, known as the transfer monitor, see Section C.1.2.

For compatibility with earlier versions of Replication Option, the transfer monitor can be started without specifying a version number for Rdb. When this is done, the default system version of Rdb is used. Prior to starting the transfer monitor, you must ensure that the system version of Rdb is properly defined. For example, to set the default system version of Rdb to 6.1, enter the following command:

```
$ @SYS$LIBRARY:DECRDB$SETVER.COM 6.1 /SYSTEM
```

Also, when starting the transfer monitor using this method, you must ensure that your current process environment is set for the same version of Rdb.

The second method of starting the transfer monitor is used when you specify the Rdb version as one of the startup parameters. When this is done, a different method of starting the transfer monitor is used. The version of Rdb that you specify in this way is not required to be defined as the system version.

Regarding the setting of the proper Rdb version, also note the following:

1. If you are a system manager or database administrator and are executing one of the Replication Option command procedures, make sure that your interactive environment is set for the appropriate Rdb version. Examples of such procedures are the Replication Option startup and shutdown files and the transfer database creation and conversion files.

   If you submit these procedures as batch processing jobs, make certain that the accounts under which these jobs are run are set to the appropriate Rdb version also.

2. The Replication Option copy process performs data transfers on behalf of the person who creates the transfer definition. Make sure that when the copy process executes, the account has the appropriate Rdb environment set.

4.3 Requirements for Locating Replication Option at Various Sites

You do not always have to install Replication Option at each site involved in a transfer. Depending on the data transfer method you choose, restrictions affect where Replication Option software must be installed and run. For example, if site A has Replication Option installed, you can transfer data from site B to site C even if neither site B nor C has Replication Option installed. Restrictions to extraction, extraction rollup, and replication transfers are discussed in Chapters 5 and 6.

When the copy process runs on the source node, data extracted from that source is said to be **pushed** to the remote target node. When the copy process runs on the target node and must copy data from a remote source database, the extracted data is said to be **pulled**.

Replication transfers always push data because the privileges required on a remote source database to access data cannot be granted over the network.

In general, the pull method is more efficient than the push method.

For more information on locations where Replication Option is installed, see Section 2.2.2.2.

4.4 Moving Applications to Target Sites

The potential benefits of making copies of your central database tables available at remote sites are discussed in Chapter 2. Moving data is just one consideration in using Replication Option. Moving applications that use the data is another. Typically, you will need to make a few changes to those applications because the data is in a different environment. Chapter 6 discusses the changes that you may need to make to your applications when you move them to target sites.

If you plan to maintain these applications or to develop new ones at the target sites, you will want to make the database definitions available at those sites by using one of the following methods:

- Use a local data dictionary.
- Transfer empty databases to the target site. (**Empty databases** have table definitions but no stored data.)

4.4.1 Using a Data Dictionary with Replication Option

When you access target databases with products such as Rdb, you may wish to maintain consistency between the data definitions stored in the system tables of the database and the data definitions stored in the data dictionary. To ensure this consistency, you use the SQL INTEGRATE statement to copy database definitions to the data dictionary.

Replication Option does not require the use of a data dictionary. By using the DICTIONARY IS REQUIRED clause of the SQL ALTER DATABASE statement, however, you can require the use of a data dictionary path name for data definition operations on both source and target databases. For more information on the SQL ALTER DATABASE statement, refer to the Oracle Rdb SQL Reference Manual for the version of Oracle Rdb you are using.

When the copy process creates data definitions in a target database, none of the definitions for that target database is stored in the data dictionary on the target node.

To change database definitions in a target database, you should invoke the database by specifying its database dictionary path name.

When the copy process creates a new version of the target database, the definitions you added become inconsistent with those in the new version. This means that the data dictionary contains definitions that are not yet in the new version of the target database.

See Section 7.3.5 for information on using the SQL INTEGRATE statement to copy database definitions to the data dictionary and using an epilogue command procedure to integrate the most current data definitions into the target data dictionary.

4.4.2 Using Transfer Definitions to Provide Remote Copies of Database Definitions

If your organization uses the data dictionary for database definitions, you can refer to the source database definitions stored in the data dictionary to ensure consistency between the source and target databases. However, when the data dictionary is not available, you can still provide a consistent set of database definitions to target applications. You can use transfers to provide data definitions in target databases without also having to transfer data. You can establish the target tables before you need the actual data. Application programmers can design and build programs using the database definitions for precompilation. Then, when data becomes available, programs are ready for testing.

To create target tables without data, create a transfer definition and specify the tables you want to transfer. The procedure for doing this is described in subsequent chapters. Prior to executing the transfer, define the environment variable, DDAL$EXTRACT_DATA, to inhibit the transfer of data for your particular transfer. Defining environment variables is described in Appendix B, and DDAL$EXTRACT_DATA in particular, is described in Section B.9. In this way, you can ensure consistency among all copies of the source database definitions. Programmers located at the target nodes can use these definitions to develop programs. Once the applications at the target site are ready to receive actual data, you must delete the environment variable or redefine it to permit data transfer.

You can also create a replication database in a target directory from a source database that contains no data yet. Replication Option will forward the selected data rows to the replication database after they are stored in the source database.

4.5 Preparations at the Transfer Definition Site

When you define a transfer, Replication Option associates your OpenVMS user identification code (UIC) and account name with that transfer. When the transfer is due to be executed, Replication Option creates an OpenVMS process under your account name. The copy process image runs within that process.

The copy process runs on your behalf with whatever privileges, quotas, and defaults your account affords. As a transfer definer, you automatically have the privileges needed to run, alter, or drop the transfer. You can also grant these privileges to other users so that they can run, alter, or drop a transfer you have defined (including the schedule definition). See Section 9.7 for more information on granting privileges.

If the situation changes after granting privileges to another user such that you no longer want that user to have transfer or schedule definition privileges, you can revoke their privileges. See Section 9.10 for more information on revoking privileges.

Note however, that transfer and schedule definition privileges are not the same as privileges to access the files in the source and target databases. The copy process must have the appropriate privileges to access the source and target databases for a transfer to run successfully. See Section 4.6 for more information on privileges needed to access the source database. See Section 4.7 for more information on privileges needed to access the target database.

You might want to tailor your LOGIN.COM procedure to accommodate the execution of the copy process. Refer to Section 6.4 for information on how to do this.

4.6 Preparations at the Source Sites

If your source is an Rdb database, you need to consider the following items before creating a transfer. If your source is accessed through an Rdb Transparent Gateway, you need to consider these items, and also refer to the gateway's documentation for any additional considerations.

- What database privileges are needed?
- What subset of the database to transfer?
- When to start the transfer?

- How often to repeat the transfer?
- What operating system accounts at the source sites will be used for the data transfer?
- What privileges and, for OpenVMS systems, what quotas are needed for those source accounts?

When you define a transfer using SQL, SQL looks up the data definitions in the source database and, if it is an Rdb database, verifies that you have the necessary access rights. In particular, you must have SELECT access to all tables and views you wish to transfer. Additionally, if you are defining a replication transfer, you must have both SELECT and CREATE privileges on the source database. These additional privileges are required on replication transfers for two reasons:

- Replication transfers can affect update and retrieval performance at the source database and can affect disk space utilization. Therefore, granting a user permission to create replication transfers should be carefully considered.
- The copy process may have to create the Replication Option system tables in the source database.

You may already have these privileges on a local database. However, to define an extraction rollup transfer, you will need privileges on all of the source databases. In this case, you will need to obtain the necessary privileges for your account on each of the other source databases.

You need to decide whether to transfer the entire database or a subset, what method of transfer to use, when to begin performing the transfer, and how often to repeat it. To help you decide which transfer method to use, refer to Chapter 2.

At each source site, you need to use an operating system account for database operations performed during data transfer. If the source database or gateway is located at a remote site, you can gain access to an account in several ways:

- Enable a proxy account.
- Specify an access control string.
- Use the default RDB$REMOTE account.

Refer to Section 4.8 for details about remote access and security considerations. If one of the Rdb Transparent Gateway products manages the database, refer to the documentation for that particular product for details about setting up an account on your system.

4.7 Preparations at the Target Site

At the target site, where there can be a native database, such as Rdb, or an Rdb Transparent Gateway product, you need to consider the following items before creating a transfer:

- Who will administer the target database?
- What operating system account at the target site will you use for the data transfer?
- What privileges do you need in the target database?
- What privileges and, for OpenVMS, what quotas do you need for the target account?

If someone at the target site is assigned to manage the target database, that person must have access privileges to the target database equal to those of the database owner. This person may also grant access rights for other database users.

At the target site, you need to use an account for database operations performed during data transfer. If the target tables are created at a remote OpenVMS site or by means of an Rdb Transparent Gateway at a remote OpenVMS site, you can select from several methods to provide access to this account. Refer to Section 4.8 for information about accounts used for remote access.

Regardless of what type of account is used at the target site, the account will need process privileges and, for OpenVMS, quotas that meet the minimum requirements of the target system or transparent gateway. Refer to the documentation for the target database product or gateway product for these requirements.

This account must also have read/write access privileges for the target disk directory if Replication Option needs to create the target database. To accommodate the target database, you should determine if the target disk has sufficient free space. If the transfer will be repeated, the replication target database can grow larger; extraction and extraction rollup transfers may create new databases each time. You and the person in charge of the target database should plan how to manage the disk space.

If the transfer creates its own target database, the copy process performing the transfer is given unlimited database privileges. If the transfer is to an existing database, you must request the appropriate privileges from the owner or administrator of the target database. The minimum required privileges include:

- SELECT
- CREATE
- DROP
- INSERT
- UPDATE
- DELETE
- ALTER

4.8 Security and Environment Considerations for Remote Access

Transfers involve three or more databases: one or more source databases, a target database, and a transfer database. The transfer database is located on the transfer system, the system on which Replication Option is installed and run. The source and target databases can be located on the transfer system, but typically one or more of them will be on a remote system (the source or target system).

Replication Option can access remote Rdb databases, remote Distributed Option databases, and remote Rdb Transparent Gateway databases. These exist on OpenVMS or Digital UNIX systems. The Rdb Transparent Gateways, in turn, provide access to a wide variety of database types, not limited to OpenVMS and Digital UNIX. When you access a remote Rdb or Distributed Option database or a remote Rdb Transparent Gateway database, an Rdb server process is created on the remote system to act on your behalf. A following section discusses the remote server account on the remote system into which Oracle Rdb logs in to establish the Rdb server process.

To gain access to a database, you must provide, either explicitly or implicitly, a valid user name and password. This is used for user authentication and determines your access privileges to the database system.

In a distributed environment, it is common for different user accounts to be used on these different systems, rather than a single account with the same user name, password, and login characteristics. Consider the following example. Replication Option is installed on system REPOPT, the source database on system SOURCE, and the target database on system TARGET. The transfer might be created and executed on system REPOPT under user account SYSTEM. Access to the source database on system SOURCE, done by a server process on that system, could be under user account PROD. Similarly, access to the target database on system TARGET might be done by a server process running under the user account MIS. The point to be made is this:

when you create a transfer definition, you must decide under what user accounts the source and target databases will be accessed. Account privileges, available resources (for example, OpenVMS quotas), and environment (for example, a database variant) are important considerations. How you identify which accounts to use is the subject of this section.

The only type of database system that Replication Option can access directly is Rdb. All other databases must be accessed indirectly. Indirect access can be done using one of the Rdb Transparent Gateways. It can also be done using a Distributed Option database. Distributed Option uses Rdb Transparent Gateways to access the actual underlying database systems.

A transfer is defined using the SQL CREATE TRANSFER statement. The source databases are identified by first issuing one or more ATTACH statements. The target database is identified in the CREATE TRANSFER statement in the TO clause (TO NEW ... or TO EXISTING ...). The TO clause is an attach statement for the target database. The accounts used to access the source and target databases can be specified explicitly within these attach statements or in a more indirect form:

- Explicitly name the user account and password in the attach string.
- Explicitly name the user account and password in a configuration file and name the configuration file in the attach string.
- Explicitly name a proxy account in the attach string.
- Explicitly specify the account in an Rdb configuration file.
- Explicitly use one of several user accounts in a Replication Option configuration file.
- Implicitly use your local account as authorization into the remote database.

These methods are not necessarily interchangeable. Some are available only with certain network software or certain operating systems. Some methods of specifying explicit account names and passwords do not work with certain database systems.

4.8.1 Remote Server Security and Environment Considerations

The remote server process can be created under different OpenVMS accounts when DECnet is used: an explicitly named account, a proxy account, or the default RDB$REMOTE account. Under OpenVMS using TCP/IP, the server is created under the RDB$REMOTE account, unless an alternate service name is defined. In Digital UNIX systems, it is always run under the dbsmgr account. Details are in the installation and configuration guide for Oracle Rdb.

Knowledge of which server account is used can be important in some transfers. A transfer definition that uses the TO NEW FILENAME clause instructs Replication Option to create the target database files. File creation will be performed on your behalf by the remote Rdb server process. Therefore, the account under which that process executes must have the necessary permissions/privileges to create files in the target directory specified in the transfer. Also, depending on the database system used, temporary work files might be created, such as, journal files, recovery files, sort work files. These files must be created from the remote Rdb server account, and that account must have the permissions/privileges to be able to do so.

Note too that other environmental factors play a role. If the remote database file specification contains logical names or configuration parameters, you must be sure that those can be translated within the context of the remote server process. If you need to use a variant version of the database product, the proper environment needs to be established within the server process. Also, in an OpenVMS system, process priority and quotas are usually determined by the account under which you log in.

4.8.2 Remote Access Using an Explicit User Account and Password

When you attach to a source database or when you identify the target database in a TO clause of a CREATE TRANSFER statement, you can explicitly name the user account(s) and password(s) to access the remote database. An example of how to do this is shown in Example 4–1.

Example 4–1 Explicitly Named Remote Account Using Syntax for DECnet

```
SQL> ATTACH 'FILENAME SOURCE"PROD SECRET"::PROD$DISK:[DIR]PROD.RDB';

SQL> CREATE TRANSFER MIS_DATA TYPE IS EXTRACTION
cont> MOVE TABLES *
cont> TO NEW FILENAME TARGET"MIS PWD"::MIS$DISK:[DIR]MIS.RDB
cont> LOG FILENAME IS MIS_DATA_LOG;
```

In the first line, you attach to an Rdb source database, PROD.RDB, on a remote system named SOURCE. The user name is PROD and the password is SECRET. At the remote site, an Rdb server process is created using that user name and password. The syntax shown in Example 4–1 is applicable to DECnet, but not to TCP/IP. The CREATE TRANSFER statement illustrates the equivalent form used in the TO clause.

Example 4–2 shows the alternative to Example 4–1 that is used when the network transport is TCP/IP and the remote operating system is Digital UNIX. The user name and password are explicitly specified, but by a different syntax. Do not rely on this method; see the note that follows.

Example 4–2 Explicitly Named Remote Account Using Syntax for TCP/IP

```
SQL> ATTACH 'FILENAME source::/usr/users/production/prod.rdb -
cont> USER ''prod'' USING ''secret''';

SQL> CREATE TRANSFER MIS_DATA TYPE IS EXTRACTION
cont> MOVE TABLES *
cont> TO EXISTING FILENAME 'target::/usr/users/misdept/mis.rdb-
cont> USER ''mis'' USING ''pwd'''
cont> LOG FILENAME IS MIS_DATA_LOG;
```

Note

The method used in Example 4–2 cannot be used when creating a transfer using Rdb SQL version 7.0 or Replication Option Version 7.0. This is due to limitations in both Rdb SQL and in Replication Option. If you create a transfer using the notation in Example 4–2, transfer execution will fail. Instead, see Section 4.8.4 and Section 4.8.5. This restriction might be removed in future product releases.

In Example 4–3 the database is accessed by the Rdb Transparent Gateway to Oracle. The gateway software is executed within the copy process in the transfer system and is therefore run from the user account that creates the transfer definition.

The user name and password in the attach strings in this example are used to gain access to the Oracle database itself. This is an Oracle account unrelated to the operating system account within which the copy process executes. Because the gateway software is executed within the local copy process, a remote Rdb server process is not used for the transfer.

Example 4–3 Explicitly Named Remote Database Account with Local Transparent Gateway

```
SQL> ATTACH 'FILENAME /TYPE=ORACLE/NODE=SOURCE-v7P/TABLE=prod.*-
cont> /USER=prod/PASSWORD=secret';

SQL> CREATE TRANSFER MIS_DATA TYPE IS EXTRACTION
cont> MOVE TABLES *
cont> TO EXISTING FILENAME '/TYPE=ORACLE/NODE=TARGET-v7P/TABLE=mis.*-
cont> /USER=mis/PASSWORD=pwd'
cont> WITH NO CHECKING
cont> LOG FILENAME IS MIS_DATA_LOG;
```

Example 4–4 is similar to Example 4–3. In this case the transparent gateway software is located on the remote OpenVMS source or target system; it is not local to the transfer system. Thus an Rdb server process must be created at the remote site. It is the Rdb server process that executes with the transparent gateway software to access the remote database. Two sets of user name and password are needed. The first set names the remote user account used to create the remote Rdb server process. The second set names the database system account used to gain access for the transparent gateway software operating on your behalf.

Example 4–4 Explicitly Named Remote Accounts with Remote Database and Transparent Gateway

```
SQL> ATTACH 'FILENAME SOURCE"PROD SECRET"::/TYPE=ORACLE-
cont> /NODE=SOURCE-v7P/TABLE=OraProd.*-
cont> /USER=OraProd/PASSWORD=OraSecret';

SQL> CREATE TRANSFER MIS_DATA TYPE IS EXTRACTION
cont> MOVE TABLES *
cont> TO EXISTING FILENAME 'TARGET"MIS PWD"::/TYPE=ORACLE-
cont> /NODE=TARGET-v7P/TABLE=OraMis.*-
cont> /USER=OraMis/PASSWORD=OraPwd'
cont> WITH NO CHECKING
cont> LOG FILENAME IS MIS_DATA_LOG;
```

Note

In all the previous examples, be aware that user names and passwords are not hidden. Anyone with the privileges to view a transfer definition can see the passwords. Perhaps for this reason you might prefer to use one of the other methods for remote access described in the following sections.

Example 4–5 is the same as Example 4–3 with the exception that many of the attach parameters have been removed. In their place are references to transparent gateway configuration files. Within these configuration files would be statements that define the missing parameters, such as, user name and password. Use of a configuration file has the advantage of shortening the attach string and hiding the account password. For information about setting up a transparent gateway configuration file see the Oracle Rdb Distributed Technology Product Family User's Guide for the version of Oracle Rdb you are using.

Example 4–5 Explicitly Named Remote Database Account with Local Transparent Gateway

```
SQL>ATTACH 'FILENAME /TYPE=ORACLE/CONFIG=ORACLE_SOURCE_CONFIG';

SQL> CREATE TRANSFER MIS_DATA TYPE IS EXTRACTION
cont> MOVE TABLES *
cont> TO EXISTING FILENAME /TYPE=ORACLE/CONFIG=ORACLE_TARGET_CONFIG
cont> WITH NO CHECKING
cont> LOG FILENAME IS MIS_DATA_LOG;
```

4.8.3 Remote Access Using a DECnet Proxy Account

If you are using DECnet to transfer data from or to a remote Rdb database on a remote OpenVMS system, you can use a proxy account. Proxy accounts permit one or more users on one OpenVMS system to obtain access privileges to another OpenVMS system without having a private account on the other system. The user accounts in this section are those within which transfers (copy processes) must execute.

Use of proxy access simplifies database attach strings. You do not have to specify passwords, and in some cases you do not have to give an account name.

Example 4–6 Remote Access Using a DECnet Proxy

```
SQL> ATTACH 'FILENAME REMNOD"JADAMS"::DISK1:[DIR1]PERSONNEL.RDB';

SQL> CREATE TRANSFER Q1_SALES TYPE IS EXTRACTION
cont> MOVE TABLES *
cont> TO EXISTING FILENAME REMNOD"JADAMS"::DISK1:[DIR1]PERSONNEL.RDB
cont> WITH NO CHECKING
cont> LOG FILE IS Q1_SALES_LOG;
```

In Example 4–6, following the node name, REMNOD, the name of the proxy account on that node appears in quotes, JADAMS. When proxy access has been established for you at system REMNOD, you attach to the remote database as if you were user JADAMS. DECnet also has the concept of a default proxy account. You can log in to a remote system without having to specify an account name; you will be logged in under your default proxy account, if one has been defined for you.

4.8.3.1 Setting Up DECnet Proxy Accounts on Remote OpenVMS Systems

The system manager controls the use of proxy accounts by using the OpenVMS Authorize utility (AUTHORIZE) to create and modify the permanent proxy database (NETPROXY.DAT). To ensure that the copy process uses a proxy account and its associated privileges, your system manager must use AUTHORIZE to specify an explicit association between the node from which the copy process can execute and the user account on the remote node. For example, you might want to associate three accounts (source nodes) where the copy process executes with a remote account where a target database is created:

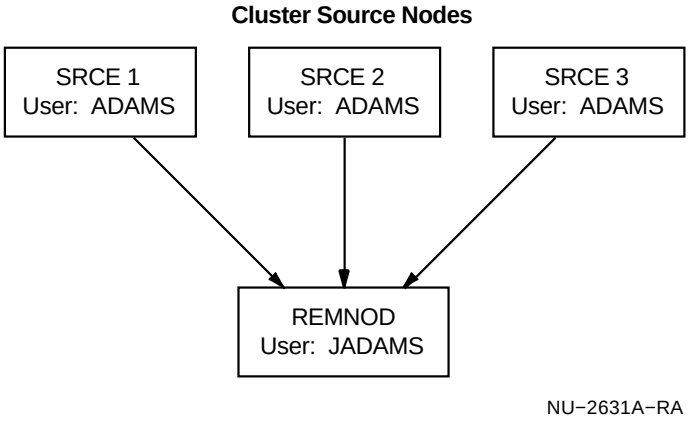
```
Source Nodes            Target Node
------------            -------------
SRCE1::ADAMS            REMNOD::JADAMS
SRCE2::ADAMS            REMNOD::JADAMS
SRCE3::ADAMS            REMNOD::JADAMS
```

Each proxy login record maps a set of local accounts to a remote node name and user name (in the form node::user). For example, on the remote node the system manager might use AUTHORIZE to enter the following command allowing user ADAMS on nodes SRCE1, SRCE2, and SRCE3 to access the account associated with the user name JADAMS on the remote node REMNOD:

```
UAF> ADD/PROXY SRCE1::ADAMS,SRCE2::ADAMS,SRCE3::ADAMS JADAMS/DEFAULT
```

If the transfer to be executed is stored in a transfer database maintained on a cluster, the copy process can execute on any node in the cluster. Figure 4–1 shows the source nodes in a cluster configuration on which the copy process can execute, SRCE1, SRCE2, and SRCE3. The figure also shows the target node, REMNOD, which contains the target directory belonging to user JADAMS where the copy process can create the target database.

Figure 4–1 DECnet Proxy Access for Remote OpenVMS Nodes



The user, in this case the copy process with the user name ADAMS, can access the target node (and directory) as though it were user JADAMS and have sufficient privileges to create a database.

You can avoid specifying proxy access for the same user account on each node in a cluster by using a cluster alias name. For example, if the alias name for the SRC1, SRC2, and SRC3 cluster is SOURCE, the AUTHORIZE command in the previous example can be simplified to the following:

```
UAF> ADD/PROXY SOURCE::ADAMS JADAMS/DEFAULT
```

If you use a cluster alias name, make certain that the RDBSERVER network object has an alias enabled for both incoming and outgoing connections.

Using proxy accounts is equally applicable when the target database is created at your local site and the source database or databases are remote. This perspective is important because you have to define transfers at the target site to pull data from a source database that might reside at a remote site.

Individuals at your site who are responsible for defining transfers and schedules can arrange with system managers at the remote sites to have proxy accounts set up.

If multiple transfers have been defined to a single target database and that database is running Rdb Version 5.1 or higher, it may be advisable to have the system manager on the target database node set up a single proxy account to serve all transfers. This is because Rdb Version 5.1 and higher, by default, grants access privileges to the database creator only; all other users are granted no access privileges. By using a single proxy account to serve all transfers, each transfer definer can access, and grant access, to all transferred tables. Whether single or multiple proxy accounts are used, you can include an epilogue procedure in your transfer definition to make sure users and applications can access target tables.

Whenever the copy process creates an Rdb target database, Rdb establishes access privileges to the database by default; Replication Option does not specify access information as part of the database creation process. By default, Rdb gives all privileges to the OpenVMS user account used to create the database and no privileges to anyone else. If the target database is created using a remote proxy account, that OpenVMS account becomes the owner of the target database and is the only account that can alter database access privileges to allow others to use it. Chapter 7 has an example showing how to grant access to others.

Refer to the OpenVMS documentation about system security, networking, and system management for complete details and examples that show how your system manager can use the OpenVMS Authorize utility to prepare, create, and maintain proxy accounts.

4.8.4 Remote Access Using an Rdb Configuration File

You can provide a user name and password for a remote system in a local Rdb configuration file, RDB$CLIENT_DEFAULTS.DAT, on OpenVMS. The parameters are SQL_USER NAME and SQL_PASSWORD.

For example, if your user name and password on the remote system are "ejones" and "travels", add them to your Rdb configuration file on the local, transfer system:

```
SQL_USER NAME ejones
SQL_PASSWORD travels
```

If you specify a user name and password in the Rdb configuration file, it is used to log in to the remote server account when you are using DECnet to access the remote system and when your attach statement does not include an explicit user name and password (see Section 4.8.2) or a proxy name (see Section 4.8.3).

You can name only one account in the Rdb configuration file. This is not suitable when your source and target databases require different accounts to access them. See Section 4.8.5 for an alternative method.

4.8.5 Remote Access Using an Account in a Replication Option File

When accessing an Rdb database on a remote Digital UNIX system using TCP/IP, you cannot specify the USER USING syntax (see Example 4–2) with Replication Option. This is due to restrictions in Rdb SQL and in Replication Option. Instead, you must specify a user account either in an Rdb configuration file (see Section 4.8.4) or in a Replication Option configuration file.

In the Rdb configuration file you can name only one user account. If your transfers require access to more than one Rdb database on Digital UNIX using TCP/IP, and if the accounts needed for user authentication by Rdb are not the same in all cases, you must use the method described in this section rather than the one in Section 4.8.4.

The copy process, which is responsible in Replication Option for performing a transfer, reads a configuration file, SYS$LOGIN:DDAL.RC, if one exists, to obtain (among other things) user account names and passwords. These are specified with the following three environment variables:

- DDAL_DATABASE_n—Specifies a database attach string
- DDAL_USER NAME_n—Specifies the user name used in attaching to a database
- DDAL_PASSWORD_n—Specifies the password used in attaching to a database

These environment variables must be specified in three separate lines in the configuration file as follows:

```
DDAL_DATABASE_n  "attach string"
DDAL_USER NAME_n  "user name"
DDAL_PASSWORD_n  "password"
```

These environment variable names must always be entered in uppercase and using the underscore character. Other environment variables can be entered in upper- or lowercase and can use a $ character in place of the first underscore character.

Repeat these three entries, using different definitions, for each account and database you need to mention. Each group of three variables is distinguished from the others by a numeric suffix in the names, shown previously as _n. Replace the letter *n*, with an integer between 1 and 20. There is no requirement that the numbers be assigned consecutively starting with 1, but numbers chosen must normally be between 1 and 20.

In Example 4–7, the user name and password for Rodriguez are associated with the Rdb source database, prod.rdb, located on system "source". The account for user Sayles is associated with the mis.rdb database located on the "target" system. The attach string for the database as entered in the configuration file must exactly match the name stored in the transfer definition.

Example 4–7 DDAL.RC Configuration File with Two User Accounts and Databases

```
DDAL_DATABASE_1  "source::/usr/users/production/prod.rdb"
DDAL_USER NAME_1  "Rodriguez"
DDAL_PASSWORD_1  "secret"
DDAL_DATABASE_2  "target::/usr/users/misdept/mis.rdb"
DDAL_USER NAME_2  "Sayles"
DDAL_PASSWORD_2  "mispwd"
```

The information stored in the transfer definition can be different from what you enter in the ATTACH and CREATE TRANSFER statements. Logical names might be replaced by their equivalent strings. This is affected by whether or not you use the SET LOGICAL_NAME TRANSLATION FOR TRANSFER statement. Rely on what you see in the output of a SHOW TRANSFER (DEFINITION) statement. Two rules follow from this:

1. Do not mix physical path names and links

   If your CREATE TRANSFER statement shows the physical path name for a database, do not use a UNIX link as the definition in the configuration file. Although UNIX permits different paths to lead to the same physical file, for the purpose of matching user accounts with databases in Replication Option, the same names must be used in the transfer definition and the configuration file.

2. Do not mix logical names and full path names

   For reasons similar to that mentioned previously, if you refer to a database in the transfer definition using a logical name, you must use that same logical name in the attach string for the database variable in the configuration file. String comparison is done by Replication Option with what is stored in the transfer definition, not with the string that name translation might result in at run time.

If you need to specify more than 20 accounts and databases, insert an additional environment variable, DDAL_NUMBER_OF_DATABASES, to indicate the new maximum. For example, the following line changes the range of valid numeric suffixes from 1 to 20 to 1 to 30.

```
DDAL_NUMBER_OF_DATABASES  30
```

Remember, this only specifies the maximum number of account/database entries you may have. You may in fact have fewer entries than the maximum. As with the other variables named in this section, DDAL_NUMBER_OF_ DATABASES must be entered only in uppercase and only using the underscore character.

At run time, before it attaches to a source or target database, a copy process will look in the DDAL.RC file for environment variables of the form DDAL_ DATABASE_n, where *n* is an integer from 1 to the designated maximum. First it will look for DDAL_DATABASE_1, then DDAL_DATABASE_2, and so on. For each such variable found, the corresponding attach string is compared with the attach string from the transfer definition. If a match is found, the corresponding user name and password will be used to attach to the Rdb database. This takes precedence over any of the other methods of attaching to a database described in this section.

Note

Because the DDAL.RC file contains sensitive information, such as user names and passwords, the file should be protected from being read by unauthorized users. The responsibility for doing that is with the individual who creates the file.

4.8.6 Remote Access Using Implicit Authorization

The preceding sections discuss how to access a remote database by giving user names and passwords or proxy names. This section explains what happens when none of these is specified.

Example 4–8 applies only for remote Rdb databases on OpenVMS, where the network software is either DECnet or TCP/IP. In Example 4–8 the Rdb databases PROD.RDB and MIS.RDB are located on remote systems SOURCE and TARGET, respectively. An Rdb server process will be created on the remote system. Because no account is named in the attach strings and because we assume that no accounts are named in any of the configuration files, the server will attempt to sign onto the remote Rdb database under the same account name and password used at the local OpenVMS site. In other words, authorization is implicit. It works provided that the user name and password are the same on both OpenVMS systems.

Example 4–8 Implicitly Named Remote Server Account

```
SQL>ATTACH 'FILENAME SOURCE::PROD$DISK:[DIR]PROD.RDB';

SQL> CREATE TRANSFER MIS_DATA TYPE IS EXTRACTION
cont> MOVE TABLES *
cont> TO EXISTING FILENAME TARGET::MIS$DISK:[DIR]MIS.RDB
cont> LOG FILENAME IS MIS_DATA_LOG;
```

4.9 Preparing Database Access Using the RMU Open Command

Note

This section only applies if you are using an Rdb database.

The first time an Rdb database is attached (declared) from a particular node, activity on that database is momentarily frozen to allow the Rdb monitor to prepare the database for general use and for proper recovery of failed transactions should that subsequently occur. Overhead, normally charged to the first user of that database from a given node, can instead be absorbed by Rdb if you use the RMU Open command.

The use of the RMU Open command is more than just a convenience; it avoids one source of potential access conflict that can cause a Replication Option transfer to fail. The following example illustrates one situation that can arise if you do not use the RMU Open command.

Consider a two-node cluster on which you have a source database that you want to transfer to two destinations. You can use any of the Replication Option transfer methods. The first transfer executes on NODE1 while the other executes on NODE2. If the NODE1 transfer compiles a database query at about the time that the NODE2 transfer attaches to the source database, the momentary freeze on database activity can cause the request compilation of the NODE1 transfer to fail with the following error message:

```
RDMS-F-LCKCNFLCT, lock conflict on freeze
```

You can avoid the conflict error if you manually open the source database with the RMU Open command at system startup of both NODE1 and NODE2.

Target databases can be treated in the same way. If a Replication Option transfer to an Rdb target database is about to compile a database request and a user or application from another node at the target site is the first to attach to the target database from that node, the transfer can fail with a lock conflict error.

A transfer that fails because of lock conflict will be retried if the transfer has an appropriately defined schedule. Chapter 6 describes how to create transfer schedules.

4.10 DECnet Network Tuning Parameters

On some DECnet systems, you may find that heavy network traffic causes transfers to fail. The following steps describe a typical failure under these circumstances:

1. The copy process starts.
2. Approximately 70 seconds later, the copy process terminates. The copy process log file contains the message "Remote node is not currently reachable". However, investigation indicates that the remote system is still functioning properly and is accessible through DECnet (or DECnet/OSI).
3. A NETSERVER.LOG file appears at the remote node in the target account's top-level directory. A line in the file states that a request to connect to RDBSERVER was received at a particular time. This time is approximately 8 seconds after the copy process on the requesting node exited.
4. Finally, the message "Network partner aborted logical link" appears in the NETSERVER.LOG file.

This type of failure can occur in cluster environments that use a cluster alias name.

You can alleviate this condition by specifying transfer retries in the schedule definition and by increasing DECnet or DECnet/OSI timers.

When a transfer fails, as described, the Rdb server process continues to exist on the remote node for several minutes after failure. If you define the transfer specifying retries at a short interval, such as every minute or less, the retry is usually successful.

You can also ask your system manager to increase the DECnet or DECnet/OSI outgoing timer value. The outgoing timer specifies the maximum time to wait for an acknowledgment to connect to a remote network object, in this case RDBSERVER.

4.11 Performing Database Maintenance, Backup, Recovery, and Conversion

Routine maintenance and protection of your source, target, and transfer databases are essential parts of database operations. It is important to protect your data from such problems as accidental deletion, hardware failure, and software failure. Also, from time to time database formats must be converted as new versions of the database management systems are released.

Note

This section discusses tools specific to Rdb. For Rdb Transparent Gateway products, refer to the native database management system documentation for the equivalent tools. Rdb is used as an example; however, these types of maintenance operations are typical of those used with all database management systems. Only the tools and the procedures will differ. Also, some concepts may be specific to the OpenVMS operating system.

For Rdb databases, the tools include the following:

- Backup and restore tools

  The RMU Backup and RMU Restore commands

- Database migration tools

  The SQL IMPORT and SQL EXPORT statements

- Database conversion tools

  The RMU Convert command for converting a database to the format required by the current version of Rdb

- Database recovery tools

  The RMU Recover command

The following sections discuss how to back up, restore, and convert your source, target, and transfer databases. This information supplements information in the Rdb documentation set.

4.11.1 Maintenance, Backup, Recovery, and Conversion of Source Databases

Maintenance, recovery, and conversion procedures are essential for source databases. The following sections describe in general terms the backup, recovery, and conversion procedures for source databases. For detailed information about such procedures, refer to the documentation that accompanies the source database management system.

4.11.1.1 Backing Up and Recovering a Source Database

The RMU Backup command is recommended for backing up Rdb source databases. Online backup and incremental backup can be done using this operation.

If a source database becomes corrupted, suspend all transfers that use the corrupted database. If you have kept an after-image journal file (.AIJ) for that database, then you can recover the database up to the point of corruption by restoring its backup and using the RMU Recover command to apply AIJ file entries to the restored copy. In addition to restoring the source database, you may also need to drop some transfers and re-create them. For example:

- You must re-create any transfer in which the definitions of the transferred tables and columns are inconsistent with the restored copy.
- You must re-create any transfer in which the source database has been physically moved in such a way that the file specification changes. For instance, if the disk drive and disk pack for your source database become unavailable and you restore a backup copy on another drive, all transfers referring to that database must be re-created.

  You can avoid re-creating the transfers if:

  – The source disk is referred to by a concealed device logical name. In this case, redefine the logical name to point to the new disk drive. If you use concealed logical names, Oracle Corporation recommends you define them in the system logical name table.

  – The source disk is referred to by a (nonconcealed) logical name and you have specified the OFF option in an SQL SET LOGICAL_NAME TRANSLATION FOR TRANSFER statement. In this case, you can redefine the logical name to point to the new disk drive. (You must redefine the logical name so that it is visible to the copy process; redefine it at the system or group level or in the LOGIN.COM file of the transfer definer for OpenVMS systems.)

Once you have restored a source database, you must re-create those transfers that were dropped and restart them, as well as those that were suspended.

4.11.1.2 Converting Source Databases

To convert an Rdb source database to the latest Rdb format, use the RMU Convert command. The SQL EXPORT and IMPORT statements can also perform this conversion, but they are not recommended for the source databases of replication transfers.

The SQL EXPORT and IMPORT statements cause the source database and corresponding replicated target databases to be out of synchronization. If you convert a source database by using the SQL EXPORT and IMPORT statements, you would have to reinitialize all of its corresponding replication transfers. Replication databases require different treatment because:

- The SQL EXPORT/IMPORT operation changes the database keys (dbkeys) of rows in the source database. Those dbkeys are stored with the rows in the target replication databases to identify the corresponding source database rows. Once the source database is restored, the dbkeys of its rows no longer match those kept with the corresponding rows in the replication target databases.
- The SQL IMPORT and EXPORT statements do not capture the special system tables (such as RDB$CHANGES) created by Replication Option in a source database for replications. When an SQL IMPORT operation is performed, the special system tables are no longer present in the source database; thus, the replication transfer definitions do not appear, though they still exist in the transfer database.

Before converting a database, you should back up the database in case errors occur during the conversion process. Using the SQL STOP TRANSFER statement, suspend all transfers referring to the database you want to convert. Perform the conversion. Then, using the SQL START TRANSFER statement, restart the transfers you suspended. If you used the SQL EXPORT and IMPORT statements to perform the conversion, you must reinitialize all the replication transfers pertaining to this source database.

For more information on converting databases, refer to your native database management documentation.

4.11.2 Maintenance, Backup, Recovery, and Conversion of Target Databases

Maintenance, backup, recovery, and conversion procedures are essential for target databases. The following sections describe in general terms the backup, recovery, and conversion procedures for target databases. For detailed information about such procedures, refer to the documentation that accompanies the target database management system.

Note

Target databases can also serve as sources for other transfers. When you are using a target database as a source database also, follow the rules for source databases. Those rules may be more restrictive in some situations.

4.11.2.1 Backing Up and Recovering a Target Database

For backing up an Rdb target database, use the RMU Backup command. The RMU Backup operation provides online and incremental backup capabilities.

Extraction transfers created with the TO NEW qualifier of the CREATE TRANSFER statement and extraction rollup transfers create a target database at the beginning of the transfer. Once the transfer has completed, the target database is no longer connected with the source databases; it is an independent database.

If the extraction is performed only once (that is, there is no periodic repetition of the extraction transfer), the target database can be maintained as a normal Rdb database. You can use backups and after-image journaling to protect that database in the event of subsequent corruption. However, if the extraction or extraction rollup transfer is scheduled to repeat regularly, the target database will be replaced by a newer copy as soon as the extraction or extraction rollup transfer repeats. The newer copy serves the same function as a backup copy. You might still want to maintain backups and after-image journaling on the latest copy of the target database to recover from corruption. Recovering a backup copy of the target database might be faster than waiting for the next transfer of a new copy.

Replication databases created as part of an original transfer, unlike extraction and extraction rollup databases, maintain an association with the source database. Subsequent executions of a replication transfer update the original target database files rather than create new database files. If, for some reason the replication database becomes corrupt, you can reinitialize the replication transfer. (Refer to Chapter 9 for a description of the SQL REINITIALIZE TRANSFER statement.) If you maintain periodic backups and enable after-image journaling for recovery purposes, instead of reinitializing the transfer, you can reconstruct the most current version of the database. Your individual circumstances will help you determine whether to reinitialize the transfer or to use a backup copy with after-image journaling.

4.11.2.2 Converting Target Databases

To convert a target database to the latest Rdb format, use the RMU Convert command or the SQL EXPORT and IMPORT statements. Before converting a database, you should back up the database in case errors occur during the conversion process. Use the SQL STOP TRANSFER statement to suspend all transfers referring to the database you want to convert. Perform the conversion operation. Then, use the SQL START TRANSFER statement to restart the suspended transfer.

For more information on converting databases, refer to your native database management documentation.

Target database conversion is also required for conformance to the current format required by Replication Option. This conversion involves adding columns to existing Replication Option tables, modifying existing rows for active transfers, and adding tables. This type of conversion is only done automatically when a transfer executes on an Rdb target database.

4.11.3 Maintenance, Backup, Recovery, and Conversion of Transfer Databases

This section describes the backup, recovery, and conversion of transfer databases, using Rdb to illustrate typical methods. For transfer tables stored in Oracle or Sybase databases, refer to the literature for those products for equivalent tools and procedures.

4.11.3.1 Backing Up and Recovering a Transfer Database

As a precaution, you should make a periodic backup of each Rdb transfer database using the RMU Backup command. You should also keep after-image journaling enabled. The RMU backup operation provides a means of online and incremental backup. Refer to the Rdb documentation for further information.

If you need to restore a transfer database from a backup copy, first shut down the Replication Option using the SYS$MANAGER:DDAL$STOP_TR_MON.COM command procedure on each node in your cluster. Then restore the backup copy using the RMU Restore command, and issue the RMU Recover command to roll forward, if you had after-image journaling enabled.

Note

If you did not have after-image journaling enabled, or if you cannot roll forward to the present time, the transfer definitions might be out of date. This problem is likely to occur for a replication transfer because part of the transfer definition is duplicated in the source database. In

such cases, you may have to reinitialize and restart the replication transfer. You might even need to drop and re-create that transfer.

After restoring or rolling forward the transfer database, you must execute the DDAL$START_TR_MON.COM command procedure on each node of the cluster to restart the transfer monitors.

4.11.3.2 Converting Transfer Databases

To convert an Rdb transfer database to the current format required by Rdb and Replication Option, you must perform the following steps:

1. Convert the transfer database to the current Rdb format using either the RMU Convert command or the SQL EXPORT and IMPORT statements.
2. Run the SYS$MANAGER:DDAL$CONVERT_TR_DB.COM command procedure to convert the transfer database tables to the latest Replication Option format.

4.11.4 Changing Data Definitions and Restructuring Databases

Changing source and target databases can involve additions, modifications, and deletions of data definitions (columns, tables, and views) and associated data structures (indexes, constraints, and triggers). It can also involve restructuring the physical database.

4.11.4.1 Changing Definitions in and Restructuring Source Databases

This section describes a situation where an existing Replication Option transfer definition uses a source database that you are considering restructuring or for which you plan to make data definition changes. It is possible that making such changes would affect the data transfer, and you need to be aware of the following consequences:

- Adding tables or views

  If you add the definitions of tables or views to a source database, the new definitions will not be transferred to target databases by existing transfer definitions. If you want them to be included when the next transfers execute, you must change the transfer definitions. In particular, you must change any definition that uses the MOVE TABLES * or the MOVE VIEWS * clause. These clauses move all tables (or views) that exist in the source database at the time you create the transfer.

- Adding columns to existing tables

If you change a table by adding columns, the new columns will not be transferred to the target database unless you change the transfer definition. Even if you use the wildcard notation MOVE TABLES * or the MOVE TABLES SELECT * FROM table-name clause, the transfer definition includes only those columns that existed at the time you entered the definition. Note that once a replication transfer has executed at least once, changes to any source tables named in the transfer definition will be disallowed by Rdb.

- Altering or dropping column definitions

  If you intend to change a transferred table by altering or dropping column definitions, you should plan to change the transfer definition as well. If you do not, the transfer might fail. Again, once a replication transfer has executed at least once, changes to any source tables named in the transfer definition will be disallowed by Rdb.

- Dropping tables or views

  If you plan to drop a transferred table or view, change the transfer definition also. Deletion of source tables and views named in a transfer definition will be disallowed by Rdb, once a replication transfer has executed at least once.

- Transferring of indexes, constraints, triggers, query outlines, stored procedures, and access privileges is not supported in transfer definitions

  There are a number of objects in a database that cannot be named directly in a transfer definition. These are indexes, constraints, triggers, query outlines, stored procedures, and access privileges. When Replication Option executes a transfer, it does not attempt to re-create any indexes, constraints, triggers, query outlines, stored procedures, or access privileges in the target database, even if they appear in a source database. Therefore, if you decide to add or change these things within a source database, this ought not to affect Replication Option.

- Physical relocation of the source database files

  If you need to move source database files from one disk to another, it is possible to do it without changing the transfer definition that refers to that database.

  You can do this if:

  – The source disk is referred to by a concealed device logical name and you either have not used the SET LOGICAL_NAME TRANSLATION FOR TRANSFER statement, or you have used it and specified the DEFAULT or ON option. In these cases, redefine the logical name to point to the new disk drive.

Concealed device logical names can be used with both single file and multifile databases and must be defined before you create the transfer definition. By default, when a transfer definition is recorded in the transfer database, the file specification for the source database is expanded; that is, logical names are translated immediately by default (or when the ON options has been specified in the SET LOGICAL_NAME TRANSLATION FOR TRANSFER statement). If the translation results in yet another logical name, it too is translated. These translations continue until a concealed device logical name is reached. The concealed device name is preserved and is not translated. For further information about concealed device names, refer to the OpenVMS documentation set. If you use concealed logical names, Oracle Corporation recommends you define them in the system logical name table.

- The source disk is referred to by a (nonconcealed) device logical name and you have specified the OFF option in an SQL SET LOGICAL_ NAME TRANSLATION FOR TRANSFER statement. In this case, redefine the logical name to point to the new disk drive.

  The logical name must be visible to the copy process, which runs as a detached process. For the logical name to be visible to the copy process, it must be defined at the system or group level or must be defined in the LOGIN.COM file of the transfer definer.

  You only need to be concerned about concealed devices on OpenVMS when you want to display the definition of a logical name. If a device name is concealed, OpenVMS will not display the definition of that device if it is a concealed logical device.

- Physical restructuring of the source database

  Changes to physical and operational characteristics of the database, such as those you would make using the SQL ALTER DATABASE statement, should not affect Replication Option transfers. However, you might want to temporarily suspend them while you make database changes.

  In general, any changes to the source database that will change the dbkeys of existing rows will affect replication transfers. For example, using the SQL EXPORT and IMPORT statements affects replication transfers because their use changes dbkeys in the source database. For further information, refer to Section 4.11.1.2.

4.11.4.2 Changing Definitions in and Restructuring Target Databases

If you have an existing Replication Option transfer, you might want to make data definition changes in the target database or restructure the physical database. This section describes how to make these changes and notes any consequences that could affect data transfer.

The major consideration is whether the transfer creates its own target database or uses an existing database. As a rule, you can add data definitions to a target database or modify definitions that you own in an existing Rdb target database using the SQL ALTER TABLE statement. If you need to alter data definitions in a non-Rdb target database, use the equivalent tools native to your database system. Normally, you should not alter the definitions of objects transferred by Replication Option if the transfer will execute more than once. If the transfer executes only once, you can add to or modify whatever you want in the target database.

For example, if you create and execute an extraction transfer to an existing Rdb database, Replication Option creates table definitions in the target database to match the definitions of tables in the source database. If, between the time you first run the transfer and the next time you run the transfer, someone alters a table definition in the source database but does not make the corresponding change in the target database, Replication Option compares the table definitions in the source database and target database. If the databases are different, Replication Option alters the target definition to match the source definition. This is normal behavior for Replication Option, though it can be suppressed by using the DDAL$CHECK_TARGET_DEFINITIONS environment variable (described in Section B.4). Remember that in a transfer to an existing database, repeating the transfer does not create new database files. In this case, you do not need to alter the target definitions to match the source definitions because Replication Option does that for you. However, if you want to make those changes yourself before re-executing the transfer, you may do so.

What if you add a column to an existing target table that is *owned* by the transfer? If Replication Option sees a column in the target table that does not exist in the source table, Replication Option normally assumes that the column was recently deleted from the source; therefore, it deletes the column from the target. Your changes are lost. By contrast, if the extraction transfer were to a new target database each time rather than to an existing database, Replication Option would create new target database files. Again, any changes you made to the target database, including tables you added, would be lost.

If you have a non-Rdb target database, you can encounter other problems if you alter definitions in the source database tables without altering the definitions in the target database. Normally, Replication Option will detect these altered definitions and issue an error message before stopping the transfer. In this case, you *must* alter the target definitions yourself.

For replication transfers and for shared-write transfers, the target tables will have columns that the source might not have: DDAL$DBKEY and DDAL$ROW_OWNER. See Section 6.14 for more information on these columns. You may have to add the DDAL$ROW_OWNER column to the target table in some circumstances. That would make the source and target table definitions different; however, these two columns are exceptions and are ignored when source and target definitions are compared.

The following general rules apply to making changes to target databases:

- Changes you make on your own

  If these changes make the source and target table definitions look different, you should not make them except in the special case explained at the beginning of this section.

  If necessary, you can suppress detection of data definition changes by using the DDAL$CHECK_TARGET_DEFINITIONS environment variable. But remember that if you suppress comparison of transfer definitions, you must be sure to manually make any changes to the target tables that you make to the source tables. Otherwise, the transfer will fail. See Section B.4 for more information.

- Changes Replication Option asks you to make

  These changes might include making the target database definitions match the source database definitions or adding columns that Replication Option needs in order to manage certain types of transfers.

- Adding tables or views

  You can add tables and views to a target database that was not created by Replication Option. However, if Replication Option does create the target database, repeated executions of a transfer to that target might create new versions of the database files for the following transfer types:

  - Extraction transfers
  - Extraction rollup transfers
  - Initial execution of replication transfers
  - Reinitialization of replication transfers

Replication Option does not create new database files for replication updates.

You can add tables and views to a target database created by Replication Option. Use an epilogue procedure associated with the transfer so when newer versions of the database files are created by a subsequent execution of the transfer, your tables and views will be added as well. You must copy any data that existed in your tables in the old target database files to the latest version of the target database. Replication Option does not do this automatically for you because it does not manage these tables.

You should not use an epilogue procedure to add tables and views during replication update transfers because the definitions must already exist.

If you want to add a view that references a table transferred by Replication Option and the transfer is to an existing database, use an epilogue procedure. A prologue procedure may be required to drop the view prior to transfer execution. For more information, refer to Chapter 8.

- Altering table and view definitions

  You can alter or add columns in tables and views in Rdb databases that you own. If the tables and views are managed by Replication Option, do not alter or add columns unless you suppress comparison of definitions. To suppress comparison, use the DDAL$CHECK_TARGET_DEFINITIONS environment variable, as described in Section B.4.

- Dropping tables and views

  You can drop tables and views that you own. Do not drop tables and views managed by Replication Option. Doing so can cause subsequent executions of a transfer to fail.

- Adding domains

  You can add a domain to the target database at any time. Adding domains requires the same considerations as adding tables. See the discussion on adding tables and views in this section.

- Altering domain definitions

  You should not alter a domain definition if the domain is owned by a Replication Option transfer. You can alter domain definitions that you own, but be aware that Replication Option tables might have columns based on your domain. This type of domain sharing is permitted for transfers to existing databases. If you decide to alter a domain definition and the domain is used by Replication Option tables, make certain that the changed definition does not cause loss of data precision.

- Dropping domains

You should not drop domains that are owned by Replication Option; doing so can cause a transfer to fail. You might receive an error message when you try to drop a domain that you created if the domain is used by a Replication Option transfer. Transfers to existing databases can reference domains that already exist in the target database if the data type, length, and scale of the domain matches the domain in the source database.

- Copying existing indexes, constraints, triggers, query outlines, stored procedures, or access privilege information

  Replication Option has no direct means within a transfer definition of copying existing indexes, triggers, query outlines, stored procedures, or access privilege information to a target database. If you want such items in a target database, you must add them yourself. You may do so either manually, or as part of the transfer process within an epilogue procedure.

  See Chapter 7 for more information on using command procedures on OpenVMS.

  To manage the transfer process, Replication Option normally adds and drops indexes that it requires on different tables (see Section C.5 for a description of these indexes). When Replication Option is unable to manage the indexes itself, you will be asked to do so. This is described in Section 5.5.9.

  You can add indexes for tables transferred by Replication Option. If the transfer is to an existing database, you can add constraints on your local tables based on information in the transferred (reference) tables. You can also create triggers on the transferred tables that cause some action to be taken in your local tables when changes occur in the transferred tables. If you do so, be aware that a trigger failure causes the transfer to fail. This type of failure might require manual rather than automatic recovery.

  You can modify the ACLs of tables and views transferred by Replication Option to grant the necessary privileges to users who need data access.

  If your transfer is to an existing database, use an epilogue procedure to add indexes, constraints, or triggers, or to modify access control lists for transferred objects. You might need a prologue procedure to drop a view prior to transfer execution. Refer to Chapter 8 for information on dropping views.

- Relocation of the target database files

  If you need to move target database files from one disk to another, you do not have to change the transfer definition that refers to that database. When you create a transfer, use a logical name to represent the target database file name. The logical name must be created before you create the transfer definition. Then, if you want to move the target database

files, redefine the logical name to point to the new disk. Note however that if you have used the SET LOGICAL_NAME TRANSLATION FOR TRANSFER statement and selected the ON attribute, your logical name for the transfer database will be translated and stored in its expanded format in the source database. In this case, you will not be able to simply move the target database files and redefine the logical name to point to the new disk. See Section 9.11 for more information on the SET LOGICAL_NAME TRANSLATION FOR TRANSFER statement.

- Restructuring of the target database

  Changes to physical and operational characteristics of the database, such as those you would make using the SQL ALTER DATABASE statement, should not affect Replication Option transfers. You might want to suspend them temporarily while you make database changes. If the transfer created the original target database and is scheduled to repeat execution, a new copy of the database will be created (unless the transfer is a replication update). The new copy will not show changes made to the previous copy unless those changes are included in an epilogue procedure associated with the transfer.

5
Creating Database Transfers

This chapter provides additional information on extraction, extraction rollup, and replication transfers. It describes how to create a transfer definition for each transfer type. It also shows how definitions identify the data used for extraction, extraction rollup, and replication databases.

When a transfer executes, Replication Option either creates a new target database (unless the transfer is a replication update or a replication transfer with NO DELETE) or copies tables into an existing target database. This chapter discusses transfers that create new target databases. Chapter 8 describes how to use Replication Option to copy tables into an existing target database.

Chapter 6 describes how to work with transfers after they are defined and explains how to create a schedule definition, identify transfer states, run and stop transfers, modify transfer schedules, and use log files.

Chapter 7 introduces you to command procedures on OpenVMS that can perform tasks before and after a transfer.

An understanding of transfers is not complete without some knowledge of these operations.

This chapter and others throughout this manual use SQL statements in examples. You can find descriptions of SQL statements for Replication Option and syntax information in the following locations:

- Chapter 9 in this manual
- The Oracle Rdb SQL Reference Manual for the version of Oracle Rdb you are using
- On line, using interactive SQL HELP

5.1 Extraction and Extraction Rollup Transfers

An extraction transfer creates a target database, also known as an extraction database, that contains rows selected from tables in a single source database. To create an extraction database, you choose the EXTRACTION option of the CREATE TRANSFER statement.

An extraction rollup transfer creates a target database, also known as an extraction rollup database, that contains rows selected from tables in one or more source databases. To create an extraction rollup database, you choose the EXTRACTION ROLLUP option of the CREATE TRANSFER statement.

Each execution of an extraction or extraction rollup transfer creates a new target database. A target database can be on the same node as the source database or on a different node.

Data updates are written to a source database. When you define a transfer, you specify which rows or subsets of rows are extracted (copied) from the source database and stored in the target database. You must create one transfer for each target database you need. An extraction into multiple target databases requires that you create as many transfers as you want target databases. You should also be aware of the following two points about source and target databases of extraction or extraction rollup transfers:

- Changes you make to a target database are not transferred back to the original source database or databases by Replication Option. However, changes applied to an extraction or an extraction rollup target database can be transferred to a new target database. The extraction and extraction rollup databases thereby become the source for the data copied to the new target.
- Changes applied to a source database after an extraction or an extraction rollup database has been created are never transferred to the new target. Once an extraction or an extraction rollup transfer is done, Replication Option severs all ties between the source database and its target. At the target site, you can access updates made to the source database only by scheduling new extraction or extraction rollup transfers to supersede the databases in the target directories.

When the copy process runs on a source node, the data from a database on that source to a database on a remote target node is said to be **pushed**. When the copy process runs on a target node, the data from a database on a remote source node to the target node is said to be **pulled**.

For extraction and extraction rollup transfers, you can install Replication Option in one of the following configurations:

- If Replication Option is located at the site of the source database, data is pushed to the target database.
- If Replication Option is located at the site of the target database, data is pulled from the source database.
- If Replication Option is located at neither the source nor the target site, data is pulled over the network from the source database and pushed over the network to the target database.

Data from replication transfers are always pushed because the privileges required on a remote source database to access data cannot be granted over the network.

In general, the pull method is more efficient than the push method.

Sections 5.1.1 through 5.1.5 describe how to create a transfer definition for an extraction or an extraction rollup transfer.

5.1.1 Extraction and Extraction Rollup Transfer Definition Elements

To enter the definition for an extraction or extraction rollup transfer, use the CREATE TRANSFER statement. Elements of an extraction or extraction rollup transfer definition include the following:

- The identification of the source database or databases from which the data definitions and the data are read
- The identification of the target database into which the definitions and data will be written
- A specification of the tables to be transferred
- Optional expressions to select specific rows and columns from these tables
- Optional parameter values for the creation of the target database
- An optional log file for recording activity during the transfer
- Optional prologue and epilogue procedures to be executed by Replication Option before and after the actual database transfer executes

5.1.1.1 Identifying Source Databases

The CREATE TRANSFER statement does not explicitly name the source database or databases; they are established when you first attach to the database(s). There is only one source database for an extraction transfer. The following example shows two SQL ATTACH statements. The first statement attaches to a DB2 database by means of Rdb Transparent Gateway to DB2, using the TYPE qualifier in the attach string to identify the gateway type. The second statement attaches to an Rdb database, which is the default when there is no TYPE qualifier.

```
SQL> ATTACH 'FILENAME /TYPE=DB2/DATABASE=PERSONNEL  . . .  ';
SQL> ATTACH 'FILENAME DISK2:[DEPT2]PERSONNEL';
```

See the documentation for the specific transparent gateway product for the format of an ATTACH statement and the valid parameters. The Rdb attach string names a full or partial OpenVMS file specification. Oracle Corporation recommends you use OpenVMS logical names rather than complete attach strings or file specifications, so you can move database files without altering the associated transfer definitions. You must use a SET statement to direct SQL to leave the logical names intact when stored in a transfer definition, as shown in the following example:

```
$ DEFINE DB2_SOURCE "/TYPE=DB2/DATABASE=PERSONNEL  . . .  "
$ SQL
SQL> SET LOGICAL_NAME TRANSLATION FOR TRANSFER OFF;
SQL> ATTACH 'FILENAME DB2_SOURCE';
```

For extraction rollup transfers, you must invoke the source database or databases by using different database aliases. Then, in order to indicate which databases store which tables, you must specify different database aliases in each rollup-move-clause. The following example identifies two Rdb databases for an extraction rollup transfer, using the aliases PERS1 and PERS2:

```
$ DEFINE PERSONNEL1 "NODE1::DISK1:[DEPT1]PERSONNEL"
$ DEFINE PERSONNEL2 "NODE2::DISK2:[DEPT2]PERSONNEL"
$ SQL
SQL> ATTACH 'ALIAS PERS1 FILENAME PERSONNEL1';
SQL> ATTACH 'ALIAS PERS2 FILENAME PERSONNEL2';
```

Alternatively, you can attach the database by using a data dictionary path name.

5.1.1.2 Target Nodes and Directories

You must ensure that the target nodes and directories that will receive the database extractions exist and that they have sufficient storage space to receive the data.

You can create a new set of files for the target database in Replication Option if you specify the TO NEW clause in the CREATE TRANSFER statement, instead of the TO EXISTING clause. Each time such a transfer executes, a new copy of the extraction or the extraction rollup database is placed in the target directory. Make sure that successive versions of the target database do not exceed existing storage quotas or capacities.

If the target database is on OpenVMS and you execute such a transfer more than once, several versions of the target database files will exist. You should periodically delete the unwanted older file versions to recover disk space. Replication Option does not automatically perform such operations on OpenVMS systems, unless explicitly instructed to do so in an epilogue command procedure.

If the target database is on Digital UNIX, you should only execute a transfer with the TO NEW clause once. Re-executing the transfer will fail, unless you specify in the copy process prologue file that the Rdb files (and their directory) are to be deleted first.

If storage space in the target directory is critical, you can include specific values for the target database parameters that limit the size of the Rdb snapshot file or that disable snapshots completely. In this way, you can reduce storage requirements for the extraction database. Reducing storage demands has another advantage. If you must maintain extensive index definitions for the extraction database, reducing the size of the snapshot file, for example, might leave enough disk space to store additional index structures.

Other target node considerations include:

- Verifying that target databases have privileges set to allow the copy process write access
- Establishing user accounts on target nodes to permit applications to execute locally, thus using the extraction databases rather than the source database
- Establishing system privileges and, for environments that use them, quotas, for all users and applications requiring access to target databases

5.1.1.3 Target Database Parameters

The transfer definition syntax enables you to specify values for the database parameters that are allowed in the target-db-param-clause of the CREATE TRANSFER statement. For example, you can specify a limit to the number of users who can concurrently access the target database. You can also specify the number of buffers or disable writing to the database snapshot file. The transfer definition syntax allows only those database parameters that are listed in the target-db-param-clause. Replication Option does not allow all CREATE DATABASE statement parameters permitted by SQL.

You can rename the target table in an extraction or extraction rollup transfer by specifying a target table with a name different from the name of the source table. You can roll up several source tables into a single table on the target.

In this example (which is a fragment of a CREATE TRANSFER statement), EXEMPT_EMPLOYEES1 in database DB1, EXEMPT_EMPLOYEES2 in database DB2, and EXEMPT_EMPLOYEES3 in database DB3, roll up into a single target table, ALL_EXEMPT_EMPLOYEES:

```
   .
   .
   .
cont> MOVE TABLES
cont>   SELECT EXEMPT_EMPLOYEES1 FROM DB1.EMPLOYEES
cont> UNION ALL
cont>   SELECT EXEMPT_EMPLOYEES2 FROM DB2.EMPLOYEES
cont> UNION ALL
cont>   SELECT EXEMPT_EMPLOYEES3 FROM DB3.EMPLOYEES
cont> INTO ALL_EXEMPT_EMPLOYEES
   .
   .
   .
```

The preceding example merged three source tables into a single target table. The next example shows how to transfer the source tables into separate target tables.

```
   .
   .
   .
cont> MOVE TABLES
cont>   SELECT EXEMPT_EMPLOYEES1 FROM DB1.EMPLOYEES
cont>     INTO EMPLOYEES1
cont>   SELECT EXEMPT_EMPLOYEES2 FROM DB2.EMPLOYEES
cont>     INTO EMPLOYEES2
cont>   SELECT EXEMPT_EMPLOYEES3 FROM DB3.EMPLOYEES
cont>     INTO EMPLOYEES3
   .
   .
   .
```

5.1.1.4 Using the Wildcard Character

In an extraction transfer, you can refer to tables, columns, and views implicitly by using the wildcard character (*). The following examples are applicable to any type of database:

```
MOVE TABLES *
MOVE TABLES SELECT * FROM EMPLOYEES
MOVE VIEWS *
```

SQL supports a multilevel naming convention for Rdb called multischema, consisting of database, catalog, schema, and object name. If wildcards are used with database, catalog, schema, or object names, the wildcard must be to the right of the name. For example, all the following are valid uses of the wildcard character:

```
MOVE TABLES *
MOVE TABLES "*.*".*.*
MOVE TABLES *.*
MOVE TABLES "DB1.CORPORATE_DATA".*
MOVE TABLES CORPORATE_DATA.*
MOVE TABLES "DB1.CORPORATE_DATA".PERSONNEL.*
MOVE TABLES PERSONNEL.*
MOVE VIEWS  "DB2.ADMINISTRATION".*
```

The following *is not* a valid use of the wildcard character:

```
SQL> MOVE TABLES *.CANDIDATES
```

In an extraction rollup transfer, you must name the tables, columns, and views explicitly. You cannot use MOVE TABLES * or SELECT * in the rollup-move-clause.

5.1.1.5 Database Aliases

You can use database aliases in a CREATE TRANSFER statement for extraction transfers, but you can include only one for each transfer. You must use database aliases in the rollup-move-clause for an extraction rollup transfer.

5.1.1.6 Location of Copy Process Log Files

You might want to locate all copy process logs in a single directory. Such a strategy allows for central control and file management and prevents important log files from getting lost in other directories. If a directory is specified in the copy process log file specification, the file will be created there. If not, the default directory for the copy process log is the home directory of the transfer definer.

5.1.1.7 Using Command Procedures

You should consider whether or not to include prologue and epilogue command procedures in your transfer definition. These command procedures allow you to perform many tasks as part of the CREATE TRANSFER statement, such as the following:

- Test for the success or failure of the transfer execution
- Execute other command procedures
- Invoke interactive SQL or run an SQL application
- Invoke the Mail utility
- Integrate metadata into the data dictionary
- Create indexes, constraints, triggers, and other objects in the extraction and extraction rollup databases

For more information on prologue and epilogue command procedures on OpenVMS, see Chapter 7.

5.1.2 Selecting Subsets of Rows to Transfer

Before you create the extraction or extraction rollup transfer, you need to decide what subset of the source database you want to copy to the target. You select the tables you want copied to the target, and you select the columns and rows in each of those tables.

For example, suppose that the New Hampshire facility normally provides employee job status reports for all departments located there. The New Hampshire facility requests data about employees and their corresponding job history information from the PERSONNEL database, as shown in Figure 5–1.

Figure 5–1 Identifying Rows for an Extraction

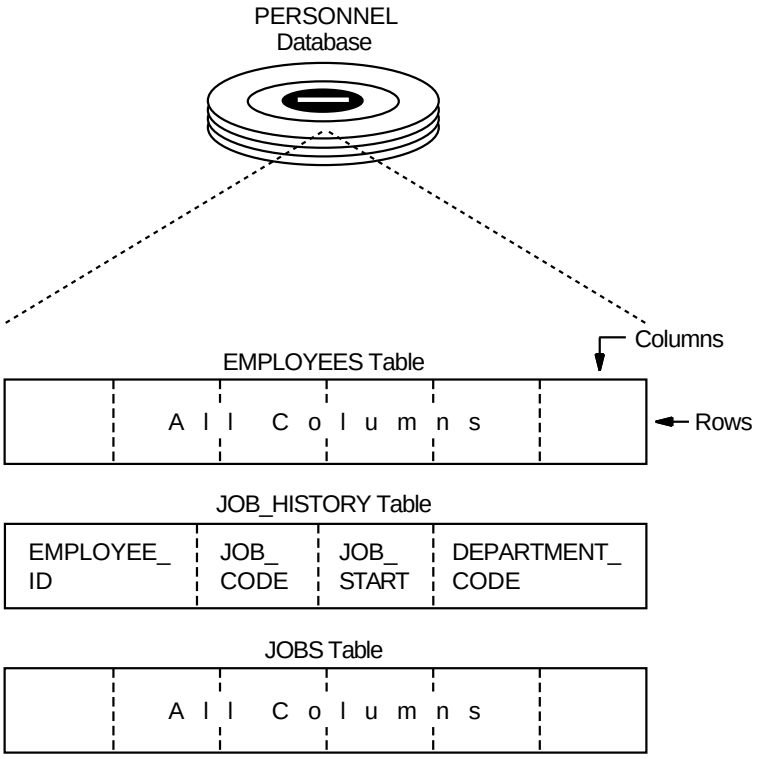


Figure 5–1 shows the identification of tables and columns from the PERSONNEL database that the New Hampshire facility needs to complete the job status reports. Employees of the New Hampshire facility live either in New Hampshire or Massachusetts. The STATE column in the EMPLOYEES table contains either NH or MA. All employees who work in New Hampshire might have ID numbers between 00164 and 00199. The New Hampshire facility requires the columns from three tables: EMPLOYEES, JOB_HISTORY, and JOBS.

5.1.3 Forming Select Expressions

After identifying the selection criteria for the database extraction, you can translate these requirements into select expressions needed by the transfer definition. In the select expressions you specify the tables and columns you want to transfer.

The following three clauses show a more formal representation of the selection criteria needed to retrieve rows required by the New Hampshire facility:

```
SELECT * FROM EMPLOYEES WHERE STATE = 'NH' OR STATE = 'MA'

SELECT * FROM EMPLOYEES WHERE EMPLOYEE_ID BETWEEN '00164' AND '00199'

SELECT EMPLOYEE_ID, JOB_CODE, JOB_START, DEPARTMENT_CODE
       FROM JOB_HISTORY WHERE EMPLOYEE_ID BETWEEN '00164' AND '00199'
```

To create an extraction database, you transfer columns, tables, and views, using the move-tables-clause. You can also transfer views for an extraction with the move-views-clause. To create an extraction rollup database, you transfer columns, tables, and views, using the rollup-move-clause. Refer to Chapter 9 for a description of the rules for use of these clauses.

The format of the select expression that can be included in the transfer definition is more limited than those allowed by SQL. Refer to Chapter 9 for details about forming a valid select expression in the MOVE clause.

Because you cannot use certain expressions to identify required rows, the resulting groups of rows transferred to the extraction database might be larger than the number of extracted rows users will actually use. However, local users on the target node can then include precise select expressions accessing the extraction database.

5.1.4 Transferring Views

You can transfer views for both extraction and extraction rollup databases. When you want to transfer views for an extraction database, you can use either of two clauses in the CREATE TRANSFER statement:

- The move-views-clause

  When you specify the MOVE VIEWS view-name or MOVE VIEWS * clause, Replication Option transfers the definition of the views and creates the same views in the target database. In addition to using the move-views-clause, you must explicitly or implicitly name all the tables, views, and columns on which the transferred view definitions are based. The transfer will fail if the underlying tables, views, and columns are not also transferred.

- The move-tables-clause

  When you specify the MOVE TABLES view-name clause, Replication Option creates a real table (rather than a view) in the target database that has the same definition and data as the named view. You do not need to transfer tables and views on which the source view is based. Once a view becomes materialized as a base table, it is no longer dependent on other tables and views.

When you want to transfer views for an extraction rollup database, only use the rollup-move-clause. You can use an epilogue command procedure to define views on the extraction rollup database.

Chapter 7 contains information about using epilogue command procedures on OpenVMS.

You can materialize a view with computed columns by naming it in the MOVE TABLES clause of a CREATE TRANSFER statement. Materializing a view means that a view in a source database becomes a real table (one with actual data stored in it) in a target database. In the target database, it is no longer defined to be a view. For example:

```
CREATE TRANSFER DATA_WAREHOUSE TYPE IS EXTRACTION
   MOVE TABLES
      SELECT * FROM A_VIEW
          . . . ;
```

For a view in an Rdb database to be transferred, Replication Option requires you to have SELECT privilege for the view. You must also have SELECT privilege for the table to which the view refers. If you do not have SELECT privilege for views you specify in the move-views-clause of a CREATE TRANSFER statement and for the tables to which the views refer, the transfer will fail. You need to obtain SELECT privilege for these views before including them in a CREATE TRANSFER statement.

For data accessed through one of the transparent gateway products, Replication Option does not check for SELECT privilege for views.

5.1.5 Creating and Starting an Extraction or an Extraction Rollup Transfer

Example 5–1 illustrates how to attach to a source database and interactively enter the CREATE TRANSFER statement on OpenVMS. The select expression for the JOB_HISTORY table selects all employees whose identification numbers are from a specific range. The select expression for the EMPLOYEES table identifies the rows of employees from a particular range of identification numbers.

Example 5–1 Extraction Transfer Definition on OpenVMS

```
SQL> ATTACH 'FILENAME PROD$DISK:[ADMIN.PERSONNEL]PERSONNEL';
SQL> CREATE TRANSFER EMPLOYMENT TYPE IS EXTRACTION
cont> MOVE TABLES
cont>   SELECT
cont>        JH.EMPLOYEE_ID,
cont>        JH.JOB_CODE,
cont>        JH.JOB_START,
cont>        JH.DEPARTMENT_CODE
cont>        FROM JOB_HISTORY JH
cont>   WHERE
cont>        EMPLOYEE_ID BETWEEN '00164' AND '00199',
cont>   SELECT *
cont>        FROM EMPLOYEES
cont>   WHERE EMPLOYEE_ID BETWEEN '00164' and '00199',
cont>   SELECT *
cont>        FROM JOBS
cont>   TO NEW FILENAME REMNOD::WORK$DISK:[PROD.PERSONNEL]NH_EMPLOYMENT
cont>   NUMBER OF USERS IS 10
cont>   SNAPSHOTS DISABLED
cont>   LOG FILE IS DISK$A:[TESTER]EMPLOYMENT.LOG
cont>   COMMENT IS 'Employment information copied to NH location'
cont>   ;
```

After you have created a transfer, you can start it immediately using the START TRANSFER NOW statement. There are several sections in this handbook that further discuss the following topics around creating and starting transfers:

- Chapter 6 discusses starting a transfer by defining a schedule that determines the time and the frequency of extraction and extraction rollup transfers.
- Chapter 4 discusses using an extraction transfer to forward to a target site only the database definitions that your applications require.
- Chapter 4 also describes how to prepare to distribute data from a source database.
- Chapter 9 provides complete details about the CREATE TRANSFER syntax, including optional clauses and examples. After you have addressed the details necessary to run the transfer successfully, you can complete the transfer definition using the options in the CREATE TRANSFER statement syntax.

5.2 Replication Transfers

Replication Option can either create the target database or use a database that already exists. When a replication transfer creates the target database, that database is called a replication database. You can replicate all the tables from the source database or a select set of those tables. You can replicate all the data in a given table or you can select only the data you need. View definitions may also be transferred. Replication can be done only from a single source database for each transfer. However, you can have multiple replication transfers from the same source database to the same target database. This can be useful if you have some tables that need to be updated more frequently than other tables in the target database.

Replication Option creates the replication database during the initial execution of a replication transfer. During each subsequent execution of the replication transfer, Replication Option forwards to the replication database all insertions, updates, and deletions made to selected rows in the source database in the interim. This is done either on demand or according to a predefined schedule. Unlike extraction and extraction rollup transfers, Replication Option maintains ties between the source database and the target replication database.

To create a replication database, you use the REPLICATION option of the CREATE TRANSFER statement.

Sections 5.2.1 through 5.2.5 describe how to create a replication transfer definition. The examples refer to the PERSONNEL database.

5.2.1 Replication Transfer Definition Elements

To enter the definition for a replication transfer, use the CREATE TRANSFER statement. Elements of a transfer definition include the following:

- The identification of the source database from which the data definitions and the data are read. You can have only a single source database for replication transfers.
- The identification of the target database into which the definitions and data will be written.
- A specification of the tables to be transferred.
- Optional expressions to select specific rows and columns from these tables.
- Optional parameter values for the creation of the target database.
- An optional log file for recording activity during the transfer.

- Optional prologue and epilogue procedures to be executed by Replication Option before and after the actual database transfer executes.

5.2.1.1 Identifying the Source Database

The source node for a replication transfer must be the same node or cluster on which Replication Option is installed and the replication transfer defined. The source node cannot be a remote node.

The CREATE TRANSFER statement does not explicitly name the source database. Instead, the source database is established when you first attach to the database. There is only one source database for a replication transfer. The following example shows an SQL ATTACH statement that attaches to an Rdb database:

```
SQL> ATTACH 'FILENAME DISK2:[DEPT2]PERSONNEL';
```

The format of an ATTACH statement and the valid parameters are documented in the Oracle Rdb SQL Reference Manual. The Rdb attach string names a full or partial OpenVMS file specification. Oracle Corporation recommends you use OpenVMS logical names rather than complete attach strings or file specifications, so you can move database files without altering the associated transfer definitions. You must use a SET statement to direct SQL to leave the logical names intact when stored in a transfer definition, as shown in the following example:

```
$ DEFINE RDB_SOURCE "DISK2:[DEPT2]PERSONNEL"
$ SQL
SQL> SET LOGICAL_NAME TRANSLATION FOR TRANSFER OFF;
SQL> ATTACH 'FILENAME RDB_SOURCE';
```

Alternatively, you can invoke the database by using a data dictionary path name.

After you start a replication transfer, update transactions on the Rdb source database incur added I/O overhead. Inserting, updating, or deleting rows from replicated source tables causes the rows to be copied into the RDB$CHANGES table for later transfer to one or more target databases. There is also additional I/O writing to the recovery-unit journal (.RUJ) file and, if used, the after-image journal (.AIJ) file. On a source database where there is intense update activity, a noticeable decrease in throughput may be experienced when replication is enabled.

Note

If you define a replication transfer that is infrequently executed, you may see that the size of the RDB$CHANGES system table in the source database continues to grow, even as the transfer remains unexecuted.

This is because changes made to the Rdb source database continue to be tracked and recorded while Rdb and Replication Option await the time when this transfer is next executed. For this reason, Oracle Corporation recommends the following, depending on your needs:

- Drop unneeded replication transfer definitions.
- If a replication transfer is needed, but only infrequently, run it on a regular schedule anyway (even if you know that no changes will take place) so that the RDB$CHANGES table can be updated and its size kept manageable.

Single-file source databases experience roughly twice the decreased throughput of a multifile database. Therefore, if you are replicating a single-file database and find that you have reached the limit of updates that can be applied in a given time, execute both of the following procedures:

- Convert the source into a multifile database.
- Spread the database files over several disk volumes to help achieve more parallelism and reduce disk contention.

In order for Rdb databases to be used as sources for replication transfers, you must ensure that the source database has snapshot files enabled. Otherwise, when you execute a replication transfer, Replication Option can encounter a deadlock error on snapshot cursor 0 if you have the source database set to SNAPSHOT IS ENABLED DEFERRED or SNAPSHOT IS DISABLED. This deadlock can occur in the Replication Option copy process leading to a failure of the transfer execution. The following example shows the error messages you can receive if this problem occurs:

```
%RDB-E-DEADLOCK, request failed due to resource deadlock
-RDMS-F-DEADLOCK, deadlock on snapshot cursor 0
```

You must have SNAPSHOTS set to ENABLED IMMEDIATE in the source Rdb database to avoid these deadlocks with replication transfers.

5.2.1.2 Target Nodes and Directories

You must ensure that the target nodes and directories that will receive the initial or reinitialized replication databases exist and that they have sufficient storage space to receive the data.

If you need to reinitialize a replication transfer, a transfer that creates the target database files, a new replication database is created in the target directory.

If the target database will be created on OpenVMS, you must ensure that the target directory has sufficient space to store more than one version of the replication database.

Additionally, if many new rows are stored in the replication database, the database file itself will require increased storage space.

Do not disable writing to the Rdb snapshot file of the replication target database. If you disable the snapshot file, Rdb grants a special read/write transaction to all subsequent read-only transactions. This special mode allows your transaction shared access with other read-only transactions; however, users may encounter lock conflicts with other active update transactions.

To prevent read locks from being placed on rows in the Rdb database file, you should specify a small allocation for the snapshot file. If you expect many applications to retrieve data concurrently from the replication database, make sure that the snapshot file is enabled to increase concurrency.

Reducing storage demands has another advantage. If you must maintain extensive index definitions for the replication database, reducing the size of the snapshot file, for example, might leave enough disk space to store additional index structures.

Other target node considerations include:

- Verifying that target databases have protection set to allow the copy process write access
- Establishing user accounts on target nodes to permit applications to execute locally, thus using the replication database rather than the source database
- Establishing system privileges and quotas for all users and applications requiring access to target databases

5.2.1.3 Target Database Parameters

The transfer definition syntax allows you to specify values for all database parameters that are allowed in the target-db-param-clause of the CREATE TRANSFER statement. This is meaningful only when directing Replication Option to create the target database files. Currently, this can be done only if the target database system is Rdb. For example, you can specify a limit to the number of users who can concurrently access the target database. You can also specify the number of buffers or disable writing to the database snapshot file. The transfer definition syntax allows only those database parameters that are listed in the target-db-param-clause. It does not allow all the parameters that are allowed in the SQL CREATE DATABASE statement.

5.2.1.4 Wildcard Character

In a replication transfer, you can refer to tables, columns, and views implicitly by using the starred notation as in MOVE TABLES *, SELECT *, or MOVE VIEWS *. You can also refer to tables, columns, and views explicitly by naming each table, column, or view. See Section 5.1.1.4 for examples of valid uses of the wildcard character.

5.2.1.5 Database Aliases

You can use database aliases in a CREATE TRANSFER statement for replication transfers, but you can include only one for each transfer because there is only one source database.

5.2.1.6 Location of Copy Process Log Files

It might be helpful to locate all copy process logs in a single directory. This strategy allows for central control and file management and prevents important log files from getting lost in other directories. If a directory is specified in the copy process log file specification, the file will be created there. If not, the default directory for the copy process log is the home directory of the transfer definer.

5.2.1.7 Using Command Procedures

You should consider whether or not to include prologue and epilogue command procedures (also known as scripts) in your transfer definition. These command procedures allow you to perform additional tasks as part of the CREATE TRANSFER statement, such as the following:

- Execute other command procedures
- Invoke interactive SQL or run an SQL application program
- Invoke the Mail utility
- Integrate metadata into the data dictionary
- Create indexes and grant access privilege to the replicated tables

For more information on pre-transfer and post-transfer command procedures on OpenVMS, refer to Chapter 7.

5.2.2 Identifying Rows

Before you create the replication transfer, you need to decide what subset of the source database you want to copy to the target. You select the tables you want copied to the target, and you select the columns and rows in each of those tables.

All columns that you name in the search condition of the WHERE clause must also be named in the SELECT select-list-clause. For example, the PERSONNEL database contains information about all departments in an organization. Each department has employees assigned to it. One department might be responsible for monitoring the salary activity of all the engineering departments. By replicating this information to a local, target database, the monitoring department can automatically receive updates made to the engineering section of the source database.

In Example 5–2, the monitoring department is interested in rows from three tables in the PERSONNEL database: JOB_HISTORY, SALARY_HISTORY, and EMPLOYEES. Within those rows selected, the department requires only a few columns. Having identified the required data, the department forms select expressions and specifies the required columns:

- From the JOB_HISTORY table, the department selects rows that contain the following DEPARTMENT_CODE values:

  ELEL
  ELGS
  ELMC
  ENG

  The required columns are:

  EMPLOYEE_ID
  DEPARTMENT_CODE
  JOB_CODE

- From the SALARY_HISTORY table, the department is interested in rows that fall into a particular time range. Either the SALARY_START column in the row should contain a value in the range 01-JUL-1976 to 30-JUN-1977 or the SALARY_END column should contain a value in the range 01-JUL-1976 to 30-JUN-1977. The columns required from this table are:

  EMPLOYEE_ID
  SALARY_AMOUNT
  SALARY_START
  SALARY_END

- From the EMPLOYEES table, the three columns that will be copied are:

  EMPLOYEE_ID
  LAST_NAME
  FIRST_NAME

5.2.2.1 Effect of Inserting, Updating, and Deleting Rows

In a replication transfer, the copy process reads information about changes to the source database and applies those changes to the target database. You can define transfers to be selective about which rows you want in the target database. If, for example, you want rows where a column called DAYS_OVERDUE contains a value of 90 or more, consider the following:

- If you insert a row in the source database where the value of DAYS_OVERDUE is 105, that row will be inserted in the target database. If you insert a row where the value of DAYS_OVERDUE is 31 or is unspecified, that row will not be transferred to the target because it does not meet the selection criterion.
- If you delete a row from the source database and its DAYS_OVERDUE value is 90, that row will be deleted from the target database. If you delete a row with a DAYS_OVERDUE value of 0, nothing is done at the target because the row, failing to match the selection criterion, should not be there.
- If the DAYS_OVERDUE column initially contains 99 and is updated to contain 100, the row is updated at the target site. The before and after values of the selection column both match the selection criterion.
- If DAYS_OVERDUE initially contains 89 and is updated to contain 90, the row is inserted in the target database, not updated. An update in the source database results in an insertion in the target database because the initial value of the column indicates the row does not exist in the target database.
- If DAYS_OVERDUE is updated from 95 to 85, the row is deleted from the target database. An update in the source database results in a deleted row in the target database because the initial value meets the selection criterion but the final value does not.

5.2.3 Forming Select Expressions

After identifying the selection criteria for the replication database, you can translate these requirements into select expressions needed by the transfer definition. In the select expression, you specify the tables, columns, and rows you want to transfer.

To create a replication database, you transfer columns, tables, and views from the source database to the target database. You use the move-tables-clause of the CREATE TRANSFER statement to transfer columns and tables.

The format of the select expression that can be included in the transfer definition is more limited than the select expression allowed by SQL. Refer to the CREATE TRANSFER statement in Chapter 9 for details about forming a valid select expression in the MOVE clause.

Because you cannot use certain expressions to identify required rows, the resulting groups of rows transferred to the replication database might be larger than the number of rows users will actually use. However, local users on the target node can then include precise select expressions when using SQL to access the replication database.

There are limits to the number of transfers and selection criteria you can specify for each source database for a replication update transfer. If the limits are exceeded, Rdb returns the error message "RDB$_IMP_EXC, implementation limit exceeded" when the application compiles its request. If limiting the number of replication update transfers is not feasible, consider designing your source database and applications so that the number of selection comparisons for each transfer is minimized.

5.2.4 Transferring Views

You can transfer views for a replication database by using the move-views-clause. (You cannot transfer views for a replication database with the move-tables-clause.) In a replication transfer, Replication Option transfers the definition of the view when you specify MOVE VIEWS view-name or MOVE VIEWS *. You must also explicitly or implicitly name in the CREATE TRANSFER statement all the tables, views, and columns on which the transferred view definitions are based. The transfer will fail if the underlying tables, views, and columns are not named.

Transferring views has the following limitations:

- COMPUTED BY columns

  Replication Option transfers those COMPUTED BY columns that depend only on column values that can be computed from the values in the current data row. This rules out the use of aggregate expressions that require computation involving all rows in the table. Views that contain unsupported COMPUTED BY columns cannot be transferred. If views that contain unsupported COMPUTED BY columns are specified in the move-views-clause of a CREATE TRANSFER statement, the transfer definition fails. You need to redefine the transfer, excluding the views that refer to the unsupported COMPUTED BY columns.

- SELECT privilege for the view

For a view in an Rdb database to be transferred, Replication Option requires that the user have SELECT privilege. You must also have SELECT privilege for the table to which the view refers. If you do not have SELECT privilege for views you specify in the move-views-clause of a CREATE TRANSFER statement and for the tables to which the views refer, the transfer will fail. You need to obtain SELECT privilege for these views before including them in a CREATE TRANSFER statement.

5.2.5 Creating and Starting a Replication Transfer

Because a source database cannot be specified explicitly in the CREATE TRANSFER statement, you must attach to the source database containing the rows to be selected by the transfer definition before entering the CREATE TRANSFER statement. Because Replication Option does not store transfer definitions in the data dictionary, you will get the same result whether you invoke the source database using its file name or its path name.

Example 5–2 illustrates how to attach to a source database and interactively enter the CREATE TRANSFER statement. The SALARY_HISTORY table's select expression identifies all salary changes that occur during a specific fiscal year. The select expression for the JOB_HISTORY table specifies only the JOB_HISTORY rows that belong to one of four engineering departments. The replication database is called DEPARTMENT_INFO on the target directory of the monitoring department.

Example 5–2 Replication Transfer Definition

```
SQL> ATTACH 'FILENAME PROD_PERSONNEL';
SQL> CREATE TRANSFER DEPARTMENT_INFO TYPE IS REPLICATION
cont>  MOVE TABLES
cont>    SELECT JH.EMPLOYEE_ID,
cont>         JH.DEPARTMENT_CODE,
cont>         JH.JOB_CODE
cont>         FROM JOB_HISTORY JH
cont>    WHERE
cont>         JH.DEPARTMENT_CODE = 'ELEL' OR
cont>         JH.DEPARTMENT_CODE = 'ELGS' OR
cont>         JH.DEPARTMENT_CODE = 'ELMC' OR
cont>         JH.DEPARTMENT_CODE = 'ENG',
```

(continued on next page)

Example 5–2 (Cont.) Replication Transfer Definition

```
cont>    SELECT SH.EMPLOYEE_ID,
cont>         SH.SALARY_AMOUNT,
cont>         SH.SALARY_START,
cont>         SH.SALARY_END
cont>         FROM SALARY_HISTORY SH
cont>    WHERE
cont>         SH.SALARY_START BETWEEN '01-JUL-1976' and '30-JUN-1977'
cont>         OR SH.SALARY_END BETWEEN '01-JUL-1976' and '30-JUN-1977',
cont>    SELECT E.EMPLOYEE_ID,
cont>         E.LAST_NAME,
cont>         E.FIRST_NAME
cont>         FROM EMPLOYEES E
cont>  TO NEW FILENAME DEPARTMENT_INFO
cont>  NUMBER OF USERS IS 10
cont>  LOG FILE IS MONITOR_DEPT_INFO
cont>  COMMENT IS 'Replication of department-specific information'
cont>  ;
```

After you have created a transfer, you can start it immediately using the START TRANSFER NOW statement. There are several sections in this handbook that further discuss the following topics around creating and starting transfers:

- Chapter 6 discusses starting a transfer by defining a schedule that determines the time and the frequency of replication transfers.
- Chapter 4 discusses using a transfer to forward to a target site only the database definitions that your applications require.

  You can also create a replication database in a target directory from a source database that itself contains no data yet. Replication Option then will forward the selected data to the target database after the data is stored in the source database.
- Chapter 4 also describes how to prepare to distribute data from a source database.
- Chapter 9 provides complete details about the CREATE TRANSFER syntax, including optional clauses and examples. After you have addressed the details necessary to run the transfer successfully, you can complete the transfer definition using the options in the CREATE TRANSFER statement syntax.

5.3 Character Set Considerations with Database Transfers

Rdb supports several character sets (in support of the Multivendor Integration Architecture (MIA)) in addition to Digital Equipment Corporation MCS. This includes support for the Kanji Character Set as defined by JIS X0208-1983. For information about the supported character sets, see the Oracle Rdb SQL Reference Manual for the version of Oracle Rdb you are using.

5.3.1 Terminal Device Characteristics Restrict Use of Character Sets

A problem may occur when you attempt to display characters on a terminal that is operating in a character set other than the one in which your data is stored. For example, if you have a terminal that displays Kanji characters, it is unlikely to correctly display Korean characters. You might see this problem when you execute the SQL SHOW TRANSFER statement or when you display error messages or log files containing embedded database object names.

The following sections discuss the factors you need to take into account when defining transfers to or from different databases with different character sets or different Rdb versions.

5.3.2 Effects of Transferring Database Objects with Different Character Sets

When Replication Option transfers to an existing database, it is possible to transfer database tables and other objects that use different identifier character sets than those used by the tables and objects already stored in the existing database.

Replication Option does not prevent this mix of character sets. However, the results in the target databases will not make sense when they are displayed. For example, if you transfer data from a database that uses the DEC_KANJI character set to a target database with tables and objects that use the DEC_ MCS character set, Replication Option performs the transfer, but you will not be able to understand the data from the target database.

5.3.3 Using Different Versions of Rdb for Source and Target Databases

When you transfer data and both source and target databases use Rdb Version 4.2 or higher, Replication Option creates the domain in the target database with the same character set subtype as the domain has in the source database. Versions of Rdb earlier than 4.2 do not support character set subtypes.

If the transfer is to an already existing target database, the database default character set is already established. If the CREATE TRANSFER statement is to create the target database, you can specify the default character set for the target database.

Under these circumstances, Replication Option creates domains in the target database without specifying the character set for the data in the domain. When a domain is created without passing a character set identifier for the domain, Rdb gives the domain the character set identifier of the database default character set. If you require a different character set for the domain, change the domain attribute within an epilogue procedure associated with the transfer definition.

For information on creating epilogue procedures on OpenVMS, see Chapter 7.

5.4 Using Multischema Databases as Source Databases

Some Rdb databases have an attribute called multischema. A multischema database consists of catalogs, each of which contains one or more schemas. You can use the same table name in multiple schemas, but you must qualify table names with their catalog and schema names. An example of a qualified table name from the sample CORPORATE_DATA database is ADMINISTRATION.PERSONNEL.EMPLOYEES.

Rdb also assigns an external name to all tables. If a table name is used in two different schemas, the table name in the second schema will have a 1 appended to the end of its name. For example, in CORPORATE_DATA there is a DEPARTMENTS table in both the PERSONNEL and ACCOUNTING schemas. The external name for ACCOUNTING.DEPARTMENTS is DEPARTMENTS1.

You include the catalog-schema name string as part of the CREATE TRANSFER statement. This information is stored in the transfer database, but only the external name is displayed when you issue a SHOW TRANSFER statement. The following example is created using the full multischema syntax (ADMINISTRATION.ACCOUNTING.DEPARTMENTS). However, the display from a SHOW TRANSFER statement lists only the external name (DEPARTMENTS1):

```
SQL> ATTACH 'FILENAME CORPORATE_DATA MULTISCHEMA IS ON';
cont> CREATE TRANSFER DEPTS TYPE IS EXTRACTION
cont> MOVE TABLE ADMINISTRATION.ACCOUNTING.DEPARTMENTS
cont> INTO DEPT2
cont> TO NEW FILENAME TRANS.RDB;
SQL> SHOW TRANSFER (DEFINITION) DEPTS
```

```
Definition for transfer DEPTS:
Definer                 ABUSHAKRA
Type                    EXTRACTION
From                    DISK1:[CORP]CORPORATE_DATA.RDB
To                      New TRANS.RDB
No log file
No prologue file
No epilogue file
Move table
    Select All
            DEPARTMENT_CODE,
            DEPARTMENT_NAME,
            MANAGER_ID,
            BUDGET_PROJECTED,
            BUDGET_ACTUAL
        From DEPARTMENTS1
        Into DEPT2
```

Multischema is an optional feature; it is not set on by default. To create transfers from a multischema database, you must set multischema on.

If, in addition to naming catalogs and schemas, you must name databases (by alias) to identify a particular table in a transfer definition, you must also enable ANSI quoting. For example:

```
SQL> ATTACH 'ALIAS DB1 FILENAME CORPORATE_DATA1 MULTISCHEMA IS ON';

SQL> ATTACH 'ALIAS DB2 FILENAME CORPORATE_DATA2 MULTISCHEMA IS ON';

SQL> SET ANSI QUOTING ON;

SQL> CREATE TRANSFER CORP_ROLLUP TYPE IS EXTRACTION ROLLUP
cont> MOVE TABLES
cont> SELECT * FROM "DB1.ADMINISTRATION".PERSONNEL.EMPLOYEES,
cont> SELECT * FROM "DB2.ADMINISTRATION".ACCOUNTING.DEPARTMENTS
cont> INTO DEPT3
cont> TO NEW FILENAME TRANSFER.RDB;
SQL>
```

For a description of how to identify tables and views in a multischema database, see the documentation of the multischema-name-clause in Section 9.4.

5.5 Transfers to Distributed Option for Rdb and Rdb Transparent Gateway Databases

The following sections discuss special considerations for Distributed Option for Rdb and Rdb Transparent Gateway databases used as the target of Replication Option transfers.

When transferring into a Distributed Option database, keep the following in mind. Distributed Option allows you to define a virtual database that appears to be a single database, but in reality is a link to several underlying databases. Only one of those underlying databases can be written into using Replication Option. To do so you must declare one of the database links to be the default link. Normally, you could do this either when attaching to the Distributed Option database or when creating it. However, if you intend to use Replication Option to transfer into the Distributed Option database, you must declare the default link when the database is created.

Transfer definitions entered with the CREATE TRANSFER statement must be formatted in specific ways. Statements used to attach to the target database, and associated configuration files (if used), must be set up in specific ways. For replication transfers, these statements and files must be changed after execution of the initial transfer.

For certain classes of target database, you must manually add indexes to the target tables.

All these topics are discussed in the following sections.

5.5.1 Manually Create the DDAL$TRANSFER_INFO Table

To manage the transfer process, Replication Option uses two or three tables in a target database: DDAL$TRANSFER_INFO, DDAL$TRANSFER_OWNR, and DDAL$INDEXES. The DDAL$INDEXES table is needed to perform shared-write replication transfers; otherwise, it is not needed. See Section 8.1.3 for an explanation of shared-write transfers. If you are transferring directly to an Rdb database, or if you are transferring to an Oracle database or to a Distributed Option database, Replication Option will create these tables for you during the execution of a transfer. For any other target database type, prior to initially executing a transfer, you must manually create the DDAL$TRANSFER_INFO table and add an index on one of its columns. Later, during execution of the transfer, Replication Option will create the DDAL$TRANSFER_OWNR and DDAL$INDEXES tables, as needed.

To create the DDAL$TRANSFER_INFO table, use the SQL interface to attach to the target database. Then enter the following statements:

```
CREATE TABLE DDAL$TRANSFER_INFO(
   DDAL$TRANSFER_TSER INTEGER,
   DDAL$TRANSFER_TIME DATE,
   DDAL$TRANSFER_NODE CHAR(255),
   DDAL$TRANSFER_NAME CHAR(31),
   DDAL$TRANSFER_ID   SMALLINT,
   DDAL$TRANSFER_BUSY CHAR(1));
COMMIT;
```

The Rdb SQL interface, not the native table definition tool of the target system, should be used to create this table. If you were to use the native table definition tool, the column data types might not match. For example, in Microsoft Access version 2.0, you can create a column of data type Text or Memo for character data. Neither of these maps to the Rdb SQL CHAR data type. By contrast, when the definition is entered using the Rdb SQL interface, the CHAR data type is preserved, even though the column is listed in Microsoft Access as having a Text data type.

5.5.2 Manually Create the DDAL$TRANSFER_IDX1 Index

After you create the DDAL$TRANSFER_INFO table using the Rdb SQL interface, you must add a unique index, DDAL$TRANSFER_IDX1, on a column of that table. Except for Rdb Transparent Gateway to PC Data, use the Rdb SQL interface to add the index, as follows:

```
CREATE UNIQUE INDEX DDAL$TRANSFER_IDX1
  ON DDAL$TRANSFER_INFO(DDAL$TRANSFER_ID);
COMMIT;
```

For transfers using Rdb Transparent Gateway to PC Data, you cannot create or drop indexes. Therefore, you must use the target database system directly to create the index.

5.5.3 Add Access Privileges to Target Tables

Permission to read, write, and update the DDAL$TRANSFER_INFO table must be granted to the account or accounts under which Replication Option transfers will operate within the target database. Granting such privileges is done using the native tools in the target database systems.

This stipulation also applies to all other tables that will be created in the target database when the transfer executes. The account used to execute the transfer in the target system will require the necessary privileges. Remember to add privileges as needed to allow other accounts to access the target tables, including the DDAL$TRANSFER_OWNR and DDAL$INDEXES tables. This must be done after the transfer executes successfully once.

5.5.4 Create Transfer Definitions

When creating transfer definitions in which the target is a Distributed Option database or an Rdb Transparent Gateway database, you must include the WITH NO CHECKING and TO EXISTING clauses in the definition.

The WITH NO CHECKING clause suppresses checks performed by SQL that are currently valid only for Rdb target databases. The TO EXISTING clause ensures that Replication Option does not attempt to create the target database files. Replication Option can only create target database files for Rdb databases.

The following is an example of a CREATE TRANSFER statement using these clauses:

```
CREATE TRANSFER SALES TYPE IS EXTRACTION
    MOVE TABLES
        SELECT * FROM PROSPECTS
TO EXISTING FILENAME GATEWAY_TARGET WITH NO CHECKING
LOG FILENAME IS SALES_LOG;
```

The name GATEWAY_TARGET is assumed to be an OpenVMS logical name. The logical name, defined separately from the transfer definitions, would translate to the attach string necessary to attach to the target database. The attach string can list all the attach parameters directly or it can list some parameters and have the rest listed in a configuration file.

The following example shows a typical target database attach string:

```
/TYPE=ORACLE/CONFIG_FILE=PROD$DISK:[CONFIG_FILES]CONFIG.TGT
```

5.5.4.1 Limit to Number of Tables Per Transfer

Oracle and DB2 have a built-in limit to the number of active cursors they will support. Refer to the Distributed Option for Rdb documentation for information about these limits. This limit restricts how many tables can be named in a single CREATE TRANSFER statement. If you need to transfer more tables than the limit allows, create more than one transfer, and have each transfer manage a different set of source tables.

5.5.4.2 Data Type Limitations

Not all data types supported by a given database system are transportable to all other database systems. For example, the Rdb VARBYTE LIST data type currently has no counterpart in other database systems. When you create a transfer definition, SQL does not check the source data types and the capabilities of the target system to determine if the transfer can succeed or not. If you attempt to transfer data with a type that cannot be supported by the target system, you will receive a run-time error message at the time you execute the transfer. The transparent gateway products transform some data types from one form to another but there are some data types they will not accept. Refer to the Distributed Option for Rdb and the Rdb Transparent Gateway products documentation for information about supported data types.

5.5.5 Create the Configuration File

This step assumes your target attach string will name a configuration file, similar to that shown in the following example:

```
/DATABASE=PROD_TARGET
/USER=repl_opt
/PASSWORD=secret
/TABLES=(*,
         DDAL$TRANSFER_INFO/DBKEY_MODE=(DDAL$TRANSFER_ID)
        )
```

Note that entries in lowercase in the configuration file do not need to be enclosed in quotation marks.

This configuration file is not meant to show every parameter you might need to include. The part being highlighted here is how the Replication Option table is called out and one of its columns is designated as providing a database key (dbkey). The dbkey column must have unique values and can therefore be used to identify individual data rows. This is a prerequisite for being able to perform updates to existing rows. Refer to the transparent gateway documentation for complete information about database attach strings.

5.5.6 Create a DDAL.RC File

If you plan to allow shared writing of the target tables with other transfers or with local user application programs, you must enable shared writing for the transfers. To do so, create a DDAL.RC file in your home directory (SYS$LOGIN). The details for enabling shared writing are explained in Appendix B.

5.5.7 Execute the Transfer

At this point you are ready to execute the transfer(s). If they are extraction transfers, there are no additional steps needed. Extraction transfers can be repeated without any changes required to the configuration file or to the target database.

If you are executing replication transfers, execute them once so that the initial transfers have completed successfully. Following the initial transfers, there are more changes that must be made, as described in the following sections.

5.5.8 Modify the Configuration File

If you are transferring to an Oracle database using Rdb Transparent Gateway to Oracle, you can skip this section. It applies to all other Distributed Option and Rdb Transparent Gateway target databases.

If you intend to transfer from an Rdb database into a database using the REPLICATION method, Replication Option will need the ability to update rows in the target tables to reflect changes as they occur in the source tables. After successfully executing the initial replication transfer, you must edit the configuration file used to access the target database.

Each table transferred by the REPLICATION method will have one or two additional columns added to it in the target database: DDAL$DBKEY and, if performing a shared-write transfer, DDAL$ROW_OWNER. These columns form a unique key that is used to locate individual data rows to be updated or deleted. You must declare this key for each table by making one entry per transferred table in the configuration file. For example, if you were replicating the JOB_CODES table from an Rdb source database into a DB2 target database, you would need to use a target access string and configuration file similar to the ones shown in Example 5–3, Example 5–4, and Example 5–5.

Example 5–3 Sample Target DB2 Attach String

```
/TYPE=DB2/CONFIG_FILE=PROD$DISK:[CONFIG_FILES]CONFIG.TGT
```

The configuration file shown in Example 5–4 is for nonshared-write transfers.

Example 5–4 Sample Configuration File for Nonshared-Write Replication Transfer

```
/DATABASE=PROD_TARGET
/USER=data_dist
/PASSWORD=secret
/TABLES=
   (*,
   DDAL$TRANSFER_INFO/DBKEY_MODE=(DDAL$TRANSFER_ID),
   JOB_CODES/DBKEY_MODE=(DDAL$DBKEY)
   )
```

For shared-write transfers, add the DDAL$ROW_OWNER column, as shown in Example 5–5.

Example 5–5 Sample Configuration File for Shared-Write Replication Transfer

```
/DATABASE=PROD_TARGET
/USER=data_dist
/PASSWORD=secret
/TABLES=
   (*,
   DDAL$TRANSFER_INFO/DBKEY_MODE=(DDAL$TRANSFER_ID),
   JOB_CODES/DBKEY_MODE=(DDAL$DBKEY,DDAL$ROW_OWNER)
   )
```

5.5.9 Manually Create Indexes on Target Tables

Replication Option requires a number of columns to have unique values. If you are transferring to a database using Rdb Transparent Gateway to PC Data, you must add these indexes yourself using the tools native to your target database system.

If your target is a Microsoft Access database, you can add indexes in Table Design View. The following examples show the syntax you would use to create indexes in a Microsoft SQL Server database:

1. Index on the DDAL$TRANSFER_ID column in the DDAL$TRANSFER_ INFO table. This step is also described in Section 5.5.2.

   ```
   create unique index DDAL$TRANSFER_IDX1
    on DDAL$TRANSFER_INFO(DDAL$TRANSFER_ID)
   go
   ```

2. Index on the DDAL$TABLE_NAME column in the DDAL$INDEXES table, when shared writing is enabled.

   ```
   create unique index DDAL$INDEXES_IDX1
    on DDALINDEXES(DDAL$TABLE_NAME)
   go
   ```

3. Index on the DDAL$INDEX_NAME column in the DDAL$INDEXES table, when shared writing is enabled.

   ```
   create unique index DDAL$INDEXES_IDX2
    on DDAL$INDEXES(DDAL$INDEX_NAME)
   go
   ```

4. Index on the DDAL$DBKEY column in each transferred table when the REPLICATION method of transfer is used and shared writing is not enabled.

   Index name format: DDAL$KEY_IDXn_m

   The letters n and m represent numbers, with the lowest number for each being 1 and leading zeros not used. m is the transfer ID number, which you can determine by finding the transfer's row in the DDAL$TRANSFER_INFO table and looking at the DDAL$TRANSFER_ID column. The numbers n for a given transfer are assigned to tables in sequence, 1, 2, 3, and so forth. In the following example, an index is created on the JOBS table. It is the fifth table index created for transfer number 2.

```
create unique index DDAL$KEY_IDX5_2
 on JOBS(DDAL$DBKEY)
go
```

5. Index on the DDAL$DBKEY and DDAL$ROW_OWNER columns in each transferred table when the REPLICATION method of transfer is used and shared writing is enabled.

   Index name format: DDAL$INDEXn

   The letter n represents a number, with the lowest number being 1 and leading zeros not used. The number n is determined by examining the DDAL$INDEX_NUMBER column of the DDAL$INDEXES table, taking the maximum value found in that column, and adding one to it. If the table has no rows in it, start with the number 1.

```
create unique index DDAL$INDEX14
 on JOBS(DDAL$DBKEY,DDAL$ROW_OWNER)
go
```

   After the index has been created, record its existence in the DDAL$INDEXES table.

```
Insert into DDAL$INDEXES(DDAL$TABLE_NAME,
                         DDAL$INDEX_NAME,
                         DDAL$INDEX_NUMBER)
          values         ('JOBS',
                          'DDAL$INDEX14',
                          14)
go
```

5.5.10 Manually Drop Indexes After Transfer Reinitialization

There are occasions when replication transfers must be restarted from the beginning. At those times, you must reinitialize the transfer and then re-execute it. A reinitialized transfer to a Distributed Option target will first delete all the data transferred before and replace it with a fresh set of data from the source database.

Deletion and insertion of data typically can be done faster when indexes are not defined on the target tables. Because of this, Replication Option will normally drop target indexes on the transferred tables before deleting and reinserting data. Once that is done, it re-creates the target indexes. Other Replication Option behavior can be chosen if you define the DDAL$TARGET_INDEX_OPTIONS environment variable, described in Section B.19.

Transfers that use Rdb Transparent Gateway to PC Data are an exception. The CREATE INDEX and DROP INDEX statements are not supported to some targets. Therefore, Replication Option assumes it is not supported for all database systems accessed by that transparent gateway. As a result, Oracle recommends that, for performance reasons, you manually drop such indexes after a transfer is reinitialized and before it is re-executed. Then re-create the indexes manually after the transfer is complete. You must do this using the tools native to your target database system.

5.5.11 Multiple Transfers and Uniqueness of Object Names

Different database management systems can have other names and concepts for databases, ways to partition those databases, and names for database objects in those databases. This section is an example of these differences in Rdb, DB2, Oracle, and Sybase databases. This section points out naming conflicts that can occur and suggests ways to avoid these conflicts using these particular databases as examples.

All database systems require object names (tables, views, indexes, and so forth) within some environments to be unique. For Rdb, that environment is a physical database. For DB2 and Oracle, that environment is the database catalog. For Sybase, that environment is the master database. The database catalog may contain references to objects in several physical databases. Oracle refers to this environment as the table name space. DB2 refers to it as a subsystem.

Replication Option creates indexes with names of its own choosing. One such index is on a column in the DDAL$TRANSFER_INFO table. Another such index is on the DDAL$DBKEY column that appears in each replicated table. The names chosen by Replication Option are unique within each physical database but naming conflicts can occur within an Oracle or DB2 catalog.

The problem and the recommended method of avoiding the problem will be discussed later in this section.

Objects within a DB2 or Oracle catalog are identified by an object name and qualified by an authorization identifier. The authorization identifier is the name of the user account that creates the object. If you logged into one of these systems as user ROR and created a table named EMPLOYEES, an object named ROR.EMPLOYEES would be entered in the catalog.

When Replication Option attaches to an Oracle or DB2 database by means of a transparent gateway, with the access identifier USER=ROR, any objects created by Replication Option will be qualified with the name ROR. A problem can occur when you create two or more transfers, executed under the same USER account, and target different physical databases within the same catalog. For each transfer, the names will be qualified with the authorization identifier ROR. For each transfer, Replication Option will attempt to create indexes with the same name. This violates the uniqueness rule for catalogs.

To avoid this problem, you must create different user accounts for each transfer. This is not necessary if you have multiple transfers that target the same physical database. In that case, the name uniqueness within the physical database is guaranteed by Replication Option. You also do not need separate accounts if each transfer goes to a database in a different catalog from the other transfers.

5.5.12 Source Definition Changes

Neither Distributed Option for Rdb nor the Rdb Transparent Gateways support altering data definitions. Therefore, you must ensure that changes to source definitions are mirrored in the target definitions whenever the changes are made after the first execution of a transfer.

6
Working with Transfers

This chapter supplements the discussion of extraction, extraction rollup, and replication transfers provided in Chapter 5. The previous chapter describes how to create transfer definitions and select data for target extraction, extraction rollup, and replication databases. This chapter extends the description of transfers with discussions about scheduling transfers, transfer states, and running and stopping transfers, among other topics.

In addition, the chapter provides guidelines for evaluating current applications that need modification before they are used in extraction, extraction rollup, and replication databases.

6.1 Creating a Schedule for Transfers

Once created, you can start the transfer immediately using the START TRANSFER NOW statement. Alternatively, you can create a schedule that determines the time and frequency of the transfer, including the initial execution.

This section describes how to create a schedule for an extraction, extraction rollup, or replication transfer definition. There is no difference syntactically between a schedule definition created for an extraction (or extraction rollup) transfer and a replication transfer. The syntax of the CREATE SCHEDULE statement provides four optional clauses that start with the following keywords:

- START

  The START clause requires a parameter value that specifies the date and time of the initial execution of the transfer. If you omit the START clause, the transfer starts immediately after you enter the CREATE SCHEDULE statement.

- EVERY

  The EVERY clause requires a parameter value that controls when subsequent transfer executions occur. You can specify every day, week, or month, and you can name days of the week, each with its own time of day.

- RETRY

  The RETRY clause requires a parameter value that limits the number of attempts Replication Option makes when a recoverable error prevents the transfer from completing. If you omit this clause, Replication Option does not attempt to retry the transfer until the next scheduled transfer time specified in the EVERY clause. See Table C–6 for a list of recoverable errors.

- RETRY EVERY

  The RETRY EVERY clause requires a parameter value specifying a length of time between retry attempts.

In Example 6–1, users at the remote node request that a new database extraction be created every day to support the following day's query activities. A schedule is created that calls for the first extraction to execute at 11 a.m. on 27-OCT-1996. If the transfer fails, Replication Option should make up to 3 attempts to execute the transfer with 30 minutes between each attempt. Thereafter, the copy process provides a new database extraction to the target directory 24 hours after the completion of the last NH_EMPLOYEES transfer.

The schedule definition for the NH_EMPLOYEES transfer in Example 6–1 shows 3 retry attempts, with 30 minutes between each attempt:

Example 6–1 Schedule Definition

```
CREATE SCHEDULE FOR NH_EMPLOYEES
   START                          ! 11:00 AM
     27-OCT-1996 11:00:00.00
   EVERY                          ! Every 24 hours
     1 00:00:00.00
   RETRY                          ! Maximum of 3 retries
     3 TIMES
   RETRY EVERY                    ! Every half hour
     0 00:30:00
;
```

The START clause of the CREATE SCHEDULE statement allows you to specify an absolute-time or a delta-time value indicating the day and time to begin the transfer. The schedule for the NH_EMPLOYEES transfer in Example 6–1 includes the clause START 27-OCT-1996 11:00:00.00, causing the transfer to execute at 11:00 a.m. on October 27, 1996.

The EVERY clause specifies when subsequent executions of the NH_ EMPLOYEES transfer will occur. The example includes the parameter value EVERY 1 00:00:00.00. Other possible values include WEEK, MONTH, or particular days of the week. Refer to the CREATE SCHEDULE statement in Chapter 9 for details about valid values for all clause parameters.

The RETRY clause controls what happens when a transfer fails on the first attempt of a scheduled transfer. For example, if the target node is unavailable, the transfer cannot take place. The RETRY clause specifies how many attempts will be made to complete the transfer. If all attempts fail, the next transfer attempt will occur according to the value provided in the EVERY clause.

Associated with the RETRY clause of the schedule definition is the RETRY EVERY clause. The parameter value supplied for this clause determines the time interval between attempts specified in the RETRY clause to complete the transfer successfully. Example 6–1 calls for half an hour between retries.

You should note carefully the difference between specifying delta time and absolute time in the EVERY clause of the transfer schedule. When you include a delta time, such as EVERY 1 00:00:00.00, instead of specifying a time of day, the transfer monitor uses the completion time of the last transfer as the base for computing the start time of the next scheduled transfer.

In Example 6–1, because the transfer is governed by EVERY 1 00:00:00.00, the execution of the next transfer is determined by the completion time of the last transfer. If the NH_EMPLOYEES transfer takes 40 minutes to complete and the first execution starts at 11:00 a.m. on the specified date, the transfer finishes at 11:40 a.m. Therefore, the transfer monitor will schedule the next execution at 11:40 a.m. and not 11:00 a.m. on the following day.

When you want the transfer to execute at regular intervals at a particular time of day, specify an absolute time. The transfer monitor will use the absolute time you specify to begin transfer execution. Example 6–2 shows a schedule definition that starts each execution of the NH_EMPLOYEES transfer at 5:30 p.m. every day.

An absolute time causes the transfer monitor to always schedule subsequent executions at that absolute time and not at a time dependent on the completion time of the previous transfer.

Example 6–2 Schedule Definition Specifying Absolute Time

```
CREATE SCHEDULE FOR NH_EMPLOYEES
  START                     ! 5:30 PM
    5-JUN-1996 17:30:00.00
  EVERY                     ! Every day at 5:30 PM
    DAY AT 17:30:00.00
  RETRY                     ! Maximum of 3 retries
    3 TIMES
  RETRY EVERY               ! Every hour
    0 1:00:00.00
;
```

6.2 Transfer States

Transfers can have several different **states**. Each state restricts the behavior of a transfer in certain ways. You use the SHOW TRANSFER statement to find out the current state. In Example 6–3, the SHOW TRANSFER STATUS statement and its output show that the INVENTORY transfer is currently scheduled to execute at a later time.

Example 6–3 Transfer in the SCHEDULED State

```
SQL> SHOW TRANSFER (STATUS) INVENTORY;

Status for transfer INVENTORY:
State                  SCHEDULED since 23-JUN-1996 14:31:27.66
Next transfer          26-JUN-1996 14:30:00.00
Last completion status  transfer was successful
Last executed          23-JUN-1996 14:30:00.00
```

Four of these transfer states are:

- Unscheduled

  A schedule does not exist for this transfer. A transfer in the unscheduled state can only be executed when the START TRANSFER NOW statement is issued.

- Scheduled

  A schedule exists for this transfer. It remains scheduled until the next transfer time is reached, and then it becomes active.

- Active

  A transfer is currently executing.

- Suspended

  The transfer is not currently executing. It will not execute even if it is set to do so according to an associated schedule. It can execute only if you issue a START TRANSFER statement and if the transfer has an associated schedule definition. If a schedule definition does not exist for a transfer and you issue a START TRANSFER statement, Replication Option places the transfer in the unscheduled state and executes the transfer only when you enter a START TRANSFER NOW statement.

Refer to Appendix C for information about transfer states and the events that change transfers from one state to another.

6.3 Executing the Transfer

This section describes how to initiate transfer execution and how to ensure that the transfer completes before other operations are allowed. The section also describes how to determine that the transfer is complete.

Depending on its transfer state, a transfer will either execute or will require some other action that will cause it to execute. Transfer execution occurs either when a START TRANSFER NOW statement is processed or when a transfer schedule triggers execution at a predetermined time.

The START TRANSFER NOW statement has the following two options:

- NO WAIT

  In interactive SQL, as soon as you press Return to process the START TRANSFER NOW statement, the next SQL> prompt displays so you can enter other statements while the transfer is executing.

  In an application program, your program will continue to execute without waiting for the transfer to finish.

  NO WAIT is the default.

- WAIT

  In interactive SQL, you must wait until the transfer finishes before you can continue your interactive SQL session.

  In an application program, your program will stall until the transfer completes (successfully or not).

When you issue the START TRANSFER NOW statement in SQL, the named transfer begins immediately regardless of the time specified in the transfer schedule, if one exists. You can then resume other Replication Option tasks or exit SQL. The following statement starts the CARGO transfer immediately:

```
SQL> START TRANSFER CARGO NOW NO WAIT;
```

If your transfer is currently active and you issue a START TRANSFER NOW statement to cause the transfer to begin execution immediately, Replication Option issues an error message indicating that a new transfer cannot be started because the transfer is already active. If the transfer is not currently executing when you issue a START TRANSFER NOW statement, Replication Option will not follow the preset transfer schedule. Instead, the transfer starts immediately.

Note

Do not issue a START TRANSFER NOW WAIT statement when you are attached to a source database and the transfer type is either of the following:

- A replication transfer that has not yet been initialized (that is, has not yet successfully executed once)
- A replication reinitialization

The copy process will not complete until you first disconnect from the database.

In the WAIT option case, the copy process waits indefinitely for you to release the source database. Because you specified that you want to wait for the copy process to complete, the transfer operation will never complete.

Instead, issue a DISCONNECT statement to detach from the database, and then issue one of the following statements:

- START TRANSFER NOW WAIT
- START TRANSFER NOW NO WAIT

The transfer monitor uses the time values specified in the transfer schedule to compute the next scheduled transfer and schedules the transfer to execute at that time.

Entering a START TRANSFER NOW statement can affect the time value used for the next execution of the transfer if a delta time is specified in the schedule. For example, the following schedule for NH_EMPLOYEES specifies that the transfer execution begin on 27-OCT-1996 at 11:00 a.m. and execute again every hour after that. The delta time (1:00:00.00 or 1 hour) and the time that the last transfer ended execution together determine when the next execution occurs. If the START TRANSFER NOW statement is issued at 10:30 a.m. and the transfer finishes at 11:15 a.m., the next execution will occur one hour from 11:15, or at 12:15 p.m.

```
SQL> CREATE SCHEDULE FOR NH_EMPLOYEES
cont>  START
cont>     27-OCT-1996 11:00:00
cont>  EVERY
cont>     0 1:00:00
cont>  ;
SQL> START TRANSFER NH_EMPLOYEES NOW;
```

When you include the WAIT option in the START TRANSFER NOW statement, you do not receive the SQL> prompt again until the transfer is finished. The WAIT keyword causes the SQL> prompt to return only after the transfer completes successfully or has exhausted the number of retries specified in the transfer schedule.

Although Replication Option attempts to begin execution of the transfer as close as possible to the specified time, any one of several conditions can prevent the transfer from beginning precisely at the time specified in the schedule definition. For example:

- If the transfer monitor is busy, it cannot respond to the request until a task that cannot be interrupted completes.

- When a STOP TRANSFER statement causes the transfer status to change to suspended, the transfer will not execute again until you enter a START TRANSFER statement. At that point, if the transfer has a schedule created for it, the next scheduled transfer time is calculated.
- A condition can exist that causes the transfer execution to enter the retry state. For example, the target node might be off line, thus preventing the transfer from completing successfully.
- When the source node is unavailable, the transfer must wait until the node is available again. Then the transfer will either execute or continue to retry.
- If the maximum number of copy processes has been reached, execution of the next transfer will not occur until one of the executing transfers finishes. Replication Option currently allows up to 20 copy processes to run on a particular node at the same time.

The transfer monitor records the status for every event or task performed during the transfer operation and writes entries in the transfer monitor log file.

In addition, the copy process creates a new log file every time a transfer executes, if a log file is specified in the transfer definition.

You can use the copy process and the transfer monitor log files to determine when and where a failure occurs. Refer to later sections describing how to use the log files of both the transfer monitor and the copy process.

6.4 Modifying Your LOGIN.COM File for the Copy Process

When you create and then execute a transfer, a copy process is created as an OpenVMS detached process. It then runs under your account for the following reasons:

- System accounting

  If your site charges users for the use of system resources, you will be charged for the resources that the copy process uses.

- Environment variable definition

  When you create a transfer, you might enter in the transfer definition environment variables that are defined only in your process. These environment variables should be defined from within your LOGIN.COM file. Because environment variables can be stored in the transfer definition without being translated, the copy process must be able to see those environment variable definitions at the time it executes.

You might want to customize your LOGIN.COM file taking into account that it will be used when a copy process executes. Your LOGIN.COM file might include sections that you want the copy process to not execute. For example, the copy process would have no use for environment variables you define for your own use when you work at a terminal. Tailoring the execution of your LOGIN.COM procedure for the copy process avoids writing the entire LOGIN.COM procedure into the copy process log file.

If there are sections of your LOGIN.COM file that you want the copy process to skip, you can include the DCL code in Example 6–4.

Example 6–4 Customizing LOGIN.COM File to Skip Portions of Its Code

```
$ PROCESS_NAME = F$GETJPI("","PRCNAM")
$ IF F$EXTRACT(0,10,PROCESS_NAME) .EQS. "DDAL_COPY_" THEN -
    GOTO ITS_A_COPY_PROCESS
```

The first line asks the system to return the name of the OpenVMS process. The transfer monitor assigns to all copy processes names that begin with "DDAL_COPY_". If you change your process name within your LOGIN.COM file, make sure this check occurs before the line that changes the process name.

Another test you can make is to check the mode in which your process is running:

```
$ IF F$MODE() .EQS. "OTHER" THEN GOTO ITS_A_DETACHED_PROCESS
```

When you are running a detached process, such as a copy process, the mode of operation is shown as "OTHER" rather than "INTERACTIVE", "BATCH", or "NETWORK".

If your LOGIN.COM procedure redefines standard OpenVMS commands such as SHOW, RUN, or LOGOUT, those commands will revert to their normal definitions because the transfer monitor redefines the commands it needs prior to starting the transfer.

6.5 Stopping Transfers

Replication Option provides the STOP TRANSFER statement to explicitly terminate transfer execution or prevent a transfer from starting. Reasons for stopping a transfer can include the following:

- The source database is temporarily unavailable.
- Routine batch updates to the source database have been delayed, thereby reducing the benefit of the transfer attempt.

- The target applications are not prepared for the next data transfer.

You can issue the STOP TRANSFER statement at any time. Executing the STOP TRANSFER statement changes the transfer state to suspended [1]:

- Prevents scheduled execution

  If you stop a transfer and the transfer has a defined schedule, unless you take further action the next scheduled execution of that transfer will not take place. Stopping a transfer suspends further execution indefinitely.

- Might render the target database unusable

  Transfers that create new Rdb database files are accomplished by using batch-update transactions. Batch-update transactions execute faster than read/write transactions because write operations are not performed to either a recovery-unit journal file or to a snapshot file. If a batch-update transaction fails, the partially created target database becomes corrupt and unusable.

  If you stop a replication update transfer in midexecution or any transfer to an existing target database, the target database is not compromised. Replication updates are performed using read/write transactions, not batch updates. You lose only the current transaction when you stop such a transfer during execution. That aborted transaction is eventually rolled back. Any changes made to the target database within that transaction are not visible to other applications. The target data remains consistent.

Note

Stopping a replication transfer does *not* stop the capture of data row changes in the source database. Changes continue to be written to the RDB$CHANGES table regardless of a transfer's state. Rows are added to RDB$CHANGES even for transfers that are suspended for a long time. Therefore, the number of rows in RDB$CHANGES and the overall size of the database both continue to grow.

[1] When the STOP TRANSFER statement changes the state of a transfer from active to suspended, the last transfer completion status is not updated and does not reflect the incomplete transfer.

6.6 Changing and Dropping Transfers

You can change some clauses in a transfer definition by using the SQL ALTER TRANSFER statement. If you want to make a change that is not supported by the ALTER TRANSFER statement, you must drop the transfer definition, and then create a revised definition using the same transfer name. The following two sections describe the ALTER TRANSFER statement (and what clauses it allows you to alter) and how to drop transfer definitions.

6.6.1 Changing Transfer Definitions Using the SQL ALTER TRANSFER Statement

The SQL ALTER TRANSFER statement allows you to:

- Add, remove, or change a prologue file, an epilogue file, log file, or CHECKPOINT attribute
- Change the target file specification
- Add or change the description of the transfer

The ALTER TRANSFER statement does not allow you to change the fundamental transfer definition. That is, it does not allow you to:

- Add or remove tables or columns from a transfer definition
- Alter the TYPE IS clause of a transfer definition
- Alter the move clause of a transfer definition
- Alter the target-db-params clause of a transfer definition
- Alter the target database from NEW to EXISTING or from EXISTING to NEW
- Alter the source database file specification

Those items specified in the second list can be changed only by dropping the transfer and redefining it. See Section 6.6.2 for information on dropping a transfer definition.

To alter a transfer definition, you must take the following steps:

1. Make sure you are not in the middle of a transaction in SQL. The first statement in Example 6–5 ends the current transaction with a ROLLBACK statement.
2. Determine the transfer's current state with the SHOW TRANSFER statement. If the transfer is active and you want it to finish, you will have to check the status until the transfer is no longer active.

3. Suspend the transfer by entering a STOP TRANSFER statement. You can verify the change of transfer state by again issuing the SHOW TRANSFER statement.
4. Alter the transfer definition by issuing an ALTER TRANSFER statement.

When you issue the ALTER TRANSFER statement to change a transfer definition, the transfer definition is updated in the transfer database.

When a transfer is altered, whatever has already been copied to the target database remains intact:

- Data definitions created by the transfer remain in effect.
- Data written to the tables remains in place.

The following example shows the current definition of the NH_EMPLOYEES transfer, followed by an ALTER TRANSFER statement that adds the CHECKPOINT attribute.

```
SQL> ATTACH 'FILENAME PERSONNEL';
SQL> CREATE TRANSFER NH_EMPLOYEES TYPE IS EXTRACTION
cont>  MOVE TABLES
cont>     SELECT EMPLOYEE_ID,
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>  FROM EMPLOYEES WHERE STATE = 'NH'
cont>  TO NEW FILENAME DISK1:[TARGET]NH_EMP
cont>  LOG FILE IS NH_EMPLOYEES.LOG
cont>  COMMENT IS 'EMPLOYEES WHO LIVE IN NEW HAMPSHIRE'
cont>  ;
SQL> STOP TRANSFER NH_EMPLOYEES;
SQL> ALTER TRANSFER NH_EMPLOYEES
cont> CHECKPOINT;
```

6.6.2 Dropping Transfer Definitions

If you want to change the fundamental transfer definition, that is, do any of the following, you must drop the transfer definition first, and then redefine it:

- Add or remove tables or columns from a transfer definition
- Alter the TYPE IS clause of a transfer definition
- Alter the move clause of a transfer definition

- Alter the target-db-params clause of a transfer definition
- Alter the target database from NEW to EXISTING or from EXISTING to NEW
- Alter the source database file specification

After dropping the transfer, you can enter a revised definition using the same transfer name. Transfers must be in the suspended state before you can drop them. You can use the SHOW TRANSFER statement to find out what the current state is.

To drop a transfer definition, you must take the following steps:

1. Make sure you are not in the middle of a transaction in SQL. The first statement in Example 6–5 ends the current transaction with a ROLLBACK statement.
2. Determine the transfer's current state with the SHOW TRANSFER statement. If the transfer is active and you want it to finish, you will have to check the status until the transfer is no longer active.
3. Change the transfer to a suspended state by entering a STOP TRANSFER statement. You can verify the change of transfer state by again issuing the SHOW TRANSFER statement.

The final steps in this process differ depending on the type of transfer you are dropping:

- For extraction or extraction rollup transfers, simply delete the transfer definition by issuing a DROP TRANSFER statement.
- For replication transfers, detach from the source database before you issue the DROP TRANSFER statement.

  If the source database is inaccessible or no longer exists, and you need to drop the transfer, create a new database with the same name, columns, and tables as the inaccessible source database. Locate this new database at the address of the old database. Reinitialize and then start the transfer. These actions will enter the transfer definition in the source database.

  After the definition has been entered in the source database, stop the transfer by entering a STOP TRANSFER statement.

The sequence of statements in SQL deletes the transfer called NH_ EMPLOYEES in Example 6–5.

Example 6–5 Dropping a Transfer Definition

```
SQL> ROLLBACK;
SQL> SHOW TRANSFER (STATUS) NH_EMPLOYEES;

Status for transfer NH_EMPLOYEES:
State                   ACTIVE since 26-JUN-1996 14:30:01.64
Last completion status  transfer was successful
Last executed           23-JUN-1996 14:30:00.00

SQL> SHOW TRANSFER (STATUS) NH_EMPLOYEES;

Status for transfer NH_EMPLOYEES:
State                   SCHEDULED since 26-JUN-1996 14:31:54.68
Next transfer           27-JUN-1996 14:30:00.00
Last completion status  transfer was successful
Last executed           26-JUN-1996 14:30:00.00

SQL> STOP TRANSFER NH_EMPLOYEES;
SQL> SHOW TRANSFER (STATUS) NH_EMPLOYEES;

Status for transfer NH_EMPLOYEES:
State                   SUSPENDED since 26-JUN-1996 14:32:57.62
Last completion status  transfer was successful
Last executed           26-JUN-1996 14:30:00.00

SQL> DROP TRANSFER NH_EMPLOYEES;
SQL> SHOW TRANSFER (DEFINITION) NH_EMPLOYEES;
No transfers found
```

When you drop a replication transfer definition, the transfer monitor might require a copy process to perform the deletion in the source database. That is, if the transfer has never executed, no copy process is needed. If the replication transfer has executed at least once, the transfer monitor requires a copy process to drop the transfer definition. Therefore, when a copy process is required to drop a replication transfer, a copy process log is created and written to the directory specified in the transfer definition.

When you issue the DROP TRANSFER statement to drop a transfer definition, the transfer definition and any associated schedule are removed from the transfer database. When a transfer is dropped, whatever has been copied to the target database remains intact:

- Data definitions created by the transfer remain in effect.
- Data written to the tables remains in place.
- Data definitions continue to be listed in the RDBVMS$TRANSFER_OWNER or DDAL$TRANSFER_OWNR table and the DDAL$INDEXES table if shared writing is enabled, as owned by the transfer, even though the transfer definition no longer exists.

Leaving the data definitions in place is important because they may be referenced by other data definitions not owned by this transfer.

If you drop a transfer, redefine it, and then re-execute the changed transfer, Replication Option drops any tables that were included in the original definition but dropped in the new definition. Note that if the original transfer had shared writing enabled, the subsequent transfer cannot drop tables included in the original definition. When transfers create tables with shared writing enabled, only the transfer that created the table can drop it. This is illustrated in the following steps:

1. Transfer XYZ, naming tables A and B, is executed. Tables A and B end up in the target database.
2. Transfer XYZ is dropped. All the data definitions and data related to transfer XYZ remain intact in the target database.
3. Transfer XYZ is re-created. This time only table A is named.
4. Transfer XYZ is executed. Table A is repopulated with data because it is owned by transfer XYZ. The copy process notices that table B exists in the target database and it is owned by transfer XYZ. Because the new definition of XYZ no longer mentions table B, that table is dropped from the target database. This only applies to tables and views. If there are any domains in the target database that exist solely to support table B, they are not deleted.

If you intend to modify the transfer definition that you just dropped, you enter the new transfer definition with a CREATE TRANSFER statement. You can then create a schedule for the transfer or you can execute it on demand with the START TRANSFER NOW statement.

Follow the steps outlined in Chapter 5 about entering the transfer definition to reenter the new replication transfer definition. Follow the steps outlined in Chapter 8 about entering the transfer definition to reenter the new replication transfer with the NO DELETE attribute.

6.7 Changing and Dropping Schedules

You can change all of the clauses in a schedule definition by using the SQL ALTER TRANSFER statement. Alternatively, you can drop the schedule and redefine it or leave the transfer without a schedule. The following two sections describe the SQL ALTER SCHEDULE statement and the DROP SCHEDULE statement.

6.7.1 Changing Schedules

To change a parameter in a transfer schedule for NH_EMPLOYEES, use the SQL ALTER SCHEDULE statement. To issue an ALTER SCHEDULE statement, the transfer must be in the stopped state. Therefore, if the transfer is currently active, you might want to let the transfer complete execution before you issue an ALTER SCHEDULE statement.

When you alter a schedule definition, changes are made to the transfer database residing on the node or cluster to which you logged in. In the following example, you stop a transfer, alter a transfer schedule, and use the SHOW TRANSFER statement to show that the changes are made:

```
SQL> STOP TRANSFER NH_EMPLOYEES;
SQL> ALTER SCHEDULE FOR NH_EMPLOYEES
cont> START 30-AUG-1996 17:00:00.00
cont> EVERY DAY AT 17:00:00.00
cont> RETRY 3 TIMES RETRY EVERY 0 03:00:00;
SQL> SHOW TRANSFER (SCHEDULE) NH_EMPLOYEES;

Schedule for transfer NH_EMPLOYEES:
Start                   30-AUG-1996 17:00:00.00
Frequency               Every day at 17:00:00.00
Number of retries       3 times
Retry frequency            0 03:00:00.00
```

6.7.2 Dropping Schedules

To delete a transfer schedule for NH_EMPLOYEES, use the DROP SCHEDULE statement. To issue a DROP SCHEDULE statement, the transfer must be in the stopped state. Therefore, if the transfer is currently active, you might want to let the transfer complete execution before you issue a DROP statement.

When you drop a schedule for an inactive transfer, the transfer changes its state. That is, a transfer that has a schedule created for it is in the scheduled or active state. When you delete the schedule for the transfer and the transfer is not currently executing, the transfer state changes to unscheduled. When you create a new schedule for that transfer, the status of the transfer changes from unscheduled to scheduled.

When you add or delete a transfer or schedule definition, you make the changes to the transfer database residing on the node or cluster to which you logged in. In the following example, you stop a transfer, delete a transfer schedule, and use the SHOW TRANSFER statement to show that the schedule no longer exists:

```
SQL> STOP TRANSFER NH_EMPLOYEES;
SQL> DROP SCHEDULE FOR NH_EMPLOYEES;
SQL> SHOW TRANSFER (SCHEDULE) NH_EMPLOYEES;
No transfers found
SQL>
              .
              .
              .
```

6.8 Changing Data Definitions After a Transfer Is Defined

In an extraction transfer, you can change table definitions in the source database or definitions of columns on which the transferred tables are based. However, those changes are not reflected in the transfer definition.

In replication transfers, you can make such changes until the initial transfer begins. Thereafter, any attempts to change or delete definitions for tables or columns in the replication transfer are not allowed, with the exception of the following four column attributes; these attributes can be modified even after the initial transfer begins:

- Edit string
- Default value for DATATRIEVE
- Query name for DATATRIEVE
- Query header

However, note that if you do change the attributes listed in a source column, a comparable change is not made automatically to the target database column. If you want that to happen, you must make the change in both the source and the target databases.

In both extraction and replication transfers, if you make changes that cause the source tables or columns to be incompatible with the transfer definitions, Replication Option detects these differences at the time of transfer execution. The transfer fails, the transfer status reflects the failure, and the transfer is suspended. You can suppress the comparison of source and target definitions by defining the event variable, DDAL$CHECK_TARGET_DEFINITIONS. See Section B.4 for more information about DDAL$CHECK_TARGET_ DEFINITIONS.

Note

If you suppress comparison of source and target definitions, you must remember to manually make any changes to the target tables that you

make to the source tables. Otherwise, the transfer might fail.

In extractions and initial replications, the copy process does not detect changes to the data type of a column. For example, you can change a column from the data type CHAR(10) to the data type CHAR(20) without causing a failure.

6.9 Displaying Transfer Information

You can use the SHOW TRANSFER statement in SQL to display the complete definition, status, or current schedule for all transfers you created.

To view information about other transfers, you must have the ALTER, DROP, SELECT, or DBCTRL transfer privileges or you must have additional privileges available in the OpenVMS operating system.

See Section 9.12 for information about the SHOW TRANSFER statement. See Section 9.7 for information about the GRANT statement

Examples 6–6, 6–7, and 6–8 illustrate the different types of information you can display about a transfer by using the SHOW TRANSFER statement.

6.9.1 Showing a Transfer Definition

Although you need to be attached to the source database to enter a transfer definition, the association between the transfer and the source database is implicit. The SHOW TRANSFER (DEFINITION) display, however, includes an explicit reference to the source database in the line containing FROM.

Example 6–6 Displaying a Transfer Definition on OpenVMS

```
SQL> SHOW TRANSFER (DEFINITION) NH_EMPLOYEES;

Definition for transfer NH_EMPLOYEES:
Definer                 TESTER
Type                    EXTRACTION
Comment                 Employees who live in New Hampshire
From                    PROD$DISK:[ADMIN.PERSONNEL]PERSONNEL.RDB
To New                  REMNOD::WORK$DISK:[PROD.PERSONNEL]NHEMP
Log file                DISK$A:[TESTER]NHEMP_EXT.LOG
No prologue file
No epilogue file
Move table
    Select All
            EMPLOYEE_ID,
            LAST_NAME,
            FIRST_NAME,
            MIDDLE_INITIAL,
            ADDRESS_DATA_1,
            ADDRESS_DATA_2,
            CITY,
            STATE,
            POSTAL_CODE
        From EMPLOYEES
        Where
            STATE = "NH"
```

Although the names of the target tables in this example are the same as those in the source database, the structure of the tables created in the extraction, extraction rollup, or replication database is likely to be different. Therefore, applications that used these names to access tables in the source database cannot be expected to access the tables in the target databases without recompiling them.

6.9.2 Examining Target Database Information

In each target database, Replication Option maintains a table for keeping information about transfers to that database. For Rdb, the table is called RDB$VINTAGE; for all other databases, the table is called DDAL$TRANSFER_INFO. Table 6–1 lists two columns from each table that contain useful reference information for target users and applications.

Table 6–1 Tables and Columns Useful for Target Users and Applications

Table name	Transfer Time Column	Transfer in Progress Column
DDAL$TRANSFER_ INFO	DDAL$TRANSFER_ TIME	DDAL$TRANSFER_BUSY
RDB$VINTAGE	RDB$VINTAGE_ TIMESTAMP	RDBVMS$VINTAGE_ TRANSFER_BUSY

When an extraction, extraction rollup or initial replication transfer begins, the date and time column is set to 17-NOV-1858 00:00:00.00. Once the main part of the transfer has completed, but before execution of any epilogue procedure, that column contains the date and time that the transfer last executed. The date and time are relative to the system on which the transfer definition is recorded. If you anticipate transfers to recur on a regular basis, you can examine this column to determine if a transfer has occurred as recently as expected.

When the data transfer is about to begin in an extraction, extraction rollup or initial replication transfer, the transfer-in-progress column is set to the value "T", meaning TRUE. Once the main body of the transfer is finished, but before the epilogue procedure is executed, the column value is reset to "F", meaning FALSE. While data is being transferred, the database may be in an inconsistent state. Therefore you might wish to delay execution of your application when the in-progress flag is set to "T". Note that inconsistent states do not occur during replication update transfers. So for them, the in-progress flag remains "F".

See Table A–21 for the columns in the RDB$VINTAGE and DDAL$TRANSFER_ INFO tables.

6.9.3 Showing a Transfer Schedule

You can review the schedule currently entered for a transfer by issuing the SHOW TRANSFER (SCHEDULE) statement. Example 6–7 shows a schedule created for the NH_EMPLOYEES transfer. The first transfer began on 6-JUN-1996 at 10:24:00 a.m. Subsequent transfers will occur every day. If the initial daily transfer fails, one retry will be attempted every 30 minutes after the failure of the initial transfer.

Example 6–7 Displaying a Schedule Definition

```
SQL> SHOW TRANSFER (SCHEDULE) NH_EMPLOYEES;

Schedule for transfer NH_EMPLOYEES:
Start                    6-JUN-1996 10:24:00.00
Frequency                Every day at 14:30:00.00
Number of retries        3 times
Retry frequency          0 00:30:00.00
```

6.9.4 Showing Transfer Status

Example 6–8 displays the status of the transfer NH_EMPLOYEES. The state of the transfer is scheduled; the transfer changed to this state on 26-JUN-1996 at 15:21:13.07. The completion status tells you that the last attempt to transfer data to the target database was successful. The last completed transfer occurred on 26-JUN-1996 at 15:20:10.50.

Example 6–8 Displaying the Status of a Transfer

```
SQL> SHOW TRANSFER (STATUS) NH_EMPLOYEES;

Status for transfer NH_EMPLOYEES:
State                   SCHEDULED since 26-JUN-1996 15:21:13.07
Next transfer           27-JUN-1996 14:30:00.00
Last completion status  transfer was successful
Last executed           26-JUN-1996 15:20:10.50
```

6.9.5 Showing All Transfer Information

The following SHOW TRANSFER statement includes all available information about the transfer—the definition, the status, and the schedule—in a single display:

```
SQL> SHOW TRANSFER NH_EMPLOYEES;
```

Similarly, the next statement includes the transfer definition, status, and schedule for every transfer in the transfer database that you are authorized to view. If you do not have sufficient privileges to display information about a particular transfer, Replication Option simply shows you the name of the transfer and the definer's name. Because all transfer names must be unique within a particular transfer database, you can, without special privileges, review all transfer names currently stored in the transfer database to ensure uniqueness. You can use the following form of the SHOW TRANSFER statement to display definitions for all transfers you are permitted to see:

```
SQL> SHOW TRANSFER *;
```

When you use the wildcard (*) character with SHOW TRANSFER, the statement lists all of the transfer names in the transfer database.

You can create a complete report on all transfers currently defined when you add the SQL SET OUTPUT file-spec statement and direct the output to a disk file:

```
SQL> SET OUTPUT ALL_TRANSFERS.LIS;
SQL> SHOW TRANSFER (SCHEDULE) *;
SQL> SET NOOUTPUT;
```

6.10 Diagnostic Tools

Two components of Replication Option create log files that record event and status information. The transfer monitor creates a transfer monitor log file (see Section 6.10.3). Also, if you specify the LOG FILE IS clause in the transfer definition, a new copy process log is created every time a transfer executes (see Section 6.10.1). Whenever failures occur in a transfer or in Replication Option as a whole, you should examine these log files. They will contain detailed information and error messages that will aid in diagnosing problems.

You can also instruct Replication Option to log specific events that will help you to diagnose communication problems between the transfer monitor and the user interface. See Section 6.10.4.

6.10.1 Using Copy Process Log Files

The copy process writes information about the tasks it performs into a log file. A log file for the copy process is located on the host/node where a transfer is defined. Therefore, the copy process log file is stored as follows:

- On the source node for replication transfers
- On the source node for extraction transfers of the push type
- On the target node for extraction transfers of the pull type

Push transfers are defined on the source node. Pull transfers are defined on the target node.

When a transfer executes, the transfer monitor starts a copy process. To resolve logical names and symbols, the copy process executes the LOGIN.COM file of the user who created the transfer and writes information to the copy process log.

Example 6–9 shows a copy process log file. The sample log file shows the activities of the copy process during the execution of an extraction transfer. Embedded in Example 6–9 are numbers, or callouts, that mark key areas of the log. Descriptions keyed to the callouts follow the log example.

Example 6–9 Sample Copy Process Log File on OpenVMS

```
   $ SET VERIFY
❶  $ SHOW = "SHOW"
   $ RUN = "RUN"
   $ LOGOUT = "LOGOUT"
   $ SHOW PROCESS

❷  16-MAR-1996 13:18:53.71   User: DBA                    Process ID:   47C0EE58
                             Node: LOCNOD                 Process name: "DDAL_COPY_01   "

   Terminal:           RTA5:
   User Identifier:    [LOCNOD,DBA]
   Base priority:      4
   Default file spec:  USD04:[LOCNOD.DBA]

❸  $ DEFINE DDAL$CP_CONTINUE "SUCCESS"
   $ DEFINE/TRANS=TERMINAL DDAL$TRANSFER_NAME NH_EMPLOYEES
   $ DDAL$CP_ACTION :== E
   $ DDAL$CP_RETRY_ITERATION == 00
   $ RUN DDAL$COPY_PROCESS

❹  Product version:        Replication Option V7.0

❺  ----- 16-JAN-1996 15:07:25.00 ----- Process        ------------------------
   Process name:  DDAL_COPY_01                        Priority:  4
   Username:  DBA                                     UIC:  [LOCNOD,DBA]
   Nodename:  LOCNOD                                  PID:  3F4044AA
   Privileges currently enabled:
       TMPMBX, NETMBX

   Image privileges:
       SYSPRV, BYPASS, SYSLCK, TMPJNL

   Image name:
       SYS$COMMON:[SYSEXE]DDAL$COPY_PROCESS.EXE;1


❻  Transfer database = SYS$SYSTEM:DDAL$TR_DB
   Transfer name     = NH_EMPLOYEES
   Transfer option   = E (Extraction transfer)
```

(continued on next page)

Example 6–9 (Cont.) Sample Copy Process Log File on OpenVMS

```
❼ 15:07:36  %DDAL-I-READTRDEF, reading the transfer definition from the transfer
  database
  15:07:44  %DDAL-I-ATTACHSDB, attaching to source database
  USD04:[DBA.RDB]PERSONNEL.RDB
  15:07:48  %DDAL-I-CRETARGDB, creating target database
  REMNOD::USD04:[DBA.RDB]NHEMP
  15:08:16  %DDAL-I-ATTACHTDB, attaching to target database
  REMNOD::USD04:[DBA.RDB]NHEMP
  15:08:16  %DDAL-I-STADATTRM, starting data transfer/modification

            %DDAL-I-CREDOMAIN, creating domain RDB$VINTAGE_TIMESTAMP
            %DDAL-I-CREDOMAIN, creating domain RDBVMS$VINTAGE_TSER
            %DDAL-I-CREDOMAIN, creating domain RDBVMS$VINTAGE_TRANSFER_ADDR
            %DDAL-I-CREDOMAIN, creating domain RDBVMS$VINTAGE_TRANSFER_NODE
            %DDAL-I-CREDOMAIN, creating domain RDBVMS$VINTAGE_TRANSFER_BUSY
            %DDAL-I-CREDOMAIN, creating domain RDBVMS$VINTAGE_TRANSFER_ID
            %DDAL-I-CREDOMAIN, creating domain RDBVMS$VINTAGE_TRANSFER_NAME
  15:08:17  %DDAL-I-CREATABLE, creating table RDB$VINTAGE
  15:08:21  %DDAL-I-LOATABDAT, loading data for table RDB$VINTAGE

            %DDAL-I-CREDOMAIN, creating domain DEPARTMENT_CODE
            %DDAL-I-CREDOMAIN, creating domain FIRST_NAME
            %DDAL-I-CREDOMAIN, creating domain ID_NUMBER
            %DDAL-I-CREDOMAIN, creating domain LAST_NAME
            %DDAL-I-CREDOMAIN, creating domain MIDDLE_INITIAL
            %DDAL-I-CREDOMAIN, creating domain SALARY_AMOUNT
            %DDAL-I-CREDOMAIN, creating domain SALARY_END
            %DDAL-I-CREDOMAIN, creating domain SALARY_START
  15:08:22  %DDAL-I-CREATABLE, creating table EMPLOYEES
  15:08:23  %DDAL-I-CREATABLE, creating table SALARY_HISTORY
  15:08:24  %DDAL-I-CREATABLE, creating table EMPLOYEES
  15:08:25  %DDAL-I-LOATABDAT, loading data for table EMPLOYEES

❽ 15:08:26  %DDAL-I-TAB_STATS,                TABLE  EMPLOYEES

            %DDAL-I-ROWSFMSRC,        90 rows copied from USD04:[DBA.RDB]PERSONNEL.RDB

            %DDAL-I-ROWSTOTGT,        90 rows copied into REMNOD::USD04:[DBA.RDB]NHEMP

❾ 15:08:26  %DDAL-I-LOATABDAT, loading data for table SALARY_HISTORY

            %DDAL-I-ROWSFMSRC,        25 rows copied from USD04:[DBA.RDB]PERSONNEL.RDB

            %DDAL-I-ROWSTOTGT,        25 rows copied into REMNOD::USD04:[DBA.RDB]NHEMP

  15:08:26  %DDAL-I-LOATABDAT, loading data for table RDB$VINTAGE

❿ 15:08:28  %DDAL-I-SUMRYSTAT,              SUMMARY STATISTICS

            %DDAL-I-ROWSFMSRC,       115 rows copied from
  USD04:[DBA.RDB]PERSONNEL.RDB

            %DDAL-I-ROWSTOTGT,       115 rows copied into
  REMNOD::USD04:[DBA.RDB]NHEMP
```

(continued on next page)

Example 6–9 (Cont.) Sample Copy Process Log File on OpenVMS

```
15:08:28  %DDAL-I-ENDDATTRM, ending data transfer/modification

15:08:28  %DDAL-I-COPY_EXIT, normal copy process termination

$ LOGOUT
$
  DBA         job terminated at 28-JAN-1996 15:08:39.42

  Accounting information:
  Buffered I/O count:          437      Peak working set size:  2267
  Direct I/O count:            372      Peak page file size:    7319
  Page faults:                6311      Mounted volumes:           0
  Charged CPU time:     0 00:00:07.35   Elapsed time:     0 00:01:07.69
```

When the copy process runs, it executes your LOGIN.COM procedure. You might want to tailor your LOGIN.COM file so that only relevant portions of it are executed for the copy process. Doing so avoids having the entire LOGIN.COM procedure appear in the copy process log file. For an explanation of how to tailor your LOGIN.COM procedure, refer to Section 6.4.

The callouts in Example 6–9 are keyed to the descriptions that follow:

❶ The SHOW, RUN, and LOGOUT commands are reset to ensure that the copy process runs properly. The SET VERIFY command displays the contents of command procedures on a terminal or in a log file. The SHOW PROCESS command displays information about the process under which the copy process runs.

❷ The copy process runs as a detached process. The copy process log file stores information about the detached process under which the copy process runs.

❸ DDAL$CP_CONTINUE is defined assuming the transfer will be successful. The prologue and epilogue procedures, as well as the copy process image, can change this in case of error. The DDAL$CP_ACTION logical name indicates this is an extraction transfer. DDAL$CP_RETRY_ITERATION shows that this is the first execution (zero retries). These logical names can be tested by the prologue and epilogue procedures.

❹ The copy process log file documents the version of Replication Option under which the copy process is currently running.

❺ Information is written to the copy process log file about the OpenVMS process under which the copy process image is running: enabled privileges, image privileges, and the image name of the copy process.

❻ The log file lists the name of the transfer database, the transfer, and the transfer type.

❼ The copy process log file attaches a timestamp to many copy process events to help track the activities of the copy process running the transfer. Some events, such as "creating domain", are not stamped with a time. In older versions of Replication Option, global field was another name for an SQL domain.

❽ The copy process log file includes summary statistics for each table. In this case, it is the EMPLOYEES table.

❾ The copy process log file includes summary statistics for each table. In this case, it is the SALARY_HISTORY table.

❿ Summary statistics appear at the end of the log file. Row counts are not included for tables other than those you specified in the transfer definition. For example, counts are not given for the RDB$VINTAGE table despite the fact that rows are written to it.

The copy process also creates a log file each time you delete a replication transfer.

The copy process log file can also display information not shown in Example 6–9, including:

- A replication update log, which lists (not by default) the source rows that cause some operation to be performed on the target database. For each row, the information listed includes:
  - The database key of the source row
  - The target operation (insert, update, or delete)
  - The target table name

  Refer to Section 6.10.2 for information about changing the default to permit logging of information about source database rows during a replication update transfer.
- Replication update statistics, including the transaction count that provides the number of transactions applied on the target database during a replication transfer update.

  The largest value that a log file can show in a summary display field is 4,294,967,295. Values that exceed this number appear as a string of asterisks, for example **********. The copy process log file displays asterisks under these conditions for all transfer types: extraction, extraction rollup, and replication.

Example 6–10 shows part of a copy process log file that contains summary statistics with overflow values.

Example 6–10 Sample Summary Statistics with Overflow Values Recorded

```
16:13:29  %DDAL-I-SUMRYSTAT,              SUMMARY STATISTICS
          %DDAL-I-ROWSFMSRC, ********** rows copied from DISK8:[CP]PERS.RDB
          %DDAL-I-ROWSTOTGT, ********** rows copied into EXTRACT_TARGET
16:13:30  %DDAL-I-ENDDATTRM, ending data transfer/modification
16:13:30  %DDAL-I-COPY_EXIT, normal copy process termination
```

All counts in the statistical summary can overflow, and any of the totals based on individual counts can overflow as well.

- It is possible that when your transaction numbers reach or exceed 4,000,000,000, Replication Option will enter a message similar to the following in the copy process log file:

  ```
  ** Warning: Source transaction number 4000000000 is near its
  ** maximum value. Please consult the Replication Option Handbook
  ** for instructions about how to reset the source database and
  ** its associated transfers.
  ```

  You will only see this message if you examine the copy process log files. This message will not affect any currently executing transfers. However, it is intended to alert you that you need to reset the source database and its transfers to avoid exceeding the maximum transaction number allowed. (The maximum transaction number supported by Replication Option may be different from the number supported by Rdb.)

  If you do get this message, you should at your convenience, take the following steps to reset the source database and its transfers:

  1. Stop all applications that would make data inserts, updates, or deletes to the source database.
  2. Stop all transfers to and from the source database.
  3. Delete all rows from the RDB$CHANGES table.
  4. Update the row and column in the RDB$CHANGES_MAX_TSER table to the value 1. (Note that this table has only 1 row and 1 column.)
  5. Perform an export/import so that the transaction numbers are reset to start at number 1.

6. Reinitialize all replication transfers that transfer data from the source database. Replication transfers that transfer data into that database use it as a target of the data transfer, not as a source. The latter transfers do not have to be reinitialized.

7. Restart the replication transfers.

6.10.2 Copy Process Option for Logging Individual Row Keys

The copy process can log information about source database rows during a replication update transfer. This information includes:

- The database key of the source row (hexadecimal format)
- The type of operation done at the target database (insert, update, or delete)
- The name of the target table affected

The copy process attempts to translate the environment variable DDAL$CP_LOG_OPTIONS_transfername in the standard logical table search list, LNM$FILE_DEV. Typically, this is set up for names to be searched in the PROCESS, JOB, GROUP, and then SYSTEM environment variable tables on OpenVMS.

For example, if the transfer being executed is named XYZ, the environment variable will be DDAL$CP_LOG_OPTIONS_XYZ.

The environment variable, if defined, must translate to one of two keywords (either uppercase or lowercase is permitted):

- SRC_DB_KEYS . . . enables logging
- NO_SRC_DB_KEYS . . . disables logging

If the environment variable associated with the transfer is undefined, the copy process attempts to translate the more generic environment variable DDAL$CP_LOG_OPTIONS in the same way. If neither environment variable is defined, the default is to not log information.

As an alternative to defining logical names for this procedure, you could make use of the configuration file, DDAL.RC, to specify environment variables. See Section B.1 for more information on DDAL.RC.

6.10.3 Using Transfer Monitor Log Files

The transfer monitor creates a log file that contains information about events that occur during a transfer. The log file typically contains a series of hexadecimal lists that duplicate mailbox messages passed between an interface product (such as SQL) and the transfer monitor, or between the transfer monitor and a copy process. OpenVMS mailboxes serve as a way for the transfer monitor to communicate with user interfaces, such as SQL, and with copy processes. Each hexadecimal list is annotated with a message to explain the operation reflected within the list.

A header marks the beginning of the log file and lists the Replication Option version number and information about the process under which the transfer monitor is running. Following the header and interspersed with hexadecimal lists, the log file can contain messages that indicate:

- A request has been made
- The beginning and end of a request
- The beginning and end of a copy process
- A commit request
- A commit message
- Error conditions, when they occur
- A rollback transaction request
- A rollback transaction message

Which messages appear in the transfer monitor log depends on the activity of the transfers that the log file tracks.

Every time your system starts, the transfer monitor starts and creates a transfer monitor log file. The location of the transfer monitor log is specified during the installation of Replication Option on your system.

The transfer monitor log file name is a system logical name, DDAL$TM_LOG_FILE, which translates to a file specification that has the following format:

```
DDAL$node_name_TR_MON.LOG
```

Example 6–11 shows a simple but complete transfer monitor log file. The sample log file shows messages received to start a transfer and shows the status of the transfers. The log also shows the creation of a copy process to perform the transfer and completion of that process. Embedded in Example 6–11 are numbers, or callouts, that mark key areas of the log. Descriptions keyed to the callouts follow the log example.

Example 6–11 Transfer Monitor Log File

```
                Copyright (c) 1986, 1996 by
             Oracle Corporation, Redwood Shores, CA

❶ Product version: Replication Option V7.0

  logging is enabled for TIMESTAMP, PROCESS, ERROR, TRANSACTIONS, INTERRUPTS,
   TRANSFERS, FAILURES


❷ ----- 16-MAR-1996 12:46:56.71 ----- PROCESS        -------------------------
  Process name:  SYSTEM_TM                   Priority:  5
  Username:  SYSTEM                          UIC:  [LOCNOD,SYSTEM]
  Nodename:  LOCNOD                          PID:  47C0ECD2

  Privileges currently enabled:
      CMKRNL, SYSNAM, DETACH, SETPRV, TMPMBX, WORLD, SYSPRV, BYPASS, SYSLCK

  Image privileges:


❸ Image name:
      $222$DUA25:[SYSTEM.DDD061]TR_MON.EXE;357

❹ transfer monitor should be run as a detached process

  ----- 16-MAR-1996 12:46:58.49 ----- INTERRUPTS     -------------------------
❺ configuring complete; resuming (or beginning) normal operation


❻ Maximum copy processes for this node set at 20

  ----- 16-MAR-1996 15:05:39.67 ----- INTERRUPTS     -------------------------
❼ received message from user with PID 47C0EE58  at 16-MAR-1996 15:05:39.60

❽ 00000001 0C000001 484E073D 504D455F 45594F4C ........=.NH_EMPLOYE 01C4
  20205345 20202020 20202020 20202020 20202020 ES                   01D8
  00000220 00000000 00000000 00000000 00000000  ..................... 01EC
   followed by 452 hexadecimal zeros (%x'00').


❾ Establishing link with transfer monitor


❿ Request to START TRANSFER has been received

  ----- 16-MAR-1996 16:32:19.90 ----- TRANSFERS      -------------------------

⓫ copy process spawned for transfer NH_EMPLOYEES
  scheduled to be executed at  16-MAR-1996 16:32:19.84
  there are 01 copy processes active


  ----- 16-MAR-1996 16:32:20.63 ----- INTERRUPTS     -------------------------
⓬ sent message to user with PID 47C0EE58
```

(continued on next page)

Example 6–11 (Cont.) Transfer Monitor Log File

```
00010001 00000000 00000000 00000000 00000000 ..................... 01D8
00000000 00000000 0016E907 00000001 00000000 .........é.......... 01EC
 followed by 472 hexadecimal zeros (%x'00').

⓭ Started transfer named NH_EMPLOYEES

----- 16-MAR-1996 15:05:44.07 ----- TRANSFERS       -------------------------

next transfer scheduled is DOC_MANAGER


----- 16-MAR-1996 15:05:44.71 ----- INTERRUPTS      -------------------------
received message from user with PID 47C0EE58  at 16-MAR-1996 15:05:44.63

00000001 0A000001 00000000 00000000 00000000 ..................... 01EC
 followed by 492 hexadecimal zeros (%x'00').

Request to commit a transaction has been received


----- 16-MAR-1996 15:05:45.59 ----- INTERRUPTS      -------------------------
sent message to user with PID 47C0EE58

00010001 00000000 00000000 00000000 00000000 ..................... 01D8
00000000 00000000 20FC000A 00000001 00000000 ..........ü ........ 01EC
 followed by 472 hexadecimal zeros (%x'00').

Committed transaction


----- 16-MAR-1996 15:06:48.14 ----- TRANSFERS       -------------------------

⓮ execution of transfer NH_EMPLOYEES
finished at  16-MAR-1996 15:06:48.01

⓯ transfer was successful
there are 00 copy processes active

----- 16-MAR-1996 15:07:47.21 ----- INTERRUPTS      -------------------------
received message from user with PID 47C0EE58  at 16-MAR-1996 15:07:47.12
```

(continued on next page)

Example 6–11 (Cont.) Transfer Monitor Log File

```
00000001 0C000001 484E063D 504D455F 45594F4C ........=.NH_EMPLOYE 00D4
20205345 20202020 20202020 20202020 20202020 ES                   00E8
FF030020 32434F44 4D5B3A31 48535241 5D4C4C41  .....DUA15:[SYSTEM] 00FC
244C5153 574F4853 4152545F 5F46534E 30433734 SQL$SHOW_TRANSF_47C0 0110
38354545 504D542E 2020203B 20202020 20202020 EE58.TMP;           0124
20202020 20202020 20202020 20202020 20202020                     0138
20202020 20202020 20202020 20202020 20202020                     014C
20202020 20202020 20202020 20202020 20202020                     0160
20202020 20202020 20202020 20202020 20202020                     0174
20202020 20202020 20202020 20202020 20202020                     0188
20202020 20202020 20202020 20202020 20202020                     019C
20202020 20202020 20202020 20202020 20202020                     01B0
20202020 20202020 20202020 20202020 20202020                     01C4
20202020 20202020 20202020 20202020 20202020                     01D8
20202020 20202020 20202020 20202020 00202020                    . 01EC
 followed by 212 hexadecimal zeros (%x'00').

Establishing link with transfer monitor

16 Request to SHOW TRANSFER has been received

----- 16-MAR-1996 15:07:49.38 ----- INTERRUPTS      -------------------------
user SYSTEM with PID 47C0EE58 is connected to transfer monitor

----- 16-MAR-1996 16:32:40.96 ----- INTERRUPTS      -------------------------
sent message to user with PID 47C0EE58
```

(continued on next page)

Example 6–11 (Cont.) Transfer Monitor Log File

```
19000001 0000021E 00000C1A 52414D00 4C414853 ..............SYSTEM 0000
2020204C 001F0320 484E0000 504D455F 45594F4C .    .....NH_EMPLOYE 0014
20205345 20202020 20202020 20202020 20202020 ES                   0028
00010420 05010000 00000023 6C706D65 6565796F  .......#...employee 003C
68772073 696C206F 69206576 654E206E 61482077 s who live in New Ha 0050
6873706D 06657269 00000029 00240001 32434F44 mpshire.).....$.DUA1 0064
4D5B3A31 48535241 2E4C4C41 30444444 505D3136 5:[SYSTEM.DDD061]P 0078
4F535245 4C454E4E 4244522E 00210700 654E0000 ERSONNEL.RDB..!...Ne 008C
6F642077 3A313263 72616D5B 6C616873 64642E6C w .dua15:[system.dd] 00A0
31363064 5F686E5D 08706D65 00000020 32434F44 d061]nh_emp. ...DOC2 00B4
4D5B3A31 48535241 5D4C4C41 455F484E 4F4C504D 1:[SYSTEM]NH_EMPLO 00C8
53454559 474F4C2E 00013D0C 766F4D00 61742065 YEES.LOG.=...Move ta 00DC
0D656C62 2020200A 6C655320 20746365 0D6C6C41 ble..    Select All. 00F0
2020200A 20202020 20202020 504D4520 45594F4C .            EMPLOYE 0104
44495F45 200A0D2C 20202020 20202020 4C202020 E_ID,..           L 0118
5F545341 454D414E 200A0D2C 20202020 20202020 AST_NAME,..          012C
46202020 54535249 4D414E5F 0A0D2C45 20202020    FIRST_NAME,..     0140
20202020 20202020 4444494D 495F454C 4954494E        MIDDLE_INITI 0154
0D2C4C41 2020200A 20202020 20202020 44444120 AL,..           ADD 0168
53534552 5441445F 2C315F41 20200A0D 20202020 RESS_DATA_1,..       017C
20202020 44412020 53455244 41445F53 325F4154      ADDRESS_DATA_2 0190
200A0D2C 20202020 20202020 43202020 2C595449 ,..            CITY, 01A4
20200A0D 20202020 20202020 54532020 2C455441 ..           STATE, 01B8
20200A0D 20202020 20202020 4F502020 4C415453 ..           POSTAL 01CC
444F435F 200A0D45 20202020 46202020 206D6F72 _CODE..        From  01E0
4C504D45 4545594F 200A0D53 00000000 00000000 EMPLOYEES.. ........ 01F4

Output from SHOW TRANSFER - more to come

----- 16-MAR-1996 16:32:43.49 ----- INTERRUPTS      -------------------------
sent message to user with PID 47C0EE58
```

(continued on next page)

Example 6–11 (Cont.) Transfer Monitor Log File

```
20010002 20202020 68572020 0D657265 2020200A ...       Where..    0084
20202020 20202020 61747320 3D206574 484E2720          state = 'NH 0098
00581B27 1F030000 4E000000 4D455F48 594F4C50 '.X........NH_EMPLOY 00AC
20534545 20202020 20202020 20202020 20202020 EES                   00C0
08102020 00000000 DE7D650C 120098A8 00000008   .......e}¤....... 00D4
CF1DCC00 FFFFFFFB 00000111 02130200 03000000 .Ì.Ïû.............. 00E8
00081400 F4000000 FEB4EA31 5A1CFFFF 03000000 .......ô1ê...Z.... 00FC
0000001F 455F484E 4F4C504D 53454559 20202020 ....NH_EMPLOYEES     0110
20202020 20202020 20202020 1D202020 00000001                ..... 0124
00041E01 83AB0000 081F0126 00000000 37DEA2BD ......«.&.......½¢7 0138
210098D7 00000008 F8CA2E00 0098D743 00000820 ..!......ÊøC.. ... 014C
CD3B4000 98D743F8 008E3000 1F030000 4E000000 .@;ÍøC..0.........N 0160
4D455F48 594F4C50 20534545 20202020 20202020 H_EMPLOYEES          0174
20202020 20202020 65312020 20000000 28202020           1e...    ( 0188
4E454449 49464954 5B3D5245 4C495551 414D2C4C IDENTIFIER=[LOCNOD,S 019C
41485352 2C5D4C4C 45434341 533D5353 43454C45 YSTEM],ACCESS=SELEC  01B0
4C412B54 2B524554 504F5244 4342442B 294C5254 T+ALTER+DROP+DBCTRL) 01C4
20200A0D 49282020 544E4544 45494649 2A5B3D52 ..    (IDENTIFIER=[* 01D8
2C5D2A2C 45434341 4E3D5353 29454E4F 000000FF ,*],ACCESS=NONE).... 01EC
 followed by 132 hexadecimal zeros (%x'00').
```

⓱ SHOW TRANSFER information finished

```
----- 16-MAR-1996 15:07:51.26 ----- INTERRUPTS      -------------------------
sent message to user with PID 47C0EE58

00010001 00000000 00000000 00000000 00000000 ..................... 01D8
00000000 00000000 00000006 00000001 00000000 ..................... 01EC
 followed by 472 hexadecimal zeros (%x'00').
```

⓲ Transfer monitor can now output result of SHOW TRANSFER

```
----- 16-MAR-1996 15:07:52.16 ----- INTERRUPTS      -------------------------
received message from user with PID 47C0EE58  at 16-MAR-1996 15:07:52.09

00000001 0A000001 00000000 00000000 00000000 ..................... 01EC
 followed by 492 hexadecimal zeros (%x'00').

Request to commit a transaction has been received


----- 16-MAR-1996 15:07:53.08 ----- INTERRUPTS      -------------------------
sent message to user with PID 47C0EE58

00010001 00000000 00000000 00000000 00000000 ..................... 01D8
00000000 00000000 20FC000A 00000001 00000000 ..........ü ........ 01EC
 followed by 472 hexadecimal zeros (%x'00').
```

(continued on next page)

Example 6–11 (Cont.) Transfer Monitor Log File

```
----- 16-MAR-1996 15:07:53.12 ----- INTERRUPTS      ------------------------
⓳ user with PID 47C0EE58 has exited
```

The callouts in Example 6–11 are keyed to the descriptions that follow:

❶ The version of Replication Option that the transfer monitor is currently running under.

❷ Information about the process under which the transfer monitor is currently executing and the privileges assigned to that process.

❸ The name of the transfer monitor image and its directory location.

❹ A message indicating that the transfer monitor runs as a detached process and no attempt should be made to run the image interactively.

❺ Initialization of the transfer monitor is complete, and the monitor is ready to accept requests.

❻ The number of concurrent transfers allowed to execute on this CPU is 20 (the default).

❼ The transfer monitor has received notification that a user has issued a request. In this case, the request is to start a transfer.

❽ A hexadecimal listing that represents the actual messages sent by way of OpenVMS mailboxes between the transfer monitor and the user or the transfer monitor and the copy process.

❾ The user is initiating a connection to the transfer monitor.

❿ The transfer monitor has received a request to start a transfer.

⓫ The transfer monitor spawns a copy process to administer the transfer and checks the number of currently active copy processes.

⓬ The transfer monitor has sent a message to the user indicating that it has processed the user request.

⓭ The transfer monitor has started a transfer.

⓮ Execution of the transfer has ended at the specified time.

⓯ The transfer ended successfully, leaving no copy processes active.

⓰ The transfer monitor has received a request to show transfer information.

⓱ The transfer monitor has finished gathering information from the transfer database.

⓲ The transfer monitor has received the information requested and is ready to display it.

⓳ The user has exited the language interface and has disconnected its link to the transfer monitor.

To protect sensitive information, such as user passwords, from access by unauthorized users, the transfer monitor sets protection on its log file according to the following rules on OpenVMS:

- For UIC-based protection

  The monitor log file protection is (S:RWED,O:W,G,W).

- For ACL-based protection

  The monitor log file protection inherits the access control list (ACL) of the subdirectory containing the log file. The transfer monitor does not attempt to delete any access control entry (ACE) in the ACL. The database administrator or system manager is expected to determine an ACL that is suitable for all authorized Replication Option applications and users.

When Replication Option executes in a VMScluster configuration, a transfer monitor executes on each node in the cluster. The transfer monitor records in its log file changes in the cluster configuration or changes in the number of transfer monitors currently operating in the cluster. The following entries in the transfer monitor log indicate a change in the number of monitors running in the cluster:

```
-----  16-MAR-1996 13:07:53.88 ----- INTERRUPTS  ----------------------
transfer monitor is reconfiguring due to node loss or addition.

-----  16-MAR-1996 13:07:53.90 ----- INTERRUPTS  ----------------------
configuring complete; resuming (or beginning) normal operation.
```

If the VMScluster node you are using fails, Replication Option uses an automatic failover mechanism to transfer control of the transfer execution to another monitor in the cluster. In this event, if the transfer is in the ACTIVE or RETRYING state, the transfer monitor will restart the transfer almost immediately. If the transfer monitor restarts an active transfer because of a node failure, Replication Option considers this restart an initial attempt rather than a retry.

6.10.4 Using Event Logs

In cases where there are communication problems between the transfer monitor and the user interface (for example, interactive SQL), and an intermediary called DDAL$SHR.EXE, it can be helpful to log certain events. Enabling event logging allows you to see:

- When messages are received by DDAL$SHR.EXE from the user interface (SQL)
- When DDAL$SHR.EXE sends and receives messages to/from the transfer monitor
- When DDAL$SHR.EXE replies to the user interface

Replication Option sends the events it logs to SYS$OUTPUT. You can capture that output to a log file for later review. Events can be logged for the following commands:

```
ALL
ALTER
CREATE
DROP
GRANT
MESSAGE
REINITIALIZE
REVOKE
SHOW
START
STOP
```

The ALL and MESSAGE statements are not SQL statements that direct Replication Option. ALL enables all event classes. MESSAGE enables the logging of messages between SQL and DDAL$SHR.EXE and between DDAL$SHR.EXE and the transfer monitor.

See Section B.10 for information about the option used to enable event logging, DDAL$LOG_TRMON_CLIENT_EVENTS.

6.11 Using Checkpoints for Quicker Recovery from Some Transfer Failures

The Replication Option CHECKPOINT option on the CREATE TRANSFER statement (and ALTER TRANSFER statement) allows you to specify that Replication Option should periodically commit changes it has made to the database as the database transfer progresses. In the event of a recoverable error involving access to the target database, Replication Option will be able

to restart the transfer at the point where data was last committed and try to restart the transfer for the time period you specify (up to one hour). The checkpoint option is available for all transfers.

For a replication update to support the checkpoint option, the copy process must be able to store all the data for the current transaction in virtual memory so that it can restart the transaction without also having to restart the source transaction. This means that the size of the virtual memory buffer may have to be changed. You can do this using the DDAL$CHANGES_BUFFER_SIZE environment variable, as discussed in Section B.2.

The CHECKPOINT option is particularly useful when a transfer is likely to last several hours. For example, suppose your transfer typically takes 20 hours to complete. If you do not specify the checkpoint option and the transfer fails between the 19th and 20th hour, all that work is lost and the transfer must be re-executed from the beginning.

Suppose instead that you specify the CHECKPOINT option in your transfer definition and state that recovery is to occur within 60 minutes. Now if a recoverable error occurs between the 19th and 20th hour, all data transferred into all but the current table will have been committed. Replication Option only has to retry transferring data to the current table and then continue with those remaining to be transferred.

The following example shows how to specify the checkpoint option in a CREATE TRANSFER statement:

```
CREATE TRANSFER DOC_MANAGER TYPE IS EXTRACTION
   MOVE TABLES
      SELECT LAST_NAME,
             FIRST_NAME,
             EMPLOYEE_ID,
             DEPARTMENT_CODE,
             SUPERVISOR_ID
      FROM CURRENT_JOB WHERE JOB_CODE = 'DMGR'
   TO NEW FILENAME USER1:[DOCUMENTATION]DOC_MANAGER
   LOG FILE IS DMGR.LOG
   CHECKPOINT RECOVER WITHIN 60 MINUTES
   COMMENT IS 'List of documentation managers';
```

The checkpoint option is ignored by Replication Option when an Rdb database is being created. An Rdb database is created during an extraction, extraction rollup, or initial replication transfer where the CREATE TRANSFER statement uses the TO NEW FILENAME clause. It is created using batch-update transactions; when a batch-update transaction fails, the database cannot be recovered and must be re-created from the beginning.

Checkpoint restart does not apply to source database errors. If an error occurs while Replication Option is accessing a source database, there is no mechanism for Replication Option to pick up where the error occurred. Restarting the transfer from the beginning is the only way to ensure consistent data.

If an error occurs while Replication Option is accessing a target database (and checkpoint restart is enabled), the failed transaction is retried regardless of the type of error that occurred. No distinction is made between soft and hard errors. A **soft error** is one that might eventually disappear, such as a network disconnection. A **hard error** is one that would probably be repeated again and again under the same circumstances; for example, a program access violation.

If you are experiencing the failure of long transfers, you should weigh the benefits of the checkpoint option (taking network reliability into account) against the benefits of a fast load operation and decide whether the checkpoint option makes sense for your business needs.

See Section 9.4 for the checkpoint syntax.

Note

In a replication transfer, if the checkpoint option is used along with database key logging, the copy process log file might contain duplicate entries. This is because information about each change applied to the target database during a replication update transfer is written to the copy process log file. If an error occurs before those changes are committed to the target database and a checkpoint restart occurs, the same changes are reapplied in a new target transaction. However, the summary transaction counts at the end of the update process will accurately reflect the number of insert, update, and delete operations committed to the target database.

6.12 User-Added Data Definitions in a Target Database

The Replication Option CREATE TRANSFER statement enables you to name tables, views, and columns to be transferred to a target database. Replication Option transfers both the table definitions and the data in these tables. Domains on which columns are based are implicitly transferred by Replication Option as well. Other data definitions and database attributes—indexes, constraints (including primary and foreign key attributes), triggers, multischema attributes, multifile storage area attributes, query outlines, stored procedures, and protection attributes on database objects—cannot be named in a CREATE TRANSFER statement, and are not transferred from a source

database to a target database. You also may want to create additional tables and views that do not exist in the source database or databases but that can be useful in the target database.

You can add such data definitions to a target database as part of the transfer process. You can create command procedures that are run immediately prior to or immediately following a transfer execution.

Refer to Chapter 7 for information on developing command procedures.

6.12.1 Effects of Transfer Re-Execution on User-Added Definitions

When you create data definitions in a target database, it is important to understand what happens to these definitions when a transfer is repeated. The type of transfer, replication or extraction and extraction rollup, also influences what happens. Transfers to existing databases require special data definition considerations and are discussed in Chapter 8.

6.12.1.1 Extraction and Extraction Rollup Transfers

An extraction or extraction rollup transfer whose definition includes the TO NEW clause, creates the target database files each time the transfer executes. If such a transfer executes only once, the resulting target database remains independent of any subsequent changes made to the source database or databases. Therefore, you can make additions and modifications to the target database definitions without considering the source database. This is not the case with transfers that may be repeated.

If a TO NEW extraction or extraction rollup transfer repeats on a regular basis, each subsequent execution of the transfer creates new versions of the database files. Replication Option does not delete prior versions of the database files nor does it copy anything from prior target databases into the latest version. Note that in a Digital UNIX system, prior to re-executing such a transfer, you must delete the target database files from the previous transfer. Any changes made to the earlier databases that you want to have appear in the latest version must be added or copied manually. Data definitions, such as table indexes, can be created automatically in an epilogue procedure (see Chapter 7) each time a transfer executes. If you add your own tables to the target database and write data to them, you must execute a procedure to re-create the tables and repopulate them in the latest target files.

Alternatively, an extraction transfer can specify the TO EXISTING clause. In this case, the target database is not created as part of the transfer execution. However, target domains, tables, and views are created by the transfer. Refer to Chapter 8 for information about adding data definitions as part of transfers to existing databases.

6.12.1.2 Replication Transfers

The first time a replication TO NEW transfer executes, it performs many of the same operations as an extraction transfer. In particular, it creates the target database files and then writes domains, tables, and views using information from the source database. Adding data definitions to a replication target database requires the same considerations as an extraction transfer. Subsequent executions of a replication transfer, called replication updates, do not create a new target database nor do they change the target data definitions. Replication updates transfer data additions, changes, and deletions that have occurred in the source database since the previous transfer.

Sometimes a replication transfer needs to be reinitialized. Reinitialization forces the next execution of the transfer to mimic the initial transfer rather than an update. If a TO NEW replication transfer is reinitialized, its next execution creates a new set of target database files, just as the re-execution of an extraction transfer would. Note that in a Digital UNIX system, prior to re-executing such a transfer, you must delete the target database files from the previous transfer. A reinitialized TO EXISTING replication transfer does not create the database files, but normally it does modify target data definitions, if necessary, and drops and retransfers all rows of a transferred table. Note that a reinitialization of a replication transfer with the NO DELETE attribute does not delete rows in the target database that have been deleted from the source database. See Section 8.2.3 for more information on replication transfers with the NO DELETE attribute. Also see Section 8.2.5 for more information about the rules concerning deletion of data owned by others.

Oracle Corporation recommends that you do not alter the definition of domains and tables transferred by a replication transfer. If you do, you run the risk of causing subsequent transfers to fail because of a mismatch in the definitions of the source tables and the target tables. An exception to this is if you need to add columns to a target table. To do so, you must first define the environment variable, DDAL$CHECK_TARGET_DEFINITIONS. See Section B.4 for more information.

6.12.2 Adding Indexes to Target Tables

Once an extraction, extraction rollup, initial replication, or reinitialized replication transfer completes the transfer process, you can add indexes for the transferred tables. Indexes help speed up the process of data retrieval.

Target indexes do not have to be identical to those in the source database. The source database might be optimized for different applications, such as data entry. You can create indexes on the target tables to optimize data retrieval.

Consider the following when you create indexes for a replication transfer database:

- Indexing replication updates

  When the copy process forwards changes that have been made to the source database, the target table indexes are automatically updated.

  Remember that indexes represent overhead that might affect the performance of some transfer operations. If you plan to make a large number of updates, you might want to delete the indexes before transferring changes and then re-create the indexes after the transfers.

For information about adding indexes using the CREATE INDEX statement, refer to the Oracle Rdb SQL Reference Manual for the version of Oracle Rdb you are using.

6.12.3 Adding Constraints for Target Tables

You may want to add constraints to transferred tables that serve as reference data. These tables can be used in constraints on other tables that your applications maintain in the target database.

Do not add constraints to the target database if the tables are exclusively written by a Replication Option transfer. Constraints on such tables should be implemented in the source database at the point of entry of the original data.

For information about adding constraints using the ALTER TABLE statement, refer to the Oracle Rdb SQL Reference Manual for the version of Oracle Rdb you are using.

6.12.4 Adding Triggers for Target Tables

A **trigger** defines actions that occur either before or after a table is updated. For example, you can create a trigger such that when a row is updated in one table, the triggered action causes additional rows to be updated in other tables that are related to the first by key values.

When creating triggers on tables managed by Replication Option transfers, beware of possible side effects. A transfer can fail if the triggered action fails. The triggered action can fail if it violates constraints on other tables, constraints not directly associated with the transferred tables.

For information about adding triggers using the CREATE TRIGGER statement, refer to the Oracle Rdb SQL Reference Manual for the version of Oracle Rdb you are using.

6.12.5 Adding Multischema Attributes to Target Tables

When a transfer is made to a new database, Replication Option creates a non-multischema database. Replication Option does not transfer multischema information from source catalogs and schemas.

After a transfer has completed, or as part of an epilogue procedure, you can alter the target database to add multischema attributes. All the tables and views that Replication Option transferred then become part of the default catalog and schema. However, you will not be able to move already existing tables from the default catalog and schema to a catalog and schema you specify.

When you transfer data to an existing multischema database, the tables get placed in the default catalog and schema automatically.

6.12.6 Adding Multifile Storage Areas to a Target Database

When Replication Option creates a target database, it is a single-file database even if the source database has multiple storage areas. There is no provision in the CREATE TRANSFER statement for specifying storage areas in a target database. However, after a transfer has executed you can use the SQL EXPORT and IMPORT statements to convert the target database to the multifile format. This can be done either manually or as part of an epilogue procedure associated with the transfer.

6.12.7 Modifying Protection for Target Databases and Tables

Objects in a database, such as tables and views, can be protected in such a way that only certain people are privileged to use them. When Replication Option transfers these objects from a source database to a target database, it does not also re-create the associated privilege lists. Instead, the objects are given whatever default privileges are conferred on them by the target database system.

For example, if the target system is Rdb, the OpenVMS account used to create the objects is granted all access privileges and no one else (the public) receives access rights.

With Rdb for Digital UNIX Alpha, only the `dbsmgr` account has the privilege to access everything in a database. Rdb does not treat the `root` account as a privileged account. The GRANT and REVOKE statements may also be used to allow or deny access to a particular database object.

After a transfer has executed, you may grant to other users of the target database access to tables, views and other database objects as necessary.

For information about modifying user privileges and database protection using the GRANT and REVOKE statements, refer to the Oracle Rdb SQL Reference Manual for the version of Oracle Rdb you are using.

6.12.8 Adding Tables to Target Databases

When Replication Option executes a transfer to an existing target database, local tables already exist. You can add tables either before or after transfer execution. You can also add tables following the transfer, if the target database is created by the transfer. However, note the implications of re-executing transfers explained in Section 6.12.1.

For information about adding tables using the CREATE TABLE statement, refer to the Oracle Rdb SQL Reference Manual for the version of Oracle Rdb you are using.

6.12.9 Adding Views to Target Databases

When Replication Option executes transfers to an existing database, local tables and views already exist. You can add views either before or after transfer execution. If you want to add views that reference the transferred tables, you can do so when the transfer completes. Note the implications of re-executing transfers explained in Section 6.12.1.

For information about adding views using the CREATE VIEW statement, refer to the Oracle Rdb SQL Reference Manual for the version of Oracle Rdb you are using.

6.13 User-Added Data in a Target Database

You can insert, update, and delete data rows in target tables not managed by a Replication Option transfer.

You can insert, update, and delete data rows in target tables created by an extraction or an extraction rollup transfer. Any such additions or changes made in the target database are not reflected in the source database. Similarly, any additions or changes made in the source database are not included in the target database unless the transfer is re-executed.

Refer to Section 6.12.1 for the implications of re-executing transfers.

You cannot insert, update, and delete data rows in target tables created by a replication transfer unless you have shared writing enabled and the transfer is to an existing database. See Section 8.2.5 for more information on using the shared-write option with transfers to existing databases. Subsequent replication updates test for the existence of rows in the target database when

such rows are either updated or deleted in the source database. Failure to locate these rows in the target database causes the transfer to fail.

6.14 Special Data Definitions Supporting Replications

When you execute a replication transfer for the first time, the copy process creates special-purpose system tables in the source database and the target database. These system tables contain information Replication Option needs to identify and record updates to the source database and to forward them to the target databases. This section includes a very brief description of part of these additional system tables. Also discussed is a database key column that is added to all transferred tables. Refer to Appendix A for a detailed description of the special-purpose system tables.

6.14.1 Storing Updates to the Source Database for Replication Transfers

The copy process adds the table RDB$CHANGES to the source database. It is used to store all updates to rows in the source database that will be forwarded to target replication databases. When a replication transfer begins execution, all rows in RDB$CHANGES are evaluated to determine which rows will be included in the transfer operation. When the transfer of updated rows completes, Replication Option checks to see if these rows are required by other transfers. If they are not, these rows are purged automatically from the RDB$CHANGES table. Replication Option provides an environment variable, DDAL$PURGE, that you can define to specify if and when RDB$CHANGES is purged. For more information, see Section B.13.

The existence of a replication transfer, regardless of its state, causes changes to RDB$CHANGES. Rows are added to RDB$CHANGES even for transfers that are suspended for a long time. As a result, the number of rows in RDB$CHANGES and the overall size of the database both continue to grow.

You can encounter disk space and performance problems with replication transfers if you have applications that never detach from the source database. Processes that update replicated tables should detach and reattach from the source database on a regular basis so new insertions of data into the RDB$CHANGES table will restart with an insertion point at the beginning of the table and space will be reused. Otherwise, the user processes will expand the logical area when they reach the end of it and they will eventually expand the physical area also when it becomes full.

This problem is especially of concern for people trying to achieve near real-time replication. This mode is typical of high-availability situations, with processes that tend to stay attached for extended periods of time, such as online transaction processing servers. In this mode, replications are frequent (every 2 to 5 minutes), but they do not transfer large amounts of data.

In these cases, the application processes continue to add rows in the RDB$CHANGES table. The copy processes execute and copy the data into the target database and then erase that data from the RDB$CHANGES table. However, the application processes do not see these erasures and continue to add rows at the end of the table.

Eventually, the RDB$CHANGES logical area becomes very large which slows the transfers because the copy process always reads the RDB$CHANGES table sequentially.

As a result, it will take longer to find the rows to be transferred because Rdb has to scan the entire RDB$CHANGES table even if you only have a small number of rows to transfer. Transfer execution becomes slower and slower over time, possibly canceling out the "near real-time" replication effect.

Even if application processes disconnect and reconnect daily for reading the source database, the transfers still will always need time and resources directly proportional to the *size* of RDB$CHANGES instead of the number of changes to transfer.

Also, once RDB$CHANGES has grown too large, the copy processes will still take too long to scan the RDB$CHANGES table even if the application processes now disconnect hourly.

The following example shows how to physically clear the pages used by RDB$CHANGES once they become empty:

```
$ RMU/Repair/Initialize=Free_Pages /Table=RDB$CHANGES
$ RMU/Repair/Spam/Abm /Table=RDB$CHANGES
```

The first RMU command purges the allocated but unused pages in the RDB$CHANGES table. The second RMU command corrects the space area management pages (SPAMs) and the logical area bitmap pages (ABMs). You must have exclusive access to the database to perform these two RMU commands. After executing these commands, you should perform a backup of the RDB$SYSTEM table, at least.

6.14.2 Tracking Data Between Target and Source Databases

One of the tables Replication Option adds to a target database is RDB$VINTAGE, for Rdb targets, or DDAL$TRANSFER_INFO, for all other database systems. Among other things, these tables track how up-to-date the data in the target database is with respect to the source data. One of the columns contains date and time: RDB$VINTAGE_TIMESTAMP for Rdb; DDAL$TRANSFER_TIME for all others. When the copy process forwards updates to the replication database, it modifies the value in this column to

indicate the time of the last update. Remember, you can use the date value in this column to determine how current is the data in the replication database.

For every table created in a target replication database, the copy process includes a special column called DDAL$DBKEY. This column contains a database key value of the corresponding source database row. Additionally, the copy process creates an index on this column, one index per table.

For replication or extraction transfers using the shared-write option, Replication Option adds the DDAL$ROW_OWNER column to each transferred table. If the transfer method is replication, Replication Option also adds a unique index based on the DDAL$DBKEY and DDAL$ROW_OWNER columns. The DDAL$ROW_OWNER column records the identifier of the transfer that owns the particular data row. When extraction transfers are re-executed (or replication transfers reinitialized and re-executed), they must delete all data that they own but not data owned by others. The DDAL$ROW_OWNER column allows this to work properly.

Table 6–2 shows the columns that are added to a target table by Replication Option to manage transfers.

Table 6–2 Replication Option Columns in Target Tables

Column Name	Transfers Using This Column
DDAL$DBKEY	All replication transfers
DDAL$ROW_OWNER	Shared-write transfers to existing databases

6.15 Application Requirements for Transfers

If you have existing applications (on the source node) that you want to run on your target databases, you should move the applications to the target site and modify them to run on the target using the resources there. The following sections contain suggestions that can help you modify your applications to run on those target databases.

6.15.1 Storing Database Definitions in the Data Dictionary

Both the location and the structure of the target database can be different from those of the source database. The location can be a different node in the network. The structure might be different in terms of the columns that are included in each of the tables moved from the source database.

If your applications use other Oracle Corporation information management products on OpenVMS systems, you can make the target database definitions known to the data dictionary. You can do this using the SQL INTEGRATE statement to incorporate database, table, and column definitions into the remote data dictionary at the target site. If you do not change the database definitions in the target database, the definitions you store in the data dictionary remain valid for all subsequent database transfers.

If you do change the database definitions in the extraction or extraction rollup database, you should delete the data dictionary node using the SQL DROP PATHNAME statement and use the SQL INTEGRATE statement again if you plan to create another extraction or extraction rollup transfer.

If you expect to make further extraction or extraction rollup transfers, you should determine if the data definition changes have been made to the source database. If the changes have been made to the source database, you must also change the transfer definition. If the changes have not been made to the source database, you must make the data definition changes to the new target database and to the local dictionary as well.

Replication Option provides a REINITIALIZE TRANSFER statement that causes the copy process to create a new replication database on the target node (when the transfer is defined as being to a new database). Because reinitialization causes a new version of the database to be created in the target directory, any indexes and protection definitions created in the previous version of the replication must be re-created in the new replication database. You must issue the SQL DROP PATHNAME and SQL INTEGRATE statements to ensure consistency between the current version of the replication database and the entries in the data dictionary.

When you rely on the data dictionary to make all database entity definitions available to other information management products, you should be the only user accessing the database when you use the following steps to update the entries in the data dictionary:

1. Invoke the target database using either the database file name or path name.
2. Enter local definitions for all indexes and constraints, and commit these definitions.
3. Enter the DISCONNECT ALL statement to detach from the database.
4. Issue the SQL DROP PATHNAME statement to erase the current database definitions from the data dictionary.

5. Issue the SQL INTEGRATE statement to store the complete set of target database definitions in the data dictionary.

Failure to update the data dictionary in this way might result in data definitions stored either in the dictionary or in the database, but not in both. Follow the previously listed steps to avoid inconsistencies between the data dictionary and target databases.

Refer to Chapter 7 for information on using epilogue command procedures on OpenVMS to integrate target metadata into the data dictionary.

6.15.2 Identifying the Location of the Database

Change either the file specification or path name of the database or the logical name that refers to the database you invoke in host language programs so that it refers to the local target database.

For example, the following ATTACH statement in an application program uses the dictionary path name to refer to a central PERSONNEL database in the network:

```
EXEC SQL
 ATTACH 'CDD$TOP.CENTRAL.PERSONNEL';
END-EXEC
```

Because you modified the database path name specification in the ATTACH statement, the application program now refers to a local target database. Furthermore, if you use an OpenVMS logical name to refer to the database path name, you can change its value instead:

```
$ DEFINE DB_PATH CDD$TOP.LOCAL.PERSONNEL
   .
   .
   .
EXEC SQL
 ATTACH 'PATHNAME DB_PATH';
END-EXEC
```

Similarly, the file specification in the ATTACH statement can refer to the central PERSONNEL database:

```
EXEC SQL
 ATTACH 'FILENAME BIGNOD::DB$DISK:[ADMIN.PERSONNEL]PERSONNEL';
```

The following ATTACH statement includes a new file specification that could refer to a target database:

```
EXEC SQL
 ATTACH 'FILENAME PERS$DISK:[PERSONNEL]PERSONNEL';
```

6.15.3 Determining the State of a Target Database

When you are accessing data in a target database, it can be useful for you to know if a transfer execution is underway and also how recently a transfer was executed.

One of the tables Replication Option adds to a target database is RDB$VINTAGE, for Rdb targets, or DDAL$TRANSFER_INFO, for all other database systems. Among other things, these tables track how up-to-date the data in the target database is with respect to the source data. One of the columns contains date and time: RDB$VINTAGE_TIMESTAMP for Rdb; DDAL$TRANSFER_TIME for all others. After the copy process completes execution of a transfer, it records the date and time of the transfer execution in this column. You can use the date value in this column to determine how current the data in the target database is.

Another column, RDBVMS$VINTAGE_TRANSFER_BUSY, for Rdb, and DDAL$TRANSFER_BUSY, for all others, lets you know if a transfer is currently executing or not. When a transfer is executing, the column value is set to "T" (meaning TRUE); when not, to "F" (meaning FALSE). This is done only when the transfer proper is executing, not during prologue or epilogue execution. For replication update operation the column value remains "F".

6.15.4 Modifying Row Descriptions in Application Programs

After Replication Option creates the target database, local applications can process the locally stored data. On OpenVMS systems, you can access the local database and issue the SQL INTEGRATE statement to copy all of the database definitions to the data dictionary. When you complete that task, recompile your local application programs against the data dictionary descriptions.

Because target databases might contain only selected columns from the original tables defined in the source database, you might have to recompile the application programs that you intend to execute against target databases. When a COBOL program includes a COPY statement, for example, a row structure is created in the host program that describes the complete set of columns defined in a particular table. This row description can be used with INSERT, ALTER, or FETCH statements to store, modify, or retrieve an entire row at a time rather than specifying each column explicitly.

Thus, the values of all of the columns in the EMPLOYEES row in the application program are applied to all of the corresponding columns in the EMPLOYEES table. The following COPY statement creates a row description from the data dictionary equivalent to all of the columns defined in the EMPLOYEES table of the central PERSONNEL database, when remote database access is required:

```
COPY "CDD$TOP.CENTRAL.PERSONNEL.RDB$RELATIONS.EMPLOYEES"
        FROM DICTIONARY.
```

Because the database path name now refers to the definitions of the local target database, recompiling the application program will now include the local version of the EMPLOYEES table:

```
COPY "CDD$TOP.LOCAL.PERSONNEL.RDB$RELATIONS.EMPLOYEES"
        FROM DICTIONARY.
```

The following embedded data manipulation statements operate on an entire row at a time. After you precompile the application programs against the local database definitions, the INSERT, ALTER, and FETCH statements will work correctly with the new row descriptions copied from the data dictionary on the local node:

```
   .
   .
   .
&RDB&  STORE E IN EMPLOYEES
&RDB&   USING
&RDB&    E.* = EMPLOYEES
&RDB&  END_STORE
   .
   .
   .
```

6.15.5 Modifying Transaction Share Modes and Lock Types for Extraction or Extraction Rollup Databases

For an Rdb database, a transaction can include both a **share mode** (SHARED, PROTECTED, or EXCLUSIVE) and a **lock type** (READ or WRITE). Application programs accessing the target database might benefit from:

- Less restrictive share modes and lock types than those required to lock specific database resources from competing applications in remote access to the central database.

  You might use the SHARED mode rather than the PROTECTED mode because the target applications no longer have to compete with so many other transactions. For example, using the PROTECTED lock type to access a central database remotely might be necessary to prevent read/write transactions from changing data values while a particular remote application query executes. Local control of an extraction database, however, means that you can determine, with share modes and lock types, which applications are allowed to access the data. In addition, you have more control over when those applications execute. Executing application queries against a locally controlled copy of the data reduces the demand

on the source database and allows greater concurrent access to both the source database and the local extraction database.

- More restrictive share modes and lock types than remote access to the snapshot (read-only) portion of the central database.

  Because the snapshot file (file type .SNP) can contain earlier versions of data values that other read/write transactions might be changing currently, some data values may not be up-to-date. Using a more restrictive share mode might allow the target application to prevent other target applications from changing values until it finishes, temporarily restricting changes other transactions might make. Therefore, you might change the transaction from read-only to read/write when data in the extraction database itself experiences frequent changes due to local (target) user access and modification.

6.15.6 Modifying Transaction Share Modes and Lock Types for Replication Databases

Your applications might include transaction types that permit updates to occur. New transaction types can prevent possible changes to data values in the replication database.

For an Rdb database, a transaction can include both a share mode (SHARED, PROTECTED, or EXCLUSIVE) and a lock type (READ or WRITE). Because the copy process is the only process permitted to update the replicated tables, your application programs must include share modes (SHARED and PROTECTED) and lock types (READ and read-only) that will not interfere with it.

While the preceding information deals with Rdb, it may be applicable to other products. Refer to the *Oracle Rdb7 Distributed Technology Handbook* for information about transaction parameters and how they are enforced within the transparent gateways.

6.15.7 Evaluating Application Schedules for Extraction or Extraction Rollup Databases

If scheduling a routine application to run against the central database results in infrequent execution because of competing transactions from other parts of the network, you can schedule applications that access the extraction databases at regular intervals and run routine applications more often than you could against the central database.

6.15.8 Evaluating Application Schedules for Replication Databases

Because replication databases routinely receive updates according to a predefined schedule, you might need to adjust when applications are scheduled to execute. That is, certain applications can benefit from data that is as current as possible. Therefore, you want to schedule those applications to execute *after* the copy process has completed the latest scheduled update to the replication database.

Consider the following when determining how often to repeat a transfer:

- Replication Option does not impose a minimum time interval for scheduling transfer repetition.
- If a transfer is already running at the time the next scheduled execution is to take place, nothing happens. The current transfer is allowed to complete and the transfer is run again at the next scheduled execution time following the latest completion.
- Tests have shown that Replication Option can re-execute transfers as often as every 20 seconds. This depends on hardware, network connections, storage capacity, system usage, type of target database, activity at the target site, and so on.

6.16 Performance Options for Replication and Extraction Transfers

Replication Option provides several options for enhancing transfer performance. These options apply to both replication and extraction transfers:

- Specifying how many data rows are read and written at a time
- Combining the deleting and writing of data into one transaction
- Transferring data starting with particular tables
- Limiting Concurrent Transfers on a CPU

Each of these options is discussed in the following sections. See Section 6.17 for other performance options that only apply to replication transfers.

6.16.1 Specifying How Many Data Rows Are Read and Written at a Time

When Replication Option extracts data from a source database and writes data to a target database, it reads and writes several rows of data at a time. This reduces the overhead in moving the data and improves execution speed of transfers rather than reading one row and writing one row at a time. This is true for those database and transparent gateway products that support the ability to fetch and write multiple rows of data at a time.

You can specify how many data rows are read and written at a time by Replication Option during an extraction, extraction rollup, or initial replication transfer with the environment variable DDAL$RW_ROW_COUNT. See Section B.14 for more information on this variable.

If you do not define the row count variable, Replication Option reads and writes 16 rows of data at a time by default. If Replication Option reads or writes to a database or transparent gateway that only supports row-at-a-time operation, changing the default row count has little effect on performance. As you change the row count value in an attempt to get faster transfer execution speed, start with row count values close to the default. Gradually increase the row count until you reach a point where further tuning yields no improvement. The larger the row count value, the more dynamic memory will be used by the copy process during transfer execution. You might need to adjust system parameters that affect process memory utilization.

6.16.2 Combining the Deleting and Writing of Data into One Transaction

When an extraction or initial replication transfer to an existing database is repeated, data in the transferred tables must be deleted to make room for the new data snapshot. The default transaction behavior during a transfer is to delete the data in one table and then commit, delete the data in the next table and then commit, until the data in the last table has been deleted, then insert the new data into the first table and commit, and so forth, and finally, insert the new data into the last table and commit. If the target tables have foreign key constraints on them, even though they may be declared to be deferrable (that is, normally evaluated at transaction commit time), the transfer will cause a constraint violation. That is because data from the tables containing the primary keys might be deleted and the transaction committed while a foreign key constraint from some other table was still in effect.

Oracle Corporation recommends you drop all constraints in the transfer prologue, transfer the data, and reapply the constraint definitions in the transfer epilogue. Doing so causes transfers to take longer to complete than would otherwise be the case. As an alternative, you can define the environment variable DDAL$SINGLE_DATA_COMMIT so Replication Option

will delete data and write data into the target tables all in one transaction. See Section B.16 for details about DDAL$SINGLE_DATA_COMMIT.

Deleting and writing data within a single transaction has both benefits and drawbacks.

The benefit is that you do not have to drop and redefine the constraints every time the transfer re-executes. This can prevent large amounts of CPU time from being used and also prevent large numbers of lock conflicts.

The drawback is that you can increase disk space utilization. It is typical for database systems to journal the delete and insert operations in a before-image file. This file exists to allow the tables to be restored to their original state if the transaction failed or was rolled back. Putting the delete and insert operations for all tables into one transaction increases the size of the space needed in the before-image file. If the number of tables and the amount of data per table are relatively small, this might not matter.

A second potential drawback is that the space in the database occupied by the deleted rows might not be reused for the insert data. Some database systems only free the space for reuse after the transaction is committed.

Finally, this option can only be effective if the constraints are declared to be deferrable. If they are declared to be non-deferrable, the constraints will be violated despite Replication Option employing the single-data-commit method of transfer.

6.16.3 Transferring Data Starting with Particular Tables

Normally, a Replication Option copy process begins transferring data from the first table in its list. However, overall performance (decreased elapsed time) for concurrently executing extraction or initial replication transfers can be improved using the DDAL$START_WITH_SOURCE_TABLE environment variable. When this variable is enabled, it instructs a copy process to transfer data starting with a particular table, thus forcing execution of concurrent transfers to start working on different areas of the source database.

See Section B.17 for information on how to use this variable.

6.16.4 Limiting Concurrent Transfers on a CPU

Replication Option has a built-in limit to the number of concurrent transfers it will attempt to execute. The limit is 20. This limit is enforced by Replication Option's transfer monitor, as the monitor is responsible for starting up the copy processes. If more than 20 transfers are scheduled to execute concurrently, some will not start until others have completed their tasks. This limit does not include the running of a copy process when you wish to drop a replication

transfer and erase the definitions from the source database. This situation is handled separately from the normal concurrent transfer quota.

The concurrent transfer limit does not guarantee that 20 transfers will run concurrently. System resources can lower the limit to less than 20. For example, on certain hardware configurations you can use all available CPU cycles when running less than 20 concurrent transfers.

On a small and heavily used system, the number of OpenVMS process slots can be limited. In such an environment, controlling the number of processes started can be crucial to maintaining acceptable system performance. A system manager can limit the number of concurrent transfers that can be executed on a given CPU by defining a systemwide environment variable, DDAL$MAX_COPY_PROCESS. (See Section B.11 for details about DDAL$MAX_COPY_PROCESS.)

For example, you could define DDAL$MAX_COPY_PROCESS as follows to limit concurrent transfers to 5:

```
$ DEFINE/SYSTEM DDAL$MAX_COPY_PROCESS 5
```

When a transfer monitor starts executing, it attempts to translate this environment variable. If the environment variable is not defined, the transfer monitor selects the lesser of the following two values:

- The default limit of 20
- The transfer monitor's process quota called MAXDETACH minus 1

If a transfer monitor is to do any meaningful work, MAXDETACH must have a value of 2 or more: one process to run a normal transfer and the other to perform a replication deletion, if necessary.

If the DDAL$MAX_COPY_PROCESS environment variable is defined, its value must be at least 1, and no more than the built-in limit of 20. The transfer monitor selects the lesser of the following two values:

- The value translated from DDAL$MAX_COPY_PROCESS
- The transfer monitor's process quota called MAXDETACH minus 1

Defining or redefining this environment variable while Replication Option is running has no immediate effect. In order for a transfer monitor to recognize the current setting, you must stop Replication Option and restart it.

If Replication Option is running on several or all nodes in a cluster, you must define this environment variable on each CPU. The value defined for each processor can vary. This is a potential benefit in heterogeneous VMScluster systems where processor power varies.

6.17 Performance Options for Replication Transfers

The performance options described in the following sections apply only to replication transfers:

- Specifying when RDB$CHANGES is purged
- Executing concurrent transfers from the same source database
- Protecting against concurrent writers during a transfer
- Adding indexes to the RDB$CHANGES table

Each of these options is discussed in the following sections. See Section 6.16 for performance options that apply to both replication and extraction transfers.

6.17.1 Specifying When RDB$CHANGES Is Purged

The information presented in this section applies only to replication updates. It does not apply to replication initialization or to reinitialization, for which purging is not applicable.

Deadlocks can occur when several copy processes attempt to purge rows no longer needed in the RDB$CHANGES table. You can control when, and if, a given transfer attempts to purge rows from the RDB$CHANGES table by using the DDAL$PURGE environment variable. See Section B.13 for more information about using DDAL$PURGE.

By performing the purge process during a slack period of the day, you can remove a source of interference for database applications during periods of heavy database activity.

To control when purging is done and by which transfer process, you must change the environment variable definitions at the appropriate time. One possible method is to use the SUBMIT command to submit batch jobs that will execute at any time you want.

Messages appear in the copy process log file that indicate whether or not the copy process is attempting to purge. If DDAL$PURGE or one of its variations is defined, the message indicates which environment variable applies and specifies its definition. If the exclusive purge lock is not obtained immediately, that fact is logged instead.

6.17.2 Executing Concurrent Transfers from the Same Source Database

When several replication transfers are created to copy data from the same source database, and when those transfers are started at the same time, the normal (default) behavior is for the transfers to execute one after the other, not concurrently. This is because of the sequence of operations a copy process performs during an initial replication transfer. During a short period while it is storing the transfer definition in the source database, a copy process requires exclusive access to the database. After storing the transfer definition, the copy process reattaches to the source database in normal mode and starts the long process of copying all the source data to the target database. This prevents other copy processes from gaining exclusive access to the source database so that they can store their transfer definitions. As a result, the other copy processes must wait for the first one to finish its transfer.

This situation could be avoided simply by performing the initial replication transfers at different times, but that reinforces the serial nature of the problem. Instead, you can take the following steps to greatly improve concurrent execution:

1. Disable transfer schedules because it will be necessary to perform the transfers in a specific order. If you do not disable schedules, a transfer could start automatically before the first stage of the initial transfers has been performed. Drop any schedules you might have defined for each affected transfer. You can redefine the schedules in a later step.

2. Define the following environment variable or one of its variant forms:

   ```
   DDAL$STORE_DEF_SEPARATE Yes
   ```

   See Section B.18 for more information on this environment variable.

3. Issue a START TRANSFER NOW WAIT statement for each transfer involved. The use of NOW WAIT ensures that each transfer has completed before you proceed to step 4. Once a transfer definition is stored in the source database, the transfer immediately finishes.

   Of course, you could issue a START TRANSFER NOW NO WAIT statement and then manually check for completion of each transfer by issuing a SHOW TRANSFER (STATUS) statement. However, you cannot issue a START TRANSFER NOW NO WAIT statement on all but the last transfer and then issue a START TRANSFER NOW WAIT statement on the last, assuming that the last transfer must be the last to complete execution.

When an initial transfer is executed, the copy process first checks to see if the transfer definition exists in the source database and if the definition is the same as is stored in the transfer database. This check is done without exclusive access. If the definition has not yet been stored, the copy process stores it. Storage of the definition requires exclusive access. If after storing the definition the copy process detects that the DDAL$STORE_DEF_SEPARATE environment variable is enabled, the copy process exits without performing the remainder of the transfer. This means that if you enable DDAL$STORE_DEF_SEPARATE and then execute several initial replication transfers on the same source database, they all execute one after the other, not concurrently. However, the storing of a transfer definition typically takes less than a minute to perform, so elapsed time is relatively short.

4. Now that the definitions for all transfers have been stored in the source database, issue the STOP TRANSFER and then the REINITIALIZE TRANSFER statements for each transfer. This forces the next execution of each transfer to be an initial transfer rather than a replication update.

5. Start each transfer either by using the START TRANSFER NOW NO WAIT statement or by defining a transfer schedule. The transfers can now execute in parallel. If you examine the resulting copy process transfer log files, you will see messages about storing the transfer definition in the source database and then messages indicating that the definitions have already been stored. In such cases, exclusive access to the source database is not required.

Note that should you ever need to reinitialize transfers but *not* change the transfer definitions, you can re-execute the transfers starting at step 4 (although restarting at step 1 is also acceptable).

With two transfers started concurrently and each transferring 5,000 rows of data, use of this option can make performance (elapsed time) worse by about 2 percent. With two concurrent transfers copying 50,000 rows of data, the performance gain might be around 25 percent. With four concurrent transfers each copying 100,000 data rows, the performance improvement might be close to 40 percent. These figures might vary given your specific situation. They are listed to give you a general idea of what you might expect to see using this option.

6.17.3 Protecting Against Concurrent Writers During a Transfer

A copy process does not read the RDB$CHANGES table while there are active applications writing in RDB$CHANGES. Instead, the copy process does the following:

1. Attaches to the source database twice so it can start two transactions
2. Starts a transaction for protected read on the RDB$CHANGES table
3. Starts a second transaction, this one a snapshot for read-only
4. Rolls back the first transaction and detaches once

Starting a protected read transaction on RDB$CHANGES forces the copy process to wait until there are no writers in RDB$CHANGES. It also forces any writers, who appear while the copy process is waiting, to stall. Once the protected read transaction is allowed to continue, the copy process can safely start a snapshot transaction and avoid seeing transactions committed by active writers. At that point, the protected read transaction is rolled back permitting other stalled transactions to continue.

This copy process behavior can have an effect on concurrency and perceived throughput in high-contention environments. You should keep the duration of update transactions to a minimum so delays due to stalled transactions are minimized.

If you can guarantee one of the two following cases and you need the copy process to not stall for protected read, use the environment variable DDAL$CP_READ_CHANGES (described in Section B.8).

- Ensure that update transactions on the source database are committed sequentially.
- Ensure the copy process runs when there are no applications writing to the source database.

6.17.4 Adding Indexes to the RDB$CHANGES Table

Replication Option includes an option for adding indexes to the RDB$CHANGES table in a source database for replication transfers. Certain classes of applications can realize overall performance benefits from this option. You can enable and disable this option by defining the DDAL$CHANGES_INDEX environment variable. See Section B.3 for information on this variable.

6.17.4.1 Performance Considerations

Replication transfer performance considerations can be examined by using an example. Consider two identical tables without indexes, each having the same number of stored rows. One table has 100 database pages allocated to it and the other has 300 pages. Within a snapshot (read-only) transaction, a sequential read of each table is to be done. Optimally, execution time should be nearly the same in each case; that is, a function of the number of rows retrieved. However, execution time increases as more database pages are allocated to the table, even when the number of stored rows remains constant.

This behavior can occur with the RDB$CHANGES table. This table keeps copies of rows to be transferred to target replication databases, and is read by Replication Option's copy processes. Sometimes transfers get behind and execution is delayed. For example, the network link to the remote target site might be disabled due to hardware difficulties. When this happens, more rows than normal get stored in RDB$CHANGES. Eventually, the transfers execute successfully, the backlog of rows is reduced, and the number of rows on average in that table returns to a lower value.

By contrast, the time it takes to execute a transfer that increased while RDB$CHANGES was backlogged, does not return to normal. This behavior is related to the way Rdb performs a sequential scan of a table that is not indexed in a uniform storage area. The default is that RDB$CHANGES is not indexed.

The copy processes, which read RDB$CHANGES, operate on rows grouped by transaction and sorted in commit order. Indexes on RDB$CHANGES are needed, based on the transaction serialization number (the RDB$TRANSACTION_TSER column), the transaction sequence number (the RDB$TRANSACTION_TID column), and the sequence number for rows within the same transaction (the RDB$TRANSACTION_SEQUENCE column). The TSER and TID columns are assigned values based on the following behavior. When applications update the source database, index entries are always placed at the far right side of the index B-tree structure. If there is a single update application, there is no problem, although delays can occur when the tree needs to be rebalanced from time to time. If multiple update applications and their updates are performed concurrently, the applications can experience lock conflicts on the index nodes. The probable effect would be to force the update applications to execute serially rather than in parallel.

Indexing benefits database readers and penalizes writers. If you have one primary update application and a secondary update application that runs infrequently, and you also execute numerous concurrent replication transfers from the same database, indexing RDB$CHANGES may give you better overall performance. While you can experience some lock conflicts, and while writing

index entries is added overhead for the updated applications, the gains in overall performance outweigh this disadvantage.

6.17.4.2 Optional Indexing

By default, RDB$CHANGES is not indexed. You can override the default behavior by defining environment variables.

- Define the DDAL$CHANGES_INDEX_transfername environment variable if you want to index the RDB$CHANGES table in a source database accessed by individual transfers.
- Define the DDAL$CHANGES_INDEX environment variable if you want to index the RDB$CHANGES tables in source databases accessed by all the transfers for which the environment variable definitions are visible.

6.17.4.2.1 Using the DDAL$CHANGES_INDEX_transfername Environment Variable

The DDAL$CHANGES_INDEX_transfername environment variable defines whether or not RDB$CHANGES is to be indexed for a given transfer. For example, if a transfer is named ABC, then DDAL$CHANGES_INDEX_ABC is the associated environment variable. The transfer must be a replication transfer. If it is not, defining DDAL$CHANGES_INDEX_transfername has no effect. Because a named replication transfer has one source database, identifying the transfer in this way indirectly identifies the source database in which the RDB$CHANGES table to be indexed exists.

Translation of the DDAL$CHANGES_INDEX_transfername environment variable yields an equivalence name. The environment variable definition in the following example indicates that indexing is desired:

```
$ DEFINE DDAL$CHANGES_INDEX_ABC  USE
```

If the equivalence name NONE is substituted for the equivalence name USE, it indicates that indexing is not wanted. If indexes exist and are to be used, a copy process must query the database in a different way than if indexes are not present. The equivalence string, which is the translation of this environment variable, tells the copy process which type of query to perform, indexed or not indexed.

The use of environment variables means that the copy process does not have to query the system tables to see if the indexes exist prior to each update run.

If several replication transfers use the same source database, you must define an environment variable for each transfer. Two replication transfers, A and B, can use the same source database but they can differ in their use of indexing. Transfer A can specify indexing; B can specify no indexing, either through the equivalence name NONE or by accepting the default. In this case, the indexes

are created, transfer A uses them, and transfer B does not. Transfer B does not because a copy process does not query the system tables to see if the indexes exist. However, this situation should not arise. If you decide indexing will improve performance, you should use it for all replication transfers on a source database.

The name lookup is done using the standard search list logical name, LNM$FILE_DEV, which normally defines the order of search to be the PROCESS, JOB, GROUP, and SYSTEM environment variable tables on OpenVMS.

6.17.4.2.2 Using the DDAL$CHANGES_INDEX Environment Variable

You can also use the generic environment variable DDAL$CHANGES_INDEX. When the copy process executing a transfer fails to find a definition for the environment variable DDAL$CHANGES_INDEX_transfername, it checks for the generic logical name, DDAL$CHANGES_INDEX. The advantage is that you only have to define a single environment variable that applies to all transfers on a given processor. However, this can also be a disadvantage. If there are multiple replicated source databases on a system managed by independent groups, defining a systemwide environment variable might not make sense.

Environment variables for a single process must be defined in a LOGIN.COM file on OpenVMS. The copy process is run as a detached process using the OpenVMS account of the person who created the transfer definition. It is that person's LOGIN.COM file that must be modified. Systemwide environment variables can be defined independent of the transfer definer's account.

6.17.4.3 Creating the Indexes

The copy process creates the indexes. This section describes how the copy process creates indexes during replication initialization, replication reinitialization, and replication update.

6.17.4.3.1 Index Creation for Replication Initialization and Reinitialization

During replication initialization or reinitialization, the copy process creates system tables, such as RDB$CHANGES, if they do not exist. It also creates indexes on RDB$CHANGES, if you selected that option, based on the environment variables described in Section 6.17.4.2. These indexes appear in the RDB$SYSTEM storage area, as that is the location of the RDB$CHANGES table. Refer to Section 6.17.4.2 for instructions on defining environment variables for indexing RDB$CHANGES.

6.17.4.3.2 Index Creation for Replication Updates

If you have replication transfers already established, you can have the copy process add the indexes during replication update. Normally, the copy process does not check for the existence of these indexes when performing an update. Copy process queries assume that you have specified whether or not to use indexes through environment variable definition and that these environment variables have been resolved during initialization.

If there are several replication transfers from the same source database, select one transfer to be the one to add the indexes.

Use the following statement to specify index creation during a replication update:

```
$ DEFINE DDAL$CHANGES_INDEX_transfername CREATE_ON_UPDATE
```

If there is only one replication transfer for the source database, use the following statement to specify index creation during a replication update:

```
$ DEFINE DDAL$CHANGES_INDEX CREATE_ON_UPDATE
```

Note

Creation of indexes requires that the copy process have exclusive access to the database. Therefore, if you plan to add indexes during a copy process update run, stop all other transfers and applications that use that database and let just that one transfer execute. If you forget to do so, the transfer can fail with a lock conflict error.

For example, if the transfer is named ABC, define the following logical name :

```
$ DEFINE DDAL$CHANGES_INDEX_ABC CREATE_ON_UPDATE
```

When the copy process executes for the ABC transfer, it translates this environment variable and notes that it should add the indexes. This is done prior to performing the actual update transfer.

Upon completion of the transfer, change the definition of the OpenVMS logical name as follows:

```
$ DEFINE DDAL$CHANGES_INDEX_ABC USE
```

Also, define similar environment variables for other replication transfers on that database to USE the indexes.

6.17.4.4 Dropping the Indexes

If you have created indexes that adversely affect performance, you can instruct the copy process to remove the indexes.

The copy process drops indexes whenever it translates the index environment variable and finds a definition similar to the following on OpenVMS:

```
$ DEFINE DDAL$CHANGES_INDEX_transfername DROP
$ DEFINE DDAL$CHANGES_INDEX DROP
```

Note

Dropping indexes, like adding indexes, requires that the copy process have exclusive access to the database.

6.17.4.5 Creating Indexes for Replication Transfers

You have the option of automatically adding two sorted indexes for the RDB$CHANGES table in a replicated source database. See Table 6–3.

Table 6–3 Indexes for the RDB$CHANGES Table

Index	Based on column
RDBVMS$CHANGES_INDEX1	RDB$TRANSACTION_TSER RDB$TRANSACTION_TID RDB$TRANSACTION_SEQUENCE
RDBVMS$CHANGES_INDEX2	RDB$TRANSACTION_TID RDB$TRANSACTION_SEQUENCE

You can enable or disable indexing by defining one of the following environment variables:

DDAL$CHANGES_INDEX_transfername
DDAL$CHANGES_INDEX

Note that the environment variables are prefixed with the DDAL facility code whereas the indexes themselves are prefixed with the RDBVMS facility code.

Table 6–4 shows the equivalence names that you can use for these environment variables. The equivalence names are not case sensitive.

Table 6–4 Equivalence Names for Index Variables

String	Meaning
CREATE_ON_UPDATE	Create indexes during the replication update cycle, if necessary.
DROP	Drop indexes during replication initialization, reinitialization, update, or deletion.
NONE	No indexing desired. Queries will assume indexes do not exist.
USE	Create indexes during replication initialization or reinitialization, if necessary. During replication update, assume RDB$CHANGES is indexed.

6.17.5 Avoiding Concurrent Transfers into PC Databases

If the target database system does not have automatic deadlock detection and resolution, concurrent transfers into the same PC target database could result in the transfers stopping, but not terminating. In effect, the transfers would be frozen. Because of this, it is recommended that transfers into PC databases be performed at different times from each other and at different times from the execution of other target database applications that write to the database.

6.18 Maximum Size of Source Database Table

Replication Option cannot extract or replicate a table whose row width is the maximum size allowed by the database systems. That maximum size may differ among the various database products. The maximum row size is approximately 65,000 bytes. This is because Replication Option requires a certain number of bytes for overhead, which varies depending on the number of columns being transferred.

Replication Option adds a number of columns to the target database tables that are not in the source database. These include three columns used to manage the target query: an operation code field (1 byte), a status code field (1 byte), and a database key field (8 bytes). For each column in the table, Replication Option has a 2-byte column in the query for setting the null flag in the target row. For example, if the source row has 100 columns transferred to the target, the query has 210 bytes of overhead for Replication Option.

7

Using Command Procedures with Transfers

When you use Replication Option to transfer your data, you might want to perform other tasks in conjunction with the transfer. For example, you might need to bring up the network at the start of the transfer and take it down immediately afterwards. You might want to create tables and views on the target that do not exist on the source. You might want to perform such tasks as defining indexes, integrating data definitions into the data dictionary, or coordinating other events with the transfer.

To accomplish these or similar tasks, you can specify Digital Command Language (DCL) command procedures to run automatically before and after the transfer. Using the PROLOGUE and EPILOGUE options of the CREATE TRANSFER statement, you can run procedures automatically every time an extraction, extraction rollup, or replication transfer executes, whether or not the transfer is successful.

This chapter describes the syntax of these user-written command procedures. It also suggests possible uses and provides examples for both prologue and epilogue command procedures.

7.1 Defining and Executing Command Procedures

The epilogue and prologue command procedures are OpenVMS DCL command procedures. As part of the CREATE TRANSFER statement, you can specify either a prologue or an epilogue file or both to run in conjunction with the transfer. Including a prologue or an epilogue file specification is optional. If you do not include the prologue file or the epilogue file, Replication Option assumes the defaults, NO PROLOGUE and NO EPILOGUE.

The prologue and epilogue file names must be standard OpenVMS file specifications. You can use full file specifications including node names and version numbers. If you do not specify a directory name, Replication Option assumes the default directory, SYS$LOGIN. Example 7–1 shows a CREATE TRANSFER statement that includes both prologue and epilogue file specifications.

Example 7–1 Prologue and Epilogue Files Specified in CREATE TRANSFER

```
SQL> ATTACH 'FILENAME DISK2:[MYDIR]PERSONNEL';
SQL> CREATE TRANSFER TRANSFER_ALL TYPE IS EXTRACTION
cont>  MOVE TABLES *
cont>  TO NEW FILENAME PERS_SAMPLE
cont>  PROLOGUE FILE IS MYNODE::MYDEV:[MYDIR]RETRY_NO.COM
cont>  EPILOGUE FILE IS DEFINE_INDEXES.COM;1
cont>  LOG IS PERS_SAMPLE.LOG
cont>  ;
```

By default, when you create the transfer, SQL translates any logical names you use for the prologue and epilogue file names and sends them to the transfer monitor. The transfer monitor stores the file names in the transfer database as part of the transfer definition. Should you desire it, you can override this default behavior by using the SET LOGICAL_NAME TRANSLATION FOR TRANSFER OFF statement. See Section 9.11 for more information on the SET LOGICAL_NAME TRANSLATION FOR TRANSFER statement.

Note

You can use any DCL command you want when you write a prologue or epilogue command procedure. However, you should not include a LOGOUT command in either a prologue or epilogue file.

Each time an extraction, extraction rollup, initial replication, replication update, or replication reinitialization transfer starts, the prologue and epilogue command procedures execute.

Also, when the copy process is run to delete a replication transfer definition from a source database, the prologue and epilogue command procedures execute. You can test your epilogue and prologue command procedures to see if the copy process is deleting a replication transfer. If the command procedures do not benefit you in this situation, you can have them exit using the following DCL command:

```
$ IF DDAL$CP_ACTION .EQS. "D" THEN EXIT
```

When executing a prologue or epilogue command procedure, Replication Option uses the following sequence of events:

1. When the transfer executes, the transfer monitor checks for prologue and epilogue file names and signals the copy process[1].

[1] Copy process refers to the detached OpenVMS process in which the copy process image runs, not the copy process program.

2. The copy process[1] executes the files before and after execution of the transfer, respectively.

The command procedures and the DDAL$COPY_PROCESS image communicate with one another through the DDAL$CP_CONTINUE process logical name. On the basis of the transfer status you assign to DDAL$CP_CONTINUE in the prologue command procedure you write, DDAL$CP_CONTINUE signals the copy process whether to continue or to abort the transfer. For information on how DDAL$CP_CONTINUE determines and affects transfer status, refer to Section 7.2.

7.1.1 Executing Command Procedures

While the discussion in this section is oriented to the OpenVMS system, it is possible for OpenVMS DCL prologue and epilogue procedures to trigger actions on other systems, such as an IBM operating system using RJE. After you have written the prologue and epilogue procedures, you should decide where you want to locate and execute them. You can locate your command procedure on the node where you define the transfer and execute it on that node. Or you can locate the procedure on a remote node and execute it on either the local node or the remote node, as the following table shows:

Command Procedure File Location	File Execution Location
Local Node	Local Node
Remote Node	Local Node
Remote Node	Remote Node

For example, if the source node is your local node, you can execute on the source node command procedures that exist on either the source or the target node. Or, you can execute on the target node a command procedure that is located on the target node.

7.1.1.1 Executing Remotely or Locally Stored Command Procedures on a Local Node

Whether the command procedure resides on a local node or on a remote node, the procedure itself can be executed on the local node, the one on which you define the transfer. Running the procedure causes changes at the local node or the remote node. The following examples show some possible procedures:

- A prologue file that establishes the network link by dialing an outside telephone line—stored on the local node and executed on the local node
- An epilogue file that creates indexes in the target database at the remote site—stored either on the local node or on the remote node and executed on the local node

- An epilogue file that grants access rights for users of the target database at the remote site—stored at the remote site (where users can easily keep it up-to-date) and executed on the local node

When procedures are executed on the local node to cause changes on the remote node, what is actually executed locally is the DCL command procedure in a local OpenVMS process. For example, this procedure might call on SQL, which can attach to the target database at the remote site. SQL can then perform operations on that database using the remote access capabilities of Rdb or the transparent gateway products.

Because DCL command procedures can themselves execute other command procedures, you can combine the execution of both local and remote command files.

7.1.1.2 Executing Remotely Stored Command Procedures on a Remote Node

When the command procedure is stored on a remote node, you can also execute it on that same remote node. You might want to store and execute a command procedure remotely when the remote site has software capabilities that do not exist at the local site.

7.1.2 Completing the Transfer with DDAL$EPILOGUE.COM

If your transfer includes an epilogue command procedure, Replication Option does not execute that procedure directly. Instead, Replication Option executes its own DDAL$EPILOGUE.COM file and, in turn, the DDAL$EPILOGUE.COM file executes your epilogue file.

The DDAL$EPILOGUE.COM procedure is used after DDAL$COPY_PROCESS, the copy process image, finishes executing. DDAL$EPILOGUE.COM executes your epilogue file and then exits with the status that corresponds to one of the following equivalence names it finds for the logical name DDAL$CP_CONTINUE:

- SUCCESS
- FAIL
- RETRY
- PRO_FAIL
- PRO_RETRY
- EPI_FAIL
- EPI_RETRY

Section 7.2 shows how to use these equivalence names to determine the success status of your prologue file, transfer, or epilogue file.

Before running any of the procedures associated with a transfer, the transfer monitor defines the DDAL$CP_CONTINUE logical name to be SUCCESS. If there is a prologue, that procedure can leave DDAL$CP_CONTINUE as is or it can redefine it to be PRO_FAIL or PRO_RETRY. After the prologue, the copy process image is run. The copy process will perform the transfer or not, depending on what it finds in DDAL$CP_CONTINUE. Upon exiting, the copy process sets DDAL$CP_CONTINUE to correspond with one of the five status conditions shown in Table 7–1.

Table 7–1 Status Corresponding to DDAL$CP_CONTINUE

Status	Meaning
SUCCESS	The copy process image performed the transfer successfully and either no prologue existed or one of the following actions occurred: the prologue succeeded; the prologue did not alter DDAL$CP_CONTINUE (that is, the prologue did no error handling of its own).
FAIL	The copy process image attempted the transfer upon seeing SUCCESS in DDAL$CP_CONTINUE, but the copy process encountered a severe error or one that the transfer monitor does not recognize and understand. Such an error means that Replication Option cannot retry the transfer automatically.
RETRY	The copy process image attempted the transfer upon seeing SUCCESS in DDAL$CP_CONTINUE, but the transfer failed. The transfer monitor recognized the cause of the failure as recoverable.
PRO_FAIL	The prologue failed and altered DDAL$CP_CONTINUE to reflect that failure. As a result, the copy process image intentionally exited without performing the transfer. The prologue procedure decided that the transfer cannot be automatically retried.
PRO_RETRY	The prologue failed and altered DDAL$CP_CONTINUE to reflect that failure. As a result, the copy process image intentionally exited without performing the transfer. However, the transfer can be retried automatically.

If the transfer includes an epilogue that does not change DDAL$CP_CONTINUE, the final transfer status is determined by one of the conditions listed in Table 7–1. If one of these five conditions is the final transfer status, the epilogue procedure either executed without error or did no error checking. Two additional values for DDAL$CP_CONTINUE, shown in Table 7–2, reflect epilogue failures.

Table 7–2 Epilogue Failure Status Conditions for DDAL$CP_CONTINUE

Status	Meaning
EPI_FAIL	The epilogue procedure was run, and either the epilogue procedure or DDAL$EPILOGUE.COM (if the epilogue procedure did not work as intended) detected a fatal error. EPI_FAIL can also mean that an unknown value appeared in DDAL$CP_CONTINUE upon completion of the epilogue.
EPI_RETRY	The epilogue failed. The epilogue procedure was run and either the epilogue procedure or DDAL$EPILOGUE.COM (if the epilogue procedure did not work as intended) detected an error. However, the transfer can be automatically retried.

If a transfer includes an epilogue procedure, that procedure will run whether the prologue and copy process image succeeded or failed. If what your epilogue procedure does is affected by the success or failure of the prologue or copy process image, you should have the epilogue procedure check the code in DDAL$CP_CONTINUE first. Then the epilogue can base its actions on the status it finds.

You should take care when you assign values to DDAL$CP_CONTINUE within your epilogue procedure. If the epilogue fails, you should not automatically define DDAL$CP_CONTINUE to be EPI_RETRY even though the epilogue procedure finds a nonfatal error. Instead, you should have your epilogue procedure first check the value stored in DDAL$CP_CONTINUE. If the epilogue procedure finds either FAIL or PRO_FAIL, which are fatal errors, you should not redefine the value to EPI_RETRY.

If the final status in DDAL$CP_CONTINUE is PRO_FAIL, FAIL, or EPI_FAIL, the transfer's status will change to *suspended* on completion of the transfer.

7.1.3 Executing Multiple Command Procedures

Replication Option lets you specify only one prologue and one epilogue command file in a transfer definition. However, in certain situations you might need to execute more than one command procedure as the prologue or epilogue:

- Common procedures

  There may be procedures that everyone at your site can use, such as a procedure that established a network link through the use of a dialup telephone connection. Besides the common procedure, there could also be a second procedure that is specific to your particular transfer.

- Locally and remotely managed procedures

You can keep part of your command procedure at the local site where the transfer executes and another part at the remote target site. For example, you might maintain a local procedure to define indexes on the target database. Someone at the target site might maintain a procedure that defines access control entries for the target database users.

You can achieve the effect of executing multiple command procedures by creating a **master** command procedure that includes other procedures. The master procedure is the one you name as the prologue or epilogue file in your transfer definition. It invokes, one after the other, several command procedures, each of which accomplishes a subtask. You need to maintain each command procedure in a separate file. (The command procedures can also be included in their entirety. You should select the method that is most effective for your own purposes.)

For example, you might create MA_PERSONNEL_EPILOGUE.COM, an epilogue file used in the transfer of Massachusetts personnel records. This file contains these lines:

```
$ @DISK1:[DEFINER]DEFINE_MA_PERS_INDEX.COM
$ @REMNOD::DISK2:[PERS]GRANT_MA_PERS_RIGHTS.COM
```

As the definer, you would be responsible for writing and maintaining the DEFINE_MA_PERS_INDEX.COM file, which defines indexes on the target database. Someone at the REMNOD remote node would write and maintain the GRANT_MA_PERS_RIGHTS.COM file, granting access rights to the target tables.

This example of a simple epilogue file assumes that the transfer that precedes this epilogue file and the indexing procedure will succeed. More typically, your epilogue file will check the status of the preceding operation to decide whether or not to continue. Other sections of this chapter describe how to check the status of the preceding operations.

For information on including more than one DCL procedure, refer to the OpenVMS documentation. Master procedures can be substituted for either a prologue or epilogue procedure or both.

7.1.4 Passing Parameters to Command Procedures

Replication Option does not allow you to specify parameters directly in prologue or epilogue command procedures. However, you can name the parameters in a file that you include in a shell. By executing the shell as a prologue or epilogue, you can pass the parameters in your command procedure.

For example, if you want your epilogue procedure to signal the operator's terminal that a transfer has completed, your file, SEND_OPCOM.COM, might include the transfer name, TRANSFER1, as a parameter. You can include the file in a shell procedure, TRANSFER1_EPILOGUE.COM.

The shell includes the following line:

```
$ @SEND_OPCOM TRANSFER1
```

Replication Option executes the shell procedure as the epilogue and allows you to include the transfer name.

7.1.5 Using DDAL$TRANSFER_NAME to Identify a Transfer

DDAL$TRANSFER_NAME is a process logical name defined by the transfer monitor. The transfer monitor defines this logical name before executing a transfer's prologue procedure.

You might use the DDAL$TRANSFER_NAME logical name in a prologue or epilogue command procedure to determine which transfer is currently being executed and to do conditionalized processing based on the transfer name.

For example, the following code fragment shows how to send mail to a user named SMITH to inform him that his transfer has failed:

```
$ tname = f$trnlnm("DDAL$TRANSFER_NAME")

$ MAIL -
/SUBJ = "Prologue file failed; transfer ''tname' is retrying." -
 INFO.TXT -
 SMITH
```

7.1.6 Executing a Command Procedure on a Remote Node

Replication Option allows you to define a command procedure on the local node (that is, the node where the transfer is defined) or on a remote node. However, whether the procedure you invoke resides on the local or remote node, Replication Option executes it only on the local node. If you want to execute at the remote site, a procedure that is located at the remote site, you can use the local command procedure to submit a remote batch job, to perform a remote task, or to execute a remote shell procedure.

To submit a remote command procedure on a remote OpenVMS system as a batch job, use the SUBMIT/REMOTE command. To execute a command on any remote system, you can use the RSH command.

If you are transferring to an OpenVMS system using DECnet, you can execute a remote task. For example, the command procedure you want to execute, INDEX.COM, might be in REMNOD::SYS$LOGIN of the appropriate proxy account. Your epilogue file on the local node, DO_INDEX.COM, executes the following job at the remote site:

```
$ TYPE REMNOD::"TASK=INDEX"
```

By using a remote task specification, you can execute a synchronous operation on the remote node. However, if you do use a remote task specification, anyone with access to the network can run the TYPE statement with the TASK= designation.

For more information on performing remote tasks, refer to the OpenVMS documentation on networking. Also, refer to Section 7.3.5 for information about using a remote command procedure to integrate data definitions from a target database into the data dictionary.

7.1.7 Finding Command Procedure Information in the Copy Process Log File

The copy process log file includes information that results from running prologue and epilogue command files. Example 7–2, extracted from such a log file, shows that the copy process defines the symbol names DDAL$CP_ACTION and DDAL$CP_RETRY_ITERATION before the prologue file executes.

Example 7–2 Prologue Information Within a Copy Process Log File

```
30-MAY-1995 01:34:02.42  MBA7967          User: TRANSDEF
Pid: 2080112D  Proc. name: DDAL_COPY_01   UIC:[NODE1,TRANSDEF]
Priority:   3  Default file spec:  DISK1:[TRANSDEF]
$ DEFINE DDAL$CP_CONTINUE "SUCCESS"
$ DEFINE/TRANS=TERMINAL DDAL$TRANSFER_NAME NH_EMPLOYEES
$ DDAL$CP_ACTION :== E
$ DDAL$CP_RETRY_ITERATION == 00
$ @MY$DISK:[MYDIR]MY_TRANSFER_PROLOGUE.COM
$    SHOW SYMBOL DDAL$CP_RETRY_ITERATION
  DDAL$CP_RETRY_ITERATION == 0  Hex = 00000000  Octal = 00000000000
$    IF DDAL$CP_RETRY_ITERATION .EQ. 0 THEN GOTO FIRST_ATTEMPT
$ FIRST_ATTEMPT:
$    DEFINE/NOLOG DDAL$CP_CONTINUE "SUCCESS"
$    IF DDAL$CP_ACTION .EQS. "E" THEN GOTO EXT_TRANSFER
$ EXT_TRANSFER:
```

(continued on next page)

Example 7–2 (Cont.) Prologue Information Within a Copy Process Log File

```
$!
$!  This portion of the prologue would contain logic for handling extraction
$!  transfers.
$!
   .
   .
   .
$    DEFINE/NOLOG DDAL$EPILOGUE DISK1:[DIR1]DDAL$EPILOGUE
$    WRITE SYS$OUTPUT "End of Prologue Commands"
End of Prologue Commands
$
$ EXIT
$ RUN DDAL$COPY_PROCESS
```

The letter following the DDAL$CP_ACTION symbol name indicates which kind of transfer is being executed:

E	Extraction
I	Replication initialization or reinitialization
U	Replication update
X	Extraction rollup
D	Replication deletion

Because a replication with the NO DELETE attribute is actually a subtype of a replication transfer, there is not a separate letter stored in DDAL$CP_ACTION to represent this type of transfer.

The number following the DDAL$CP_RETRY_ITERATION shows the number of the current retry. The first time the transfer executes, the number is zero.

You can test DDAL$CP_ACTION in a command procedure to determine different actions for different types of transfers. For example, if DDAL$CP_ ACTION indicates that the transfer is a replication, you might want to define indexes for an initialization, but not for an update.

The retry count provided by DDAL$CP_RETRY_ITERATION can be used within prologue or epilogue command procedures to execute part of the procedure while skipping other parts. You might use DDAL$CP_RETRY_ITERATION in a prologue command procedure, for example, to purge a log file directory on iteration 0 but not on subsequent iterations. In this way, you would keep any log files for failed transfers.

7.2 Determining Transfer Status

When you define a transfer that has a prologue and an epilogue command procedure, the transfer execution has three distinct phases:

1. The prologue executes.
2. The transfer takes place.
3. The epilogue executes.

However, if the prologue command procedure fails, it would be to your advantage to detect the failure before the transfer and epilogue execute. Detecting the failure of the prologue before the other phases execute enables you to use the epilogue file to do cleanup operations if they are needed.

To obtain information about the status of your prologue or epilogue file, the status of your transfer during execution, and how to affect the transfer during execution, you can use equivalence names that the DDAL$CP_CONTINUE logical name provides.

If you want to examine the effect of the prologue or epilogue command procedures on the transfer execution, you can issue a SHOW TRANSFER (STATUS) statement. In addition, you can use another method to obtain information about the status of your epilogue file. Replication Option provides a command file, DDAL$EPILOGUE.COM, to run the epilogue command procedure. (Replication Option does not provide a separate file to execute the prologue command procedure.) When DDAL$EPILOGUE.COM runs, additional information is entered in the copy process log file. You can examine the copy process log file to obtain information about the status of your epilogue file. This section shows how to use each method to obtain this information.

7.2.1 Using DDAL$CP_CONTINUE to Affect Transfer Execution

You can use the DDAL$CP_CONTINUE logical name in a command procedure to determine the success status of your prologue command procedure, transfer, or epilogue command procedure. At the beginning of the transfer, the transfer monitor assigns to DDAL$CP_CONTINUE the equivalence name SUCCESS for the completion status.

If your prologue command procedure detects an error during its execution and needs to signal this error so that the transfer itself (running the copy process image) does not take place, you redefine DDAL$CP_CONTINUE in your prologue command procedure. You can indicate the severity of the error by specifying whether the transfer can be retried later or not. If your prologue file does not change DDAL$CP_CONTINUE, the copy process will attempt to run, regardless of the success or failure of the prologue.

7.2.1.1 Determining Prologue Success Status with Prologue Command Procedures

To pass prologue failure information to the copy process image, your prologue command procedure should assign PRO_RETRY or PRO_FAIL to DDAL$CP_CONTINUE. The following example of a partial prologue command procedure shows how to assign these equivalence names to DDAL$CP_CONTINUE. This part of the command procedure will execute immediately after you run an image, such as a procedure to bring up the network. The example also shows how to use your prologue command procedure to write a message on the operator's terminal that your prologue execution has failed. You can use the REQUEST command in a prologue command procedure to send a message to a terminal, such as a system operator's terminal. Remember that the message will be embedded in the other information that appears on the system operator's terminal screen. If you choose to use this method of signaling a terminal, you should format the message distinctly.

In the following example, the prologue command procedure invokes a command file, STARTNET.COM, to bring up the network at the beginning of the transfer:

```
$ @SYS$MANAGER:STARTNET.COM
$ SEVERITY == $SEVERITY
$ IF SEVERITY .EQ. 0 THEN GOTO RETRY
$ IF .NOT. SEVERITY THEN GOTO FAIL
$ GOTO SUCCESS
$ !
$ RETRY:
$ DEFINE DDAL$CP_CONTINUE "PRO_RETRY"
$ REQUEST "Prologue file failed; retrying transfer MA_PERSONNEL."
$ EXIT
$ !
$ FAIL:
$ DEFINE DDAL$CP_CONTINUE "PRO_FAIL"
$ REQUEST "Prologue file failed; transfer MA_PERSONNEL not retrying."
$ EXIT
$ SUCCESS:
   .
   .
   .
```

The prologue command procedure then tests the exit status of the STARTNET.COM procedure and saves the severity level of the return status. The severity levels have the following meanings:

- Odd numbers (1,3,5,7)—Informational
- Zero (0)—Warning
- Even numbers (2,4,6)—Failure

If the severity level is an odd number, the transfer continues. If the severity level is 0, the transfer fails but will retry later. If the severity level is an even number other than 0, the transfer fails.

If you want to send the status message to a specific user, you can invoke the Mail utility (MAIL) in your prologue command file and use the process logical name defined by the transfer monitor, DDAL$TRANSFER_NAME, to identify the transfer that is being run. In place of the REQUEST commands that appear in the preceding procedure, you can use MAIL commands. (Note that you must include a file specification in the command line to enable the SUBJ qualifier.) For example, to send a failure message to a user named Smith, use the following:

```
$ tname = f$trnlnm("DDAL$TRANSFER_NAME")

$ MAIL -
/SUBJ = "Prologue file failed; transfer ''tname' is retrying." -
 INFO.TXT -
 SMITH
```

For more information on the REQUEST command, see the OpenVMS DCL documentation; for information on using MAIL, see the OpenVMS documentation on the Mail utility.

The copy process determines the status of the transfer at the time of transfer execution by translating the logical name, DDAL$CP_CONTINUE, to its equivalence name. Table 7–3 describes each transfer status.

Table 7–3 Transfer Status Conditions

Status	Meaning
SUCCESS	The copy process lets the transfer continue.
PRO_FAIL	The copy process aborts the transfer, sends the information to the transfer monitor, issues an error message, and executes the epilogue file if one exists.
PRO_RETRY	The transfer monitor initiates the retry schedule.
FAIL, RETRY, EPI_FAIL, or EPI_RETRY	The copy process acts as if the equivalence name were PRO_FAIL.

7.2.1.2 Determining Transfer Success Status with Epilogue Command Procedures

Just as you can use a prologue command procedure to signal the status of the transfer at the time of prologue execution, you can also use an epilogue command procedure to find out the status of your transfer at the time of the epilogue execution.

For example, if you want to signal the operator in case of a severe transfer failure, you can write an epilogue command file that issues a message only if the equivalence name of DDAL$CP_CONTINUE is PRO_FAIL or FAIL. The following example shows an epilogue command procedure that writes a message on the operator's terminal in the event of a prologue failure or a transfer failure:

```
$ CP_CONT = F$TRNLNM("DDAL$CP_CONTINUE")
$ IF CP_CONT .EQS. "PRO_FAIL" THEN GOTO PRO_FAIL
$ IF CP_CONT .EQS. "FAIL" THEN GOTO TR_FAIL
$ EXIT

$ PRO_FAIL:
$ REQUEST "Prologue file for MA_EMPLOYEES failed with severe failure status."
$ EXIT

$ TR_FAIL:
$ REQUEST "Transfer MA_EMPLOYEES failed with a severe failure status."
$ EXIT
```

If you choose to use this method to signal the operator's terminal, you should format your command procedure in such a way that the message will stand out on the screen.

In the following example, the epilogue file invokes MAIL and uses the process logical name DDAL$TRANSFER_NAME to notify a specific user of the transfer status:

```
$ tname = f$trnlnm("DDAL$TRANSFER_NAME")
$ CP_CONT = F$TRNLNM("DDAL$CP_CONTINUE")
$ IF CP_CONT .EQS. "SUCCESS" THEN GOTO SUCCESS

$ MAIL -
/SUBJ = "Your transfer, ''tname', has failed." -
 INFO.TXT -
 JOHNSON

$ EXIT
$ SUCCESS:

$ MAIL -
/SUBJ = "Your transfer, ''tname', has succeeded." -
 INFO.TXT -
 JOHNSON
```

7.2.1.3 Determining Epilogue Success Status with Epilogue Command Procedures

If you design your epilogue command procedure to perform an operation that is essential to the successful completion of your transfer, you might want your epilogue command file to indicate whether or not the epilogue file has failed:

- For epilogue failures

  Assign EPI_FAIL to DDAL$CP_CONTINUE and use the equivalence name to send a message that specifies the epilogue failure and to enter the failure status in the transfer database for the SHOW TRANSFER (STATUS) statement.

- For epilogue recoverable failures

  Assign EPI_RETRY to DDAL$CP_CONTINUE if you want your epilogue command file to indicate that a recoverable failure has occurred and that the transfer can be retried. (Whether or not a retry attempt is subsequently made depends on if you specify retries in the transfer schedule and if the retry count has run out.)

To assign EPI_FAIL or EPI_RETRY to DDAL$CP_CONTINUE, use the procedure described in Section 7.2.1.1.

7.2.2 Using the SHOW TRANSFER STATUS Statement

After your transfer has completed executing, you can check whether your prologue or epilogue command procedure has failed or is retrying by issuing a SHOW TRANSFER STATUS statement. Table 7–4 shows the prologue and epilogue status messages and their meanings.

Table 7–4 Prologue and Epilogue Command Procedures Status Messages and Their Meanings

Message Displayed	Meaning
Prologue failed; copy aborted; not retrying	The prologue file has failed.
Prologue failed, copy aborted; retrying	The prologue failed but the transfer can be retried.
Epilogue failed; not retrying	The epilogue file has failed.
Epilogue failed; retrying	You assigned EPI_RETRY to DDAL$CP_CONTINUE in your epilogue command procedure.

If the transfer is currently executing when you issue the SHOW TRANSFER STATUS statement, the status shown will be either ACTIVE or RETRYING.

7.2.3 Using the Copy Process Log for Status Information

Replication Option provides the DDAL$EPILOGUE.COM command file, which runs the epilogue command procedure and enters information in the copy process log file. You can examine the copy process log file for information about the status of your epilogue file.

Replication Option does not use a command file to run your prologue file; however, the copy process log file does show your prologue file being executed. It lists the command line used to execute the prologue file and any command lines and data lines in the command procedure.

DDAL$EPILOGUE.COM output in the copy process log file indicates the status of your epilogue command procedure. Example 7–3 shows output from the DDAL$EPILOGUE.COM command procedure.

Example 7–3 DDAL$EPILOGUE.COM Output in Copy Process Log File

```
❶ $ @DISK1:[FITZGERALD]EPILOGUE.COM
  $ tname = f$trnlnm("DDAL$TRANSFER_NAME")
❷ $ FINAL_STATUS == F$TRNLNM ("DDAL$CP_CONTINUE")
  $ SHOW SYMBOL FINAL_STATUS
    FINAL_STATUS == "SUCCESS"
  $ IF FINAL_STATUS .EQS. "SUCCESS" THEN GOTO SUCCESS
  $ SUCCESS:
  $ MAIL -
   /SUBJ = "Your transfer, ''tname', was successful." -
   INFO.TXT -
   FITZGERALD
❸ $ EXIT
❹ $ RESULT :== SUCCESS
❺ $ IF RESULT .EQS. "SUCCESS" THEN $EXIT %X012683AB
```

The callouts in Example 7–3 are keyed to the descriptions that follow:

❶ DDAL$EPILOGUE.COM file

❷ Begin your epilogue file

❸ Exit your epilogue file

❹ The epilogue file ran successfully.

❺ The hexadecimal value for the success status code (%X012683AB) is passed to the transfer monitor through an OpenVMS mechanism called the termination mailbox.

For additional information about the DDAL$EPILOGUE.COM command procedure, refer to Section 7.1.2.

7.3 Typical Uses of Command Procedures

The command procedures that you invoke can include or refer to any user-written program that you normally execute in DCL. As shown in previous sections, you can specify prologue and epilogue command procedures to perform such tasks as logging errors to the operator's terminal or signaling the beginning and completion of a transfer. You could also define logical names for the target database in the prologue or purge files (on OpenVMS) after the transfer in an epilogue.

An epilogue can be used to have the successful completion of one transfer start the execution of another transfer. This section gives examples of typical command procedures that you could use to make your transfers more comprehensive.

7.3.1 Creating Target Indexes and Other Database Attributes

Replication Option cannot transfer indexes, constraints, triggers, multischema attributes, query outlines, stored procedures, or protection information. However, you can use epilogue command procedures to add such attributes for your target database. This section suggests ways to use an epilogue command procedure to define indexes, and shows how to define an index in a command procedure. You can use similar procedures to add other database attributes.

There are occasions when you might want to run a command procedure to create indexes or to make the creation of indexes possible. For example:

- When indexes are created on a replication target database

  If you have created indexes on your replication target database, performance of updates to the target database might suffer. If updating a replication target database that includes indexes causes more demands on the system resources than dropping and re-creating indexes, you can use a prologue to drop the indexes and an epilogue to redefine them at the end of all updates.

- When indexes cannot be created on the target

  Depending on the software you have installed you might not be allowed to create indexes.

- When it is necessary to know if the transfer completed properly

You can use a command procedure to ensure the transfer successfully completed before you create an index. Refer to Section 7.2 for information about determining transfer status.

The following example shows how to create an index in a command procedure called DEF_INDEX.COM, which executes on the source node a procedure that resides on the target node:

```
$ RUN SYS$SYSTEM:SQL$
SQL> ATTACH 'FILENAME TARGET::DISK1:[TARGET_DIR]PERSONNEL_MA';
SQL> CREATE UNIQUE INDEX EMP_EMPLOYEE_ID ON EMPLOYEES
cont> (EMPLOYEE_ID);
SQL> COMMIT;
SQL> EXIT;
```

You can use the command procedure with extraction, extraction rollup, or initial replication transfers.

7.3.2 Granting Target Database Access Rights

All target database systems provide a security mechanism that restricts access to database objects. The SQL GRANT and REVOKE statements are used to add and remove user privileges. The discussion in this section is specifically for Rdb, but has counterparts in the other target database systems, such as DB2, Oracle, and Sybase.

For securing its databases, Rdb provides a mechanism called access control lists (ACLs). ACLs grant users specific types of access rights to the database as a whole and to each table in the database. For example, one user may be allowed to insert as well as select rows, another may be allowed only to read, and others may be denied access to the database or portions of it.

You can set or change access rights by using either the GRANT statement in SQL. For complete information about managing ACLs, refer to the Rdb documentation.

Replication Option does not duplicate in the target database the ACLs that exist in the source database (or databases, in the case of an extraction rollup transfer). When Replication Option creates a target database, it sets ACLs for the database and for each table to the default established by Rdb. Two access control entries, one for the database creator/owner and another for all other database users (denoted by [*,*]), are assigned for the database and for each table. The database owner is granted all privileges without restriction. Privileges granted to all other users depend on the version of Rdb running at the target database site.

For Rdb Version 5.1 and higher, user accounts denoted by [*,*] are granted no access privileges to the target database or its tables, including the Replication Option-created tables within that database. The database owner/creator must explicitly grant users whatever privileges are required.

The Rdb default of granting access only to the database owner/creator has further implications for extraction rollup transfers and multiple transfers to existing databases.

- If three users, each with a proxy account on the target database site, create three transfers to the same target database, each user can only access, or grant access, to the tables created by the transfer defined by that user.
- If three users use a single proxy account for their transfers to a target database site, the transfers execute under the same OpenVMS account. Any transfer owner/creator can grant access to any or all Replication Option-created tables in the target database.

For either single or multiple proxy accounts, each transfer must use an epilogue procedure to set protection on the target database so that users and applications can read the RDB$VINTAGE table and if desired, the RDBVMS$TRANSFER_OWNER and DDAL$INDEXES tables for Rdb target databases, and the DDAL$TRANSFER_INFO and, if desired, DDAL$TRANSFER_OWNR and DDAL$INDEXES tables for all other target databases.

For multiple proxy accounts, each transfer's epilogue procedure must also ensure that the other transfers have full access to RDB$VINTAGE and RDBVMS$TRANSFER_OWNER for Rdb target databases, and the DDAL$TRANSFER_INFO and DDAL$TRANSFER_OWNR tables for all other target databases.

To modify the ACLs for the database and for individual tables, you need DBCTRL privilege in SQL. Any user application that needs to access a table requires the SELECT privilege, at minimum.

After you consider the following issues about access rights, you might want to add or modify entries in your target database ACLs. You can do so in an epilogue command procedure by using SQL to add or create new protection. The preceding section on defining indexes, constraints, triggers, additional views, and protection information includes an example of using an epilogue procedure to create an index. You can use a similar procedure to define ACLs.

The account of the database owner, the one granted DBCTRL access, is the account that creates the target database; the target database is created by a Replication Option copy process. The copy process runs under the user account that defined the transfer.

Which account or UIC is listed as the target database owner depends on which of the possible methods of access to the target system is used:

- For extraction and extraction rollup transfers that create the target database and in which the target database is at the transfer definition site, the target database owner UIC is that of the transfer definer.
- For replication transfers, extraction transfers, and extraction rollup transfers that create the target database, and in which the target database is at a site remote from the transfer definer, the UIC of the target database owner is that of the account used to run the Rdb remote server. Three possibilities exist for this account:
  - The proxy account established for you at the remote site. Replication Option recommends that you select this method.
  - A private account you name explicitly in the target database file specification.
  - The default RDB$REMOTE account at the target site. You should use this method only if it is not possible to use either of the other methods.

For further information about proxy access, refer to Chapter 4.

If you create a transfer on behalf of someone else, that user should be given the access privileges, most important DBCTRL privilege, of a database owner. To do so, find out that user's UIC at the target site, and add an entry for that UIC to the several ACLs in the target database.

For a replication database, which is supposed to be accessed for retrieval only, setting the privileges of [*,*] to SELECT+SHOW would be appropriate. Even for extraction or extraction rollup transfers, selecting SELECT+SHOW privileges for the "all others" group would protect your database. Later, the target database owner can grant additional privileges to users who need them.

For example, even after the transfer executes, you might find that you want to change indexes or views in replication databases, as well as extraction or extraction rollup databases. Perhaps you want some users besides the database owner to have the necessary access rights to implement such changes.

Note

In a replication database, the copy process periodically applies changes to the target database. Therefore, even though some user other than the original database owner manages the target database, do not restrict the access rights of the original database owner, on whose behalf the copy process runs.

7.3.3 Defining Transfer and Schedule Definition Protection Using Access Control Lists

Replication Option uses a mechanism similar to that used for target database protection for securing transfer definitions and schedule definitions: access control lists (ACLs).

The user who defines a particular transfer always has privileges to alter that transfer definition and any schedule that may be associated with it as well as privileges to start or stop that transfer. This user is considered the transfer definition owner.

The transfer definition owner can set or change access rights to allow other users access to transfer and schedule definitions by using the SQL GRANT transfer-privs statement or SQL REVOKE transfer-privs statement.

You might want to grant transfer privileges to other database users so that they can run a transfer when you are not available.

If you are not the transfer definition owner, you must have the DBCTRL privilege to grant privileges for the transfer. See Table 9–1 for a complete list of privileges needed to execute the various SQL transfer and schedule control statements.

See Section 9.7 for more information on ACLs.

7.3.4 Bringing Up the Network at Transfer Time

Maintaining a leased network with continuous access can be costly. To reduce the cost, you might want to use a dialup connection and bring up the network immediately before and take it down immediately after the transfer. You can use a prologue command file to bring up the network and an epilogue command procedure to take it down automatically. The following example shows how to bring up the network for a DECnet network connection.

To bring up a network, you need the following privileges: ACNT, DETACH, CMKRNL, LOG_IO, WORLD, NETMBX, TMPMBX, SYSNAM, OPER, SYSPRV, and ALTPRI. The following example shows a typical prologue file that automatically invokes STARTNET.COM to bring up the network:

```
$ @SYS$MANAGER:STARTNET.COM
$ SHOW NETWORK
$ EXIT
```

The command procedure also issues a SHOW NETWORK statement to verify that the network is running. The output of the statement appears in the log file.

Alternatively, you can use your prologue file to check if the network is up and display a message accordingly. The return success status for the SHOW NETWORK command is 10000001 in hexadecimal. If the network is unavailable, then another return status will result from the command. If another return status results, you can exit or invoke STARTNET.COM as shown in the preceding example.

The commands that you can enter in your prologue file to check on the condition of the network follow:

```
$ SHOW NETWORK
$ SAVE_STATUS == $STATUS
$ IF SAVE_STATUS .EQ. %X10000001 THEN EXIT
$ DEFINE DDAL$CP_CONTINUE "PRO_FAIL"
$ EXIT
```

If you prefer, you can substitute the error code for the NETWORK NOT AVAILABLE message for the status returned as a result of the SHOW NETWORK statement. In which case, you fail only if the status is this error code.

For information on managing networks, see the OpenVMS documentation.

7.3.5 Storing Data Definitions in the Data Dictionary at a Target Site

The information in this section is specific to target databases on OpenVMS systems. You might want to merge data definitions into the data dictionary from a target database located at a remote site. The Replication Option transfer definition does not provide explicit syntax for directly integrating data definitions from a remote target database into a data dictionary. On the other hand, the transfer definition does allow you to execute a remote command procedure from an epilogue command file named in the definition. This section describes two methods of executing a remote command procedure from an epilogue file to integrate data definitions into a data dictionary:

- Use the SUBMIT/REMOTE command in an epilogue file to execute a remote command procedure. The RSH command may be used instead, but its use is not illustrated in the sections that follow.
- Use the TYPE command in an epilogue file to execute a remote command procedure as a network object.

Before you can integrate data definitions into a data dictionary, the following components have to be in place:

- A transfer definition that specifies an epilogue file and that creates the remote target database

- An epilogue command file to execute a remote command procedure that integrates the remote database data definitions into the dictionary
- A remote command procedure to integrate the data definitions into the dictionary on the remote node

Example 7–4 shows a sample transfer definition that specifies an epilogue command file called EPILOGUE_SUBMIT.COM. The example could have used an epilogue file named EPILOGUE_TYPE.COM. The name chosen for the epilogue file can help identify the method for executing the remote command procedure. In addition, you can execute remote command procedures from extraction rollup and replication transfers, not just extraction transfers as Example 7–4 might imply.

Example 7–4 Epilogue File Named in a Transfer Definition

```
ATTACH 'FILENAME USER7:[ADMIN.DBS]PERSONNEL';
CREATE TRANSFER REM_DBS_TYPE TYPE IS EXTRACTION
    MOVE TABLES *
    TO NEW FILENAME NODE12::USER6:[COLLEGE.DBS.TARGET_DB]TARGET_DB
  EPILOGUE FILE IS USER7:[ADMIN.DBS]EPILOGUE_SUBMIT.COM
  LOG FILE IS USER7:[ADMIN.DBS]REM_DBS_SUBMIT_XFER.LOG;
DISCONNECT ALL;
```

After creating the transfer definition (as previously described) and the two command procedures (as described in the following sections), you can start the transfer. The transfer creates the remote target database and executes the epilogue command procedure. The remote command procedure initiated by the epilogue command file integrates the data definitions from the newly created target database into the data dictionary. For this application, the data dictionary is local to the target site, and the target database is remote in relation to the source database.

The following sections show examples of the epilogue and remote command procedures.

7.3.5.1 Using the SUBMIT/REMOTE Command to Run a Remote Command Procedure

This section describes how to create an epilogue command procedure and a remote command procedure that the epilogue file executes to integrate data definitions into the data dictionary. The epilogue file invokes SQL to alter the target database so that subsequent data definition changes to the target database require access to the database by dictionary path name. It then invokes the SUBMIT/REMOTE command that runs another command procedure at the remote site to integrate the data definitions into the dictionary.

Example 7–5 shows an epilogue file that alters the target database and executes a remote command file on the remote node.

Example 7–5 Epilogue Command Procedure Using SUBMIT

```
$!
$!     E P I L O G U E _ S U B M I T . C O M
$!
$! Test for successful creation of the target schema. If not
$! successful, exit.
$
$   CP_CONT = F$TRNLNM("DDAL$CP_CONTINUE")
$   IF CP_CONT .NES. "SUCCESS" THEN EXIT
$
$! Change the database at the target site to require the use of
$! the dictionary.
$
$   DEFINE/USER SYS$COMMAND SYS$INPUT
$   SQL
     ALTER DATABASE FILE 'NODE12::USER6:[COLLEGE.DBS.TARGET_DB]TARGET_DB'
     DICTIONARY IS REQUIRED;
    EXIT;
$
$! Integrate the target database definitions into the data
$! dictionary at the target site.
$
$   SUBMIT/REMOTE NODE12::DISK8:[DBS.REMOTE]REMOTE_INTEGRATE.COM
$   EXIT
```

When using the SUBMIT/REMOTE command in OpenVMS, the command procedure that you specify in the epilogue file has to reside at the remote site. In addition, you must include the node name in the file specification.

Note

When you use SUBMIT/REMOTE, you cannot tell when the remote batch job ends, which means you do not know when to check if it succeeded or not. Also, there is no simple OpenVMS mechanism to return and make available to Replication Option the remote batch job status.

After the transfer executes to create the target database on the remote node, the epilogue file submits a remote command procedure to the remote batch queue to perform the data dictionary integration. Example 7–6 shows a sample remote command procedure that you might write to integrate data definitions into the data dictionary. This procedure resides in a directory account local to the target site but remote from where the transfer is defined and executed.

Example 7–6 Remote Command Procedure Executed by SUBMIT

```
$!
$!      R E M O T E _ I N T E G R A T E . C O M
$!
$   IF F$MODE() .EQS. "NETWORK" THEN DEFINE/USER SYS$OUTPUT SYS$NET
$!
$   DEFINE/USER SYS$COMMAND SYS$INPUT
$   SQL
     DROP PATHNAME USER7:[DBS.PERS_DICT]TARGET_DB.PERSONNEL;
     INTEGRATE DATABASE FILE 'USER6:[DBS.TARGET_DB]TARGET_DB'
      CREATE PATHNAME USER7:[DBS.PERS_DICT]TARGET_DB.PERSONNEL;
     COMMIT;
    EXIT;
$   EXIT
```

SQL requires that you use a different form of the INTEGRATE statement if you are integrating the information into the dictionary for the first time. Instead of distinguishing the first from subsequent integrations, this command file simply deletes the definitions and then inserts them as if for the first time.

When you use the SUBMIT/REMOTE method to execute a remote command procedure, the remote batch job does not:

- Check for errors
- Return status information for the copy process log

After integrating data definitions from the target database, a user at the remote target site cannot make any metadata changes to the target database unless that user specifies a path name when invoking the target database.

7.3.5.2 Using the TYPE Command to Run a Remote Command Procedure

This section describes how to create an epilogue command procedure and a remote command procedure, which the epilogue file executes to integrate data definitions into the data dictionary. The epilogue procedure invokes SQL to alter the target schema so that subsequent data definition changes to the target database require access to the schema by dictionary path name. The file then invokes the TYPE command that executes another command procedure at the remote site to perform the integration of the data definitions into the data dictionary.

Example 7–7 shows a sample epilogue file that uses the TYPE command to execute a command procedure at a remote site.

Example 7–7 Epilogue Command Procedure Using TYPE

```
$!
$!     E P I L O G U E _ T A S K . C O M
$!
$! Test the successful creation of the target schema. If not
$! successful, exit.
$
$   CP_CONT = F$TRNLNM("DDAL$CP_CONTINUE")
$   IF CP_CONT .NES. "SUCCESS" THEN EXIT
$
$! Change the database at the target site to require use of the
$! dictionary.
$
$   DEFINE/USER SYS$COMMAND SYS$INPUT
$   SQL
     ALTER DATABASE FILE 'NODE12::USER6:[COLLEGE.DBS.TARGET_DB]TARGET_DB'
      DICTIONARY IS REQUIRED;
     EXIT;
$
$! Integrate the target database definitions into the data
$! dictionary at the target site.
$
$   TYPE NODE12::"TASK=INTEGRATE"
$   EXIT
```

When using the network task method, the command procedure to be submitted must reside at the remote site, typically in the top directory for the remote account (SYS$LOGIN). The task name can contain up to 12 characters. You can specify a file directory (within the 12 character limit), but you cannot specify a device name or a file type. The device and directory defaults to SYS$LOGIN of the remote account (proxy accounts are recommended). The file type by default is .COM. Refer to the OpenVMS documentation on networking for details about command procedures as network task objects.

Using the TYPE command has the following advantages over using the SUBMIT/REMOTE command in an epilogue command file:

- Completion of the remote procedure execution is guaranteed before the epilogue procedure continues.
- Output from the remote procedure can be returned (with SET VERIFY) to the local node and can appear in the copy process log.

Note

When you use TYPE, you can tell when the remote batch job ends, but the epilogue procedure has no way to test if the remote procedure succeeded or failed. Also, there is no simple OpenVMS mechanism to

return and make available to Replication Option the remote batch job status.

Example 7–8 shows a sample remote command procedure that you might write to integrate data definitions into the data dictionary using the DECnet task object. This procedure resides in a directory account local to the target site but remote from where the transfer is defined and executed.

Example 7–8 Remote Command Procedure Executed by TYPE

```
$!
$!     I N T E G R A T E . C O M
$!
$   SET VERIFY
$   IF F$MODE() .EQS. "NETWORK" THEN DEFINE SYS$OUTPUT SYS$NET
$!
$   DEFINE/USER SYS$COMMAND SYS$INPUT
$   SQL
     SHOW VERSION
     DROP PATHNAME USD08:[ADAMS.DBS.PERS_DICT]TARGET_DB.PERSONNEL;
      INTEGRATE DATABASE FILE 'USER6:[COLLEGE.DBS.TARGET_DB]TARGET_DB'
     ALTER DATABASE FILE 'NODE12::USER6:[COLLEGE.DBS.TARGET_DB]TARGET_DB'
     CREATE PATHNAME USD08:[ADAMS.DBS.PERS_DICT]TARGET_DB.PERSONNEL;
    COMMIT;
    EXIT;
$!
$   CLOSE SYS$INPUT
$   CLOSE SYS$COMMAND
$   EXIT
```

SQL requires that you use a different form of the INTEGRATE statement if you are integrating the information into the dictionary for the first time. Instead of distinguishing the first from subsequent integrations, this command file simply deletes the definitions and then inserts them as if for the first time.

When the procedure is executed as a DECnet task, the SET VERIFY command at the beginning of the procedure allows output to be sent back to the originating node by assigning the output to SYS$NET. The output then appears in the copy process log for the transfer.

After integrating data definitions from the target database, a user at the remote target site cannot make any metadata changes to the target database unless that user specifies a path name when invoking the target database.

8

Transferring Data to Existing Databases

Replication Option enables you to define and execute transfers that copy tables to an existing target database. This target database can be one created by a previous Replication Option transfer, or it can be an existing database that was never connected to Replication Option. Furthermore, you can define several transfers to copy tables into a single, existing target database. This chapter describes the benefits of transferring data to existing databases and how to use this feature.

The ability to create transfers to existing databases is implemented through the SQL CREATE TRANSFER statement.

Note

Using an existing database as a target for a transfer introduces operational complexities such as questions of data and data definition ownership, and possibilities for database access conflict.

If you are a new user of Replication Option, Oracle Corporation recommends you read this entire handbook from front to back before using Replication Option. Although the first chapters discuss transfers that create the target database files, much of that information is relevant to Distributed Option and the Rdb Transparent Gateway products that cannot create target database files. Note that non-Rdb systems can only transfer data to existing databases.

8.1 Overview

You can use Replication Option to create transfers to an existing target database to produce the following benefits:

- Merge reference data with existing data that depends on it
- Roll up data from multiple source databases into one target database

- Share data among multiple databases
- Create an archive database

8.1.1 Merging Reference Data with Existing Data

Businesses that are geographically distributed often maintain common information in one database located at a central site. Such information could include price catalogs, job classifications, or salary ranges. Sharing this common information with databases at remote sites is a valuable asset; the information can be used as reference data, joined with other data in queries and in views, and serve as validation data in constraints on other tables. Rather than create a new target database, Replication Option re-creates the selected source tables in an existing target database.

8.1.2 Rolling Up Multiple Sources to One Target Database

Rolling up is the taking of data from multiple sources and merging that data into a single target table in a target database.

Replication Option supports a number of options for sharing the writing of information in databases. Multiple Replication Option transfers and customer applications can read the same data from database tables. This discussion is about writing or modifying data. Table 8–1 shows the varying degrees of data sharing, from most restrictive to least restrictive, where mode 1 is the most restrictive and mode 4 is the least restrictive. The table shows if Replication Option supports each mode and if multiple transfers can write to the same database, table, and/or row in each mode.

Table 8–1 Modes of Data Sharing in Replication Option

	Mode 1	Mode 2	Mode 3	Mode 4
Supported Mode?	Yes	Yes	Yes	No
Database	No	Yes	Yes	Yes
Table	No	No	Yes	Yes
Row	No	No	No	Yes

These modes are described in the following list:

- Mode 1: Only one transfer can write to a given target database.

  This is the most restrictive mode of operation. Replication Option creates the target database files as part of executing a transfer. If the transfer is repeated on a regular schedule, the old database files are superseded by the new files. Effectively, anything written into the old database files is

lost unless it is re-created by executing the transfer again. For this reason, no other transfer or user application should be writing to that database.

This mode of operation is specified when you use the TO NEW FILENAME clause in a CREATE TRANSFER statement.

For mode 1, Replication Option must create the RDB$VINTAGE or DDAL$TRANSFER_INFO table in the target database. Also, to support replication transfers, Replication Option must add a DDAL$DBKEY column to each transferred table.

- Mode 2: Only one transfer can write to a given target table.

  By contrast, if transfers are created using the TO EXISTING FILENAME clause in the CREATE TRANSFER statement, several transfers and user applications can write to the same target database. However, the tables must have different names. Unless the shared-write option is enabled (discussed under mode 3), each transfer must write to its own set of tables. For example, transfer A can write to tables P and Q, while transfer B can write to tables R and S, and application C can write to tables T and U. They can all read one another's tables; it is only the writing or modifying of data in the tables that is restricted.

  For mode 2, Replication Option must create the RDB$VINTAGE and RDBVMS$TRANSFER_OWNER tables for Rdb target databases or the DDAL$TRANSFER_INFO and DDAL$TRANSFER_OWNR tables for non-Rdb target databases. Also, to support replication transfers, Replication Option must add a DDAL$DBKEY column to each transferred table.

- Mode 3: Only one transfer can write or modify a given target data row.

  In this mode of operation, known as the shared-write option, transfers (and in certain cases, customer applications) can write to the same target table. However, once a transfer has written a row to a table, only that transfer is permitted to update or delete the row. To select this mode of operation, use the TO EXISTING FILENAME clause in the CREATE TRANSFER statement. In addition, you have to define the DDAL$SHARED_WRITE environment variable for each participating transfer using either the transfer-specific form or the global form of that environment variable. See Appendix B for the valid forms of this environment variable.

  If the transfers all use the extraction method, user applications can also write to the same tables. If the transfers use the REPLICATION method, user applications cannot write to the same tables. The transfers must all use either one method or the other; that is, you cannot have one transfer performing an extraction and another a REPLICATION to the same table.

While mode 3 provides a greater degree of data integration, transfers and user applications are more likely to experience lock conflicts and deadlock errors.

For mode 3, Replication Option must create the RDB$VINTAGE, RDBVMS$TRANSFER_OWNER, and DDAL$INDEXES tables for Rdb databases or the DDAL$TRANSFER_INFO, DDAL$TRANSFER_OWNR, and DDAL$INDEXES tables for non-Rdb databases in the target database. Also, to support replication transfers, Replication Option must add a DDAL$DBKEY column to each transferred table and a DDAL$ROW_OWNER column to support any type of shared-write transfer. The DDAL$ROW_OWNER column identifies which transfer owns the data in a table. Only that transfer can update or delete those rows.

- Mode 4: Anyone can write or modify data in any table.

  Mode 4 is not currently supported by Replication Option. In this mode, any transfer or user application can write to a given target table and can update or delete any data row. In this mode of operation, transfers and user applications can experience lock conflicts and deadlocks as in mode 3. Additionally, it is possible for updates to be lost if changes to a data row are made concurrently by two different processes.

Replication Option has three formal transfer methods that can be used in a CREATE TRANSFER statement:

- Extraction Rollup

  An **extraction rollup transfer** enables you to copy tables from several source databases and roll them up into a single, composite target database. You can perform extraction rollup either by using the CREATE TRANSFER statement to have a single transfer definition that names multiple source databases or by defining several extraction transfers to the same target database. The advantages and disadvantages of extraction rollups are discussed in Chapter 2.

- Extraction

  An **extraction transfer** enables you to copy tables from a single source database to a target database. The tables copied to a target database by an extraction transfer maintain no ongoing relationship with the source database tables. Unlike a replication transfer, Replication Option severs the connection between the source database and the target database after the extraction transfer completes. Each time an extraction transfer executes, Replication Option copies the latest version of the selected tables to the target database. Although you can write to tables created by Replication Option in a target database, any future execution of an

extraction transfer that copies the same tables from the source database will not include data written only to the target tables.

- Replication

  A replication transfer enables you to copy tables from a single source database to a target database. After the initial transfer, the target database is periodically updated to reflect changes made to the source database tables. When a replication transfer executes, Replication Option creates the tables 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 database. Subsequent transfers involve only those additions, modifications, and deletions that have been made to the selected rows in the source database.

Replication and extraction transfers can be used where the target database already exists, whereas extraction rollup transfers cannot. Extraction rollup transfers can only be used when creating a new Rdb target database. When rolling up data from multiple source databases into a single target database (whether for shared-write or not), you must create a separate transfer definition for each source database.

A replication rollup transfer combines the update feature of the replication transfer with the rollup feature of the extraction rollup transfer. You can perform this type of transfer, even though Replication Option does not explicitly offer this choice, by transferring to an existing database. Instead of creating a single transfer to manage all the source databases, as in an extraction rollup transfer, you create one replication transfer for each source database. Each of these replication transfers would name the same target database for a destination. Refer to Section 8.2.4 for more information and a discussion of the differences between extraction rollup and replication rollup transfers.

You can have several extraction transfers or replication transfers write to the same target database tables using the shared-write transfer option.

8.1.3 Sharing Data Among Multiple Databases

Replication Option allows you to share data among two or more databases. For example, one can create a transfer of data from database A to database B and also create a transfer from database B to database A. This is not restricted to just two databases. There are two options to sharing data:

- Nonshared-write—This transfer option is selected by defining the DDAL$SHARED_WRITE environment variable (see Section B.15 for details) to "No" or by not defining the variable at all. With shared writing disabled, you can have multiple transfers moving data among participating databases. However, you cannot transfer data into an existing table

unless that table was created by a prior execution of the same transfer. In other words, the target table must be "owned" by that transfer. If the participating databases have tables with the same names, each transfer must assign a different name to each table it creates to avoid conflict with the existing tables.

Two-way data transfers are illustrated by the following example. Consider two separately managed databases: database A and database B. Each has data that needs to be shared with the other. To accomplish this, you can create two transfers of either the replication or extraction type. One transfer uses database A as a source and database B as a target. The second transfer uses database B as a source and database A as a target. Each source database serves as the target database for the other. With shared writing disabled, you cannot achieve a physical union of data between existing tables in the two databases. If two identical tables exist in databases A and B, the table transferred from database A to database B must be renamed. If you want to join the table copied from database A with the table already in database B, you can define a view in database B that combines the two tables.

This type of transfer is not limited to two databases; you can also share data using this method with three or more databases.

- Shared-write—This option allows you to write data from one or more source databases to a common target table. This transfer option is chosen by defining the DDAL$SHARED_WRITE environment variable (see Section B.15 for details) to "Yes". Each writer that inserts data into a shared table must mark each data row with a distinctive identifier. Only the owner of a given data row should be responsible for updating or deleting it. See Section 8.2.5 for the rules that apply to shared-write transfers and examples of sharing data between two or more databases. As with the nonshared-write option, you can share data between two or more databases.

8.1.4 Archive Databases

A replication transfer where the target data is never deleted is called an archive transfer. By specifying an archive transfer, you can create a target database that holds all data currently stored in the source database, as well as any data that has been deleted from the source database after the transfer executes for the first time. Thus, over time, the target database will become an archive of data from the source database.

With a few exceptions, an archive transfer works in the same way as a standard replication transfer. See Section 8.2.3 for details.

8.2 Using Transfers to Existing Databases

Before you use the transfer to existing databases feature, you should understand the basic operations of Replication Option described in previous chapters of this handbook. The syntax for executing transfers to existing databases is described in Chapter 9 in the CREATE TRANSFER statement. This section discusses how transfers to existing databases are performed and the role you play to make them work properly.

As a general rule, anyone who creates, executes, alters, and schedules transfers should maintain close communication with the administrators of the source and target databases. All must agree on when and what to transfer. You must discuss the performance of normal database maintenance and how that maintenance might affect the execution of transfers.

Replication Option is unaware of what storage areas may be used to house transferred tables. For Rdb, then, an existing target database can have either the single-file or multifile attribute.

8.2.1 Extracting Reference Tables

The following sections describe an extraction transfer to an existing database. For example, you might want to create an extraction transfer to move a salary-range table to a remote personnel database and have the transfer include the following options:

- The target table should be indexed.
- A constraint should be established on some existing table in the personnel database for the salary column, based on the salary-range data to be transferred.
- Views should be created that join the salary-range data with rows from other tables in the personnel database.

For this example, it is assumed that shared writing is disabled. Sections 8.2.1.1 through 8.2.1.8 discuss various aspects of this transfer that you should consider.

8.2.1.1 Using an Epilogue Procedure to Add Indexes, Constraints, and New Views

In the transfer definition proper, you indicate that you want to transfer to some existing, remote database and you name the table to be transferred. Replication Option does not copy indexes and constraints, however. Therefore, in the transfer definition you should reference an epilogue command procedure of your own creation. This epilogue procedure creates the table index once the data has been transferred. The procedure can also establish table constraints and create additional views.

To transfer into an existing database and to perform these operations, you must have the necessary database privileges for the user account used at the target site by the transfer process. This requires close cooperation between the person creating the transfer definition and the person administering the target database.

8.2.1.2 Updating the Target Data Dictionary

In some applications, a target database may require updates to the local data dictionary whenever data definitions are added or altered. Replication Option does not automatically integrate data definitions into a data dictionary. If you want this done as part of the transfer process, use an epilogue procedure. Refer to Section 7.3.5 for detailed information.

8.2.1.3 Using Prologue Procedures for Transfer Re-Execution

You may need to repeat a transfer for one of the following reasons:

- To transfer new data on a scheduled basis
- To retry following a transfer failure

In either case, you may need to reference a prologue procedure in the transfer definition so that certain considerations, outlined in Sections 8.2.1.4 through 8.2.1.8, can be taken into account prior to the execution of the copy process.

When a copy process performs an extraction transfer to an existing database, the copy process first checks for the existence of the data definitions to be transferred. Unless the initial transfer has failed, those data definitions already exist in the target database.

When an extraction transfer re-executes, Replication Option erases the existing data before copying the new data into the tables. Extraction transfers do not read data from source and target databases, search for differences, and then copy only the new data. That operation is a replication transfer update. For a replication transfer update, the differences are known ahead of time and recorded in the source database in the RDB$CHANGES table.

8.2.1.4 Changing Table Definitions

Before an extraction transfer to an existing database executes, Replication Option's copy process checks the source and target database table definitions and notes any changes in the source definitions. If changes exist, the copy process alters the target table definition to match the source table definition, unless there is no way for such changes to be made. Currently, you can only do this with Rdb. Consequently, if a source table does not match a target definition in a Distributed Option or transparent gateway product, Replication Option will log an error and the transfer will be terminated. In

these situations where you change source table definitions, also change the target table definitions to match before executing a transfer.

Changing a table definition in the target database can fail if other definitions exist that depend on that table. These dependent definitions can include views, constraints, indexes, and triggers. If all the dependent definitions are managed by the transfer, use a prologue procedure to drop the dependent definitions. If some definitions are managed at the target database site independent of the transfer, changing data definitions must be coordinated among the persons managing the various databases.

8.2.1.5 Changing Domain Definitions

After you create an extraction transfer, it is possible for someone to change data definitions in the source database. For example, a postal code domain could be changed to increase its size. For Rdb target databases, Replication Option's copy process examines the domains previously created in the target database and compares the source and target domain definitions for the current transfer. If the definitions differ, the target definition is changed to reflect the source definition. You must determine whether or not the definition change is advisable. Changing data type or reducing data size can result in loss of data precision. For non-Rdb target databases, altering domain definitions is not currently supported.

Local data definitions and data definitions created by other transfers to the same target database can reference a domain owned by and altered by another transfer. In this case, a data definition change in a source database can affect data definitions owned by others in a target database. This situation requires close communication among database administrators.

8.2.1.6 Changing View Definitions

You can transfer a view in one of three ways:

- For extraction transfers, name a view in a MOVE TABLES clause. The view is created in the target database as a table of its own and contains real data.
- For extraction or replication transfers, transfer a view as a view definition, based on other tables and views within the transfer definition proper.
- Add views to a target database by using an epilogue procedure associated with a transfer.

If you use the view definition method or an epilogue procedure, a view definition rather than a table definition is created. When a transfer repeats, some database systems do not allow the alteration of view definitions. Instead, you must drop the view definition and then create a new one. Because views can be defined in terms of other views, you must drop them in the prologue procedure in the reverse order of their addition. Replication Option's copy process does this for views named within the transfer definition, first examining the definitions to determine the dependency hierarchy.

8.2.1.7 Dropping Indexes and Constraints

You cannot drop all the rows from a table if doing so violates any established constraints. This applies to primary and foreign key attributes also, because these keys are implemented as constraints in SQL. Wholesale deletion of data from tables occurs whenever an extraction or initial replication transfer is repeated to an existing target database. The old data must be dropped before it is replaced with the current source data.

Replication Option normally drops data from each table in a separate transaction. To avoid this causing a constraint violation, you must drop the constraint definitions from the target database prior to executing the transfer and redefine the constraints once the transfer has completed successfully. This can be done within prologue and epilogue procedures associated with the transfer.

There is an alternative provided that your constraints are all declared to be deferrable. You can request that Replication Option drop data from and rewrite data to all tables within a single transaction. This allows transfers to be re-executed without you having to drop constraints and without causing constraint violations. Whether or not this will improve transfer execution time is something one must determine by experiment. For some database systems this option increases the amount of target disk space used. The "single data commit" option is discussed in detail in Appendix B.

If the table is indexed, deleting all rows will take longer than if the table is not indexed. For tables with only a few rows, this might not matter. As a general rule, drop a table index before performing wholesale changes to the data. Once the transfer has repopulated the table, you can re-create the indexes by using an epilogue procedure.

8.2.1.8 Database Access Conflicts

When Replication Option performs a transfer to an existing target database, there is the potential for database access conflicts to arise. Such conflicts can occur between the transfer's copy process and some local target application that references the tables managed by Replication Option. Conflicts can also arise between two copy processes, each independently performing its own transfer. It is normal to expect such conflicts from time to time.

In some cases a copy process can deal with the conflict by rolling back its transaction, detaching from the target database if necessary, hibernating for half a minute, and then retrying the operation that caused the error. In other cases the copy process stops the transfer. The transfer fails but will be retried according to the retry parameters in the transfer schedule. If the latter type of failure occurs frequently, make sure that the maximum number of retries defined in the transfer schedule is sufficiently high to permit a reasonable number of retries. Also, consider changing the retry delay to see if that can decrease the number of conflict errors or decrease the elapsed time to achieve transfer completion.

8.2.2 Replicating Reference Tables

The considerations you make for extraction transfers to existing databases, described in Section 8.2.1, are the same considerations you must make for replication of reference tables.

8.2.3 Archiving Production Data

A replication transfer in which the target data is never deleted is called an archive transfer. Such a transfer is specified in a transfer definition as follows:

```
CREATE TRANSFER transfer-name TYPE IS REPLICATION WITH NO DELETE
. . .
```

Note

Archive transfers can only be performed when the target database is managed by one of the Oracle database managers, either Rdb or Oracle. If you attempt an archive transfer to a non-Oracle or non-Rdb database, an error is reported. This is because the archiving capability depends on the existence of native database keys in the target database system, something that only Oracle Corporation products currently have.

The considerations you must make for an archive transfer include all those you must make for extraction transfers to existing databases, plus the following:

- How database keys are used

  Replication Option uses the dbkey value for rows in the Rdb source database to locate the corresponding row in the target database. This is done by adding a DDAL$DBKEY column to all tables in the target database involved in the replication. When a replication transfer without the NO DELETE attribute executes, each time a source row is written into the target database, the DDAL$DBKEY value of the source row is also recorded. In addition, Replication Option expects that this value will be unique and defines an index on the DDAL$DBKEY column with the UNIQUE attribute or expects you to add a column constraint to enforce dbkey uniqueness.

  When a replication transfer is defined with the NO DELETE attribute, the DDAL$DBKEY value in the target table is used to flag the rows that have been deleted in the source database. This flagging is accomplished by setting the DDAL$DBKEY value to NULL in the target database. Thus, the values in the DDAL$DBKEY column are not unique in a target database, resulting from a replication transfer with the NO DELETE attribute. Once the DDAL$DBKEY value is set to NULL, the row is no longer active and Replication Option will not touch the row again in the target database.

  Because the DDAL$DBKEY value is used to flag inactive rows, the same select expression used against both the target and source database may not return the same data. You must specify a WHERE DDAL$DBKEY IS NOT NULL clause on your select expression for the target database to have SQL return the same data as the source database for the same query. See Example 8–1.

- The target database must be an existing database

  When you first set up your transfer definition, be sure that you define the TO-clause to be an existing database. This is necessary in case your transfer ever needs to be reinitialized. When a transfer to a new database is reinitialized, the source database tables are copied to a new target database even though an old target still exists; in the case of an archive database, this would mean that all the archived data would be lost. When a transfer to an existing database is reinitialized, all rows in the existing target are deleted and recopied from the source database *if, and only if, those rows have a DDAL$DBKEY value that is not NULL*. Therefore, when a transfer to an existing database is reinitialized, all the archived data (data that has been deleted from the source database) is maintained in the target database.

Note, however, that when you first define a replication transfer with NO DELETE, the existing target database need not include table definitions. Only the database file in its simplest form needs to exist. Example 8–1 demonstrates how to create an archive transfer and access data in the archived database.

Once the DDAL$DBKEY column for rows in the target database have been set to NULL, Replication Option will ignore those rows. Therefore, over time, the target database may grow considerably larger than the source database. If, at some point, you find that the target database is too large, you will need to design a process for deleting the archived data from the target database and moving it to an offline storage device.

Example 8–1 uses the following steps to create and maintain an archive of the PERSONNEL database. Target database creation is not shown because that is specific to the type of target database system to be used. Also, file specifications are abbreviated here and normally would include operating-system specific information for locating the file.

1. Defines a replication transfer with the NO DELETE attribute and moves all the tables from a PERSONNEL database to an ARCHIVE_PERS database.
2. Starts the transfer, which will be a replication initialization transfer, to copy the data from PERSONNEL to ARCHIVE_PERS.
3. Deletes the row with an EMPLOYEE_ID number of 00165 from all the tables in the PERSONNEL database.
4. Re-executes the transfer. This time it will be a replication update transfer to update the target database. In this case, because a row was deleted in the source database, the corresponding row will have its DDAL$DBKEY column set to NULL in the target database.
5. Attaches to the target database to check that rows have been updated as expected.

Example 8–1 Creating an Archive Database Using Replication with NO DELETE

```
SQL> ATTACH 'FILENAME PERSONNEL';
SQL> CREATE TRANSFER ARCHIVE
cont> TYPE IS REPLICATION WITH NO DELETE
cont> MOVE TABLES *
cont> TO EXISTING FILENAME ARCHIVE_PERS
cont> LOG IS ARCHIVE_PERS.LOG
cont> COMMENT IS 'Create and Maintain Archive of PERSONNEL Database';

SQL> DISCONNECT ALL;
SQL> START TRANSFER NOW WAIT;

SQL> ATTACH 'FILENAME PERSONNEL';

SQL> DELETE FROM EMPLOYEES WHERE EMPLOYEE_ID = '00165';
1 row deleted

SQL> DELETE FROM JOB_HISTORY WHERE EMPLOYEE_ID = '00165';
4 rows deleted

SQL> DELETE FROM DEGREES WHERE EMPLOYEE_ID = '00165';
1 row deleted

SQL> DELETE FROM EMPLOYEES WHERE EMPLOYEE_ID = '00165';
1 row deleted

SQL> DELETE FROM SALARY_HISTORY WHERE EMPLOYEE_ID = '00165';
12 rows deleted

SQL> COMMIT;
SQL> SELECT LAST_NAME FROM  EMPLOYEES WHERE EMPLOYEE_ID = '00165';
0 rows selected

SQL> START TRANSFER NOW WAIT;

SQL> DISCONNECT ALL;

SQL> ATTACH 'FILENAME ARCHIVE_PERS';

SQL> SELECT LAST_NAME FROM  EMPLOYEES WHERE EMPLOYEE_ID = '00165';
 LAST_NAME
 Smith
1 row selected

SQL> SELECT LAST_NAME FROM EMPLOYEES WHERE EMPLOYEE_ID ='00165'
cont> AND DDAL$DBKEY IS NOT NULL;
0 rows selected
```

8.2.4 Rollup Transfers

The ability to transfer to existing databases enables you to define multiple transfers that use the same database as a target. Each transfer, whether an extraction or a replication, has its own source database, transfer definition, and transfer schedule. Each transfer creates its own set of target tables unless you have the shared-write option enabled. If the tables contain similar data and you want to achieve a union of this data and you do not have shared writing enabled, create a view that performs the union of data from like tables. This type of transfer causes a rollup of data from databases physically removed from each other.

Compare this type of rollup with that performed by an extraction rollup transfer and note the differences:

- Single versus multiple transfers and schedules
  - An extraction rollup transfer uses a single transfer definition and a single schedule.
  - Multiple transfers to an existing database require one transfer definition for each source database and one transfer schedule for each transfer.
- All or none versus independent transfers
  - All data transfers in an extraction rollup transfer must execute at one time. If one of the source databases is not accessible, the transfer cannot proceed.
  - Multiple transfers to the same database execute independent of each other. If one transfer cannot run because a database is unavailable, the other transfers are not impeded.
- Physical union versus virtual union
  - An extraction rollup transfer can transfer like tables from each source database, create a composite target database, and achieve a true, physical union of the data. For example, if one source contains personnel records from employees in California and another contains personnel records from employees in Texas, a single employees table containing the employee records from both states can be created in the target database.
  - Multiple transfers to an existing database can only create separate tables when shared writing is disabled. Therefore, like tables must be given new names to achieve name uniqueness. Using the preceding example, you would have to create a CALIFORNIA_EMP table and a

TEXAS_EMP table. Union of the data rows occurs when the rows are read using a union view. Alternatively, with shared writing enabled, these multiple transfers can write to the same target tables. A union view is therefore unnecessary.

8.2.5 Rules for Multiple Writers to the Same Tables

Replication Option allows two or more transfers to write into the same set of target tables if the shared-write option is enabled. In some situations, user applications can write into the transferred tables and transfers can write into local user application tables. However, there are several restrictions that must be observed.

Section 8.2.5.1 shows how you could define two transfers to write to a single target database.

Section 8.2.5.2 shows how you could transfer data among multiple databases.

Oracle Corporation recommends you read and follow these rules carefully:

1. You must have Replication Option 7.0 or some later version installed on your system to use the shared-write feature.

   For example, if you have two sites where one site has Replication Option 7.0 and the other has a version prior to 7.0, both systems cannot transfer data into the same target table. Systems with versions of Replication Option prior to 7.0 cannot use the shared-write feature.

2. Each transfer manages its own set of data.

   Replication Option allows you to transfer different sets of data into a common target table. However, the only transfer allowed to subsequently alter or delete a given row of data is the transfer that wrote the data in the target table originally. In some cases (discussed later in this section), user applications will also be able to manage their own set of data in these common tables.

   To achieve this degree of data isolation, Replication Option adds a DDAL$ROW_OWNER column to shared-write tables as a method of labeling each data row with an identifier that indicates who owns the data. Each transfer will write its own transfer identification code in this column so you can distinguish data written by transfer A from data written by transfer B. This is especially important in the following two situations:

   - One situation is when extraction transfers or initial replication transfers execute a second time. When they do, they must erase all data they had transferred previously so the data can be replaced by a current snapshot of the source data. When the tables are used with

shared writing, the transfers can erase only their own data but not *all* data in the tables, so they need a way to identify the rows they own.

- The second case is for performing replication update transfers. A replication update transfers changes that have occurred to data within a source table. These changes could be insertions, updates or deletions. To perform updates and deletions, Replication Option must be able to locate specific rows in the target database. In previous versions of Replication Option, this was done using a unique value for the DDAL$DBKEY column. However, if more than one transfer replicates data into the same target table, the database keys in DDAL$DBKEY are no longer necessarily unique. Replication Option uses the DDAL$ROW_OWNER and DDAL$DBKEY columns to make unique keys.

3. If you intend to transfer data, using the shared-write option, to tables that will be written into by other transfers or by local applications, the CREATE TRANSFER statement must specify the following clauses:

   - Use the TO EXISTING clause to specify an existing target database.
   - Use the WITH NO CHECKING clause to disable SQL one-writer checks.

   Each transfer that writes into a target table shared by multiple writers must have the TO EXISTING clause in its transfer definition.

   For transfers to Rdb databases, you can specify either the TO NEW or the TO EXISTING clause in a transfer definition. The TO NEW clause instructs Replication Option to create a new set of target database files at the start of an extraction, extraction rollup, or initial replication transfer.

   On OpenVMS systems, The previous target database files are not deleted; another set of files is created with a higher version number of the file.

   On Digital UNIX, re-creating existing Rdb files is not permitted. You must delete the existing files before executing a transfer that uses the TO NEW FILENAME option.

   Creating new target database files implies that all previous target tables and data are in effect erased. One transfer would be erasing the information written by another transfer. Therefore, for shared writing of data to work, all participating transfers must include the TO EXISTING clause in their CREATE TRANSFER statements.

4. All participating transfers must have the shared-write option enabled.

By default, Replication Option assumes that each transfer is the only writer into a given target table and ownership checks are performed at run time to enforce this assumption. If you want transfers to write into a common target table, you must enable the shared-write option for those transfers. See Section B.15 for details on enabling shared writing.

If a target table is to be shared for writing by more than one transfer, it has to be shared for writing by all transfers. For example, you cannot have one transfer declare nonshared-write (either explicitly or by default) and the other transfers specify shared-write. The transfer that executes with no shared-write and that needs to refresh its entire set of data (such as for an extraction or initial replication transfer) will erase *all* the data in the target table, not just the data that it has written into the target table.

5. A transfer to a new database will ignore the shared-write option.

   A transfer that creates its own target database files is performed when you specify the TO NEW FILENAME clause in the CREATE TRANSFER statement. If shared writing is declared for such a transfer, the shared-write attribute will be ignored.

6. Do *not* change a shared-write declaration.

   Once a transfer has executed, do not change from nonshared-write to shared-write or from shared-write to nonshared-write. This will lead to incorrect results or a run-time error.

7. Do not mix extraction and replication transfers.

   If you want to have several transfers write into a given target table, all transfers must use the same transfer method. They can all be extraction transfers or they can all be replication transfers but you cannot have some transfers be of one type and some the other type.

   This restriction exists because Replication Option adds a DDAL$DBKEY column to the target table for replication transfers. That column together with the DDAL$ROW_OWNER column contains a unique value for each replicated data row. As a safeguard to catch software or human errors, either Replication Option adds an index with a built-in uniqueness constraint or you have to add a column constraint to achieve the same results. When an extraction transfer is performed, no value is written into the DDAL$DBKEY column (this is only needed for replication transfers). Therefore, there will be many duplicate (NULL) values for that column for the data transferred by the extraction method. Because that would violate the uniqueness constraint, the extraction transfer would fail the test.

8. Shared writing between user applications and Replication Option transfers is allowed, with restrictions.

If you have an existing target database with tables managed by local application programs, you can transfer data into those tables from some other data source. Similarly, your applications can write data into a table created by a Replication Option transfer. Normally, this type of sharing is allowed only when the transfers use the extraction method, not the replication method.

Extraction transfers will work because the transferred rows will have a value in the DDAL$ROW_OWNER column and user application data will have a NULL value in that column. The user applications will have to be modified to be selective about the data they update or delete so they only look for rows where the DDAL$ROW_OWNER column has a NULL value.

Normally, replication transfers will not work because of the uniqueness requirement on the combination of the DDAL$DBKEY and DDAL$ROW_OWNER columns. User applications, which would not write into these columns, would generate multiple data rows with the same (NULL) value for these columns, thereby generating duplicate index values. As a result, the user applications would immediately fail the uniqueness test.

Trying to work around this situation by removing the uniqueness constraint is not advisable. It is there to detect errors, whether caused by an operational mistake, a hardware malfunction, or a defect in the database software.

A better solution would be to create an index on Replication Option columns and on user application columns and maintain unique values throughout. This solution does require that the target table have another column (or columns) that can be used by local applications to store unique values. For example, assume the target table has the DDAL$ROW_OWNER, DDAL$DBKEY and LOCAL_KEY columns, as follows (DDAL$DBKEY column values are typically not simple numbers such as these, but these are shown to illustrate the point):

DDAL$ROW_OWNER	DDAL$DBKEY	LOCAL_KEY	
1	1	null	<– transfer 1's row
2	1	null	<– transfer 2's row
null	null	1	<– application's row
1	2	null	<– transfer 1's row
null	null	2	<– application's row

This example shows that:

- The replication transfers write values into the first two of these columns and leave the LOCAL_KEY column empty (null).
- The user application writes into the LOCAL_KEY column and leaves the other two columns empty.

If you were now to create a unique index or constraint on all three columns, the uniqueness criterion would not be violated. If the index or constraint were created on only the first two columns, the uniqueness criterion would be violated.

It might be possible to avoid violating the uniqueness criterion if you suppress the creating and dropping of target indexes during transfer execution. However, if you do so, you must remember that once the target tables are in place, it is your responsibility to create the indexes (and constraints, if necessary) on the target tables.

For information on how to suppress the creating and dropping of indexes during a transfer, read about the DDAL$TARGET_INDEX_OPTIONS environment variable in Section B.19.

9. Indexes must be unique for replication transfers whether Replication Option creates the indexes or you create them.

   As mentioned previously, Replication Option requires values be unique in the combined DDAL$DBKEY and DDAL$ROW_OWNER columns for replication transfers. To ensure uniqueness, Replication Option creates an index for each target table based on these columns. For some target systems, the uniqueness criteria is part of the index definition. For others it has to be done by you, the customer, by adding a column constraint.

   The index is also created to provide better performance for transfers when performing updates and deletions of rows in the target tables.

10. You may add indexes for extraction transfers that use the shared-write option.

When extraction transfers with the shared-write option enabled are run to a common target table, Replication Option will not add any indexes of its own for that target table. You may add indexes to such a table if so desired.

11. The DDAL$ROW_OWNER column must be added to shared-write tables.

    If Replication Option creates the target table and shared writing is enabled, Replication Option adds a DDAL$ROW_OWNER column to the table definition. If a target table already exists and is to be used for shared writing, Replication Option will not alter the table definition to add the DDAL$ROW_OWNER column (even if Replication Option created the table originally). You have to add the DDAL$ROW_OWNER column before running the transfer.

12. Data definitions must be the same among source databases.

    If you have two or more source databases and you want to transfer data from table T in each source database into a single table T in some target database, the table definitions for the source tables must be identical. The column definitions must be the same as well as the number of columns and the column names.

    If the source definitions do not match, Replication Option will not explicitly detect this fact (that is, Replication Option will not go out and compare the source definitions for you and inform you they are different). Instead, as each transfer executes, Replication Option will check to see if a data definition exists in the target database. If so, Replication Option will compare the source and target definitions. If they do not match, Replication Option will report an error. Note this is only the case for shared-write transfers. For transfers that do not use shared writing, if Replication Option detects a difference between the source and target definitions, Replication Option will alter the target definition accordingly provided that the target system (including the transparent gateways) supports the ability to alter definitions.

    This does not mean that the list of tables named in each transfer must be identical. It is valid for a transfer to copy a subset of a list of tables or to copy a common set of tables plus a set of tables unique to that transfer.

13. Changing a target table is your responsibility.

    Given the restriction of rule 12, what happens if shared-write transfers have been in place and running and you now need to make some change to source data definitions? Prior to rerunning any of the affected transfers, you must modify the definitions in all participating source databases and also in the target database.

14. Dropping a target table is your responsibility.

    When Replication Option performs a nonshared-write transfer, Replication Option will detect the situation where the transfer definition was changed to exclude some table that had been transferred in the past. In this case, Replication Option will drop that table from the target database. The behavior when performing a shared-write transfer is different. A table no longer mentioned in a transfer definition will *not* be dropped from the target database because the other transfers doing shared writing into that target table might still be referencing that table. Therefore, if a shared-write target table is no longer needed by any of the transfers, you need to drop the target table and the record of that table in the DDAL$TRANSFER_OWNR or RDBVMS$TRANSFER_OWNER table.

15. Circular transfers are permitted if you use selection clauses.

    Common scenarios for transferring data among peer databases include the following:

    - Two databases with transfers going from each to the other. See Section 8.2.5.1 for an example of this scenario.
    - Three databases with database A transferring to database B, database B to database C, and database C to database A. See Section 8.2.5.2 for an example of this scenario.

    Shared-write transfers can be used in these two cases. Each shared table is both the source of data for a transfer to other databases and the target of transfers of data from the other databases. Remember, rule 2 states that each transfer manages only its own set of data and must not manage the data owned by other transfers. You follow this rule by including a SELECT expression in the transfer definition. The local data rows to be transferred will have a NULL value in the DDAL$ROW_OWNER column. So, for example, the following shows a transfer definition for transferring all locally owned rows from the INVENTORY table:

```
CREATE TRANSFER TO_DATABASE_B TYPE IS EXTRACTION
 MOVE TABLES
  SELECT * FROM INVENTORY
   WHERE DDAL$ROW_OWNER IS NULL  . . .  ;
```

    The use of the SELECT expression is crucial when performing replication transfers. If you did not use the SELECT expression, changes to data rows would circulate among the databases endlessly. An insert into database A would be transferred to database B. From database B, that insert would be transferred to the next database, and so on back to database A to repeat the cycle.

8.2.5.1 Transferring Data from Multiple Source Databases into a Common Target Table

You can perform a rollup of tables from multiple source databases into a common target table in a target database by executing multiple transfers independently. In order to perform this type of rollup transfer, you have to enable the shared-write option. This section illustrates the steps you would use and the results you could expect to see.

1. Enable the shared-write option by defining DDAL$SHARED_WRITE as described in Section B.15. If there are going to be several transfers using the same target account and writing to the same target tables, you need to enable shared writing for each of the transfers or enable it globally for all transfers.

2. The following are two source databases, ma_employees and nh_employees, each having an employees table with the following rows of data in them.

```
SQL> attach 'filename ma_employees';
SQL> select * from employees;
 NAME
 Christine Connolly
 James Herbener
2 rows selected
SQL> disconnect all;
SQL> attach 'filename nh_employees';
SQL> select * from employees;
 NAME
 Edward Keisling
 Hope Lapointe
2 rows selected
SQL> disconnect all;
```

3. Create a transfer such as the following to transfer the data in the ma_employees database to the target database, which is named all_employees in this example:

```
SQL> attach 'filename ma_employees';
SQL> create transfer ma_employees_names_tr type is extraction
cont> move tables * to existing filename all_employees
cont> with no checking
cont> log file is ma_employees_names_tr;
SQL> disconnect all;
```

4. Create another transfer to transfer the data in the nh_employees database to the all_employees target database:

```
SQL> attach 'filename nh_employees';
SQL> create transfer nh_employees_names_tr type is extraction
cont> move tables * to existing filename all_employees
cont> with no checking
cont> log file is nh_employees_names_tr;
cont> disconnect all;
```

5. You can execute each transfer using either the CREATE SCHEDULE statement to have the transfer occur on a fixed schedule or the START TRANSFER NOW statement to have the transfer execute on demand.

   You might find that when two or more transfers write concurrently to the same target tables, access conflicts or deadlocks occur. There are two ways to deal with this.

   - Schedule transfer execution so that only one transfer writes to the target tables at a given time.

     Or

   - Enable checkpoint restart in each transfer's definition. This is done by adding the CHECKPOINT clause to the CREATE TRANSFER statements. Or you could add the CHECKPOINT clause later by using the ALTER TRANSFER statement. The checkpoint option does not prevent access conflict or deadlock errors; it permits transfers to recover from such errors and continue execution.

6. Examine the tables in the target database after executing the two transfers:

```
SQL> attach 'filename all_employees';
SQL> show tables;
User tables in database with filename all_employees
     EMPLOYEES
     RDB$VINTAGE
     RDBVMS$TRANSFER_OWNER
SQL> select * from employees;
 DDAL$ROW_OWNER   NAME
              1   Christine Connolly
              1   James Herbener
              2   Edward Keisling
              2   Hope Lapointe
4 rows selected
```

   Note the DDAL$ROW_OWNER column in the target table. Replication Option creates this column when it performs a shared-write transfer. The numbers in the DDAL$ROW_OWNER column indicate the transfer that inserted a particular row. Only the transfer that wrote the data to the database can alter or delete those rows of data. So, only transfer 1 (ma_ employees) can alter or delete the rows containing Christine Connolly and

James Herbener. Only transfer 2 (nh_employees) can alter or delete the rows containing Edward Keisling and Hope Lapointe.

If the target database was not an Rdb database, you would see the DDAL$TRANSFER_INFO and DDAL$TRANSFER_OWNR tables instead of the RDB$VINTAGE and RDBVMS$TRANSFER_OWNER tables.

You can determine the name of the actual transfer that owns the data from a given row owner ID. The row owner ID is taken from the transfer ID. You can find the transfer ID in the DDAL$TRANSFER_INFO or RDB$VINTAGE table in the RDBVMS$VINTAGE_TRANSFER_ID column for Rdb databases or the DDAL$TRANSFER_ID column for non-Rdb databases. The row associated with that column value will tell you the network node from which the transfer originated and the transfer name.

For example, in the preceding query you see that the row with Hope Lapointe has a row owner code of 2. To determine which transfer inserted this row, assuming the target is an Rdb database, issue the following query:

```
SQL> select rdbvms$transfer_name, rdbvms$vintage_transfer_node
cont> from rdb$vintage
cont> where rdbvms$vintage_transfer_id=2;
 RDBVMS$TRANSFER_NAME   RDBVMS$VINTAGE_TRANSFER_NODE
 MA_EMPLOYEES_TR        NOEL
```

This shows that the row was inserted by the MA_EMPLOYEES_TR transfer, which originated from node NOEL. If the target is not an Rdb database, the table and column names you use would be different (see Section A.3.1 for details).

7. Now, delete the rows in the first source database and insert the following new data:

   Susan Gray
   Jesse Crain

8. Rerun the first transfer, ma_employees_tr, and re-examine the target database.

```
SQL> attach 'filename all_employees';
SQL> select * from employees;
 DDAL$ROW_OWNER   NAME
              1   Susan Gray
              1   Jesse Crain
              2   Edward Keisling
              2   Hope Lapointe
4 rows selected
```

This example shows that the changes made in one source database, when transferred to the shared target table, only affect the data rows created by that transfer. Notice the target data owned by the second transfer did not change after re-execution of the first transfer.

8.2.5.2 Transferring Data from Multiple Databases to Each Other

You can perform rollup transfers of tables from multiple source databases into each other using the shared-write feature. In order to perform this type of rollup transfer, you have to enable the shared-write option for each transfer.

The following is an example of performing multiple rollup transfers from three databases (NY, NH, and MA) to each other. Each database has an OFFICE table, with rows originating from all three databases. The example shows how Replication Option assigns values to a column to indicate which transfer owns the data in the table. Only the transfer that wrote the data to a table in a database can delete or change that data.

You can perform multiple rollup transfers by performing the following steps:

1. Enable the shared writing option by defining DDAL$SHARED_WRITE as described in Section B.15. If there are going to be several transfers using the same target account and writing to the same target tables, you need to enable shared writing for each of the transfers or enable it globally for all transfers.

2. Add the DDAL$ROW_OWNER column to the OFFICE table in each database before you use the shared-write feature. Assume that each database already has the NAME column. For each database, NY, NH, and MA, you can use statements similar to the following to add the DDAL$ROW_OWNER column. This shows the NY database; you would do the same for the NH and MA databases.

   ```
   SQL> attach 'filename ny';
   SQL> alter table office add column ddal$row_owner smallint;
   SQL> commit;
   ```

   These statements create the DDAL$ROW_OWNER column with a value of NULL in each of the three databases.

3. The following are three databases named NY, NH, and MA with tables labeled NAME and the following rows of data in them:

```
SQL> attach 'filename ny';
SQL> select * from OFFICE;
 NAME                                                DDAL$ROW_OWNER
 New York City                                                NULL
1 row selected
SQL> disconnect all;
SQL> attach 'filename MA';
SQL> select * from OFFICE;
 NAME                                                DDAL$ROW_OWNER
 Boston                                                       NULL
1 row selected
SQL> disconnect all;
SQL> attach 'filename NH';
SQL> select * from OFFICE;
 NAME                                                DDAL$ROW_OWNER
 Manchester                                                   NULL
1 row selected
SQL> disconnect all;
```

Note the null values in the DDAL$ROW_OWNER column. This indicates data rows that originated from local application programs.

4. Create the six transfers in the following list, so that data is transferred from all three databases to each other:

   - NY to MA
   - NY to NH
   - MA to NY
   - MA to NH
   - NH to NY
   - NH to MA

   The following example shows the transfers from the NY database. Similar entries would be made for the NH and MA databases.

```
SQL> attach 'file NY';
SQL> create transfer NY_TO_MA type is extraction
cont> move tables select * from OFFICE where DDAL$ROW_OWNER  is null
cont> to existing filename prod$disk:[prod.personnel]MA with no checking;
SQL>
SQL> create transfer NY_TO_NH type is extraction
cont> move tables select * from OFFICE where DDAL$ROW_OWNER  is null
cont> to existing filename prod$disk:[prod.personnel]NH with no checking;
```

Note all the transfers are of type extraction and are to an existing file. Also note the "where DDAL$ROW_OWNER is null" clause. The purpose of this is to transfer only data that originates from the given source database. You do not want to transfer data that came from other databases.

5. You can execute the transfers using either the CREATE SCHEDULE statement to have the transfers occur on a fixed schedule or the START TRANSFER NOW statement to have the transfers execute on demand.

   Examine the tables in the three databases after executing the six transfers. For example, the NY database looks as follows:

```
SQL> attach 'filename NY';
SQL> select * from OFFICE;
 NAME                                                DDAL$ROW_OWNER
 New York City                                                 NULL
 Boston                                                           1
 Manchester                                                       2
3 rows selected
```

   Each of the three databases will now contain the same data in the NAME column. The values in the DDAL$ROW_OWNER column show the ID's of the transfers that inserted the data into the target table. Do not expect the owner ID for a particular office name to be the same in each database. For example, in the NY database, the Boston row has a row owner ID of 1. In the MA database, which is where that row originates, the owner ID value is null.

6. Insert new data into each database. For example, insert the city names Albany and Buffalo into the NY database, as follows:

```
SQL> attach 'filename NY';
SQL> insert into OFFICE(NAME) values ('Albany');
SQL> insert into OFFICE(NAME) values ('Buffalo');
SQL> commit;
```

7. If you then examine the NY database, you see the new Albany and Buffalo entries, but they have not yet been transferred to the other two databases.

```
SQL> attach 'filename ny';
SQL> select * from OFFICE;
 NAME                                                DDAL$ROW_OWNER
 New York City                                                 NULL
 Boston                                                           1
 Manchester                                                       2
 Albany                                                        NULL
 Buffalo                                                       NULL
5 rows selected
```

   Remember, rows where the DDAL$ROW_OWNER value is NULL are owned by (originate from) that database.

8. Run the six transfers again and examine the databases. The new data has been transferred to the other databases. For example, the MA database now contains the Albany and Buffalo entries that were originally added to the NY database. It also contains new entries made in the MA database itself and rows transferred from the NH database.

```
SQL> attach 'filename ma';
SQL> select * from OFFICE;
 NAME                                                DDAL$ROW_OWNER
 Boston                                                        NULL
 New York City                                                    2
 Albany                                                           2
 Worcester                                                     NULL
 Framingham                                                    NULL
 Buffalo                                                          2
 Manchester                                                       1
 Concord                                                          1
 Nashua                                                           1
9 rows selected
```

   Now, all three databases have the same data in the NAME column again. Note Replication Option inserted transfer_id values into the DDAL$ROW_OWNER column to identify which transfer owns the data row.

9. If you were to update or delete rows in the databases that originated them and then perform the transfers again, those changes would be reflected in all three databases.

8.2.6 Rollup of RMS Files into a Single Database

You can use Replication Option with Rdb Transparent Gateway to RMS to roll up several RMS files into a single database. One way to do this is to create an empty database to serve as the target of several Replication Option transfers.

For each RMS file, create a transfer to this existing target database. The operative clause in the transfer definition is "TO EXISTING FILENAME . . . ". Each such transfer is executed independently of the other. The transfers can occur in parallel, if you so choose, or they can occur at different times of the day.

One problem can arise that results in a "transfer not owner" error message. The message could come from SQL at the time the transfer is created, or it could come from Replication Option's copy process during execution of the transfer. An example of such a message is:

```
%SQL-F-TRANOTOWNER, This transfer is not the owner of domain
NSDS$_GF0000000057
```

When Replication Option transfers to an existing database, it records the names of all metadata objects added to that database by the transfer. The transfer is noted as the owner of those objects. Among these objects are domains. If other transfers subsequently target the same database, they may share domains owned by others provided that the domain definitions match. For example, if transfer A owns a domain of type CHAR(10), transfer B may reference that domain provided that transfer B's domain is of type CHAR(10) as well.

Rdb Transparent Gateway to RMS obtains data definitions from the data dictionary definitions of RMS file records. If record fields are defined with a BASED ON clause, you can control the correspondence between domain (also known as global field) names and their data types. If you omit the BASED ON clause in the field definitions, Rdb Transparent Gateway to RMS invents domain names of the form NSDS$_GFnnnnnnnnnnn.

If the definitions of two, unrelated RMS files are transferred to the same target database, and each record's fields do not have an explicit BASED ON clause, each record uses the same set of domain names for domains of completely different data types. Then when Replication Option attempts to transfer from these RMS files, the previously noted error can occur.

Simply using a BASED ON clause for the record field definitions in the dictionary does not guarantee that you will avoid the error. However, it does make explicit the domain names and their data types. You must ensure that domains of the same name have the same attributes.

8.2.7 Writing to Target Databases

The following sections describe the considerations to take into account when you or your applications are writing to a new or existing database as opposed to Replication Option performing the transfer.

8.2.7.1 New Databases

If you define a transfer that creates a new target database, the tables that Replication Option creates should be considered read-only. You can add your own local tables to the target database and write to them. If you have the requisite privileges, it is also possible for you to write to Replication Option-created tables. Replication Option does not enforce the read-only restriction. However, if a transfer is re-executed (extraction transfer) or reinitialized (replication transfer) such that a new target database is created, any user modifications made to the original target will not automatically be included in the new target. The transfer will create a new version of the target database files using the current source database tables.

8.2.7.2 Existing Databases

If you define a transfer that copies tables from a source database to an existing target database and you have shared writing disabled, only the tables copied by the transfer should be considered read-only. If reinitialization or re-extraction becomes necessary, Replication Option does not create a new target database; only those tables specified in the transfer definition are recopied to the existing target database. Tables in the existing target that are not owned by Replication Option are unaffected by reinitialization. If, on the other hand, shared writing is enabled, any data you inserted in the shared tables will remain. Only data owned by the repeated transfer will be deleted and replaced.

8.2.8 Transfer Failure and Storage Usage

Transfers to existing databases, unlike transfers to new databases, do not use batch update as a transaction mode. If a batch-update transaction fails for any reason, it cannot be rolled back. This is because batch-update transactions do not record data changes to a table in a database recovery file (for Rdb this is called a recovery-unit journal file). Failed batch-update transactions result in corrupt databases. Because transfers to existing databases do not use batch update, a transfer to an existing database requires more disk storage than a transfer to a new database. This extra disk storage is only needed temporarily during the execution of a transaction on a target table.

8.2.9 Database Consistency

An extraction (or initial replication) transfer to an existing target database requires several transaction steps. For example, data written into table A is committed in one transaction and data written into table B is committed in a second transaction. If your application uses tables A and B, it can see an inconsistent view of the data if it makes a query after table A's data has been written but before table B's data has been written. For example, consider a banking application where table A contains deposits and table B contains withdrawals. A transfer of funds from one account to another should show up as a deposit in one and a withdrawal in the other.

This inconsistent state does not occur during replication update transfers. Replication updates operate differently than initialization transfers. They mirror transactions that are performed on the source databases, and therefore can alter several tables within the same transaction.

Applications that are sensitive to this type of inter-table dependency should delay execution until the extraction or initialization transfer has completed. For this reason, Replication Option maintains information in the target database to inform users and applications about a transfer's state of execution.

Each transfer to a target database has one row in the RDB$VINTAGE or DDAL$TRANSFER_INFO table. RDB$VINTAGE is used for Rdb target databases and DDAL$TRANSFER_INFO is used for all others. When the copy process begins a transfer, it sets the RDBVMS$TRANSFER_BUSY or DDAL$TRANSFER_BUSY column to the value T, meaning TRUE. When the copy process completes execution, it resets that column value to F, meaning FALSE. You can read these columns to determine whether or not a transfer is executing.

If an extraction transfer fails during execution, such as when a network link disappears, the busy flag remains set. It is not reset until the next successful transfer execution.

Replication transfer updates behave differently. They mirror transactions that occurred in the source database. They do not compromise target database consistency. Therefore, replication updates do not set the busy flag. If a replication transfer update fails because the network link disappears, its transaction is rolled back automatically, so database consistency is maintained.

Note

It is very important to avoid an inconsistent database state. If you fail to ensure against database inconsistency, you can have an inconsistent database state and not realize it. To ensure that a transfer does not disrupt Rdb databases, you can reserve all the tables required by the transaction ahead of time by using the SET TRANSACTION statement. If you reserve tables for PROTECTED READ, a Replication Option transfer stalls until it is permitted to continue. In this case, transfer execution time is dependent on what applications are using the target database at the time.

A second way to ensure that a transfer does not disrupt a database is to schedule transfers to execute at a time when target applications are not running.

8.3 Restrictions on Using Transfers to Existing Databases

Section 8.2.5 explained rules for using transfers to existing databases when the shared-write option is enabled. Some of these rules can be considered restrictions on what you can and cannot do. The following are restrictions that apply to transfers to existing databases, whether shared writing is enabled or disabled.

- You cannot define an extraction rollup transfer into an existing database. If you need to merge disparate tables into one physical database, you can do so by either of the following methods:
  - First, define several transfers into that existing database. If you also require the union of data from several like tables, which currently exist in separate databases, you can do the same and then use a view to accomplish the union. Refer to Section 8.2.4.
  - Multiple transfers can write to the same target table if you enable shared writing. The result is the same as if you had performed an extraction rollup transfer.
- You cannot join source database and target database data.

  If a source and existing target database have identically defined tables, you cannot transfer data from the source table into the target table without renaming the target table. You can create a view to achieve a union of data in the two target tables.
- You cannot transfer multischema attributes into an existing database.

  When you transfer data to an existing multischema Rdb database, the tables automatically get placed in the default catalog and schema.

  After a transfer has completed, or as part of an epilogue procedure, you can alter the target database to add multischema attributes. All the tables and views that Replication Option transferred then become part of the default catalog and schema. However, you will not be able to move already-existing tables from the default catalog and schema to a catalog and schema you specify.
- You cannot transfer views based on renamed tables.

  When you name a view in the MOVE VIEWS clause of a transfer definition, the view is re-created in the target database as a view. Some types of transfers enable you to name a view in the MOVE TABLES clause, in which case the view is re-created as a table, with data of its own, in the target database.

  When transferring a view as a view, any tables or views on which this view is based must not be given different names in the target database. Replication Option does not change a view definition in the target database so that it can reference the new names of the tables and views on which it depends. To avoid this restriction, do not name the view in the transfer definition. Instead, use an epilogue procedure with the transfer and create the view from within that procedure, making sure to use the target names for the relevant tables.

- Extraction transfers after changing the data type of a column based on a domain may cause errors

  A problem may occur when you attempt an extraction transfer to an existing Rdb database if you have changed the data type of a column based on a domain. If you execute a transfer that puts the original column into the target database and then change the data type in such a way that the data stored in the column is out of range for that data type, the transfer will fail when you re-execute the transfer. For example, if you change the data in a DOUBLE PRECISION column to a data type REAL, you might receive the "%SYSTEM-F-FLTOVFLOW" error message.

  This error occurs because Rdb does not check the stored values when you alter the definition of a domain or column, but checks the stored values only when you retrieve them. Replication Option recognizes that the domain definition has changed and alters the definition in the target database. When performing an extraction transfer to an existing database, all the data in the table is erased in preparation for retransmitting the data. When Rdb erases data, it first attempts to retrieve it and convert it to the latest format. Rdb must read the data before deleting it in case the table has constraints or triggers that need to be evaluated.

9

SQL Statements for Replication Option

This chapter describes the SQL statements defined for Replication Option. SQL statements can create, alter, and delete transfers, manage transfers, and monitor transfer operation.

SQL statements for Replication Option are arranged into three categories:

- Definition

  ALTER TRANSFER
  ALTER SCHEDULE
  CREATE TRANSFER
  CREATE SCHEDULE
  DROP TRANSFER
  DROP SCHEDULE
  GRANT
  REVOKE
  SET LOGICAL_NAME TRANSLATION FOR TRANSFER

- Operation

  REINITIALIZE TRANSFER
  START TRANSFER
  STOP TRANSFER

- Information

  HELP
  SHOW TRANSFER

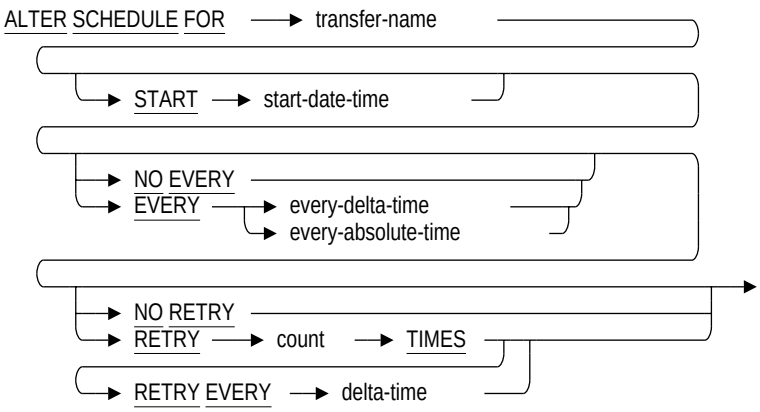
To issue any of these Replication Option statements, excluding the HELP statement, you must have the same user identification code (UIC) as the user who created the transfer, or have the appropriate transfer privilege. Table 9–1 in Section 9.7 provides a summary of the transfer privileges needed to execute the Replication Option statements against a transfer you did not create.

If you have OpenVMS BYPASS privileges, you can execute any of the SQL statements related to Replication Option. This means that you could execute and manage any transfer definition, including those that you did not author. However, you are not required to have BYPASS privilege in order to use Replication Option. You are automatically given the right to manage and execute any transfer that you create. If you want to manage a transfer that you did not create, you must be granted the necessary rights. See Section 9.7 for more information about granting rights.

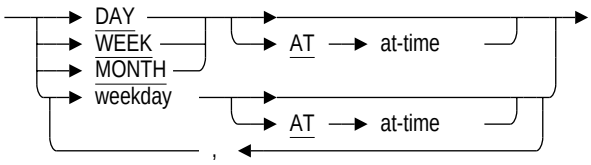
9.1 ALTER SCHEDULE Statement

Alters 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.

Format



every-absolute-time =



Arguments

transfer-name
Names the transfer for which you want to alter the schedule. This parameter can be entered in lowercase or uppercase. It is treated as uppercase by Replication Option.

9.1 ALTER SCHEDULE Statement

start-date-time

Specifies the time to execute the first transfer under the new schedule. You can specify a delta time or an absolute time.[1] If you use a delta time, Replication Option uses the current time and the delta-time value to calculate the actual transfer time. If you supply an absolute time, the first transfer under the new schedule is executed at the time specified. The START clause is optional. If you do not include a START clause, the start time is set to the start time specified in the original transfer definition.

The start-date-time parameter uses the following delta-time and absolute-time formats.

Delta time

dddd hh:mm:ss.cc

Absolute time

dd-mmm-yyyy hh:mm:ss.cc

In the delta time and absolute 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

Use 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 at the same time as specified in the start-date-time clause. If you omit the start-date-time clause, the transfer occurs at the time specified in the original transfer definition.

The valid formats for every-delta-time are shown in the preceding discussion of the start-date-time clause.

[1] See Usage Notes for information about changing default date-time formats.

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 was reached, or a fatal error occurred (one for which retries are not attempted).

every-absolute-time
Use 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 as specified in the original transfer definition.

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.

- MONTH

  The transfer occurs on a monthly cycle on the same day of the month as the initial transfer.

  If the initial transfer starts on the 31st of the month and the transfer schedule specifies EVERY MONTH, the next execution of the transfer is 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 the transfer should occur. If you specify DAY, WEEK, or MONTH, but you do not specify an at-time, Replication Option uses the time of day of the initial transfer. You can specify in a list of weekdays the at-time for each day. If you omit the at-time for any weekday, Replication Option selects as the default 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. The at-time argument has the following format:

hh:mm:ss.cc

In the at-time designation, the abbreviations have the following meanings:

hh	Hours
mm	Minutes
ss	Seconds
cc	Hundredths of a second

count
Specifies the maximum number of times Replication Option retries a transfer if a transfer execution fails. If you do not include a retry count, the retry count specified in the original transfer is used. If the count value is set to zero, Replication Option does not attempt to retry the transfer until the next scheduled transfer time, as indicated by the EVERY clause.

If your transfer definition is defined with a checkpoint clause that includes the RECOVER WITHIN option and transfer execution fails, recovery is attempted first. If recovery is successful, the transfer will complete the data transfer since the last checkpoint. If the recovery operation fails, then Replication Option executes the RETRY operation, if specified, in the schedule definition; however in this case, the transfer is restarted from the beginning.

For a list of recoverable errors, see Table C–6.

Note

Replication Option does not retry a transfer if a fatal error caused the transfer to fail. You can recover from a fatal error only by manual intervention.

delta-time
Specifies the length of time Replication Option 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, Replication Option 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 alter a transfer schedule, you must have the same user identification code (UIC) as the user who defined the transfer, or have the ALTER transfer privilege, or have the OpenVMS BYPASS privilege. If you try to alter a schedule for a transfer created by a user with a different UIC, and if you do not have the ALTER transfer privilege, or the OpenVMS BYPASS privilege, Replication Option ignores the schedule alteration, does not record it in the transfer database, and returns an error message. If you use the OpenVMS BYPASS privilege to create a schedule for a transfer associated with a different UIC, the UIC associated with the transfer does not change.
- You must execute the ALTER SCHEDULE statement outside the scope of a transaction. If you issue this statement when a transaction is active, SQL returns an error message.
- If a transfer is scheduled, you must execute the STOP TRANSFER statement before executing the ALTER SCHEDULE statement. In other words, the transfer must be in the suspended state.
- You can change the default date-time format that was established before OpenVMS V5.0 to a date-time format that complies with the LIB$ international date and time formatting routines of OpenVMS V5.0 and higher. For example, you might want to change the input format from 1-APR-1995 to one that allows entry of dates in the form April 1, 1995. You can change the output format as well.

  To change the input date-time format, define the LIB$DT_INPUT_FORMAT logical name, as discussed in the OpenVMS Run-Time Library documentation. To change the output date-time format, define the logical name LIB$DT_FORMAT. If you change the date-time format from the one used prior to OpenVMS V5.0, then you must place the date and time

in quotation marks. If you continue to use the date-time format in effect before OpenVMS V5.0, you can enter the date and time with or without quotation marks, as you wish.

For more information about the date-time routines provided by the OpenVMS Run-Time Library, see the OpenVMS Run-Time Library documentation. For details specific to the use of international date-time formats in SQL, see the Oracle Rdb SQL Reference Manual for the version of Oracle Rdb you are using.

- When you define a one-time-only transfer schedule and specify a transfer time that already passed, issuing a START TRANSFER statement changes the transfer status from suspended to scheduled. However, when you issue a SHOW TRANSFER STATUS statement, the "next transfer to be executed" phrase is not included. To avoid this problem and to execute the transfer, use the START TRANSFER NOW statement.
- Replication Option does not guarantee the re-execution of a transfer at the specified rate. If the copy process is tied up for any reason and cannot finish its task before the next transfer repetition cycle, the copy process ignores the repetition. The cycle is reset to the next expected transfer time.
- A replication transfer with a schedule that runs on an infrequent basis can consume a considerable amount of disk space in the source database. For example, if a replication transfer is scheduled to occur once a month, any new or changed rows to be transferred are duplicated in the RDB$CHANGES table in the source database and are retained for the remainder of the month until the schedule starts the transfer. This space is released only after the duplicate rows have been transferred and then dropped. Thus, for the entire month that the duplicate rows are stored, they take up disk space. In general, do not choose an infrequent transfer cycle if the number of daily changes to the source database is high.

  In addition, even if this monthly replication transfers only a single row, another problem can arise if other replication transfers use the same source database. Although those transfers might execute more frequently than once a month, their data rows are not necessarily dropped from the RDB$CHANGES table right away as Replication Option erases the rows only if rows entered earlier are not waiting to be transferred.

Examples

Example 1: Creating a schedule definition for a transfer.

The following example creates a schedule definition for the NH_EMPLOYEES transfer to start on August 15, 1995 at 1:30 AM (13:30) and run every day at 5:30 PM (17:30). It will retry 3 times with each retry occurring every 30 minutes. Then, the schedule is changed to run every day at 6:30 PM (18:30) and to retry 4 times with each retry occurring every 45 minutes.

```
SQL> CREATE SCHEDULE FOR NH_EMPLOYEES
cont> START 15-AUG-1995 13:30:00.00
cont> EVERY DAY AT 17:30:00.00
cont> RETRY 3 TIMES
cont> RETRY EVERY 0 00:30:00;
SQL>
SQL> ALTER SCHEDULE FOR NH_EMPLOYEES
cont> EVERY DAY AT 18:30:00.00
cont> RETRY 4 TIMES
cont> RETRY EVERY 0 00:45:00.00
cont> ;
```

Example 2: Altering a schedule using an absolute time with the date field omitted.

In this example, a schedule for a transfer is changed with the leading date field omitted. This transfer will start today at 1 PM (13:00).

```
SQL> ALTER SCHEDULE FOR NH_EVALUATION
cont>   START "-- 13:00"
cont>  ;
```

Example 3: Altering a schedule using a delta time.

In this example, a schedule for a transfer is changed using a delta time. This transfer will start 4 hours after the current time.

```
SQL> ALTER SCHEDULE FOR NH_EVALUATION
cont>   START 0 4:00
cont>  ;
```

9.2 ALTER TRANSFER Statement

Allows you to make the following changes to a transfer definition:

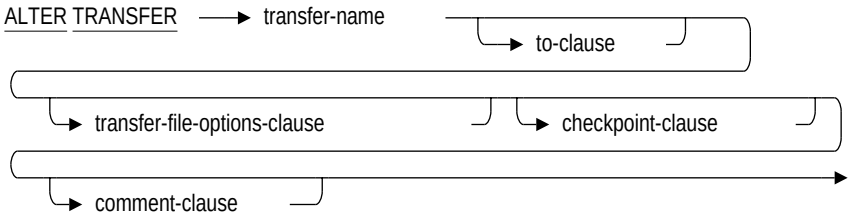
- Add, remove, or change a prologue file
- Add, remove, or change an epilogue file
- Add, remove, or change the log file
- Change the target file specification
- Add, remove, or change the checkpoint attribute
- Add or change the description of a transfer

Note that the ALTER TRANSFER statement does not allow you to:

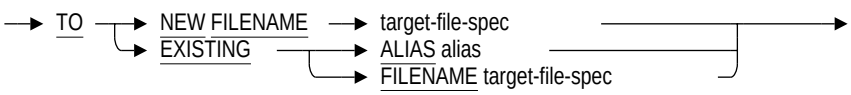
- Add or remove tables or columns from a transfer definition
- Alter the TYPE IS clause of a transfer definition
- Alter the move clause of a transfer definition
- Alter the target-db-params clause of a transfer definition
- Alter the target database from NEW to EXISTING or from EXISTING to NEW
- Alter the source database file specification

If you want to alter any of the attributes for which this statement does not support altering, you must drop the transfer and redefine it.

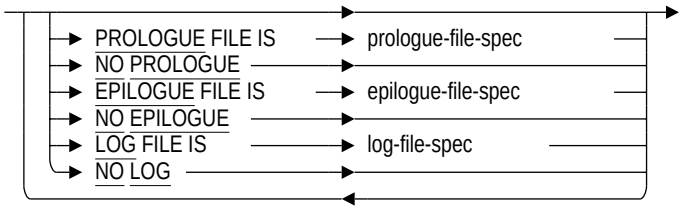
Format



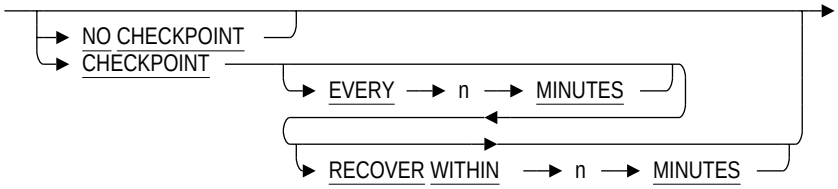
to-clause =



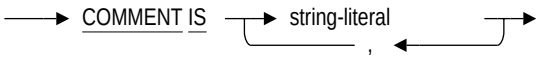
transfer-file-options-clause =



checkpoint-clause =



comment-clause =



Arguments

transfer-name
Specifies the transfer that you want Replication Option to alter. This parameter can be entered in lowercase or uppercase. It is treated as uppercase by Replication Option.

to-clause
Specifies the name of an existing target database or the name of a new target database to be created by Replication Option. You cannot change the database type (NEW or EXISTING) from that which was defined in the original transfer definition. That is, if the original transfer was to an existing database, you cannot alter it to transfer it to a new database.

TO NEW FILENAME target-file-spec
Declares that the transfer copies tables to a new, Replication Option-created target database.

TO EXISTING ALIAS alias
TO EXISTING FILENAME target-file-spec
Declares that the transfer copies tables to an existing target database.

You can identify the target database by its alias by using the ALIAS keyword if you have already issued an ATTACH statement. Alternatively, you can use the FILENAME keyword and the target-file-spec parameter.

transfer-file-options-clause
Specifies the DCL command procedures to be executed before or after a transfer runs, and the OpenVMS file where logging information related to the transfer execution is located. For each transfer definition, you can specify one prologue file, one epilogue file, and one log file, or any combination of the three. See Chapter 7 for information on using command procedures.

checkpoint-clause
Specifies whether or not the checkpoint option is used. If it is used, Replication Option will periodically commit changes made to the target database. Any recoverable error that occurs after such a commit point causes Replication Option to restart the transfer at the point where it last committed. This is particularly useful if your transfer normally takes a long time to complete.

The checkpoint clause allows you to specify how often changes should be committed to the database and the maximum length of time Replication Option has to recover from an error.

If you do not specify the checkpoint clause, the status of the checkpoint option remains as it was specified (or implied) in the original transfer definition.

comment-clause
Documents your transfer definition with one or more character string literals.

prologue-file-spec
Identifies the string literal that specifies the name of the prologue command procedure file to be run before the transfer executes. The prologue-file-spec may or may not require quotation marks, depending on the keywords used:

- If you use PROLOGUE or PROLOGUE FILE, enclose the prologue file specification in quotation marks.
- If you use PROLOGUE FILE IS or PROLOGUE IS, you can omit the quotation marks.

This file runs at the beginning of each execution of the transfer. The prologue file is a DCL command file. You can include a remote node name.

The default file type is .COM. The default directory for the prologue file is SYS$LOGIN.

If your file specification includes a node name and you are using DECnet/OSI, refer to the Usage Notes in Section 9.4 for information about node names in file specifications.

epilogue-file-spec
Identifies the string literal that specifies the name of the epilogue command procedure file to run after the transfer executes. The epilogue-file-spec may or may not require quotation marks, depending on the keywords used:

- If you use EPILOGUE or EPILOGUE FILE, enclose the epilogue file specification in quotation marks.
- If you use EPILOGUE FILE IS or EPILOGUE IS, you can omit the quotation marks.

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 remote node name.

The default file type is .COM. The default directory for the epilogue file is SYS$LOGIN.

If your file specification includes a node name and you are using DECnet/OSI, see the Usage Notes for information about node names in file specifications.

9.2 ALTER TRANSFER Statement

log-file-spec
Identifies the string literal that specifies the OpenVMS or Digital UNIX file where the Replication Option copy process writes any logging information related to the execution of the transfer. The log file spec may or may not require quotation marks, depending on the keywords used:

- If you use LOG or LOG FILE, enclose the log file specification in quotation marks.
- If you use LOG FILE IS or LOG IS, you can omit the quotation marks.

If you include a version number or a node name in the file specification, Replication Option returns an error. Replication Option creates a new log file every time a transfer executes. By default, the file specification is resolved and stored in the transfer database in its translated form when the transfer is defined.

The default file type is .LOG. The default directory for the log file is SYS$LOGIN.

NO CHECKPOINT
Specifies that the checkpoint option should be disabled.

CHECKPOINT
Specifies that the checkpoint option should be enabled.

EVERY n MINUTES
Specifies how often a checkpoint occurs (how often changes are committed to the target database). The number of minutes (represented in the syntax as *n*), must be in the range of 1 to 60 minutes. If the EVERY clause is omitted, checkpoints occur every 30 minutes.

Note

When executing a transfer, Replication Option ignores this attribute. Replication Option does not checkpoint at regular time intervals. Instead, Replication Option checkpoints after transfer or deletion of all data in each table and after creation of each table index for replication transfers.

RECOVER WITHIN n MINUTES
Specifies how long you want Replication Option to try to recover from a recoverable error. The number of minutes (represented in the syntax as *n*), must be between 1 and 60 minutes. During the recovery period, Replication

Option might attempt to restart from the last checkpoint several times. If Replication Option fails to recover from a failure after the specified period of time, the transfer is terminated and Replication Option's copy process aborts execution. After that, transfer retry attempts are determined by the transfer schedule and its RETRY EVERY clause. In this case, however, retrying the transfer means starting it over from the very beginning.

Recovery means the following steps will be performed:

1. If a transaction fails while data is being sent to a target table, Replication Option's copy process will roll back the target transaction and disconnect from the target database if the network connection to the target system still exists.
2. The copy process then hibernates for one minute and then attempts to reattach to the target database and restart the transaction.
3. If the reattach and restart occurs successfully, recovery is said to have taken place. If the reattach and restart are not successful, the cycle is repeated until either recovery is successful or until the recovery time limit (as indicated by the RECOVER WITHIN clause) has expired.

Successful recovery does not mean that the transfer has progressed any farther. For example, in the case of an intermittent network failure that allows recovery to occur (the transaction restarts within the recovery period) and yet the transaction fails due to another network interruption before the transaction can complete. To guard against this sort of behavior causing a transfer to execute indefinitely, Replication Option will only allow recovery to occur 10 times for a given transaction. After that, the transfer is terminated and the retry mechanism then reverts to that defined by the transfer's schedule.

string-literal
Specifies one or more character string literals. SQL recognizes two types of string literals: quoted character string and hexadecimal character string. A quoted character string literal is a string of printable characters enclosed in single quotation marks. A hexadecimal character string literal begins with an *X* followed by a string of up to 16 characters enclosed in single quotation marks. For more information about string literals in SQL, see the Oracle Rdb SQL Reference Manual for the version or Oracle Rdb you are using.

Usage Notes

- To alter a transfer, you must have the same UIC as the person who defined the transfer or have the ALTER transfer privilege or the OpenVMS BYPASS privilege.
- You must execute the ALTER TRANSFER statement outside the context of a transaction. If you issue this statement when a transaction is active, SQL returns an error message.
- The transfer being altered must be in the stopped state when the ALTER TRANSFER statement is executed. (That is, you must issue a STOP TRANSFER statement first.)
- The CHECKPOINT option in the CREATE TRANSFER statement has the following characteristics and limitations:
  - The checkpoint option is available when you are executing an extraction, extraction rollup, or initial replication transfer.
  - The checkpoint option is ignored by Replication Option when it is performing batch-update transfers or replication update operations. (Replication update transfers, by default, recover from errors without restarting the transfer execution from the start.)
  - The checkpoint option is not effective if an error occurs reading the source database.

    If an error occurs while Replication Option is accessing the source database, there is no mechanism for Replication Option to pick up where the error occurred. Restarting the transfer from the beginning is the only way to ensure consistent data.
  - When the checkpoint option is disabled and the transfer creates new target database files (the TO NEW FILENAME clause appears in the transfer definition), Replication Option uses batch-update transactions for the transfer operation. Batch update is the quickest mode to use for database loading, but it does not provide for recovery if an error occurs while writing to the database. When the checkpoint option is enabled, Replication Option uses read/write transactions, not batch-update. You must weigh the benefits of using the checkpoint option against the benefits of a fast-load operation and decide whether or not using the checkpoint option makes sense for your business needs.

- If the checkpoint option is used along with database key logging, the copy process log file might contain duplicate entries. This is because information about each change applied to the target database during a replication update transfer is written to the copy process log file. If an error occurs before those changes are committed to the target database and a checkpoint restart occurs, the same changes are reapplied in a new target transaction. However, the summary counts at the end of the update process will accurately reflect the number of inserts, updates, and deletes committed to the target database.

See the Usage Notes in Section 9.4 for additional information on the various issues pertaining to transfer definitions.

Examples

The following example shows the original SQL CREATE statement used to create a transfer, NH_EMPLOYEES. This is followed by the SQL ALTER statement that changes the target and log file specifications from OpenVMS file name strings to OpenVMS logical names, adds an epilogue file where one did not originally exist, enables the checkpoint option with default attributes, and changes the comment clause.

```
SQL> CREATE TRANSFER NH_EMPLOYEES TYPE IS EXTRACTION
cont>  MOVE TABLES
cont>     SELECT EMPLOYEE_ID,
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>  FROM EMPLOYEES WHERE STATE = 'NH'
cont>  TO NEW FILENAME DISK1:[TARGET]NH_EMP
cont>  LOG FILE IS NH_EMPLOYEES.LOG
cont>  COMMENT IS 'EMPLOYEES WHO LIVE IN NEW HAMPSHIRE'
cont>  ;

SQL> STOP TRANSFER NH_EMPLOYEES;
```

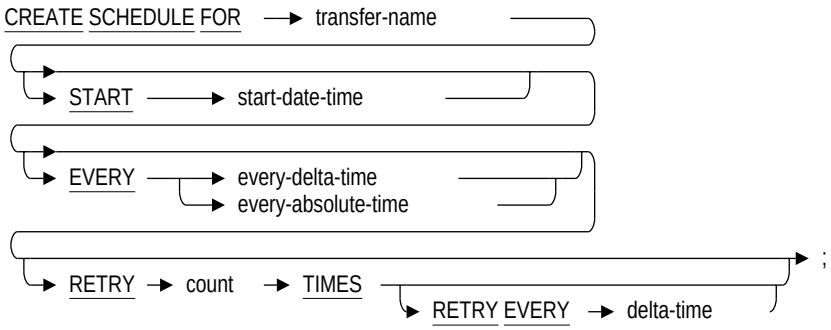
9.2 ALTER TRANSFER Statement

```
SQL> ALTER TRANSFER NH_EMPLOYEES
cont> TO NEW FILENAME  NH_EMP_TARGET
cont> EPILOGUE FILE IS NH_EMP_EPILOGUE
cont> LOG FILE IS      NH_EMP_LOG
cont> CHECKPOINT
cont> COMMENT IS 'LIST OF NEW HAMPSHIRE EMPLOYEES';
```

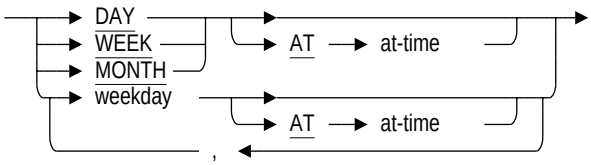
9.3 CREATE SCHEDULE Statement

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. Replication Option returns an error if you attempt to create a schedule for a transfer that already has one.

Format



every-absolute-time =



Arguments

transfer-name
Names the transfer for which this schedule is being defined. The transfer must already exist before a schedule definition can be created for it. This parameter can be entered in lowercase or uppercase. It is treated as uppercase by Replication Option.

start-date-time
Specifies the time to execute the initial transfer. You can specify a delta time or an absolute time. [1] If you use a delta time, Replication Option uses the

[1] See Usage Notes for information about changing default date-time formats.

current time and the delta-time value to calculate the actual transfer time. If you supply an absolute time, the initial transfer is executed at the time specified. The START clause is optional. If you do not include a START clause, the start time is set to the current time. The transfer then executes as soon as the transfer schedule definition is processed.

The start-date-time parameter uses the following delta-time and absolute-time formats.

Delta time

dddd hh:mm:ss.cc

Absolute time

dd-mmm-yyyy hh:mm:ss.cc

In the delta time and absolute 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

Use 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 formats for every-delta-time are shown in the preceding discussion of the start-date-time clause.

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
Use 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.

- MONTH

  The transfer occurs on a monthly cycle on the same day of the month as the initial transfer.

  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 the every-absolute-time parameter by specifying the time of day the transfer should occur. If you specify DAY, WEEK, or MONTH, but do not specify an at-time, Replication Option uses the time of day of the initial transfer. You can specify in a list of weekdays the at-time for each day. If you omit the at-time for any weekday, Replication Option selects as the default 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. The at-time argument has the following format:

At-time

hh:mm:ss.cc

In the at-time designation, the abbreviations have the following meanings:

hh	Hours
mm	Minutes
ss	Seconds
cc	Hundredths of a second

count
Specifies the maximum number of times Replication Option retries a transfer if a transfer execution fails. If you do not include a retry count or if the count value is set to zero, Replication Option does not attempt to retry the transfer until the next scheduled transfer time, as indicated by the EVERY clause. For a list of recoverable errors, see Table C–6.

Note

Replication Option does not retry a transfer if a fatal error caused the transfer to fail. You can recover from a fatal error only by manual intervention.

delta-time
Specifies the length of time Replication Option 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, Replication Option 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 create a transfer schedule, you must have the same user identification code (UIC) as the user who defined the transfer or have the ALTER access right or the OpenVMS BYPASS privilege. If you try to create a schedule for a transfer created by a user with a different UIC, and you do not have the ALTER access right or the BYPASS privilege, Replication Option ignores the schedule definition, does not record it in the transfer database, and returns an error message. If you have the OpenVMS BYPASS privilege and create a schedule for a transfer associated with a different UIC, the UIC associated with the transfer does not change.
- When you create a schedule for a transfer that exists in the unscheduled state, that transfer is placed in the scheduled state. If the transfer is in the suspended state and you define a schedule, the transfer remains in the suspended state. To place the suspended transfer in the scheduled state, use the START TRANSFER statement. When the transfer is executing, Replication Option places the transfer in the active state.
- You can modify a schedule definition using the ALTER SCHEDULE statement.
- You must execute the CREATE SCHEDULE statement outside the scope of a transaction. If you issue this statement when a transaction is active, SQL returns an error message.
- You can change the default date-time format that was established before OpenVMS Version 5.0 to a date-time format that complies with the LIB$ international date and time formatting routines of OpenVMS Version 5.0 and higher. For example, you might want to change the input format from 1-APR-1995 to one that allows entry of dates in the form April 1, 1995. You can change the output format as well.

  To change the input date-time format, define the LIB$DT_INPUT_ FORMAT logical name, as discussed in the OpenVMS Run-Time Library documentation. To change the output date-time format, define the logical name LIB$DT_FORMAT. If you change the date-time format from the one used prior to OpenVMS Version 5.0, then you must place the date and time in quotation marks. If you continue to use the date-time format in effect before OpenVMS Version 5.0, you can enter the date and time with or without quotation marks, as you wish.

For more information about the date-time routines provided by the OpenVMS Run-Time Library, refer to the OpenVMS Run-Time Library documentation. For details specific to the use of international date-time formats in SQL, refer to the Oracle Rdb SQL Reference Manual for the version of Oracle Rdb you are using.

- When you define a one-time-only transfer schedule and specify a transfer time that has already passed, issuing a START TRANSFER statement changes the transfer status from suspended to scheduled. However, when you issue a SHOW TRANSFER STATUS statement, the "next transfer to be executed" phrase is not included. To avoid this problem and to execute the transfer, use the START TRANSFER NOW statement.
- Replication Option does not guarantee the re-execution of a transfer at the specified rate. If the copy process is tied up for any reason and cannot finish its task before the next transfer repetition cycle, the copy process ignores the repetition. The cycle is reset to the next expected transfer time.

  Consider the following when determining how often to repeat a transfer:

  - Replication Option does not impose a minimum time interval for scheduling transfer repetition.
  - If a transfer is already running at the time the next scheduled execution is to take place, nothing happens. The current transfer is allowed to complete and the transfer is run again at the next scheduled execution time following the latest completion.
  - Tests have shown that Replication Option can re-execute transfers as often as every 20 seconds. This depends on hardware, network connections, storage capacity, system usage, type of target database, activity at the target site, and so on.
- A replication transfer with a schedule that runs on an infrequent basis can consume a considerable amount of disk space in the source database. For example, if a replication transfer is scheduled to occur once a month, any new or changed rows to be transferred are duplicated in the table RDB$CHANGES in the source database and are retained for the remainder of the month until the schedule starts the transfer. This space is released only after the duplicate rows have been transferred and then dropped. Therefore, for the entire month that the duplicate rows are stored, they take up disk space. In general, you should not choose an infrequent transfer cycle if the number of daily changes to the source database is high.

In addition, even if this monthly replication transfers only a single row, another problem can arise if other replication transfers use the same source database. Although those transfers might execute more frequently than once a month, their data rows are not necessarily dropped from the RDB$CHANGES table right away. Replication Option erases the rows only if no rows entered earlier are waiting to be transferred.

Examples

Example 1: Creating a schedule that specifies absolute time.

The following transfer schedule causes Replication Option to execute a transfer for the first time at 1:24 AM (01:24) on August 1, 1995. Subsequent transfers will occur every day thereafter at 5:30 PM (17:30). If a nonfatal error prevents a successful transfer, Replication Option tries every 30 minutes to execute the transfer until the transfer is successful or the maximum of 3 retries is reached. The clock for a retry begins at the end of the failed transfer execution.

```
SQL> CREATE SCHEDULE FOR NH_EMPLOYEES
cont>  ! Absolute time
cont>  START 1-AUG-1995 01:24:00.00
cont>  ! Absolute time; every day
cont>  EVERY DAY AT 17:30
cont>  RETRY 3 TIMES
cont>  RETRY EVERY 0 00:30:00
cont>  ;
```

Example 2: Creating a schedule that specifies delta time.

This example uses a delta time specification to allow 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 are 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 Replication Option to make up to five additional attempts to complete the transfer. RETRY EVERY 0 03:00:00 means Replication Option should retry the transfer 3 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 is 36 hours after the last failed attempt.

9.3 CREATE SCHEDULE Statement

```
SQL> CREATE SCHEDULE FOR PERS_EXAMP
cont>  START 15-AUG-1995
cont>  ! Delta-time: 1 day plus 12 hours
cont>  EVERY 1 12:00
cont>  RETRY 5 TIMES
cont>  ! 3 hours
cont>  RETRY EVERY 0 03:00:00
cont>  ;
```

Example 3: Creating a schedule to execute a transfer immediately.

The following schedule definition assumes that you want to execute the transfer immediately after you complete the schedule definition. The definition does not contain a START clause and, by default, the PERS_REC transfer executes immediately after you complete the CREATE SCHEDULE statement.

```
SQL> CREATE SCHEDULE FOR PERS_REC
cont>   EVERY MONTH
cont>   RETRY 3 TIMES
cont>   RETRY EVERY 0 00:30:00
cont>  ;
```

Example 4: Creating a schedule to run a transfer once a week.

This example calls for the COLLEGE_INFO transfer to execute at 3:00 PM (15:00). every Wednesday. The schedule definition calls for a maximum of 3 retries occurring at 30-minutes intervals.

```
SQL> CREATE SCHEDULE FOR COLLEGE_INFO
cont>  START 12-AUG-1995 15:00:00.00
cont>     ! Wednesdays at 3 p.m.
cont>     EVERY WEDNESDAY AT 15:00:00
cont>     RETRY 3 TIMES
cont>     RETRY EVERY 0 00:30:00
cont>  ;
```

Example 5: Creating a schedule that runs a transfer several times a week.

The following example causes an initial transfer to execute on August 15 at 5:00 PM (17:00), every Monday after the 15th at 5:00 PM (17:00), every Tuesday at 11:00 AM (11:00), and every Friday at 11:00 AM (11:00). Because a RETRY clause is not specified, the transfer does not execute until the next regularly scheduled time even if a nonfatal failure occurs.

```
SQL> CREATE SCHEDULE FOR PHD_SURVEY
cont>   START 15-AUG-1995 17:00
cont>   EVERY MONDAY, TUESDAY AT 11:00, FRIDAY AT 11:00
cont>  ;
```

Example 6: Creating a schedule to run a transfer on an as-needed basis.

In this example, a transfer starts at midnight on August 15. Because the transfer will be performed thereafter on an as-needed basis, the definer does not include an EVERY clause. The definer can replace the schedule definition if the transfer has to be run on a firm schedule at some later date.

```
SQL> CREATE SCHEDULE FOR NH_EVALUATION
cont>   START 15-AUG-1995
cont>  ;
```

Example 7: Creating a schedule using an absolute time with the date field omitted.

In this example, a schedule for a transfer is specified with the leading date field omitted. This transfer will start today at 1 PM (13:00).

```
SQL> CREATE SCHEDULE FOR NH_EVALUATION
cont>   START "-- 13:00"
cont>  ;
```

Example 8: Creating a schedule using a delta time.

In this example, a schedule for a transfer is specified with a delta time. This transfer will start 4 hours after the current time.

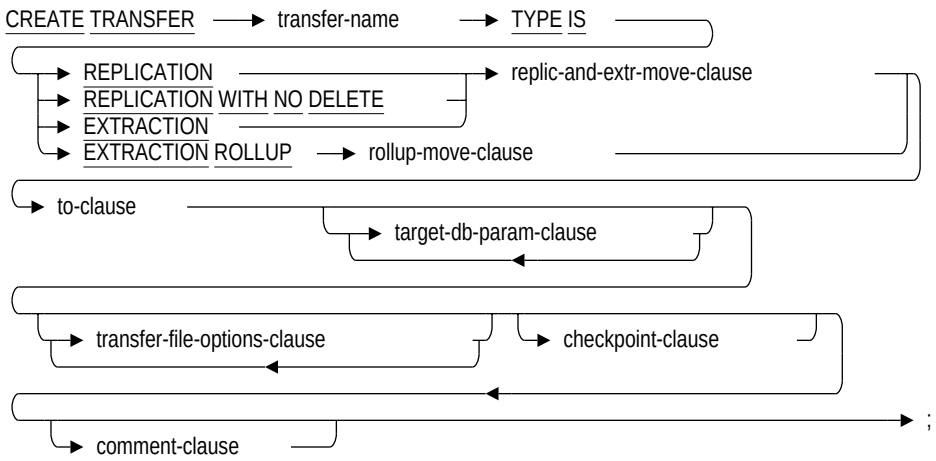
```
SQL> CREATE SCHEDULE FOR NH_EVALUATION
cont>   START 0 4:00
cont>  ;
```

9.4 CREATE TRANSFER Statement

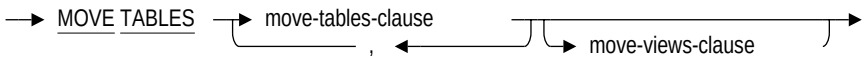
Creates a transfer definition and stores it in the transfer database. A transfer definition specifies the sources of data definitions and data, the target database to receive this information, selection of which tables, views, columns and rows to transfer, method of transfer execution, and prologue, epilogue and log files.

See Table 2–3 for the source database types and Table 2–4 for the target database types.

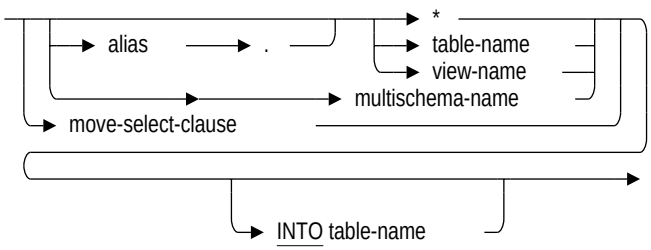
Format



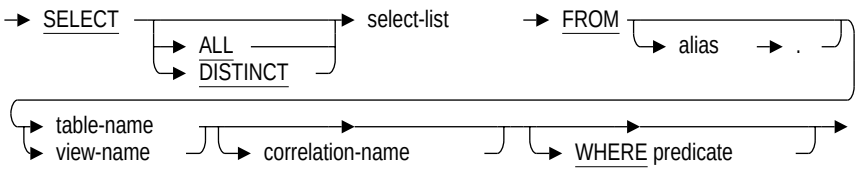
replic-and-extr-move-clause =



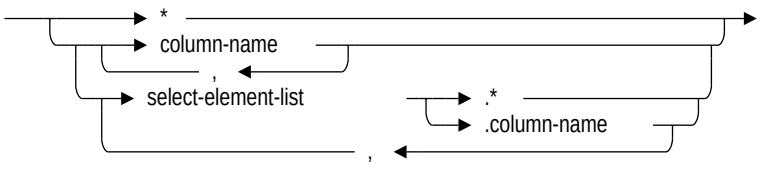
move-tables-clause =



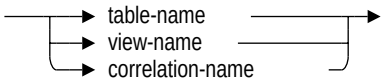
move-select-clause =



select-list =

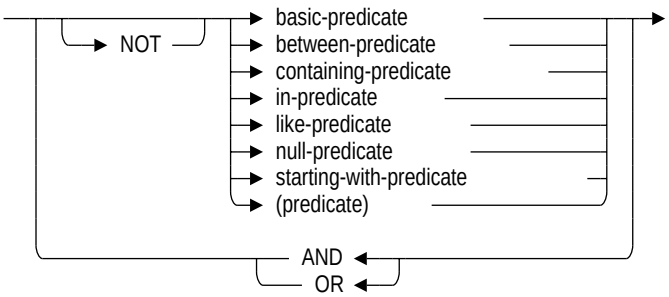


select-element-list =

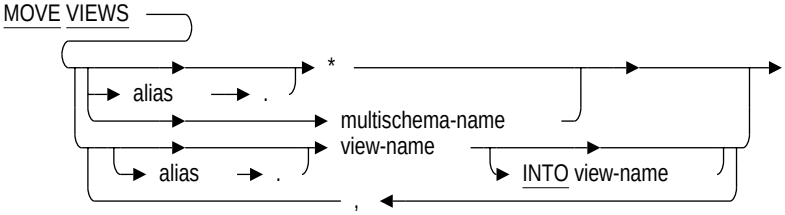


9.4 CREATE TRANSFER Statement

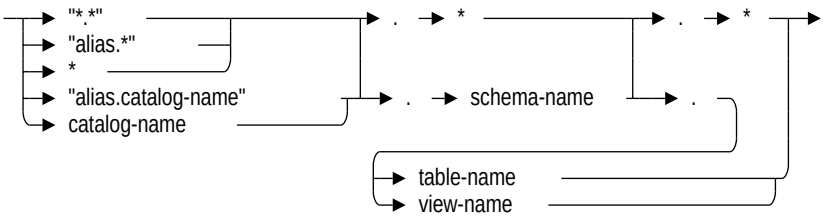
predicate =



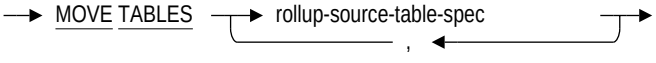
move-views-clause =



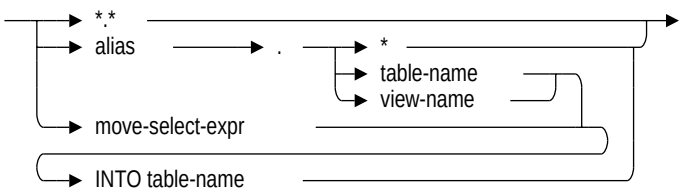
multischema-name =



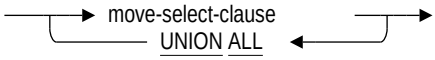
rollup-move-clause =



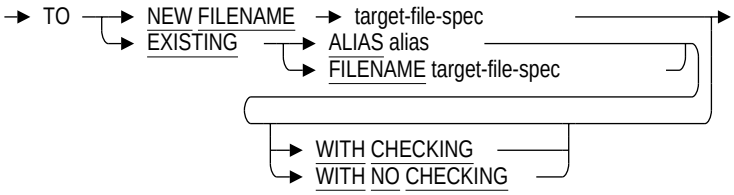
rollup-source-table-spec =



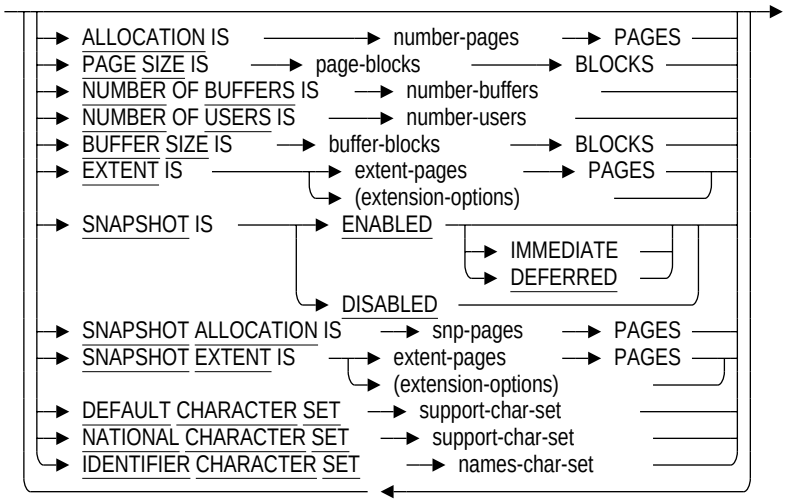
move-select-expr =



to-clause =

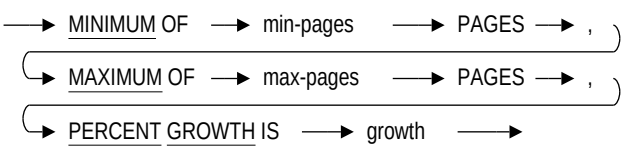


target-db-param-clause =

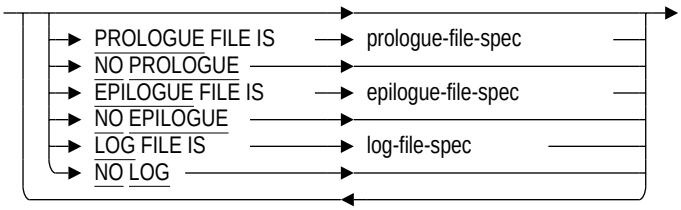


9.4 CREATE TRANSFER Statement

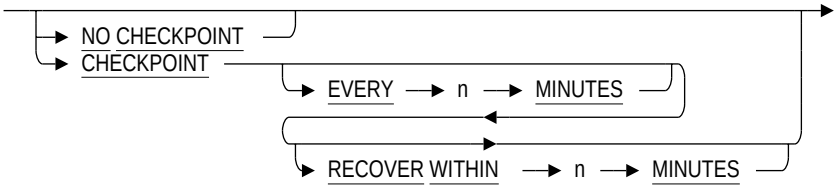
extension-options =



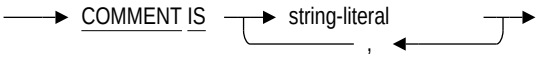
transfer-file-options-clause =



checkpoint-clause =



comment-clause =



Arguments

transfer-name
Identifies the transfer by name.

A valid transfer name has the following characteristics:

- Has a length from 1 to 31 bytes (octets).
- Can be formed using any of the set of ASCII characters:

```
A B C . . .  Z 0 1 2 . . .  9 _ $ a b c . . .  z
```

Lowercase letters may be used, but are treated as if they were uppercase letters.

- Cannot include letters with diacritical marks (which are used in European languages, for instance).
- The first character of the name must be either a letter or the dollar sign ($). However, Oracle Corporation recommends that you do not use dollar signs in a transfer name.

If you use an SQL keyword as a transfer name (or the name of any object specified in a transfer definition), Oracle Corporation recommends that you delimit the name with quotation marks to differentiate the name from a keyword. Not doing so can cause unexpected results.

TYPE IS

Names the type of transfer for the transfer definition. REPLICATION, EXTRACTION, and EXTRACTION ROLLUP are the three valid transfer types for Replication Option.

REPLICATION replic-and-extr-move-clause

Declares the transfer type to be a REPLICATION transfer. When a replication transfer executes, Replication Option creates the tables 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 database. Subsequent transfers involve only those additions, modifications, and deletions that have been made to the selected rows in the source database. If the target and source databases become inconsistent with each other, you can use the REINITIALIZE TRANSFER statement and then the START TRANSFER statement to create new versions of the target tables that are consistent with the source tables.

In addition, the execution of a replication transfer generally creates special-purpose tables in the source and target databases to handle updating. Appendix A contains specific information about these special database entities. Because the source database is modified with the addition of these system tables, the transfer definer's account needs the CREATE access right to the source database itself and the SELECT access right to attach to the database.

Tables in a target database that are created and maintained by a replication transfer should normally be treated as read-only. Exceptions to this are when shared writing is enabled. See the discussion of shared writing in the description of the move-tables-clause. Applications should not attempt to insert, update, or delete values in rows of these tables maintained by Replication Option. In some situations, it is possible for tables replicated by Replication Option to coexist with local, user-created tables. For more information on read and write restrictions, see the description of the to-clause.

9.4 CREATE TRANSFER Statement

Although data tables maintained by Replication Option should not be altered by user applications, you can enhance the target database by creating views and indexes, and by adding access rights. See Section 6.8 for more information about modifying the database definitions of a replication database.

There are a number of objects in a database that cannot be named directly in a transfer definition. These are indexes, constraints, triggers, query outlines, stored procedures and access privileges. When Replication Option executes a transfer, it does not attempt to re-create in the target database such items that might appear in a source database. Once Replication Option has created the target tables and transferred the data, you can add such objects to the target database yourself. You can create constraints, if necessary, between the tables maintained by Replication Option and local, user-created tables in a transfer to an existing database. Creating constraints in a target database that uses only tables created by Replication Option is allowed but serves little purpose because the tables should be considered read-only. You can use an epilogue command procedure to do these tasks automatically. See Chapter 7 for information describing Replication Option command procedures and their use in streamlining data distributing operations. Also refer to Section 5.2 for information about replication databases.

REPLICATION WITH NO DELETE replic-and-extr-move-clause
Declares the transfer type to be a specialized case of the REPLICATION transfer. A REPLICATION WITH NO DELETE transfer has all the same qualities as a REPLICATION transfer with the exception that rows deleted in the source database are not deleted in the target database. Instead, those rows that are deleted from the source database have their DDAL$DBKEY column set to NULL in the target database.

This option can be useful if, for example, you want to archive data; over time, your target database will hold all the data currently in the source database, plus the data that has been deleted from the source database.

Note however, that if large amounts of data are deleted from the source database over time, the target database becomes larger than the original source database. Therefore, you will probably want to move rows whose DDAL$DBKEY columns have been set to NULL (that are marked as having been deleted from the source database) to some offline storage media to prevent the database from growing beyond available disk space.

EXTRACTION replic-and-extr-move-clause
Declares the transfer type to be an EXTRACTION transfer. The tables copied to a target database by an extraction transfer maintain no ongoing relationship with the source database tables. Unlike a replication transfer, Replication Option severs the connection between the source database and the target

database after the extraction transfer completes. Each time an extraction transfer executes, Replication Option copies the latest version of the selected tables to the target database.

There are a number of objects in a database that cannot be named directly in a transfer definition. These are indexes, constraints, triggers, query outlines, stored procedures and access privileges. When Replication Option executes a transfer it does not attempt to re-create in the target database such items that might appear in a source database. Once Replication Option has created the target tables and transferred the data, you can add such objects to the target database yourself. You can also define constraints if you have transferred tables into an existing target database. Alternatively, you can use an epilogue command procedure to do these tasks automatically. See Chapter 7 for information on command procedures and extraction databases.

replic-and-extr-move-clause
Selects the rows of a table Replication Option transfers to the target database during an extraction or a replication transfer. The rows selected for replications are specified with a restricted form of a select clause. You must specify a move-tables-clause for each table moved to the target database.

For extraction transfers, when you specify a view in a MOVE TABLES clause, the rows selected by that view are created as a table in the target database. In contrast, for replication transfers, you cannot specify a view in a MOVE TABLES clause; you can only specify a view in a replication transfer with the move-views-clause. The transfer of views using the move-views-clause is also allowed for extraction transfers.

You must have SELECT privilege for the tables of an Rdb source database that you specify in a MOVE TABLES clause. For databases other than Rdb databases, such as Rdb Transparent Gateway to DB2, Replication Option does not check for SELECT privilege on tables and views. Therefore, although you may be able to create a transfer definition, it might fail on execution because you lack the required privileges.

move-tables-clause
For extraction transfers, specifies the table name or view name that you want copied from the source database. For replication transfers, specifies the table name (not view name) that you want copied from the source database. If you do not specify a table name or view name, you can specify a select clause.

When you write to tables created by Replication Option in a target database, the behavior of any future executions of an extraction transfer that copies the same tables from the source database depends upon whether or not the shared-write option is enabled. If shared writing is not enabled, only data

originating from the source database will appear in the target tables. Any data you might have inserted in the target tables using your own applications will be erased. If shared writing is enabled, any local (non-transferred) data in the target tables will remain intact. Refer to Section B.15 for more information on the shared-write option. For more information on read and write restrictions, see the description of the to-clause.

alias
Specifies a name for a particular attach to a database. In Replication Option, using an alias lets SQL statements refer to more than one database. An alias qualifies database definition names to help distinguish one database name from another.

*** (wildcard character)**
In a MOVE TABLES * clause, selects all rows and columns from all user tables, excluding views, that are in the source database at the time the transfer is created. If, after creating a transfer definition, you add a new table to the source database, that table is not transferred. To include the new table in the target database, you must delete and then create the transfer again. To move views, use the MOVE VIEWS clause or the MOVE TABLES clause without the wildcard (*) character.

table-name
Identifies the table that SQL uses to generate the result table. There are system tables and user tables (tables defined by the customer). Only user tables should be named in transfer definitions.

view-name
Identifies a view that is copied to the target database. There are system tables and user tables (tables defined by the customer). Only user tables should be named in transfer definitions.

multischema-name
Identifies a database object (table or view) or set of objects within an Rdb multischema source database. In a multischema database there is a hierarchical notation for naming objects, as shown:

```
"database-alias.catalog-name".schema-name.table-or-view-name.
```

The leftmost elements of this name hierarchy can be omitted. In such cases there is an implied use of the default database, catalog, and schema name, as required. Quotation marks are used only when the alias is included. To use the quotation marks, you must issue the statement SET ANSI QUOTING ON before entering the CREATE TRANSFER statement. Wildcard characters (*) are not permitted to the left of names. For example, *.EMPLOYEES is not

allowed. ADMINISTRATION.ACCOUNTING.* is a valid name. It refers to all tables in the ACCOUNTING schema of the ADMINISTRATION catalog of the default database.

schema-name
Identifies a schema within an Rdb multischema database.

catalog-name
Identifies a schema catalog within an Rdb multischema database.

move-select-clause
Identifies the rows from the source database that Replication Option transfers to the target database. The select clause specifies conditions that individual rows must satisfy to be included in the transfer to the target database. The move-select-clause is a restricted form of the standard SQL select clause.

SQL evaluates the clauses in a move-select-clause in the following order:

1. select-list
2. FROM
3. WHERE

After each of these clauses, SQL produces an intermediate result table that is used in evaluating the next clause.

SELECT ALL
Specifies that duplicate rows should not be eliminated from the result table. ALL is the default.

SELECT DISTINCT
Specifies that SQL should eliminate duplicate rows from the result table. The DISTINCT argument is allowed for EXTRACTION and EXTRACTION ROLLUP transfers only. If you specify DISTINCT for a replication transfer, Replication Option issues an error message.

select-list
Identifies a list of column names (to be derived from the table and views named in the FROM clause) for the final result table. You can use the asterisk (*) as a wildcard character in a select list, as described in the following arguments.

SELECT *
Specifies all column names from the intermediate result table (namely, all the columns in all the tables and views referred to in the FROM clause) at the time the transfer is created.

9.4 CREATE TRANSFER Statement

SELECT name.*
Specifies all columns in the table or view referred to by the table name, view name, or correlation name at the time the transfer is created. The name must be specified in the FROM clause of the select clause. You cannot mix this form of wildcard notation with the use of the asterisk as a wildcard character.

The implied order of the columns represents the order in which the columns were defined in the database.

FROM table-name
Identifies the table SQL uses to generate the result table. Replication Option does not let you name more than one table in a FROM clause.

You can specify a correlation name following a table name in the FROM clause to qualify column names in other parts of the select clause. If you do not explicitly specify a correlation name, SQL explicitly specifies the table name as a correlation name. Do not specify the same correlation name more than once, either explicitly or implicitly.

FROM view-name
Identifies the view SQL uses to generate the result table. Unlike other types of SQL select expressions, Replication Option does not allow you to name more than one view in a FROM clause.

You can specify a correlation name following a view name in the FROM clause to qualify column names in other parts of the select clause. If you do not explicitly specify a correlation name, SQL explicitly specifies the view name as a correlation name. Do not specify the same correlation name more than once, either explicitly or implicitly.

correlation name
Provides temporary names to tables to qualify ambiguous column names, just as an alias provides temporary names for schemas to qualify ambiguous table names.

WHERE predicate
Specifies a predicate that SQL evaluates to generate an intermediate result table. SQL evaluates the predicate for each row of the intermediate result table created by the FROM clause.

A predicate specifies conditions that must be true before a row is selected for transfer to the target database. SQL evaluates predicates to determine if they are true, false, or unknown. The predicate is also referred to as a conditional expression. Not all SQL predicates are supported by Replication Option because Replication Option requires that values be compared one at a time.

Replication Option does not support column select expressions (subqueries). Predicates that can be used with Replication Option include:

- Basic
- BETWEEN
- CONTAINING
- IN
- LIKE
- IS NULL
- STARTING WITH

For complete information on SQL predicates, see the Oracle Rdb SQL Reference Manual for the version of Oracle Rdb you are using.

INTO table-name

Specifies the name of a table in the target database, a name that can be different from the name of the source table.

For transfers to target databases other than Rdb, you must include the WITH NO CHECKING clause. The RDBVMS$TRANSFER_OWNER table does not exist in those databases.

If the transfer destination is an existing database and the table specified in the transfer definition already exists in the target database, SQL takes one of the following actions unless the WITH NO CHECKING clause is used:

- Checks the RDBVMS$TRANSFER_OWNER table, if it exists, to determine if the table was created by the current transfer. If so, the transfer definition is valid.
- If the table was not created by the current transfer, the transfer definition fails, and SQL issues an error message. In this case, you must use the INTO keyword with the table-name parameter to rename the duplicated table.

For tables owned by a given transfer, Replication Option's copy process checks the source and target table definitions to see if there are any changes in the source definition. If so, the copy process will alter the target definition to match that of the source but only for systems that allow such changes. At present, only Rdb allows table definitions to be altered; Distributed Option for Rdb and the Rdb Transparent Gateway products do not.

9.4 CREATE TRANSFER Statement

Changing a table definition in the target database may fail if other definitions exist that are dependent on that table, such as views, constraints, indexes, and triggers. You can resolve this problem by deleting such dependent definitions in the prologue procedure. If some dependent definitions are managed at the target database site independent of the transfer, you must coordinate data definition changes with the database manager at the target site.

move-views-clause
Selects the views Replication Option defines on the target database during an extraction or a replication transfer. All tables, views, and columns that the transferred views are based on must also be explicitly or implicitly named in the CREATE TRANSFER statement. The transfer definition fails if any underlying table, view, or column 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.

When views are transferred as views (move-views-clause) rather than as real tables (move-tables-clause), the following restrictions apply:

- Replication Option transfers those COMPUTED BY columns that depend only on the values of the table in which the columns are defined. Views that contain COMPUTED BY columns that violate this rule cannot be transferred. If such views are specified in the move-views-clause of a CREATE TRANSFER statement, the transfer definition fails. You need to define the transfer again, excluding the views that refer to such COMPUTED BY columns.
- Before a view in an Rdb database can be transferred, Replication Option requires that you have SELECT privilege for the view and for the underlying tables. If you do not have SELECT privilege for views you specify in the move-views-clause of a CREATE TRANSFER statement, the transfer fails.

  For databases other than Rdb databases, such as Rdb Transparent Gateway to DB2, Replication Option does not check for SELECT privilege for views. Therefore, although you might be able to create a transfer definition, it might fail on execution because you lack the required privileges.
- You can include only one move-views-clause in a CREATE TRANSFER statement. However, you can specify two or more views in the move-views-clause by separating each view name with a comma (,).

You can also move views with the move-tables-clause. However, when you use the move-tables-clause to move a view, the view becomes a table in the target database.

view-name
Specifies the name of the view transferred to the target.

INTO view-name
Specifies the name of a view in the target database, a name that can be different from the name of the source view.

For transfers to target databases other than Rdb, you must include the WITH NO CHECKING clause. The RDBVMS$TRANSFER_OWNER table does not exist in those databases.

If the transfer destination is an existing database and the view specified in the transfer definition already exists in the target database, SQL takes one of the following actions unless the WITH NO CHECKING clause is used:

- Checks the RDBVMS$TRANSFER_OWNER table, if it exists, to determine if the view was created by the current transfer. If so, the transfer definition is valid.
- If the view was not created by the current transfer, the transfer definition fails and SQL issues an error message. In this case, you must use the INTO keyword with the view-name parameter to rename the duplicated view.

EXTRACTION ROLLUP rollup-move-clause
Specifies that Replication Option create an extraction rollup database at the target site by transferring selected tables and columns from one or more source databases into a single target database. An extraction rollup database is created at the target site, and Replication Option reads data from the source databases and stores the data in a target database.

Each time an extraction rollup transfer is executed, Replication Option creates a new version of the target database and transfers a complete copy of the selected rows to the target database. Upon completion of an extraction rollup transfer, Replication Option severs the connection between the source database or databases and the target database.

See the Usage Notes for information about extraction rollup characteristics and limitations.

There are a number of objects in a database that cannot be named directly in a transfer definition. These are indexes, constraints, triggers, query outlines, stored procedures, and access privileges. When Replication Option executes a transfer it does not attempt to re-create in the target database such items that might appear in a source database. Once Replication Option has created the target tables and transferred the data, you can add such objects to the target database yourself. You can choose to have an epilogue command procedure

on OpenVMS or an epilogue shell script on Digital UNIX for information about using Replication Option command procedures. See Chapter 5 for more information on extraction rollup transfers.

rollup-move-clause
Selects the tables or views Replication Option transfers to the target database from a number of source databases during an extraction rollup transfer. You can roll up multiple tables into a single target table by using a move select expression with the UNION ALL keywords. The rollup-move-clause parameter has the following requirements:

- Replication Option requires you to declare each source database with a different alias. You must then use these aliases in the rollup-move-clause.
- The rollup-move-clause must be used with the EXTRACTION ROLLUP parameter.
- The rollup-source-table-spec clause that specifies *.* expands to a comma-separated list of alias.* for each declared database. The order of the list is undefined.
- The rollup-source-table-spec clause that includes alias.* expands to a comma-separated list of all the tables in the specified database. The order of the comma-separated list is undefined.
- All move-select-clause(s) that include the UNION keyword must also include the ALL keyword, must be of the same degree, and must match data types exactly for all columns being merged.

rollup-source-table-spec
Identifies the tables or views from the source database and names explicitly (or implicitly) the one or more target tables into which the source data is copied.

alias
Specifies a name for a particular attach to a database. In Replication Option, using an alias lets SQL statements refer to more than one database. An alias qualifies database definition names to help distinguish one database name from another.

table-name
Specifies a table name in the source database that you want transferred to the target database. To qualify a table name, precede it with an alias and a period.

view-name
Specifies a name that designates a view. A view name can be qualified by an alias, and a view name itself can qualify a column name. To qualify a view name, precede it with an alias and a period. This view-name parameter is used to generate a result table at the target database.

INTO table-name
Specifies the name of the table in the target database into which Replication Option transfers data from one or more source databases. The table name for an INTO clause must be unique within a definition. Omitting one or more INTO clauses can have one of the following effects:

- If you omit one INTO clause, Replication Option implicitly supplies a table name that matches the source table name.
- If you omit two or more INTO clauses, Replication Option also implicitly supplies table names that match corresponding source table names. However, you must ensure that each source table name is unique. Otherwise, Replication Option returns an error. When the source table is a move-select-expr clause, Replication Option takes the name of the source table from the name of the table in the FROM clause of the first move-select-clause.

move-select-expr
Transfers one or more tables or views into one or more tables. A move select expression can contain move-select-clause(s) that reference different databases; however, each move-select-clause can only reference a single database.

A move-select-expr clause for Replication Option differs from a general SQL select expression. All move-select-expr clauses for Replication Option that include a UNION ALL clause must:

- Be of the same degree
- Match data types exactly for all columns being merged

  SQL does not perform data type conversions. If the wildcard (*) character is used to refer to all columns of the specified table in the move-select-expr, the implied order of columns represented by the wildcard character is the order the columns were defined on the source database.

move-select-clause
Refer to the description of the move-select-clause described under the replication-extr-move-clause.

9.4 CREATE TRANSFER Statement

UNION ALL
Merges the results of two select expressions. UNION ALL matches columns in tables by their position in the select clause.

to-clause
Specifies the name of an existing target database or the name of a new target database to be created by Replication Option.

TO NEW FILENAME target-file-spec
Declares that the transfer copy tables to new target database files created by Replication Option.

If you define a transfer that creates a new target database, that entire database should be considered read-only. It is possible for users to add their own local tables to the target database and write to them. It is also possible for users who have the requisite privileges to write to tables created by Replication Option.

Replication Option does not enforce the read-only restriction. However, if a transfer is re-executed such that a new target database is created, any user modifications made to the original target are not automatically included in the new target. Also note that if the target files are on a Digital UNIX system, you must first delete the target files before re-executing the transfer.

TO EXISTING ALIAS alias
TO EXISTING FILENAME target-file-spec
Declares that the transfer copy tables to an existing target database.

The read-only status of data transferred to an existing target database not created by Replication Option depends on whether or not the shared-write option is enabled. For more information on the shared-write option, see the move-tables-clause description and Section B.15.

If reinitialization or re-extraction becomes necessary, Replication Option does not create a new target database; only those tables specified in the transfer definition are copied to the existing target database. Tables in the existing target that are not owned by Replication Option are unaffected by reinitialization.

You can identify the target database by its alias using the ALIAS keyword if you have already issued an ATTACH statement. Alternatively, you can use the FILENAME keyword and the target-file-spec parameter.

Note

There are two forms to the CREATE TRANSFER statement that are not supported by Replication Option:

1. Extraction rollup to an existing database is not supported by Replication Option. Although the syntax of the CREATE TRANSFER statement indicates this is a valid choice, SQL produces an error message if you try to create such a transfer.
2. The target-db-param-clause cannot be used for transfers into existing databases. SQL issues an error message if your transfer definition includes the EXISTING keyword and the target-db-param-clause. Specify target database parameters only when you are creating a new target database.

target-file-spec

Used to either create or attach to the target database. Database attach strings begin with a TYPE qualifier, which names the database or gateway product to be accessed. Refer to the Distributed Option for Rdb documentation or Rdb Transparent Gateway documentation for information about forming a valid database attach string.

If you do not specify the node, device, or directory, Replication Option assumes the home directory of the user creating the transfer. If the file type is not .RDB, the transfer to Rdb fails during access to the target database. The default file type is .RDB.

When you specify the name of a remote node as part of the file specification, you can include an access control string for login purposes.

If you use DECnet/OSI, see the Usage Notes for information on node names in file specifications.

Replication Option makes no attempt to verify whether a database or file of the same name already exists in the target directory. When the transfer is to a new database and you issue a duplicate file specification, Replication Option creates a second file with the same name, but with a higher version number.

If the Rdb database system 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.

9.4 CREATE TRANSFER Statement

WITH CHECKING
WITH NO CHECKING

If you are transferring to a database other than an Rdb database or if you are transferring to an existing database with shared writing enabled, then specify WITH NO CHECKING.

This parameter specifies whether or not SQL performs tests that determine if your transfer definition violates any restrictions when you are transferring to an existing database. The default is WITH CHECKING.

If you specify WITH CHECKING when a transfer is defined (or altered), SQL performs the following checks on the target database:

1. Verifies that the target is an Rdb database. When the target of a transfer is a database other than Rdb, you must use the WITH NO CHECKING clause in your transfer definition.
2. Verifies that you have the CREATE privilege required to create tables in the target database.
3. Determines if any domain used by a transferred table in the transfer definition already exists in the target database.
   a. If the domain definition does not exist in the target database, the transfer creates it.
   b. If the domain is already defined in the target database, SQL checks the RDBVMS$TRANSFER_OWNER table, if it exists, to determine if the domain was created by the current transfer. The RDBVMS$TRANSFER_OWNER table exists only in Rdb databases. The equivalent table in other databases is DDAL$TRANSFER_OWNR.

      If the domain was created by the current transfer, the definition is valid.

      If the domain was created by another transfer, the domain can be used if the data type, size, and scale factor exactly match in the source and target databases. Display attributes, such as the DATATRIEVE clauses, are not compared. If the domains do not match, the transfer definition fails and SQL issues an error message.
4. Determines if a table or view specified in the transfer definition already exists in the target database.
   a. If the table or view definition does not exist in the target database, the transfer creates it.

b. If the table or view is already defined in the target database, SQL checks the RDBVMS$TRANSFER_OWNER table, if it exists, to determine if the table or view was created by the current transfer. The RDBVMS$TRANSFER_OWNER table exists only in Rdb databases. The equivalent table in other databases is DDAL$TRANSFER_OWNR.

   If the table or view was created by the current transfer, the transfer definition is valid.

   If the table or view was not created by this transfer, the transfer definition fails and SQL issues an error message.

If you specify TO EXISTING and WITH CHECKING, Replication Option provides additional validity tests to support mixed-case names for database objects. (Note that transfer names are not considered database objects. SQL automatically converts transfer names to uppercase text.)

Rdb supports object names containing lowercase characters. The identifier character set for databases other then Rdb databases is DEC_MCS.

SQL checks all the names of objects to be transferred to lowercase characters if the following conditions are true:

- The CREATE TRANSFER statement contains the clauses TO EXISTING and WITH CHECKING.
- The target database system is running Rdb Version 4.2 or earlier.
- SET ANSI QUOTING ON is in effect.

If any lowercase character is found in a name, SQL reports an error and the transfer definition is not created.

This validity test is also performed at transfer execution time by the copy process.

Passing the SQL validity tests is not a guarantee of success. For example, if two transfers are created concurrently and independently, and both transfer a table of the same name, each SQL image concludes that the transfer can create the table because it does not yet exist in the target database. At transfer execution time, one of the transfers creates the table in the target database. The other transfer will fail when it attempts to do the same. The copy process explicitly checks for this situation. This situation indicates the need for the administrators of the databases to coordinate efforts. You can avoid this particular problem by specifying two different table names for the target database in the two transfer definitions.

9.4 CREATE TRANSFER Statement

When a transfer is to an existing database using the WITH CHECKING default and that database is identified by the FILENAME clause, SQL implicitly performs an ATTACH statement to attach to the target database. At the end of the transfer definition, if SQL is implicitly attached to a database, it detaches from that database.

You may choose to specify WITH NO CHECKING if the target database is inaccessible at the time you create the transfer definition. Whether or not you enable the SQL checking mechanism, the copy process that executes the transfer performs the checks.

target-db-param-clause
Specifies values for parameters that Replication Option uses to create the target database. You can only use this clause to create a new Rdb database. This clause uses syntax similar to the SQL CREATE DATABASE statement. However, Replication Option does not support all the parameters that the SQL CREATE DATABASE statement does.

Do not use the target-db-param-clause in conjunction with the TO EXISTING clause. Replication Option uses the target-db-param-clause only when it creates a new target database.

Individual arguments within the target-db-param-clause take default values when the target-db-param-clause is omitted. See the following descriptions for information about default settings for arguments.

The target-db-param-clause allows you to specify the DEFAULT CHARACTER SET, NATIONAL CHARACTER SET, or IDENTIFIER CHARACTER SET. Use these clauses to explicitly declare the default, national, or identifier character set for target databases that are running under Rdb.

The CHARACTER SET clauses are optional. If any of these clauses is left unspecified, SQL determines the missing character set as follows:

1. If, in your SQL session, you executed a SET DIALECT MIA or SET DIALECT SQL92 statement prior to executing the CREATE TRANSFER statement, the character set values established by the SET DIALECT statement are used for the target database. For the values, see the Oracle Rdb SQL Reference Manual for the version of Oracle Rdb you are using.
2. Otherwise, the target database uses DEC_MCS for all of the character set values.

Note

You can explicitly name character sets if the target system runs Rdb Version 5.1 or higher.

number-pages
Specifies the number of database pages initially allocated to the target database. Rdb 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 rows, a large page allocation helps to prevent fragmented rows.

page-blocks
Specifies the size in blocks of each database page in the target directory. Page size is allocated in 512-byte blocks. The default is 2 blocks (1024 bytes). If your largest row has more than approximately 950 bytes, you can specify more blocks for each page to prevent fragmented rows.

number-buffers
Specifies the number of buffers Rdb allocates for each process using the target database. The value must be greater than 1 buffer. The default is 20 buffers.

number-users
Specifies 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 number of users is 1.

buffer-blocks
Specifies the number of blocks Rdb allocates for each buffer. The value must be greater than zero. If you do not specify this parameter, Rdb uses a buffer size that is three times the page size value.

ENABLED IMMEDIATE
Specifies that read/write transactions write previously updated copies of rows they modify to the snapshot file, regardless of whether or not 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.

9.4 CREATE TRANSFER Statement

ENABLED DEFERRED
Specifies that read/write transactions not write previously updated copies of rows 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. Usually performance improves when snapshot writing is disabled because Rdb does not need to perform write operations to the snapshot file during update operations 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.

snp-pages
Specifies 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 ALTER DATABASE statement to truncate the snapshot file. To truncate the file, specify an allocation size smaller than the current snapshot file size.

extent-pages
Specifies the number of pages of each target database extension. This parameter provides direct control over the extension of the target database.

Specifies a number of pages for simple control over the extension. The default extension is 100 pages.

extension-options
Uses the MINIMUM, MAXIMUM, and PERCENT GROWTH parameters of extension options instead of using extent pages for greater control, particularly for multivolume databases.

min-pages
Specifies the minimum number of pages of each target database extension. The default is 100 pages.

max-pages
Specifies the maximum number of pages of each target database extension. The default is 10,000 pages.

growth
Specifies the percent of growth for each target database extension. The default is 20 percent growth.

DEFAULT CHARACTER SET support-char-set
Specifies the database default character set for this database. For a list of allowable character set names, see the Oracle Rdb SQL Reference Manual for the version of Oracle Rdb you are using.

Any character set listed as a supported character set can be specified for the DEFAULT CHARACTER SET support-char-set.

NATIONAL CHARACTER SET support-char-set
Specifies the database national character set when you create a database. For a list of allowable national character set names, see the Oracle Rdb SQL Reference Manual for the version of Oracle Rdb you are using.

Any character set listed as a supported character set can be specified for NATIONAL CHARACTER SET support-char-set.

IDENTIFIER CHARACTER SET names-char-set
Specifies the identifier character set for user-supplied database object names, such as table names and column names. The character set must contain ASCII characters. The identifier character set is used by SQL for database object names. It must be able to store ASCII characters.

For a list of allowable character sets, see the Oracle Rdb SQL Reference Manual for the version of Oracle Rdb you are using.

transfer-file-options-clause
Specifies the DCL command procedures to be executed before or after a transfer runs and the file where logging information related to the transfer execution is located. For each transfer definition, you can specify one prologue file, one epilogue file, and one log file, or any combination of the three. See Chapter 7 for information on using with the CREATE TRANSFER statement.

prologue-file-spec
Identifies the string literal that specifies the name of the prologue command procedure file to be run before the transfer executes. The prologue-file-spec may or may not require quotation marks, depending on the keywords used:

- If you use PROLOGUE or PROLOGUE FILE, enclose the prologue file specification in quotation marks.
- If you use PROLOGUE FILE IS or PROLOGUE IS, you can omit the quotation marks.

This file runs at the beginning of the first execution of the transfer and of all subsequent transfers. The prologue file is an OpenVMS DCL command file. You can include a remote node name.

You can also use a logical name for all or a part of the file specification. By default, the file specification is resolved and stored in the transfer database in its translated form when the transfer is defined. This behavior can be affected by the statement SET LOGICAL_NAME TRANSLATION FOR TRANSFER ON/OFF.

The default for the prologue-file-spec clause is NO PROLOGUE. The default file type is .COM. The default directory for the prologue file is the home directory.

If your file specification includes a node name and you are using DECnet/OSI, see the Usage Notes for information about node names in file specifications.

epilogue-file-spec
Identifies the string literal that specifies the name of the epilogue command procedure file or shell script to run after the transfer executes. The epilogue-file-spec may or may not require quotation marks, depending on the keywords used:

- If you use EPILOGUE or EPILOGUE FILE, enclose the epilogue file specification in quotation marks.
- If you use EPILOGUE FILE IS or EPILOGUE IS, you can omit the quotation marks.

This file runs after the first execution of the transfer and after all subsequent transfers. The epilogue file is a DCL command file.

The default for the epilogue-file-spec clause is NO EPILOGUE. The default file type is .COM. The default directory for the epilogue file is SYS$LOGIN.

If your file specification includes a node name and you are using DECnet/OSI, see the Usage Notes for information about node names in file specifications.

log-file-spec
Identifies the string literal that specifies the file where the Replication Option copy process writes any logging information related to the execution of the transfer. The log-file-spec may or may not require quotation marks, depending on the keywords used:

- If you use LOG or LOG FILE, enclose the log file specification in quotation marks.
- If you use LOG FILE IS or LOG IS, you can omit the quotation marks.

The default for the log-file-spec clause is NO LOG. The default file type is .LOG. The default directory for the log file is SYS$LOGIN.

Creation of the log file for the transfer, the copy process log file, is performed by the transfer monitor. The transfer monitor might be run from a different account than the one used to create the transfer definition. Therefore, if the log file specification contains a logical name, be sure that the logical name definition is visible to the transfer monitor.

checkpoint-clause
Specifies whether the checkpoint option is used. If it is used, Replication Option will periodically commit changes made to the target database. Any recoverable error that occurs after such a commit point causes Replication Option to restart the transfer at the point where it last committed. This is particularly useful if your transfer normally takes a long time to complete.

The checkpoint-clause allows you to specify how often changes should be committed to the database and the maximum length of time Replication Option has to recover from an error.

The default is NO CHECKPOINT.

NO CHECKPOINT
Specifies that the checkpoint option should be disabled.

CHECKPOINT
Specifies that the checkpoint option should be enabled.

EVERY n MINUTES
Specifies how often a checkpoint will occur (how often changes will be committed to the target database). The number of minutes (represented in the syntax as *n*), must be in the range of 1 to 60. If the EVERY clause is omitted, checkpoints will occur every 30 minutes.

Note

When executing a transfer Replication Option ignores this attribute. It does not perform a checkpoint at regular time intervals. Instead, it performs checkpoints after transfer or deletion of all data in each table and after creation of each table index for replication transfers.

RECOVER WITHIN n MINUTES

Specifies how long you want Replication Option to try to recover from a recoverable error. The number of minutes (represented in the syntax as *n*), must be between 1 and 60. During the recovery period, Replication Option might attempt several restarts from the last checkpoint. If Replication Option fails to recover from a failure after the specified period of time, the transfer is terminated and Replication Option's copy process aborts execution. After that, transfer retry attempts are determined by the transfer schedule and its RETRY EVERY clause. In this case, however, retrying the transfer means starting it over from the very beginning.

Recovery means the following steps will be performed:

1. If a transaction fails while data is being sent to a target table, Replication Option's copy process will roll back the target transaction and disconnect from the target database if the network connection to the target system still exists.
2. The copy process then hibernates for 1 minute and then attempts to reattach to the target database and restart the transaction.
3. If the reattach and restart occurs successfully, recovery is said to have taken place. If the reattach and restart are not successful, the cycle is repeated until either recovery is successful or until the recovery time limit (as indicated by the RECOVER WITHIN clause) has expired.

Successful recovery does not mean that the transfer has progressed any farther. For example, in the case of an intermittent network failure that allows recovery to occur (the transaction restarts within the recovery period) and yet the transaction fails due to another network interruption before the transaction can complete. To guard against this sort of behavior causing a transfer to execute indefinitely, Replication Option will only allow recovery to occur 10 times for a given transaction. After that, the transfer is terminated and the retry mechanism then reverts to that defined by the transfer's schedule.

comment-clause

Documents your transfer definition with one or more character string literals.

string-literal

Specifies one or more character string literals. SQL recognizes two types: quoted character string and hexadecimal character string literals. A quoted character string literal is a string of printable characters enclosed in single quotation marks. A hexadecimal character string literal begins with an *X* followed by a string of up to 16 characters enclosed in single quotation marks. For more information about string literals in SQL, see the Oracle Rdb SQL Reference Manual for the version of Oracle Rdb you are using.

Usage Notes

- Archive transfers can only be performed when the target database is managed by one of the Oracle database managers, either Rdb or Oracle. If you attempt an archive transfer to a non-Oracle or non-Rdb database, an error is reported.
- After you issue the CREATE TRANSFER statement, the transfer does not execute until one of the following events occurs:
  - The time specified in the corresponding CREATE SCHEDULE statement arrives.
  - You issue a START TRANSFER NOW statement.
- You can modify a transfer definition using the ALTER TRANSFER statement.
- You must execute the CREATE TRANSFER statement outside the scope of a transaction. If you issue this statement when a transaction is active, SQL returns an error message.
- Rdb Version 5.1 and higher supports the use of date-time data types with arithmetic operations that allow you to make calculations based on the data. You can use the built-in functions, CAST and EXTRACT, in the select clause of the CREATE TRANSFER statement. See Examples 9 and 10 to see how these might be used.

  Refer to the Rdb Transparent Gateway documentation to determine if these date-time data types are supported for specific gateways. If you attempt to transfer such data to a system that does not support it, the transfer fails during execution.
- You must have SELECT privilege for the tables and views of an Rdb source database that you specify in a CREATE TRANSFER statement. Replication Option returns an error message when you specify tables and views in a CREATE TRANSFER statement for which you do not have SELECT privilege. If you specify databases other than Rdb, such as Rdb Transparent Gateway to DB2, Replication Option does not check for SELECT privilege for the tables and views. Therefore, although you might be able to create a transfer definition, it might fail on execution because you do not have the required privileges.
- You cannot use a Distributed Option for Rdb or Rdb Transparent Gateway database as a source database for a replication transfer. Only Rdb databases can be source databases for replication transfers.

- In Rdb multischema source databases, you can refer to catalogs, schemas, tables, columns, and views in a CREATE TRANSFER statement either implicitly or explicitly. When you include the MOVE TABLES * clause, MOVE VIEWS * clause, or wildcards in the select list, you refer to catalogs, schemas, tables, columns, or views implicitly. Replication Option uses the current definitions of the catalogs, schemas, tables, columns, and views in the source database and stores those names explicitly in the transfer definition. You refer to catalogs, schemas, tables, columns, and views explicitly when you include the MOVE TABLES or MOVE VIEWS clause and supply specific catalog, schema, table, column, or view names.

  Note that Replication Option does not duplicate the catalog and schema information in the target database. If the target database is a multischema database, the transferred tables are recorded in the default catalog and schema.

- Replication Option allows you to transfer views using any of the following three methods:

  – Using the move-views-clause

    Replication Option transfers the definition of the view when you specify MOVE VIEWS view-name or MOVE VIEWS *. You must also explicitly or implicitly name, in the CREATE TRANSFER statement, all the tables, views, or columns on which the transferred view definitions are based (as well as catalogs and schemas if the database is an Rdb multischema database). The transfer definition will fail if any underlying tables, views, or columns are not named. You cannot use a move-views-clause in extraction rollup transfers. Currently, you cannot transfer view definitions to transparent gateway databases because the gateways do not support the ability to create views.

  – Using the move-tables-clause

    If you use the move-tables-clause for extraction and extraction rollup transfers (but not replication transfers), Replication Option transfers a view by creating a table in the target database containing the view's rows. This method improves query performance in the target database. For example, you can achieve the effect of a join by transferring a view that joins different tables in the source database. This method can be used for all target databases, including gateway databases.

    For replication transfers, you cannot specify a view in a move-tables-clause; you can only specify views with the move-views-clause. The transfer of views using the move-views-clause for extraction and extraction rollup transfers is also allowed.

  - For an extraction or replication transfer, you can also transfer a view to a target database by using an epilogue procedure associated with the transfer.

  When you use the move-views-clause or an epilogue procedure to transfer a view to an existing database, Replication Option creates a view definition rather than a table definition. When a transfer to an existing database is repeated, view definitions cannot be altered. Instead, you must drop the view definition and then create a new one. Because views can be defined in terms of other views, you must drop them in the prologue procedure in the reverse order from which they are added to the epilogue. Replication Option's copy process does this for views named within the transfer definition, first examining the definitions to determine the dependency hierarchy.

- You specify the tables you want to move in the move-tables-clause and the views in the move-views-clause of the CREATE TRANSFER statement. When you are doing an extraction rollup transfer, you specify the tables you want to move in the rollup-move-clause of the CREATE TRANSFER statement.
- The same clause can occur only once in a CREATE TRANSFER statement. You cannot specify the same table or view more than once in the move clauses. If a table name or view name appears in more than one move clause, Replication Option returns an error.
- Replication Option issues an error message if you do not enter a move-tables-clause in a transfer definition.
- The CHECKPOINT option in the CREATE TRANSFER statement has the following characteristics and limitations:
  - The checkpoint option is available when you are executing an extraction, extraction rollup, or initial replication transfer.
  - The checkpoint option cannot be used with batch-update transfers or with replication-update operations.
  - The checkpoint option is not effective should an error occur in reading the source database.

    If an error occurs while Replication Option is accessing the source database, there is no mechanism for Replication Option to pick up where the error occurred. Restarting the transfer from the beginning is the only way to ensure consistent data.

  - When the checkpoint option is disabled and the transfer creates new target database files (the TO NEW FILENAME clause appears in the transfer definition), Replication Option uses batch-update transactions for the transfer operation. Batch update is the quickest mode to use for database loading, but it does not provide for recovery if an error occurs while writing to the database. When the checkpoint option is enabled, Replication Option uses read/write transactions, not batch-update. You must weigh the benefits of using the checkpoint option against the benefits of a fast-load operation and decide whether or not using the checkpoint option makes sense for your business needs.
  - If the checkpoint option is used along with database key logging, the copy process log file might contain duplicate entries. This is because information about each change applied to the target database during a replication update transfer is written to the copy process log file. If an error occurs before those changes are committed to the target database and a checkpoint restart occurs, the same changes are reapplied in a new target transaction. However, the summary counts at the end of the update process will accurately reflect the number of insert, update, and delete operations committed to the target database.

- The EXTRACTION ROLLUP option in the CREATE TRANSFER statement has the following characteristics and limitations:
  - You must refer to each source database in the CREATE TRANSFER statement by its alias.
  - On the target database, table names and column names in a table must be unique. Consequently, all target tables of the same name must have the same definition, and all target columns of the same name must have the same data type. If the definitions or data types differ, Replication Option issues an error message and the transfer definition fails.
  - If any database specified in the transfer definition is on a node that is not available, Replication Option places the transfer in a waiting-to-retry state. Replication Option tries the transfer again at the time specified for retry in the schedule definition.
  - If any of the source nodes specified in the transfer definition fails during the transfer, the transfer will also fail. Replication Option tries the transfer again at the time specified for retry in the schedule definition.

  - If the transfer fails due to a recoverable error on the target node and the checkpoint option is specified, the retry starts where the last checkpoint occurred. If the checkpoint option is not specified and a recoverable error occurs on the target node, the retry starts at the beginning, not where the transfer stopped.

- The rollup-move-clause has the following characteristics and limitations:
  - All columns in each move-select-clause must reside in the same table.
  - You cannot name the same table in more than one MOVE clause of a single transfer definition.
  - For each name, view name, or move-select-expr in the rollup-source-table-spec table of a rollup-move-clause that you enter, you can supply a target table specification in the form of an INTO clause. You can create one or more tables in the target database from several source database tables by renaming the target tables in the transfer definition. You cannot, however, move tables with the same name that are from different databases to the target database without specifying either a UNION ALL or an INTO clause that provides unique names for all recurring names.
- Consider a situation where you create a transfer using the TO EXISTING keywords and the target database you specify is also the target database of a second transfer that uses the TO NEW keywords.

  Each time the TO NEW transfer executes, it creates a new version of the target database unless the transfer is a replication update. In an OpenVMS system, which supports the existence of multiple file versions, the version number for the new target database is incremented by one. If the TO EXISTING transfer specifies a version number in the target-file-spec parameter, it always transfers to the database with that file version, even though newer database files are created by the TO NEW transfer. On the next execution of the TO NEW transfer, you will lose the data from the TO EXISTING transfer. If you omit the version number in a file specification, the transfer goes to the latest version.

  Be careful when replicating into an existing database if that database is itself the target of another transfer.

  Leaving off the file version number of the target database will not help.

  If the target database becomes superseded by a later file version, the TO EXISTING replication fails. This occurs because subsequent executions of a replication transfer, called replication updates, expect the data definitions and data from the prior transfers to exist in the target database. If it

is necessary to supersede the target database file, for example, by re-executing an extraction TO NEW transfer, you must reinitialize any replication transfers into that database.

- Distributed Option for Rdb and the Rdb Transparent Gateway support only one character set, DEC_MCS, within the OpenVMS environment. Rdb Versions 5.1 and later support several character sets (in support of the Multivendor Integration Architecture (MIA)) in addition to DEC_MCS. This includes support for the Kanji character set, as defined by JIS X0208-1983. For information about the supported character sets, see the Oracle Rdb SQL Reference Manual for the version of Oracle Rdb you are using.

  The following list provides information on the factors you need to take into account when creating transfers to or from databases that do not support the same group of character sets:

  – Transferring database objects with different character sets

    When Replication Option transfers to an existing database, it is possible to transfer database tables and other objects that use different identifier character sets than those used by the tables and objects already stored in the existing database.

    Replication Option does not prevent this mix of character sets. However, the results in the target databases will not make sense when they are displayed. For example, if you transfer data from a database that uses the DEC_KANJI character set to a target database with tables and objects that use the DEC_MCS character set, Replication Option will perform the transfer, but you will not be able to understand the data from the target database.

  – Using different database systems for sources and targets

    When you transfer a database and both source and target databases, using Rdb Version 5.1 or higher, Replication Option creates domains in the target database with the same character set subtypes as the domains in the source database.

    However, when a source or target database is a Distributed Option for Rdb database or a Rdb Transparent Gateway database, you may encounter unexpected behavior because the character set information either cannot be read from the source domain subtype or cannot be recorded in the target domain subtype. In either case, Replication Option creates text domains in the target database without specifying a subtype code.

If you are transferring data from a Distributed Option for Rdb database or a Rdb Transparent Gateway database, do one of the following, depending on the version of your target system:

* Target database is a Distributed Option for Rdb database or a Rdb Transparent Gateway database.

  The only character set supported by Distributed Option for Rdb databases and Rdb Transparent Gateway databases is DEC_MCS. Data of any other character set must be interpreted by your application program.

* Target database is Rdb Version 5.1 or higher, and the source database is a Distributed Option for Rdb database or a Rdb Transparent Gateway database.

  If the transfer is to an already-existing target database, the database default character set is already established. If the CREATE TRANSFER statement is to create the target database, you can specify the default character set for the target database.

  Under these circumstances, Replication Option creates domains in the target database without specifying the character set for the data in the domain. When a domain is created without passing a character set ID for the domain, Rdb (or Rdb/VMS) gives the domain the character set ID of the database default character set. If you require a different character set for the domain, change the domain attribute within an epilogue procedure associated with the transfer definition. For information on creating epilogue procedures, see Chapter 7.

- If a transfer fails during execution, the transfer will begin again at one of the three following times:
  - If the checkpoint option is enabled, the time specified in the checkpoint RECOVER WITHIN clause
  - If the checkpoint option is not enabled, the next scheduled retry time if the failure results from a single nonfatal error
  - The next scheduled transfer time if the failure results from a series of nonfatal errors and the number of retries is exhausted, or if no retries are specified in the schedule definition

  The next extraction, extraction rollup, or initial replication transfer (but not replication update) starts from the beginning to ensure consistency with any updates that might have been made to the source database. In replication updates, however, Replication Option does not start from the beginning. Instead, because Replication Option performs updates as a

series of transactions matching the transactions performed on the source database, Replication Option starts again from the failed transaction whenever a failure occurs during a replication update transfer.

- Replication Option transfers only definitions for the database, columns, and tables specified in the transfer definition. If the source database includes additional database entities such as indexes, constraints, or access rights, these are not transferred to the target database. These entities cannot be transferred (as tables and views are) by defining them in a transfer definition, because Replication Option does not provide such syntax for indexes, constraints, or access rights. 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. See Chapter 7 for details about creating and running such command procedures.
- When tables specified in the transfer definition are created in the target database, they are given whatever default rights are granted by the target database system. You can grant and revoke privileges in the target tables after they are created.
- In a transfer definition, you can use logical 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. How Replication Option treats these logical names is determined by the SET LOGICAL_NAME TRANSLATION FOR TRANSFER statement. By default, if you do not use the SET LOGICAL_NAME TRANSLATION FOR TRANSFER statement (or you use this statement and specify DEFAULT), Replication Option treats these file names as described in the following list. For more information on the SET LOGICAL_NAME TRANSLATION FOR TRANSFER statement, see Section 9.11.
  - Source database file logical names

    If the source file name is a logical name that translates to a standard OpenVMS file name specification, Replication Option performs the logical name translation when the CREATE TRANSFER statement is processed. The SHOW TRANSFER DEFINITION statement shows the source file name in its translated form. The logical name can also translate into strings that, in turn, contain logical names. As long as the resulting string looks like an OpenVMS file specification and the logical name is not a concealed device name, the translation process is repeated.

If the logical name translates into a Rdb Transparent Gateway database access string, Replication Option does no further translations. For example, access strings for Rdb Transparent Gateway to DB2 look similar to the following:

```
/TYPE=DB2/CONFIG_FILE=DB2_TARGET_CONFIG
```

Even if Rdb Transparent Gateway to DB2 name strings contain logical name references, these references are not translated into OpenVMS file specifications. You can use a logical name in place of the entire string to identify a Rdb Transparent Gateway database.

- Target database file logical name

  If the target database name is a logical name, it is not translated by Replication Option when it is stored in the transfer definition; it is translated when the transfer is executed.

- Copy process log file logical name

  Replication Option translates the log file logical name in the same way as source file logical names, with the following exceptions:

  * Log file names must be standard OpenVMS file specifications.
  * Node names or version numbers are not allowed.

  The log file specification is translated in SQL when the CREATE TRANSFER statement is executed.

- Prologue and epilogue command procedure file logical names

  The following rules apply for prologue and epilogue files:

  * Replication Option translates the prologue and epilogue file logical names in the same way as the source file logical name and performs the translation when the CREATE TRANSFER statement is executed.
  * The prologue and epilogue file names must be standard OpenVMS file specifications. Node names and version numbers are allowed. Replication Option expects node names in file specifications to be from 1 to 6 characters long. DECnet/OSI permits node names to be much longer. To use Replication Option in a DECnet/OSI environment, you must define and use short synonyms for the long node names. Node name synonyms can be registered in either a local or distributed name service database. For details on establishing node name synonyms, see the documentation for DECnet/OSI.

- When a transfer is to a new database, Replication Option creates a nonmultischema database. Replication Option does not transfer multischema information from source catalogs and schemas, only from the table and view definitions.

  After the transfer has completed, or as part of an epilogue procedure, you can alter the Rdb target database to add multischema attributes. All the tables and views that Replication Option transferred become part of the default catalog and schema. However, you will not be able to move already-existing tables from the default catalog and schema to a catalog and schema that you specify.

  When you transfer data to an existing multischema database, the tables get placed automatically in the default catalog and schema.

- In an extraction transfer, you can change table definitions in the source database or definitions of columns on which the transferred tables are based. However, those changes are not reflected in the transfer definition.

  In replication transfers, you can make such changes until the initial transfer begins. Thereafter, any attempts to change or delete definitions for tables or columns in the replication transfer are not allowed, with the exception of the following four column attributes; these attributes can be modified even after the initial transfer begins:

  - Edit string
  - Default value for DATATRIEVE software
  - Query name for DATATRIEVE software
  - Query header

  However, note that if you do change the attributes listed in a source column, a comparable change is not made automatically to the target database column. If you want that to happen, you must make the change in the source and the target databases.

  In extraction and replication transfers, if you make changes that cause the source tables or columns to be incompatible with the transfer definitions, Replication Option detects these differences at the time of transfer execution. The transfer fails, the transfer status reflects the failure, and the transfer is suspended.

  However, with extractions or initial replications, the copy process does not detect changes to the data type of a column. For example, you can change a column from the data type CHAR to the data type INTEGER without causing a failure.

- When using the INTERVAL data type in a CREATE TRANSFER statement, you should isolate the data type expression in parentheses. For example:

```
SQL> CREATE TRANSFER TEST_SQL_EXTRACT_DATETIME_2
cont>   TYPE IS EXTRACTION
cont>   MOVE TABLES
cont>   SELECT ORDER_NUMBER, WHEN_PROMISED, WHEN_DELIVERED
cont>      FROM ORDER_INFO
cont>      WHERE (CAST (WHEN_DELIVERED AS DATE ANSI) - WHEN_PROMISED)
cont>      DAY > (INTERVAL '30' DAY)
cont>   TO NEW FILENAME DDAL$TEMP:SQL_DATETIME_TARGET_E2
```

  SQL syntax does not require that you enclose an interval qualifier, for example DAY TO HOUR, in parentheses. However, if your interval qualifier does not include the word *TO*, and you omit the parentheses, SQL interprets the TO in the TO NEW FILENAME clause as the TO in the DAY TO HOUR clause. The following syntax is unclear:

```
cont> DAY > INTERVAL '30' DAY TO NEW FILENAME  . . .
```

  The TO that precedes the NEW/EXISTING clause is optional. If you are using an interval qualifier, you can omit the *TO* in the NEW/EXISTING clause to avoid confusion, as in the following example:

```
DAY > INTERVAL '30' DAY NEW FILENAME  . . .
```

Examples

Example 1: Transferring a central reference table into an existing database.

A central, corporate database maintains price lists for inventory stored at the company's various branches. An inventory database at one of these branches has tables that are managed at the branch site. This example creates a transfer of the PARTS_PRICE table to the branch database in Atlanta. From the syntax used in the ATTACH statement for the source database and in the EXISTING clause for the target database, you cannot tell the type of database used or whether the database is local or remote from the system on which the transfer is created and executed. The source and target specifiers are OpenVMS logical names. The definitions are deliberately not shown in this example to emphasize the point that this transfer definition might refer to any source or target database system.

```
SQL> ATTACH 'FILENAME PRICE_CATALOG';
SQL> CREATE TRANSFER ATLANTA_PRICES TYPE IS EXTRACTION
cont>  MOVE TABLES
cont>    SELECT * FROM PARTS_PRICE
cont>  TO EXISTING FILENAME INVENTORY_DATABASE
cont>  WITH NO CHECKING
cont>  LOG IS ATLANTA_PRICES_TRANSFER_LOG
cont>  COMMENT IS 'Merge parts price reference data into branch database'
cont>  ;
```

Example 2: Creating an extraction database transferring all tables.

This example is of an extraction transfer that copies all tables from a Distributed Option database to an Rdb target database. The use of the TO NEW clause directs Replication Option to create the target Rdb files before transferring the tables. The transfer definition specifies some characteristics of the target database (NUMBER OF USERS and SNAPSHOT ALLOCATION).

```
SQL> ATTACH 'FILENAME /TYPE=DBI/DBNAME=PERSONNEL';
SQL> CREATE TRANSFER COLLEGE_INFO TYPE IS EXTRACTION
cont>  MOVE TABLES *
cont>  TO NEW FILENAME NODE1::DISK1:[BURTON]EMP_COLLEGE
cont>  NUMBER OF USERS IS 25
cont>  SNAPSHOT ALLOCATION IS 150 PAGES
cont>  LOG FILE 'COLLEGE_INFO.LOG'
cont>  COMMENT IS 'Info about employees'' colleges only'
cont>  ;
```

Example 3: Replicating selected columns and rows.

This replication transfer specifies three columns from the Rdb PERSONNEL source database for transfer to the Oracle target database. The columns are EMPLOYEE_ID, COLLEGE_CODE, and DEGREE. Only rows of employees with PhDs are selected for the transfer.

```
SQL> ATTACH 'FILENAME PERSONNEL';
SQL> CREATE TRANSFER PHD_SURVEY TYPE IS REPLICATION
cont>   MOVE TABLES
cont>   SELECT
cont>      EMPLOYEE_ID,
cont>      COLLEGE_CODE,
cont>      DEGREE
cont>   FROM DEGREES WHERE DEGREE = 'PhD'
cont>   TO EXISTING FILENAME '/TYPE=ORACLE/CONFIG_FILE=ORA_CONFIG'
cont>   WITH NO CHECKING
cont>   LOG IS PHD_SURVEY.LOG
cont>   COMMENT IS 'Employees with PhDs'
cont>   ;
```

Example 4: Creating an extraction rollup database from three source databases.

The following example shows an extraction rollup transfer that invokes several source databases: PERSONNEL, WORKERS, and STAFF. The transfer selects columns in the EMPLOYEES tables from the three databases specified by the aliases DB1, DB2, and DB3 and located on three different nodes and in the JOBS table from DB1. Then the transfer moves these tables to a single target database, LOCAL_DB.

```
SQL> ATTACH 'ALIAS DB1 FILENAME NODE1::DISK1:[DIR1]PERSONNEL';
SQL> ATTACH 'ALIAS DB2 FILENAME NODE2::DISK2:[DIR2]WORKERS';
SQL> ATTACH 'ALIAS DB3 FILENAME NODE3::DISK3:[DIR3]STAFF';
SQL> CREATE TRANSFER LABOR TYPE IS EXTRACTION ROLLUP
cont>  MOVE TABLES
cont>     SELECT ONE.EMPLOYEE_ID,
cont>            ONE.LAST_NAME
cont>     FROM DB1.EMPLOYEES ONE WHERE ONE.STATUS_CODE = 2
cont>        UNION ALL
cont>     SELECT TWO.EMPLOYEE_ID,
cont>            TWO.LAST_NAME
cont>     FROM DB2.EMPLOYEES TWO WHERE TWO.STATUS_CODE = 2
cont>        UNION ALL
cont>     SELECT THREE.EMPLOYEE_ID,
cont>            THREE.LAST_NAME
cont>     FROM DB3.EMPLOYEES THREE WHERE THREE.STATUS_CODE = 2
cont>  INTO EXEMPT_EMPLOYEES,
cont>     SELECT ONE.JOB_TITLE,
cont>            ONE.JOB_CODE
cont>     FROM DB1.JOBS ONE WHERE ONE.WAGE_CLASS = 4
cont>  INTO ENTRY_JOBS
cont>  TO NEW FILENAME DISK4:[DIR4]LOCAL_DB
cont>  LOG FILE IS DISK4:[DIR4]LOCAL.LOG
cont>  COMMENT IS 'Surveying wage class job titles'
cont>  ;
```

Example 5: Creating a replication database containing a set of tables and three views.

The following example creates a replication transfer that replicates the EMPLOYEES, JOB_HISTORY, SALARY_HISTORY, DEPARTMENTS, and JOBS tables in a file named NEW_TARGET. The definition also selects three views for transfer: CURRENT_INFO, CURRENT_JOB, and CURRENT_ SALARY. Because this is a replication transfer, the source must be Rdb. Because the target database clause indicates TO NEW FILENAME, the target database must also be Rdb.

9.4 CREATE TRANSFER Statement

```
SQL> ATTACH 'FILENAME PERSONNEL';
SQL> CREATE TRANSFER NH_EVALUATION TYPE IS REPLICATION
cont>   MOVE TABLES
cont>      SELECT * FROM EMPLOYEES,
cont>      SELECT * FROM JOB_HISTORY,
cont>      SELECT * FROM SALARY_HISTORY,
cont>      SELECT * FROM DEPARTMENTS,
cont>      SELECT * FROM JOBS
cont>   MOVE VIEWS
cont>      CURRENT_INFO,
cont>      CURRENT_JOB,
cont>      CURRENT_SALARY
cont> TO NEW FILENAME NHE_TARGET
cont> LOG FILE IS NHE_LOG
cont> COMMENT IS 'Evaluating job and salary history'
cont> ;
```

Example 6: Creating an extraction database containing a join of two tables.

The following example moves the CURRENT_JOB view defined in the source database as a join of the EMPLOYEES table and the JOB_HISTORY table. You can achieve the effect of a join by transferring a view that is a join in the source database. On the target database, the rows selected by the view are created as a table. (This procedure will not work for replication transfers.)

```
SQL> ATTACH 'FILENAME PERSONNEL';
SQL> CREATE TRANSFER DOC_MANAGER TYPE IS EXTRACTION
cont>  MOVE TABLES
cont>     SELECT LAST_NAME,
cont>            FIRST_NAME,
cont>            EMPLOYEE_ID,
cont>            DEPARTMENT_CODE,
cont>            SUPERVISOR_ID
cont>     FROM CURRENT_JOB WHERE JOB_CODE = 'DMGR'
cont>  TO EXISTING FILENAME MANAGERS_DB
cont>  WITH NO CHECKING
cont>  LOG FILE IS DMGR_LOG
cont>  COMMENT IS ' documentation managers'
cont>  ;
```

Example 7: Creating an extraction transfer definition with prologue and epilogue files.

The following extraction transfer includes two command files: DIALUP.COM and HANGUP.COM. The prologue command procedure establishes a connection to a remote node, and the epilogue command procedure terminates the connection to the remote node. This example shows how to specify the prologue file and epilogue file; it does not show the command procedures.

```
SQL> ATTACH 'FILENAME PERSONNEL';
SQL> CREATE TRANSFER PERSONNEL_BRIEF TYPE IS EXTRACTION
cont>  MOVE TABLES *
cont>     SELECT * FROM EMPLOYEES,
cont>     SELECT * FROM JOB_HISTORY
cont>  TO EXISTING FILENAME PERSONNEL_BRIEF
cont>  WITH NO CHECKING
cont>  PROLOGUE FILE IS USER2:[PERSONNEL.PROLOG]DIALUP.COM
cont>  EPILOGUE FILE IS USER2:[PERSONNEL.COM]HANGUP.COM
cont>  LOG IS PERS_BRIEF_LOG
cont>  COMMENT IS 'A study of employee job histories'
cont>  ;
```

Example 8: Transferring a table from a multischema database.

This example transfers the data from the DEPARTMENTS table in the ACCOUNTING schema and the ADMINISTRATION catalog of the Rdb CORPORATE_DATA database to the new database NEW_DEPTS.

```
SQL> ATTACH 'FILENAME CORPORATE_DATA';
SQL> CREATE TRANSFER DEPTS_TRANS
cont> TYPE IS EXTRACTION
cont> MOVE TABLE ADMINISTRATION.ACCOUNTING.DEPARTMENTS
cont> TO NEW FILENAME NEW_DEPTS
cont> LOG IS DISK1:[ACCOUNTING]NEW_DEPTS.LOG
cont> COMMENT IS 'Extract DEPARTMENTS data for accounting purposes'
cont> ;
```

Example 9: Using the EXTRACT function in a CREATE TRANSFER statement.

The following example uses the EXTRACT function with date-time data types to show orders posted between 11:00 AM and 2:00 PM.

```
SQL> CREATE TRANSFER MIDDAY_POSTINGS
cont>  TYPE IS EXTRACTION
cont>  MOVE TABLES
cont>    SELECT  ORDER_NUMBER, WHEN_ORDER_LOGGED
cont>    FROM    ORDER_INFO
cont>    WHERE   EXTRACT(HOUR FROM WHEN_ORDER_LOGGED) BETWEEN 11 AND 14
cont>  TO NEW FILENAME DISPATCHER
cont>  LOG IS CENTRAL_LOGGING:POSTINGS.LOG;
```

Example 10: Using the CAST function in a CREATE TRANSFER statement.

The following example uses the CAST function to show orders delivered a month or more late.

9.4 CREATE TRANSFER Statement

```
SQL> CREATE TRANSFER LATE_DELIVERIES
cont>   TYPE IS EXTRACTION
cont>   MOVE TABLES
cont>   SELECT  ORDER_NUMBER, WHEN_PROMISED, WHEN_DELIVERED
cont>       FROM    ORDER_INFO
cont>       WHERE   (CAST (WHEN_DELIVERED AS DATE ANSI) - WHEN_PROMISED)
cont>       DAY > (INTERVAL '30' DAY)
cont>   TO NEW FILENAME DISPATCHER
cont>   LOG IS CENTRAL_LOGGING:LATE.LOG;
```

Example 11: Using the CHECKPOINT option in a CREATE TRANSFER statement.

The following example uses the CHECKPOINT option. Should the transfer experience a recoverable error, Replication Option attempts to retry the transfer for 45 minutes. During that 45-minute recovery period, Replication Option might attempt to restart from the last checkpoint several times. If Replication Option fails to recover from the failure after 45 minutes, transfer retry attempts are determined by the transfer schedule and its RETRY EVERY clause. (If the transfer does not succeed within the 45-minute recovery period, the retries specified with the RETRY EVERY clause start at the beginning of the transfer, not at the last checkpoint.)

```
SQL> CREATE TRANSFER ALL_EMPLOYEES TYPE IS EXTRACTION
cont>   MOVE TABLES
cont>      SELECT LAST_NAME,
cont>             FIRST_NAME,
cont>             EMPLOYEE_ID
cont>      FROM EMPLOYEES
cont>   TO NEW FILENAME NODE1::DISK1:[EMPLOYEES]ALL_EMP
cont>   LOG FILE IS ALL_EMP.LOG
cont>   CHECKPOINT RECOVER WITHIN 45 MINUTES
cont>   COMMENT IS 'Listing of All Employees';
```

Example 12: Creating a replication transfer with NO DELETE.

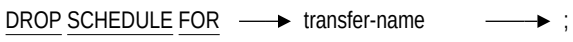
The following example specifies a type of REPLICATION WITH NO DELETE to create an archive of the PERSONNEL database. To archive the source database, the transfer must be to an existing target database. Therefore, before creating the transfer definition, an empty target database is established first.

```
SQL> CREATE DATABASE FILENAME DISK1:[TARGET]ARCHIVE_PERS.RDB;

SQL> ATTACH 'FILENAME PERSONNEL';
SQL> CREATE TRANSFER ARCHIVE
cont> TYPE IS REPLICATION WITH NO DELETE
cont> MOVE TABLES *
cont> TO EXISTING FILENAME DISK1:[TARGET]ARCHIVE_PERS
cont> LOG IS DISK1:[TARGET]ARCHIVE_PERS.LOG
cont> COMMENT IS 'Create and Maintain Archive of PERSONNEL Database';
```

9.5 DROP SCHEDULE Statement

Deletes the schedule definition associated with a transfer definition.

Format



Argument

transfer-name
Identifies the transfer whose schedule is to be deleted. This parameter can be entered in lowercase or uppercase. It is treated as uppercase by Replication Option.

Usage Notes

- To drop a schedule definition, the associated transfer must be associated with your UIC or you must have the ALTER transfer privilege or the OpenVMS BYPASS privilege.
- You must execute the DROP SCHEDULE statement outside the scope of a transaction. If you issue this statement when a transaction is active, SQL returns an error message.
- You can drop a schedule definition only when the associated transfer is in the suspended state. Refer to the Section 9.14 for information about suspending a transfer. If you issue a START TRANSFER statement (without the NOW keyword) after dropping the schedule, the transfer will be placed in the unscheduled state.
- If a transfer has a schedule definition, DROP TRANSFER also drops the corresponding schedule definition.

Examples

Stopping a transfer, then dropping its schedule definition.

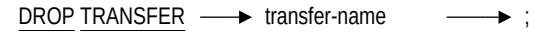
This example suspends the NH_EMPLOYEES transfer and then drops its associated schedule. The error message notifies you that a schedule definition cannot be dropped unless the transfer that the definition schedules is suspended with the STOP TRANSFER statement.

```
SQL> DROP SCHEDULE FOR NH_EMPLOYEES;
%SQL-F-DDALERR, Replication Option fatal error
-DDAL-E-TNSF_NOT_SUSPEN, transfer is not suspended
SQL> STOP TRANSFER NH_EMPLOYEES;
SQL> DROP SCHEDULE FOR NH_EMPLOYEES;
```

9.6 DROP TRANSFER Statement

Drops a transfer definition. If a transfer has a schedule definition, the DROP TRANSFER statement also drops the corresponding schedule definition.

Format



Argument

transfer-name
Specifies the transfer definition to be dropped. This parameter can be entered in lowercase or uppercase. It is treated as uppercase by Replication Option.

Usage Notes

- To issue the DROP TRANSFER statement, the transfer definition you drop must be associated with your UIC or you must have the DROP transfer privilege or the OpenVMS BYPASS privilege.
- You must execute the DROP TRANSFER statement outside the scope of a transaction. If you issue this statement when a transaction is active, SQL returns an error message.
- You can drop a transfer only if the transfer is in the suspended state. Refer to Section 9.14 for information about how to suspend a transfer.
- When you drop a transfer definition, the associated target database does not change in any way. With a transfer into an existing database this has the following implications:
  1. Data definitions created by this transfer remain in effect.
  2. Data written to the tables remains in place.
  3. The data definitions continue to be listed in the RDBVMS$TRANSFER_OWNER or DDAL$TRANSFER_OWNR table as owned by this transfer even though the transfer definition no longer exists.

The third point is important. It enables you to change a transfer definition by dropping it and then re-creating it. Leaving the data definitions in place is important in case they are referenced by other data definitions not owned by the current transfer. (Note that some, but not all, transfer definition attributes can also be changed with the ALTER TRANSFER statement.)

Examples

Stopping a transfer, then dropping its definition.

The following example suspends the NH_EMPLOYEES transfer and then drops its definition. If a transfer has an associated schedule definition, that definition is dropped as well.

```
SQL> STOP TRANSFER NH_EMPLOYEES;
SQL> DROP TRANSFER NH_EMPLOYEES;
```

9.7 GRANT Statement

Creates or adds privileges to an entry in the transfer database access privilege set, called the **access control list (ACL)**, for a transfer. Each entry in an ACL consists of an identifier and a list of privileges assigned to the identifier:

- Each identifier specifies a user or a set of users.
- The list of privileges specifies what operations that user or user group can perform on the transfer.

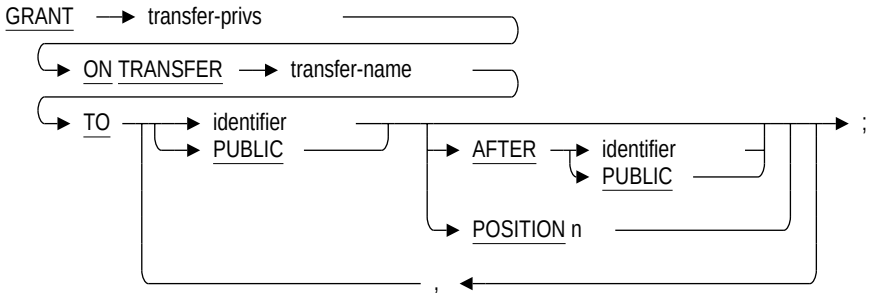
When a user tries to perform an operation on a transfer, Replication Option reads the associated ACL from top to bottom, comparing the identifier of the user with each entry. As soon as Replication Option finds the first match, it grants the rights listed in that entry and stops the search. All identifiers that do not match a previous entry "fall through" to the entry [*,*] (equivalent to the keyword PUBLIC). If no entry has the identifier [*,*], then users with unmatched identifiers are denied all access to the transfer or schedule.

For this reason, both the entries and their order in the list are important.

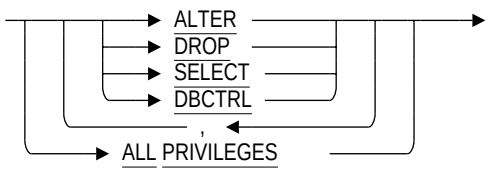
Under the Replication Option default protection scheme, when you create a new transfer you get all access rights to that transfer, including DBCTRL. All other persons are denied access to the transfer.

The DBCTRL access right enables a transfer's creator to grant access rights to other users. See the Usage Notes for information on how you can tailor the default protection for any new transfers that you create.

Format



transfer-privs =



Argument

transfer-privs
The list of privileges you want to add to an existing ACL entry or create in a new one.

Table 9–1 displays the privileges that grant you access to perform each Replication Option statement listed against a specified transfer. You need not have these privileges to execute each statement if you have the same UIC as the user who created the transfer against which you are executing a statement, or if you have the OpenVMS BYPASS privilege.

Replication Option uses the following steps in the listed order to determine whether or not you can access a transfer definition.

1. If you are the owner, you have all necessary privileges. In this case, Replication Option stops evaluation and allows you access.
2. If you have the OpenVMS BYPASS privilege, you have all necessary privileges. If this is the case, Replication Option stops evaluation and allows you access.
3. If you have been granted privileges (or have been granted privileges indirectly through the PUBLIC identifier), Replication Option checks to see if they are appropriate for the operation you want to execute. If so, access is granted.
4. If none of the preceding conditions applies, access is denied.

9.7 GRANT Statement

Table 9–1 Replication Option Privileges

Replication Option Statement	Privilege Needed to Execute
SQL ALTER TRANSFER	ALTER
SQL ALTER SCHEDULE	ALTER
SQL CREATE SCHEDULE	ALTER
SQL DROP TRANSFER	DROP
SQL DROP SCHEDULE	ALTER
SQL GRANT	DBCTRL
SQL REINITIALIZE TRANSFER	ALTER OR DBCTRL
SQL REVOKE	DBCTRL
SQL SHOW TRANSFER[1]	ALTER, DROP, SELECT, OR DBCTRL
SQL START TRANSFER	ALTER
SQL STOP TRANSFER	ALTER or DROP

[1]Privileges are not needed to execute a SHOW TRANSFER statement. Without privileges, a SHOW TRANSFER statement will display the names, but not the definitions, schedules, or protections of all transfers. With these privileges, you see the definitions and schedules of all transfers as well as the names.

ALL PRIVILEGES
Specifies that SQL should grant all privileges in the ACL entry.

transfer-name
Identifies the transfer for which privileges are to be granted. This parameter can be entered in lowercase or uppercase. It is treated as uppercase by Replication Option.

TO identifier
TO PUBLIC
Specifies the identifiers for the new or modified ACL entry. Specifying PUBLIC is equivalent to using the user identification code (UIC) identifier [*,*].

The identifiers are standard OpenVMS identifiers. There are three types:

- UIC
- General
- System defined

However, Replication Option only supports the simple UIC-type identifiers. General and system-defined identifiers are not supported by Replication Option. The following arguments briefly describe the simple UIC-type identifiers. For more information about OpenVMS identifiers, see the OpenVMS security documentation or the OpenVMS DCL documentation.

uic-identifier

A user identification code (UIC) that uniquely identifies each user on the system. The UIC can be in either numeric or alphanumeric format. The following are all valid UIC identifiers that could identify the same user:

```
[SYSTEM3, K_JONES]
K_JONES
[341,311]
```

You can use the asterisk (*) wildcard character as part of a UIC identifier. For example, if you want to specify all users in a group, you can enter [341,*] as the identifier.

When Replication Option creates a transfer, it automatically creates an ACL entry with the identifier [*,*], which specifies the privileges given to all users on the system.

AFTER identifier
AFTER PUBLIC
AFTER POSITION n

Specifies the position of the entry within the ACL to be modified or created. If you omit the AFTER or POSITION argument, SQL searches the entire ACL for an identifier list that matches the one specified in the TO clause of the GRANT statement. If it finds a match, it modifies the ACL entry by adding those privileges specified in the privilege list that are not already present. If there is no match, SQL creates a new entry at the beginning of the ACL.

With the AFTER or POSITION argument, you can specify the position in the list after which SQL searches for an ACL entry with an identifier that matches the one specified in the TO clause of the GRANT statement. Following are specifics about the AFTER and POSITION arguments:

- In the AFTER argument, the identifier specifies the entry in the ACL after which SQL begins its search for the entry to be modified or created. If none of the entries in the ACL has an identifier that matches the identifier specified in the AFTER argument, SQL generates an error and the statement fails.

Starting *after* the entry specified by the identifier in the AFTER argument, SQL searches the entries in the ACL. If an entry has an identifier that matches the identifier specified by the TO clause of the GRANT statement, SQL modifies that ACL entry by adding the privileges specified.

If none of the entries has an identifier that matches the identifier specified by the TO clause of the GRANT statement, SQL creates a new ACL entry immediately following the identifier specified in the AFTER argument.

Specifying PUBLIC in the AFTER clause is the same as specifying the UIC identifier [*,*].

- In the POSITION argument, the integer specifies the earliest relative position in the ACL of the entry to be modified or created.

  Starting *with* the position specified by the POSITION argument, SQL searches the entries in the ACL. If an entry has an identifier that matches the identifier specified by the TO clause of the GRANT statement, SQL modifies that ACL entry by adding the privileges specified.

  If none of the entries has an identifier that matches the identifier specified by the TO clause of the GRANT statement, SQL creates a new entry for that identifier at the relative position specified in the POSITION argument (even if an entry before the position at which SQL began its search had an identifier that matched).

  If you specify a position higher than the number of entries in the list, SQL places the entry last in the ACL. For example, if you specify position 12 and there are only 10 entries in the list, the new entry is placed in position 11 and given that position number.

Usage Notes

- You must execute the GRANT statement outside the scope of a transaction. If you issue this statement when a transaction is active, SQL returns an error message.
- You must have the DBCTRL transfer privilege (or be the transfer owner, which means you have the DBCTRL transfer privilege) for the specified transfer to grant access rights to other users.
- You can use the GRANT statement to modify existing ACL entries or create new ones.

  To modify an existing ACL entry, specify the same identifier in the TO clause as is in the existing entry.

To create a new ACL entry, specify an identifier that is not already part of an entry.

- When you create new transfers, they have a PUBLIC access of NONE by default.
- If you are familiar with other products that use access control lists, you might expect to be able to specify more than one identifier by combining them with plus signs (+). This is not the case for the SQL GRANT transfer-privs statement. You cannot specify combinations of identifiers.

Examples

Example 1:

This example grants all privileges on the SURVEY transfer to the SYSTEM account. Then, the SQL SHOW TRANSFER statement shows that the SYSTEM account has the SELECT, ALTER, DROP, and DBCTRL privileges.

```
SQL> GRANT ALL PRIVILEGES ON TRANSFER SURVEY TO [SYSTEM] ;
SQL> SHOW TRANSFER SURVEY;

Definition for transfer SURVEY:
Definer                 USER1
Type                    REPLICATION
From                    USER$:[USER1.DDD_TEST1]PERSONNEL.RDB
To                      New USER$:[USER1.DDD_TEST2]SURVEY2
No log file
No prologue file
No epilogue file
Move table
    Select All
            EMPLOYEE_ID,
            YEAR_GIVEN,
            DEGREE_FIELD
        From DEGREES

Protection for transfer SURVEY:
    (IDENTIFIER=[SYSTEM],ACCESS=SELECT+ALTER+DROP+DBCTRL)
    (IDENTIFIER=[USER1],ACCESS=SELECT+ALTER+DROP+DBCTRL)

Schedule for transfer SURVEY:
No schedule found

Status for transfer SURVEY:
State                   UNSCHEDULED since 30-SEP-1995 14:12:59.38
Last completion status  transfer was successful
Last executed           30-SEP-1995 14:12:55.31
```

Example 2:

This example grants all privileges on the SURVEY transfer to the PUBLIC account. Then, the SQL SHOW TRANSFER statement shows that the PUBLIC account has the SELECT, ALTER, DROP, and DBCTRL privileges. Note that the account name does not need to be enclosed in brackets.

```
SQL> GRANT ALL PRIVILEGES ON TRANSFER SURVEY TO PUBLIC;
SQL> SHOW TRANSFER SURVEY;

Definition for transfer SURVEY:
Definer                 USER1
Type                    REPLICATION
From                    USER$:[USER1.DDD_TEST1]PERSONNEL.RDB
To                      New USER$:[USER1.DDD_TEST2]SURVEY2
No log file
No prologue file
No epilogue file
Move table
    Select All
            EMPLOYEE_ID,
            YEAR_GIVEN,
            DEGREE_FIELD
        From DEGREES

Protection for transfer SURVEY:
    (IDENTIFIER=[USER1],ACCESS=SELECT+ALTER+DROP+DBCTRL)
    (IDENTIFIER=[*,*],ACCESS=SELECT+ALTER+DROP+DBCTRL)

Schedule for transfer SURVEY:
No schedule found

Status for transfer SURVEY:
State                   UNSCHEDULED since 30-SEP-1995 14:12:59.38
Last completion status  transfer was successful
Last executed           30-SEP-1995 14:12:55.31
```

Example 3:

This example grants all privileges on the SURVEY transfer to the [1,2] account. Then, the SQL SHOW TRANSFER statement shows that the [1,2] account has the SELECT, ALTER, DROP, and DBCTRL privileges.

```
SQL> GRANT ALL PRIVILEGES ON TRANSFER SURVEY TO [1,2] ;
SQL> SHOW TRANSFER SURVEY;
```

```
Definition for transfer SURVEY:
Definer                 USER1
Type                    REPLICATION
From                    USER$:[USER1.DDD_TEST1]PERSONNEL.RDB
To                      New USER$:[USER1.DDD_TEST2]SURVEY2
No log file
No prologue file
No epilogue file
Move table
    Select All
            EMPLOYEE_ID,
            YEAR_GIVEN,
            DEGREE_FIELD
        From DEGREES

Protection for transfer SURVEY:
    (IDENTIFIER=[1,2],ACCESS=SELECT+ALTER+DROP+DBCTRL)
    (IDENTIFIER=[USER1],ACCESS=SELECT+ALTER+DROP+DBCTRL)

Schedule for transfer SURVEY:
No schedule found

Status for transfer SURVEY:
State                   UNSCHEDULED since 30-SEP-1995 14:12:59.38
Last completion status  transfer was successful
Last executed           30-SEP-1995 14:12:55.31

SQL> revoke all privileges on transfer survey from public;
SQL> show transfer survey;

Definition for transfer SURVEY:
Definer                 USER1
Type                    REPLICATION
From                    USER$:[USER1.DDD_TEST1]PERSONNEL.RDB
To                      New USER$:[USER1.DDD_TEST2]SURVEY2
No log file
No prologue file
No epilogue file
Move table
    Select All
            EMPLOYEE_ID,
            YEAR_GIVEN,
            DEGREE_FIELD
        From DEGREES

Protection for transfer SURVEY:
    (IDENTIFIER=[1,2],ACCESS=SELECT+ALTER+DROP+DBCTRL)
    (IDENTIFIER=[USER1],ACCESS=SELECT+ALTER+DROP+DBCTRL)
    (IDENTIFIER=[*,*],ACCESS=NONE)

Schedule for transfer SURVEY:
No schedule found
```

```
Status for transfer SURVEY:
State                   UNSCHEDULED since 30-SEP-1995 14:12:59.38
Last completion status  transfer was successful
Last executed           30-SEP-1995 14:12:55.31
```

Example 4:

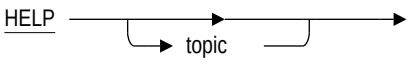
This example grants all privileges on the SURVEY transfer to all users on the system. Then, the SQL SHOW TRANSFER statement shows that all users on the system have the SELECT, ALTER, DROP, and DBCTRL privileges.

```
SQL> GRANT ALL PRIVILEGES ON TRANSFER SURVEY TO [*,*];
SQL> SHOW TRANSFER SURVEY

Definition for transfer SURVEY:
Definer                 USER1
Type                    REPLICATION
From                    USER$:[USER1.DDD_TEST1]PERSONNEL.RDB
To                      New USER$:[USER1.DDD_TEST2]SURVEY2
No log file
No prologue file
No epilogue file
Move table
    Select All
            EMPLOYEE_ID,
            YEAR_GIVEN,
            DEGREE_FIELD
        From DEGREES

Protection for transfer SURVEY:
    (IDENTIFIER=[USER1],ACCESS=SELECT+ALTER+DROP+DBCTRL)
    (IDENTIFIER=[*,*],ACCESS=SELECT+ALTER+DROP+DBCTRL)

Schedule for transfer SURVEY:
No schedule found

Status for transfer SURVEY:
State                   UNSCHEDULED since 22-SEP-1995 16:54:22.70
Last completion status  transfer was successful
Last executed           22-SEP-1995 16:54:04.94
```

9.8 HELP Statement

Displays information about Replication Option statements. When you type HELP at the SQL prompt, you can access information about Replication Option syntax diagrams, error messages, and arguments and clauses.

Format



Argument

topic
Enter the keyword REPLICATION_OPTION. SQL displays general information about Replication Option and lists topics you can select to retrieve more detailed information. If you omit the REPLICATION_OPTION keyword, HELP displays general information about SQL.

You can enter a second keyword after Replication Option to go directly to information on that subject.

Usage Notes

- An SQL HELP statement does not have to end with a semicolon. In general, SQL statements that are valid only within interactive SQL, such as HELP, do not require explicit termination.
- You can also access information about Replication Option outside of SQL. When you type HELP SQL at the DCL prompt, the Help facility displays a list of subtopics. Several SQL subtopics, namely release notes, sample database, and interactive SQL, contain information about Replication Option.

Examples

Example 1: Displaying help for Replication Option.

This example uses the REPLICATION_OPTION keyword to request general information on Replication Option. At the end of the display, HELP lists topics you can select to retrieve further information.

```
SQL> HELP REPLICATION_OPTION
```

Example 2: Displaying HELP for the Replication Option CREATE TRANSFER statement.

This example requests information on the Replication Option CREATE TRANSFER statement. The REPLICATION_OPTION keyword displays a syntax diagram for the CREATE TRANSFER statement and provides additional information if you ask for it.

```
SQL> HELP REPLICATION_OPTION CREATE_TRANSFER
```

9.9 REINITIALIZE TRANSFER Statement

Reinitializes a replication transfer in preparation for making the source and replication databases consistent with each other. You must reinitialize a transfer when someone restores a previous version of the source database.

Format

Argument

transfer-name
Specifies the replication transfer that you want to reinitialize.

Usage Notes

- If you want to reinitialize a particular transfer, the transfer definition must be associated with your UIC or you must have the DBCTRL or ALTER transfer privilege, or the OpenVMS BYPASS privilege.
- You can reinitialize a replication 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 active, SQL returns an error message.
- After you enter the REINITIALIZE TRANSFER statement, the transfer remains in the suspended state until you issue a START TRANSFER statement to restart it.
- The REINITIALIZE TRANSFER statement has a different effect depending on the type of replication transfer being reinitialized and whether the transfer is TO NEW or TO EXISTING:
  - For a standard replication transfer to an existing database, REINITIALIZE TRANSFER forces the next execution of the transfer to drop all data in the target tables and retransfer all the data specified in the transfer definition rather than update the tables with changed data. If the shared-write option is in effect, only data in the table

owned by the transfer is deleted; data written by other transfers or by customer applications is left intact.

- For a replication transfer with the NO DELETE attribute, REINITIALIZE TRANSFER forces the next execution of the transfer to drop all data in the target tables except those rows whose DDAL$DBKEY column has been set to NULL. Any row deleted from the source database has its DDAL$DBKEY column set to NULL in the target database.
- For a replication transfer specifying TO NEW FILENAME target-file-spec, new versions of the target database files are created and all tables specified in the transfer definition and their corresponding rows will be copied from the source database to the target database.

Examples

Example 1: Creating a new replication database consistent with the source database.

This example places the PHD_SURVEY 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.

```
SQL> STOP TRANSFER PHD_SURVEY;
SQL> REINITIALIZE TRANSFER PHD_SURVEY;
SQL> START TRANSFER PHD_SURVEY;
```

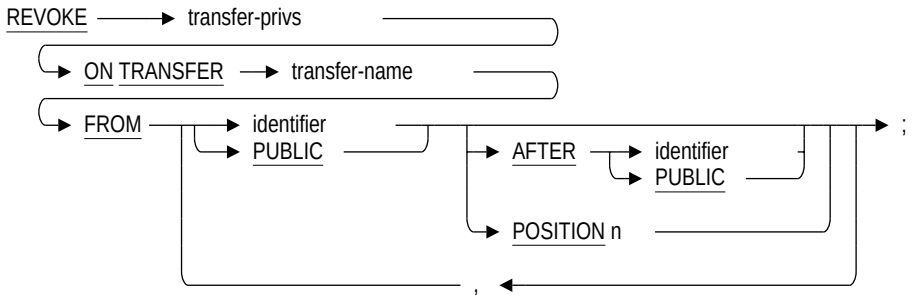
9.10 REVOKE Statement

Removes privileges from or entirely deletes an entry to the access privilege set (called the **access control list**) for a transfer. Each entry in an access control list (ACL) consists of an identifier and a list of privileges assigned to the identifier.

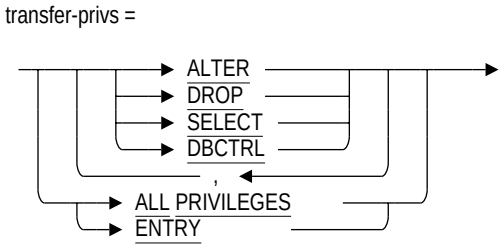
- Each identifier specifies a user or a set of users.
- The list of privileges specifies what operations that user or user group can perform on the transfer or schedule.

When a user tries to perform an operation on a transfer, Replication Option reads the associated ACL from top to bottom, comparing the identifier of the user with each entry. As soon as Replication Option finds the first match, it grants the rights listed in that entry and stops the search. All identifiers that do not match a previous entry are compared with the subsequent entry, and if no match occurs, they receive the rights of ("fall through" to) the entry [*,*], if it exists. If no entry has the UIC [*,*], then unmatched UICs are denied all access to the transfer or schedule. For this reason both the entries and their order in the list are important.

Format



9.10 REVOKE Statement



Arguments

transfer-privs
The list of privileges you want to remove from an existing ACL entry. The operations permitted by a given privilege keyword differ, depending on whether it was granted for a transfer or schedule.

See Table 9–1 for a list of the statements each privilege allows you to execute.

ALL PRIVILEGES
Specifies that SQL should revoke all privileges in the ACL entry. The REVOKE ALL PRIVILEGES statement differs from the REVOKE ENTRY statement in that it does not delete the entire entry from the ACL. The identifier remains, but without any privileges. An empty ACL entry denies all access to users matching the identifier, even if an entry later in the ACL grants PUBLIC access.

transfer-name
Identifies the transfer for which privileges are to be revoked. This parameter can be entered in lowercase or uppercase. It is treated as uppercase by Replication Option.

ENTRY
Deletes the entire entry in the ACL, including the identifier.

FROM identifier
FROM PUBLIC
Specifies the identifiers for the ACL entry to be modified or deleted. Specifying PUBLIC is equivalent to using the user identification code (UIC) identifier [*,*].

The identifiers are standard OpenVMS identifiers. There are three types:

- UIC
- General

- System defined

However, Replication Option only supports the simple UIC-type identifiers. General and system-defined identifiers are not supported by Replication Option. The following arguments briefly describe the simple UIC-type identifiers. For more information about OpenVMS identifiers, see the OpenVMS security documentation or the OpenVMS DCL documentation.

- uic-identifier

  A user identification code (UIC) that uniquely identifies each user on the system. The UIC can be in either numeric or alphanumeric format. The following are all valid UIC identifiers that could identify the same user:

  ```
  [SYSTEM3, K_JONES]
  K_JONES
  [341,311]
  ```

  You can use the asterisk (*) wildcard character as part of a UIC identifier. For example, if you want to specify all users in a group, you can enter [341,*] as the identifier.

  When Replication Option creates a transfer, it automatically creates an ACL entry with the identifier [*,*], which specifies the privileges given to all users on the system.

AFTER identifier
AFTER PUBLIC
AFTER POSITION n

Specifies the position of the entry within the ACL. If you omit the AFTER or POSITION argument, SQL searches the entire ACL for an identifier list that matches the one specified in the FROM clause of the REVOKE statement. If it finds a match, it modifies the ACL entry by deleting the privileges specified in the privilege list. If there is no match, SQL generates an error and the REVOKE statement has no effect on the ACL.

With the AFTER or POSITION argument, you can specify the position in the list from which SQL searches for an ACL entry with an identifier that matches the one specified in the FROM clause of the REVOKE statement.

- In the AFTER argument, the identifier specifies the entry in the ACL after which SQL begins its search for the entry to be modified or deleted. If none of the entries in the ACL has an identifier that matches the identifier specified in the AFTER argument, SQL generates an error and the statement fails.

Starting after the entry specified by the identifier in the AFTER argument, SQL searches entries in the ACL. If an entry has an identifier that matches the identifier specified by the FROM clause of the REVOKE statement, SQL modifies or deletes that ACL entry.

If none of the entries has an identifier that matches the identifier specified by the FROM clause of the REVOKE statement, SQL generates an error and the statement fails (even if an entry before the position at which SQL began its search had an identifier that matched).

Specifying PUBLIC in the AFTER clause is the same as specifying the UIC identifier [*,*].

- In the POSITION argument, the integer specifies the earliest relative position in the ACL of the entry to be modified or deleted. If the integer is larger than the number of entries in the ACL, SQL generates an error and the statement fails.

  Starting with the position specified by the POSITION argument, SQL searches entries in the ACL. If an entry has an identifier that matches the identifier specified by the FROM clause of the REVOKE statement, SQL modifies or deletes that ACL entry.

  If none of the entries has an identifier that matches the identifier specified by the FROM clause of the REVOKE statement, SQL generates an error and the statement fails (even if an entry before the position at which SQL began its search had an identifier that matched).

Usage Notes

- You must execute the REVOKE statement outside the scope of a transaction. If you issue this statement when a transaction is active, SQL returns an error message.
- You must have the DBCTRL access right for the specified transfer to revoke other users' access rights.

Examples

Example 1:

This example revokes all privileges on the SURVEY transfer from the [SYSTEM] account. Then, the SQL SHOW TRANSFER statement shows that the [SYSTEM] account has no privileges.

```
SQL> REVOKE ALL PRIVILEGES ON TRANSFER SURVEY FROM [SYSTEM];
SQL> SHOW TRANSFER SURVEY;

Definition for transfer SURVEY:
Definer                 USER1
Type                    REPLICATION
From                    USER$:[USER1.DDD_TEST1]PERSONNEL.RDB
To                      New USER$:[USER1.DDD_TEST2]SURVEY2
No log file
No prologue file
No epilogue file
Move table
    Select All
            EMPLOYEE_ID,
            YEAR_GIVEN,
           DEGREE_FIELD
        From DEGREES

Protection for transfer SURVEY:
    (IDENTIFIER=[SYSTEM],ACCESS=NONE)
    (IDENTIFIER=[USER1],ACCESS=SELECT+ALTER+DROP+DBCTRL)

Schedule for transfer SURVEY:
No schedule found

Status for transfer SURVEY:
State                   UNSCHEDULED since 30-SEP-1995 14:12:59.38
Last completion status  transfer was successful
Last executed           30-SEP-1995 14:12:55.31
```

Example 2:

This example revokes all privileges on the SURVEY transfer from all users on the system. Then, the SQL SHOW TRANSFER statement shows that the users on the system have no privileges.

```
SQL> REVOKE ALL PRIVILEGES ON TRANSFER SURVEY FROM PUBLIC;
SQL> SHOW TRANSFER SURVEY;

Definition for transfer SURVEY:
Definer                 USER1
Type                    REPLICATION
From                    USER$:[USER1.DDD_TEST1]PERSONNEL.RDB
To                      New USER$:[USER1.DDD_TEST2]SURVEY2
No log file
No prologue file
No epilogue file
Move table
   Select All
            EMPLOYEE_ID,
            YEAR_GIVEN,
            DEGREE_FIELD
        From DEGREES
```

9.10 REVOKE Statement

```
Protection for transfer SURVEY:
    (IDENTIFIER=[USER1],ACCESS=SELECT+ALTER+DROP+DBCTRL)
    (IDENTIFIER=[*,*],ACCESS=NONE)

Schedule for transfer SURVEY:
No schedule found

Status for transfer SURVEY:
State                   UNSCHEDULED since 30-SEP-1995 14:12:59.38
Last completion status  transfer was successful
Last executed           30-SEP-1995 14:12:55.31
```

Example 3:

This example revokes all privileges on the SURVEY transfer from the [1,2] account. Then, the SQL SHOW TRANSFER statement shows that the [1,2] account has no privileges.

```
SQL> REVOKE ALL PRIVILEGES ON TRANSFER SURVEY FROM [1,2];
SQL> SHOW TRANSFER SURVEY;

Definition for transfer SURVEY:
Definer                 USER1
Type                    REPLICATION
From                    USER$:[USER1.DDD_TEST1]PERSONNEL.RDB
To                      New USER$:[USER1.DDD_TEST2]SURVEY2
No log file
No prologue file
No epilogue file
Move table
    Select All
            EMPLOYEE_ID,
            YEAR_GIVEN,
            DEGREE_FIELD
        From DEGREES

Protection for transfer SURVEY:
    (IDENTIFIER=[1,2],ACCESS=NONE)
    (IDENTIFIER=[WHCOX],ACCESS=NONE)
    (IDENTIFIER=[SYSTEM],ACCESS=NONE)
    (IDENTIFIER=[USER1],ACCESS=SELECT+ALTER+DROP+DBCTRL)
    (IDENTIFIER=[*,*],ACCESS=NONE)

Schedule for transfer SURVEY:
No schedule found

Status for transfer SURVEY:
State                   UNSCHEDULED since 30-SEP-1995 14:12:59.38
Last completion status  transfer was successful
Last executed           30-SEP-1995 14:12:55.31
```

Example 4:

This example revokes all privileges on the SURVEY transfer from all users on the system. Then, the SQL SHOW TRANSFER statement shows that the users on the system have no privileges.

```
SQL> REVOKE ALL PRIVILEGES ON TRANSFER SURVEY FROM [*,*];
SQL> SHOW TRANSFER SURVEY;

Definition for transfer SURVEY:
Definer                 USER1
Type                    REPLICATION
From                    USER$:[USER1.DDD_TEST1]PERSONNEL.RDB
To                      New USER$:[USER1.DDD_TEST2]SURVEY2
No log file
No prologue file
No epilogue file
Move table
    Select All
            EMPLOYEE_ID,
            YEAR_GIVEN,
            DEGREE_FIELD
        From DEGREES

Protection for transfer SURVEY:
    (IDENTIFIER=[USER1],ACCESS=SELECT+ALTER+DROP+DBCTRL)
    (IDENTIFIER=[*,*],ACCESS=NONE)

Schedule for transfer SURVEY:
No schedule found

Status for transfer SURVEY:
State                   UNSCHEDULED since 22-SEP-1995 16:54:22.70
Last completion status  transfer was successful
Last executed           22-SEP-1995 16:54:04.94
```

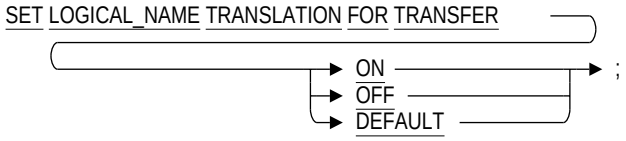
9.11 SET LOGICAL_NAME TRANSLATION FOR TRANSFER Statement

Determines which, if any, of the following file specifications SQL will translate when they are included in the CREATE TRANSFER or ALTER TRANSFER statements as logical names.

If translation is done, note that translation of logical names is an iterative process. If one logical name is defined in terms of another, SQL will translate the other logical name as well. This iterative process stops when there are no further logical names or when the name is a concealed device logical name.

Although you can change file specifications by using the ALTER TRANSFER statement, you may find it more appropriate in some cases to use this statement instead with the ON option to have all logical names translated. Use of this statement will allow the person responsible for disk management (who may not be involved with Replication Option transfers) to move a group's files from one area to another by changing the definitions of group logical names that point to disks and directories, instead of using the ALTER TRANSFER statement.

Format



Arguments

ON
Directs SQL on how to handle file specifications for the source database file(s), the target database file, the copy process log file, and the prologue and epilogue command procedure files that are expressed as logical names.

When ON is specified, SQL translates all these file specifications that are expressed as logical names.

9.11 SET LOGICAL_NAME TRANSLATION FOR TRANSFER Statement

OFF
Directs SQL on how to handle file specifications for the source database file(s), the target database file, the copy process log file, and the prologue and epilogue command procedure files that are expressed as logical names.

When OFF is specified, SQL does not translate any of these file specifications that are expressed as logical names.

DEFAULT
Directs SQL on how to handle file specifications for the source database file(s), the target database file, the copy process log file, and the prologue and epilogue command procedure files that are expressed as logical names.

When DEFAULT is specified Replication Option treats logical names as follows:

- Source database file logical names

  If the source file name(s) is a logical name that translates to a standard OpenVMS file name specification, Replication Option performs the logical name translation when the CREATE TRANSFER statement is processed. The SHOW TRANSFER DEFINITION statement shows the source file name in its translated form. The logical name can also translate into strings that, in turn, contain logical names. As long as the resulting string looks like an OpenVMS file specification and the logical name is not a concealed device name, the translation process is repeated.

  If the logical name translates into a Rdb Transparent Gateway database access specification, Replication Option does no further translations. For example, access specifications for Rdb Transparent Gateway to DB2 look similar to the following:

  ```
  /TYPE=DB2/CONFIG_FILE=DB2_TARGET_CONFIG
  ```

  Even if Rdb Transparent Gateway to DB2 name strings contain logical name references, these references are not translated into OpenVMS file specifications. You can use a logical name in place of the name string to identify a Rdb Transparent Gateway database.

- Target database file logical name

  If the target database name is a logical name, it is not translated by Replication Option when it is stored in the transfer definition; it is translated when the transfer is executed.

- Copy process log file logical name

  Replication Option translates the log file logical name in the same way as source file logical names, with the following exceptions:

  * Log file names must be standard OpenVMS file specifications.

  * Node names or version numbers are not allowed.

  The log file specification is translated in SQL when the CREATE TRANSFER statement is executed.

- Prologue and epilogue command procedure file logical names

  The following rules apply for prologue and epilogue files:

  * Replication Option translates the prologue and epilogue file logical names in the same way as the source file logical name(s) and performs the translation when the CREATE TRANSFER statement is executed.

  * The prologue and epilogue file names must be standard OpenVMS file specifications. Node names and version numbers are allowed. Replication Option expects node names in file specifications to be from 1 to 6 characters long. DECnet/OSI permits node names to be much longer. In order to use Replication Option in a DECnet/OSI environment, you must define and use short synonyms for the long node names. Node name synonyms can be registered in either a local or distributed name service database. For details on establishing node name synonyms refer to the documents for DECnet/OSI.

Usage Notes

- If you change or create transfer definitions that store file specifications as logical names, be sure that the definition of those logical names are available where needed. Those accounts that have the privilege to execute these transfers must have the definitions for the associated logical names. Logical name definitions can be systemwide, groupwide, or account specific.

Examples

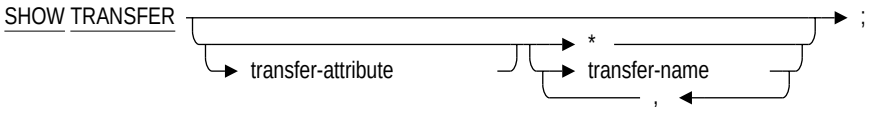
Before creating or altering a transfer definition, you should specify that logical names in OpenVMS file specifications not be translated as shown in the following example:

```
SQL> SET LOGICAL_NAME TRANSLATION FOR TRANSFERS OFF; <
```

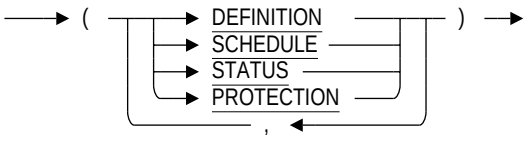
9.12 SHOW TRANSFER Statement

Displays information about the transfer definition, schedule definition, transfer status, protection, or all these transfer attributes.

Format



transfer-attribute =



Arguments

transfer-attribute
Specifies the type of information you want to display about one or more transfers. When you enter transfer attributes, separate the attributes with a comma and enclose them in parentheses. You can specify one or more of the following attributes:

- DEFINITION

  Displays transfer definition information for one transfer, all transfers, or a set of transfers you select.

- SCHEDULE

  Displays schedule definition information for one transfer, all transfers, or a set of transfers you select.

- STATUS

  Displays the following status information for one transfer, all transfers, or a set of transfers you select:

  – Transfer name

  – State of the current transfer (unscheduled, scheduled, active, waiting-to-retry, retrying, or 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)
- PROTECTION

  Displays the access control list on the specified transfer.

  See Table 9–1 in Section 9.7 for the privileges needed to execute each of the Replication Option SQL statements.

All four attributes (DEFINITION, SCHEDULE, STATUS, and PROTECTION) display the name of the transfer.

*** (wildcard character)**
The wildcard character displays information about all transfers associated with your UIC on OpenVMS, or for which you have one of the four listed privileges. For transfers to which you do not have privileges, Replication Option displays the transfer name and the name of the transfer definer.

If you omit the transfer attribute and type SHOW TRANSFER *, Replication Option displays definition, schedule, status, and privilege information for all transfers.

transfer-name
Specifies one or more transfers about which you want Replication Option to display information. This parameter can be entered in lowercase or uppercase. It is treated as uppercase by Replication Option.

Usage Notes

- 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 on OpenVMS , or you must have the ALTER, DROP, SELECT or DBCTRL transfer privilege to the transfer definition. You need not have any of these access rights to display a list of just the transfer names (using the SHOW TRANSFER statement).

- A SHOW TRANSFER statement does not have to end with a semicolon.
- Depending on the arguments you use with the SHOW TRANSFER statement, you can display information about one transfer, all transfers, or a selected number of transfers.

  If you enter the SHOW TRANSFER statement without an argument (transfer attribute, wildcard character, or transfer name), the SHOW TRANSFER statement displays a simple list of transfer names only.
- You do not have to enter all transfer attributes (DEFINITION, SCHEDULE, STATUS, and PROTECTION) to have Replication Option display definition, schedule, status, and protection information. Simply leave out the transfer attribute and enter one or more transfer names.
- Having the correct character set identified for your SQL session is important for the correct parsing and interpretation of object names.

  Before you issue a SHOW TRANSFER statement, you should establish the correct character set defaults for your SQL session. Do this by either accepting the default DEC_MCS character set or issuing either the SQL statement, SET IDENTIFIER CHARACTER SET, or the SQL statement SET DIALECT. For information on the SET IDENTIFIER CHARACTER SET and SET DIALECT statements, see the Oracle Rdb SQL Reference Manual for the version of Oracle Rdb you are using.
- When you execute the SQL SHOW TRANSFER (DEFINITION) statement, a problem may occur when you attempt to display characters on a terminal that is operating in a character set other than the one in which your data is stored. For example, if you have a terminal that displays Kanji characters, it is unlikely to correctly display Korean characters.

Examples

Example 1: Displaying a schedule definition.

This example displays information about the schedule for the NH_EMPLOYEES transfer. The transfer is scheduled to begin at 1:15 (13:15) in the afternoon and to repeat every day. If the transfer fails, the transfer will retry three times with an hour between retry attempts.

```
SQL> SHOW TRANSFER (SCHEDULE) NH_EMPLOYEES

Schedule for transfer NH_EMPLOYEES:
Start                   22-FEB-1995 13:15:00.00
Frequency               Every day at 14:30:00.00
Number of retries       3 times
Retry frequency         0 01:00:00.00
```

Example 2: Checking on the status of the NH_EMPLOYEES transfer.

This example displays information about the status of the NH_EMPLOYEES transfer. The transfer is scheduled to be performed at 2:30 (14:30) in the afternoon on February 23, 1995. The transfer ran successfully the day before.

```
SQL> SHOW TRANSFER (STATUS) NH_EMPLOYEES;

Status for transfer NH_EMPLOYEES:
State                   SCHEDULED since 22-FEB-1995 14:30:47.60
Next transfer           23-FEB-1995 14:30:00.00
Last completion status  transfer was successful
Last executed           22-FEB-1995 14:30:00.00
```

Example 3: Displaying definition, protection, schedule, and status information about a transfer.

This example displays information about the definition, protection, schedule, and status for the NH_EMPLOYEES transfer and shows the results of the statement. The example does not contain a transfer attribute, which means SQL displays information about all attributes (DEFINITION, PROTECTION, SCHEDULE, and STATUS) by default.

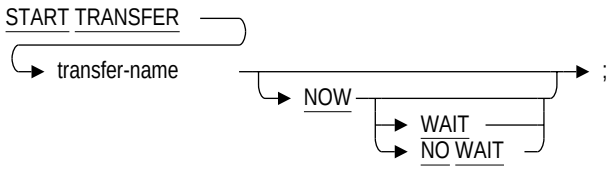
```
SQL> SHOW TRANSFER NH_EMPLOYEES

Definition for transfer NH_EMPLOYEES:
Definer                 DBADMIN
Type                    EXTRACTION
Comment                 Employees who live in New Hampshire
From                    DISK1:[DBADMIN.PERS]PERSONNEL.RDB
To                      NEW REMNOD::WORK$DISK:[PROD.PERSONNEL]NH_EMP
Log file                DISK1:[DBADMIN.PERS]NH_EMPLOYEES.LOG
No prologue file
No epilogue file
Move table
    Select All
            EMPLOYEE_ID,
            LAST_NAME,
            FIRST_NAME,
            MIDDLE_INITIAL,
            ADDRESS_DATA_1,
            ADDRESS_DATA_2,
            CITY,
            STATE,
            POSTAL_CODE
        From EMPLOYEES
        Where
            STATE = 'NH'
```

```
Protection for transfer NH_EMPLOYEES:
    (IDENTIFIER=[DBNODE,DBADMIN],ACCESS=SELECT+ALTER+DROP+DBCTRL)
    (IDENTIFIER=[*,*],ACCESS=NONE)

Schedule for transfer NH_EMPLOYEES:
Start                   22-FEB-1995 13:15:00.00
Frequency               Every day at 14:30:00.00
Number of retries       3 times
Retry frequency         0 01:00:00.00

Status for transfer NH_EMPLOYEES:
State                   SCHEDULED since 22-FEB-1995 14:30:47.60
Next transfer           23-FEB-1995 14:30:00.00
Last completion status  transfer was successful
Last executed           22-FEB-1995 14:30:00.00
```

9.13 START TRANSFER Statement

Executes a transfer on demand, or changes the state of a transfer from suspended to scheduled, unscheduled, or active.

Format



Arguments

transfer-name
Specifies the transfer to be started. This parameter can be entered in lowercase or uppercase. It is treated as uppercase by Replication Option.

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.

NO WAIT
WAIT
Specifies whether Replication Option returns control to SQL immediately after the transfer starts (NO WAIT), or waits until the transfer completes (WAIT).

The default, NO WAIT, causes Replication Option to return control immediately to the SQL command mode while the transfer executes, freeing you to do other tasks. With NO WAIT in effect, you can enter other SQL statements while the copy process completes the transfer of data rows to the target database. You can even exit SQL and log off the system, leaving the transfer to execute unattended.

The WAIT option ensures that the transfer completes before control is returned to SQL.

Usage Notes

- If you want to start a transfer, the transfer definition must be associated with your UIC, or you must have the ALTER transfer privilege, or the OpenVMS BYPASS privilege.
- 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 when you define a schedule for the transfer.
- If you specify both the NOW and WAIT arguments and your transfer has an epilogue procedure, Replication Option waits for the epilogue procedure to finish before resuming execution.
- You must place a transfer in the suspended, unscheduled, or scheduled state before you enter a START TRANSFER statement with the NOW qualifier; otherwise, Replication Option returns an error.
- You cannot enter the START TRANSFER statement when a transaction is active. You must terminate any active 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.
- When you define a one-time-only transfer schedule and specify a transfer time that has already passed, issuing a START TRANSFER statement changes the transfer status from suspended to scheduled. However, when you issue a SHOW TRANSFER STATUS statement, the “next transfer to be executed” phrase is not included. To avoid this problem and cause the transfer to execute, use the START TRANSFER NOW statement.
- Do not issue a START TRANSFER NOW WAIT statement when SQL is attached to a source database and the transfer is either a replication transfer that has not yet been initialized or a replication reinitialization. Instead, issue a DISCONNECT statement to detach from the database and then enter the START TRANSFER NOW WAIT statement. Otherwise,

the copy process waits indefinitely for you to release the source database. Because you specified that you want to wait for the copy process to complete, the transfer operation will never complete.

Examples

Example 1: Starting a transfer that has a schedule associated with it.

This example starts a transfer called NH_EMPLOYEES. If the transfer has a schedule defined for it but the transfer is suspended, issuing the START TRANSFER statement only changes the transfer state to scheduled. Execution of the transfer will not occur until the scheduled time, namely February 24, 1995 at 2:30 P.M. (14:30). The SHOW TRANSFER statement displays the transfer status before and after you enter the START TRANSFER statement.

```
SQL> SHOW TRANSFER (STATUS) NH_EMPLOYEES;

Status for transfer NH_EMPLOYEES:
State                   SUSPENDED since 23-FEB-1995 07:33:21.62
Last completion status  transfer was successful
Last executed           23-FEB-1995 14:30:00.00

SQL> START TRANSFER NH_EMPLOYEES;
SQL> SHOW TRANSFER (STATUS) NH_EMPLOYEES;

Status for transfer NH_EMPLOYEES:
State                   SCHEDULED since 23-FEB-1995 15:04:10.94
Next transfer           24-FEB-1995 14:30:00.00
Last completion status  transfer was successful
Last executed           23-FEB-1995 14:30:00.00
```

Example 2: Immediately starting a transfer with no associated schedule.

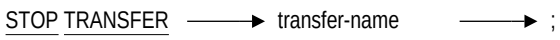
In this example, the EMPLOYEE_HISTORY transfer has been defined but has no schedule. The START TRANSFER NOW statement executes the transfer immediately, and control returns to SQL as soon as the transfer begins. Not waiting for completion is the default. The SHOW TRANSFER (STATUS) statement you enter immediately after issuing the START TRANSFER statement displays the ACTIVE status of the transfer. Because the EMPLOYEE_HISTORY transfer has no associated schedule definition, the transfer returns to the UNSCHEDULED status after the transfer completes, as shown by the display of the last SHOW TRANSFER statement.

```
SQL> START TRANSFER EMPLOYEE_HISTORY NOW;
SQL> SHOW TRANSFER (STATUS) EMPLOYEE_HISTORY;

Status for transfer EMPLOYEE_HISTORY:
State                  ACTIVE since  9-MAY-1995 15:49:42.51
Last completion status  Transfer has never executed successfully

SQL> SHOW TRANSFER (STATUS) EMPLOYEE_HISTORY;

Status for transfer EMPLOYEE_HISTORY:
State                  UNSCHEDULED since  9-MAY-1995 15:51:35.61
Last completion status  transfer was successful
Last executed           9-MAY-1995 15:49:42.76
```

9.14 STOP TRANSFER Statement

Places a transfer in the suspended state.

Format



Argument

transfer-name
Specifies the transfer to be suspended. This parameter can be entered in lowercase or uppercase. It is treated as uppercase by Replication Option.

Usage Notes

- If you want to stop a particular transfer, the transfer definition must be associated with your UIC or you must have the ALTER or DROP transfer privilege, or the OpenVMS BYPASS privilege.
- You must terminate any active transactions before you issue a STOP TRANSFER statement.
- A transfer cannot occur 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. After you stop a currently executing extraction, extraction rollup, or initial replication transfer, you cannot access the partially created target database unless you restart the transfer. This statement is true for transfers that create a new target database. If the transfer is to an existing target database, user-created tables, not specified in a transfer definition, are unaffected by the STOP TRANSFER statement. However, tables transferred by Replication Option, although they can be read, might be in an inconsistent state. For more information, refer to Chapter 8.

  If a replication update transfer is executing and you stop the transfer, users of the target database can continue their work unaffected. Furthermore, stopping the transfer does not stop the capture of the changes in the source database (written to RDB$CHANGES). The changes are transferred to the target database when the transfer is restarted. No updates to the target database are missed as a result of stopping a replication transfer for which the target tables have already been created. However, because stopping

a transfer does not stop the capture of row changes, changes continue to be written to RDB$CHANGES regardless of a transfer's state. Rows are added to RDB$CHANGES even for transfers that are suspended for a long time. The number of rows in RDB$CHANGES and the overall size of the database both continue to grow.

- When you issue a STOP TRANSFER statement for a transfer in the active state, the state changes to suspended. However, the last transfer completion status is not updated and does not reflect the incomplete transfer.

Examples

Suspending a transfer with the STOP TRANSFER statement.

The following example places the NH_EMPLOYEES transfer in the suspended state. Reappearance of the prompt indicates that Replication Option has suspended the transfer. The SHOW TRANSFER statement lets you see that the suspension took place.

```
SQL> STOP TRANSFER NH_EMPLOYEES;
SQL> SHOW TRANSFER (STATUS) NH_EMPLOYEES;

Status for transfer NH_EMPLOYEES:
State                   SUSPENDED since 23-FEB-1995 15:03:21.62
Last completion status  transfer was successful
Last executed           23-FEB-1995 14:30:00.00
```

A

Transfer Database and Special-Purpose System Tables

This appendix describes the Replication Option tables found in the transfer database, and the special system tables found in the source and target databases.

A.1 Replication Option Transfer Database Tables

The transfer database contains information about transfer definitions, transfer schedules, and the source and target databases participating in transfers. See Appendix C for specific access rights that are defined when this database is first created.

Note

Terminology used in Replication Option documentation has changed to reflect an SQL focus. Where you once read relation, field, and record, you now read table, column, and row. However, not all table names have been changed, so you may still see references to RELATION in the names.

A.1.1 Change in Transfer Database Tables

The structure of the transfer database tables has changed significantly in version 7.0. Some of the changes were necessary because of differences between Rdb and Oracle. The most significant change is in the storage of large data objects. Large data objects were originally stored in Rdb varbyte list columns, also known as segmented strings. The varbyte list data type was only available in Rdb, so the tables were redesigned to work with both Rdb and Oracle databases. In the process, table and column names were changed. See Section A.1.2 for a discussion of these large data objects.

A.1.2 Storage of Large Data Objects

Large data objects cannot always fit in one piece in a data column. For example, for some transfer definitions, the length of the stored text in the MOVE TABLES clause can be very long. A 2-step approach is used for storing such data. Stored in the main table is a count of the number of segments into which the large data object has been divided. These segments are then written into an auxiliary table, which has a column to identify the name of the transfer and a column to specify the segment number so that pieces can be reassembled in the correct order. The segment columns in the auxiliary tables are either of data type CHAR(16) or VARCHAR(512).

The following tables are used for storing large objects:

- DDAL_COMMENT (see Section A.1.12)
- DDAL_DEST_DPB (see Section A.1.13)
- DDAL_MOVE_CLAUSES (see Section A.1.14)
- DDAL_SCHED_OFFSETS (see Section A.1.15)
- DDAL_TNSF_PRIVS (see Section A.1.16)
- DDAL_TNSF_SEL_EXPR (see Section A.1.17)

Each of these tables contains:

- A VARCHAR(512) or CHAR(16) column to store one segment of the large object
- A column (either SEG_NUMBER or OFFSET_NUMBER) to store an integer value indicating the position of this segment in the entire sequence of segments
- Name column(s) (for example, TRANSFER_NAME) to associate the segment with a particular transfer

Some of these large data objects contain binary data. To be able to successfully store and retrieve such data, it is necessary to convert and store it as a sequence of printable text characters. Replication Option does this by dividing each 8-bit byte into two 4-bit nibbles. Each nibble is a binary, hexadecimal digit. It is converted into its equivalent, 8-bit ASCII form. For example, a nibble with a binary value of 1 is converted into the text character 1, which in ASCII has a binary value of 49. A nibble with a binary value of 15 (decimal) is converted into the ASCII value for the letter F, which is the hexadecimal character for the decimal 15. When the data is retrieved from the database, each hexadecimal character pair is converted back into a binary byte value.

A.1.3 DDAL_INVOKE_TIME Transfer Database Table

Table A–1 shows the columns for the DDAL_INVOKE_TIME table.

Table A–1 Replication Option Transfer Database: DDAL_INVOKE_TIME Table

Column Name	Data Type/Size	Description
TRANSFER_NAME	CHAR(31)	Name of transfer
RETRY_COUNT_LEFT	NUMBER(9)	Number of retry attempts remaining
TIME_TO_TRANSFER	Date	Next transfer execution time

A.1.4 DDAL_TRANSFERS Transfer Database Table

Table A–2 shows the columns for the DDAL_TRANSFERS table.

Table A–2 Replication Option Transfer Database: DDAL_TRANSFERS Table

Column Name	Data Type/Size	Description
TRANSFER_NAME	CHAR(31)	Name of transfer
DEFINER_NAME	CHAR(12)	Name of definer
USER_ID_CODE	NUMBER(9)	Transfer definer's user identification code[1]
TRANSFER_TYPE	NUMBER(9)	Extraction, extraction rollup, or replication transfer type[2]
SOURCE_DB_NAME	VARCHAR(255)	Name of source database[3]
DEST_DB_NAME	VARCHAR(255)	Name of target database
TNSF_TO_EXISTING	CHAR(1)	Transfer is to an existing database?[5]

[1]The number stored in DDAL_USER_ID_CODE is the decimal representation of the 32-bit value created by the OpenVMS system. If you want to convert the decimal value in DDAL_USER_ID_CODE to a UIC (octal), refer to the OpenVMS documentation about DCL concepts. There you will find information about how OpenVMS translates and stores UIC information.

[2]The DDAL_TRANSFER_TYPE column contains the number 1 for an extraction transfer, the number 2 for a replication transfer, and the number 3 for an extraction rollup transfer.

[3]No column value is stored for extraction rollup transfers. Refer to Table A–3 for source database information.

[5]The stored values are "T" or "F", signifying TRUE or FALSE. A value of "F" indicates that the transfer will create a target database.

(continued on next page)

Table A–2 (Cont.) Replication Option Transfer Database: DDAL_TRANSFERS Table

Column Name	Data Type/Size	Description
TNSF_NO_DELETE	CHAR(1)	Transfer is a replication transfer with the NO DELETE attribute?[6]
LOG_FILE_NAME	VARCHAR(255)	Copy process log file specification
PROLOGUE_FILENAME	VARCHAR(255)	Prologue file specification
EPILOGUE_FILENAME	VARCHAR(255)	Epilogue file specification
DEST_DPB_LEN	NUMBER(9)	Number of rows in the DDAL_DEST_DPB table to represent the target database parameters (DPB) for the transfer
COMMENT_LEN	NUMBER(9)	Number of rows in the DDAL_COMMENT table to represent the comment describing the transfer
MOVE_CLAUSES_LEN	NUMBER(9)	Number of rows in the DDAL_MOVE_CLAUSES table to represent the move clauses portion of the transfer definition in ASCII text format
TNSF_PRIVS_LEN	NUMBER(9)	Number of rows in the DDAL_TNSF_PRIVS table to represent the access control list containing privileges for managing the transfer
PRODUCT_VERSION	NUMBER(9)	Version of Replication Option under which transfer was defined[4]
CHECK_POINT	CHAR(1)	Is the checkpoint option enabled?[7]

[4]This column stores a 2-digit integer value for the product version number. The first digit represents the major version number, and the second digit represents the minor version number. An integer value of 61, for example, means that a transfer was created with Replication Option Version 6.1.

[6]The stored values are "T" or "F", signifying TRUE or FALSE. A value of "T" indicates that rows deleted from the source database will not be deleted from the target database; instead their DDAL_DBKEY column will be set to null.

[7]The stored values are "T" or "F", signifying TRUE or FALSE. A value of "T" indicates that the checkpoint option is enabled.

(continued on next page)

Table A–2 (Cont.) Replication Option Transfer Database: DDAL_TRANSFERS Table

Column Name	Data Type/Size	Description
CKPT_EVERY_MINUTES	NUMBER(9)	How often, in minutes, checkpoints should be executed, if the checkpoint option is enabled
CKPT_MINS_TO_RECOV	NUMBER(9)	How long Replication Option can spend, in minutes, trying to recover from an error when attempting a checkpoint

A.1.5 DDAL_SOURCE_DBS Transfer Database Table

Table A–3 shows the columns for the DDAL_SOURCE_DBS table.

Table A–3 Replication Option Transfer Database: DDAL_SOURCE_DBS Table

Column Name	Data Type/Size	Description
TRANSFER_NAME	CHAR(31)	Name of transfer
SOURCE_DB_HANDLE	CHAR(31)	Database handle of source database
SOURCE_DB_NAME	VARCHAR(255)	Name of source database corresponding to SOURCE_DB_HANDLE for this row

A.1.6 DDAL_TRANSFER_DB Transfer Database Table

Table A–4 shows the columns for the DDAL_TRANSFER_DB table.

Table A–4 Replication Option Transfer Database: DDAL_TRANSFER_DB Table

Column Name	Data Type/Size	Description
TNSF_DB_REVISION	NUMBER(9)	Version number of the transfer database[1]

[1]The version number of the transfer database is stored in the DDAL_TRANSFER_DB table when the transfer database is either converted by the DDAL$CONVERT_TR_DB.COM command procedure or when the transfer database is initially created. The transfer monitor checks the transfer database version number to ensure that the transfer database is compatible with the database requirements of the currently installed version of Replication Option.

A.1.7 DDAL_TNSF_COLUMNS Transfer Database Table

Table A–5 shows the columns for the DDAL_TNSF_COLUMNS table.

Table A–5 Replication Option Transfer Database: DDAL_TNSF_COLUMNS Table

Column Name	Data Type/Size	Description
TRANSFER_NAME	CHAR(31)	Name of transfer
SOURCE_TABLE_NAME	CHAR(31)	Table from source database
SOURCE_COLUMN_NAME	CHAR(31)	Column named in table of source database
SOURCE_DB_HANDLE	CHAR(31)	Database handle of source database[1]

[1]The column contains a value for extraction rollup transfers only.

A.1.8 DDAL_TNSF_TABLES Transfer Database Table

Table A–6 shows the columns for the DDAL_TNSF_TABLES table.

Table A–6 Replication Option Transfer Database: DDAL_TNSF_TABLES Table

Column Name	Data Type/Size	Description
TRANSFER_NAME	CHAR(31)	Name of transfer

(continued on next page)

Table A–6 (Cont.) Replication Option Transfer Database: DDAL_TNSF_TABLES Table

Column Name	Data Type/Size	Description
SOURCE_TABLE_NAME	CHAR(31)	Name of table in source database
SELECTION_EXPR_LEN	NUMBER(9)	Number of rows in the DDAL_ TNSF_SEL_EXPR table to represent the row selection criteria for a given table for a given transfer
SELECTION_EXPR_ID	INTEGER	Uniquely identifies a selection expression stored in the transfer definition under a given transfer
SOURCE_DB_HANDLE	CHAR(31)	Database handle of source database[1]
TARGET_TABLE_NAME	CHAR(31)	Name of table in target database[1]

[1]The column contains a value for extraction rollup transfers only.

A.1.9 DDAL_TNSF_SCHEDULE Transfer Database Table

Table A–7 shows the columns for the DDAL_TNSF_SCHEDULE table.

Table A–7 Replication Option Transfer Database: DDAL_TNSF_SCHEDULE Table

Column Name	Data Type/Size	Description
TRANSFER_NAME	CHAR(31)	Name of transfer
START_TIME	Date	Time for first execution of transfer
RETRY_DELTA_TIME	CHAR(16)	Time to retry transfer if failure occurs
OFFSET_TIME	CHAR(16)	Delta time from beginning of day or month, depending on time units specified in transfer schedule

(continued on next page)

Table A–7 (Cont.) Replication Option Transfer Database: DDAL_TNSF_SCHEDULE Table

Column Name	Data Type/Size	Description
NUM_OF_SCH_OFFSETS	NUMBER(9)	Number of rows in the DDAL_ SCHED_OFFSETS table representing the list of schedule offset times
SCHEDULE_OPTION	NUMBER(9)	Parameter values for the EVERY option
NUM_OF_TRANSFERS	NUMBER(9)	Number of transfers within a week
MAX_RETRY_COUNT	NUMBER(9)	Number of retries specified in transfer definition

A.1.10 DDAL_TNSF_STATUS Transfer Database Table

Table A–8 shows the columns for the DDAL_TNSF_STATUS table.

Table A–8 Replication Option Transfer Database: DDAL_TNSF_STATUS Table

Column Name	Data Type/Size	Description
TRANSFER_NAME	CHAR(31)	Name of transfer
REINI_FLAG	NUMBER(9)	Whether transfer has been reinitialized or not[1]
TRANSFER_STATE	NUMBER(9)	The state (one of six) of the current transfer[2]
LAST_COMPL_STATUS	NUMBER(9)	Status of the last completed transfer[3]
LAST_TRANSFER_TIME	Date	Time of last successful transfer execution[4]

[1]The DDAL_REINI_FLAG column contains a true or false value (binary 1 or 0) to indicate whether or not the transfer has been reinitialized. A true value indicates that a REINITIALIZE TRANSFER statement was issued but the next replication transfer has not yet executed.

[2]Table C–1 lists and describes the six states in which a transfer can exist. The table also lists the decimal values that can appear in the DDAL_TRANSFER_STATE column.

[3]Table C–3 lists and describes the status codes that indicate what occurred the last time the transfer ran.

[4]A zero value in the DDAL_LAST_TRANSFER_TIME column means the transfer has not yet run for the first time or has yet to run successfully.

(continued on next page)

Table A–8 (Cont.) Replication Option Transfer Database: DDAL_TNSF_STATUS Table

Column Name	Data Type/Size	Description
STATE_CHANGE_TIME	Date	Time of last state change

A.1.11 DDAL_TNSF_VIEWS Transfer Database Table

Table A–9 shows the columns for the DDAL_TNSF_VIEWS table.

Table A–9 Replication Option Transfer Database: DDAL_TNSF_VIEWS Table

Column Name	Data Type/Size	Description
TRANSFER_NAME	CHAR(31)	Name of transfer
SOURCE_VIEW_NAME	CHAR(31)	Name of view
TARGET_VIEW_NAME	CHAR(31)	Name of view in target database[1]

[1] Views can be created in a target database using a name other than the one appearing in the source database. This column contains a view name, padded with trailing space characters. If a view is transferred with its original name, that name appears in this column.

A.1.12 DDAL_COMMENT Transfer Database Table

Table A–10 shows the columns for the DDAL_COMMENT table. See Section A.1.2 for an explanation of the origins of this table.

Table A–10 Replication Option Transfer Database: DDAL_COMMENT Table

Column Name	Data Type/Size	Description
TRANSFER_NAME	CHAR(31)	Name of transfer
SEG_NUMBER	NUMBER(9)	Integer value indicating position of the segment in entire sequence of segments
COMMENT_SEG	VARCHAR(512)	One segment of the comment

A.1.13 DDAL_DEST_DPB Transfer Database Table

Table A–11 shows the columns for the DDAL_DEST_DPB table. See Section A.1.2 for an explanation of the origins of this table.

Table A–11 Replication Option Transfer Database: DDAL_DEST_DPB Table

Column Name	Data Type/Size	Description
TRANSFER_NAME	CHAR(31)	Name of transfer
SEG_NUMBER	NUMBER(9)	Integer value indicating position of the segment in entire sequence of segments
DPB_SEG	VARCHAR(512)	One segment of the database parameter block

A.1.14 DDAL_MOVE_CLAUSES Transfer Database Table

Table A–12 shows the columns for the DDAL_MOVE_CLAUSES table. See Section A.1.2 for an explanation of the origins of this table.

Table A–12 Replication Option Transfer Database: DDAL_MOVE_CLAUSES Table

Column Name	Data Type/Size	Description
TRANSFER_NAME	CHAR(31)	Name of transfer
SEG_NUMBER	NUMBER(9)	Integer value indicating position of the segment in entire sequence of segments
MOVE_CLAUSES_SEG	VARCHAR(512)	One segment of the transfer definition's move clauses in ASCII text format

A.1.15 DDAL_SCHED_OFFSETS Transfer Database Table

Table A–13 shows the columns for the DDAL_SCHED_OFFSETS table. See Section A.1.2 for an explanation of the origins of this table.

Table A–13 Replication Option Transfer Database: DDAL_SCHED_OFFSETS Table

Column Name	Data Type/Size	Description
TRANSFER_NAME	CHAR(31)	Name of transfer
OFFSET_NUMBER	NUMBER(9)	Integer value indicating position of the segment in entire sequence of segments
OFFSET_TIME	CHAR(16)	One segment of the list of offset times

A.1.16 DDAL_TNSF_PRIVS Transfer Database Table

Table A–14 shows the columns for the DDAL_TNSF_PRIVS table. See Section A.1.2 for an explanation of the origins of this table.

Table A–14 Replication Option Transfer Database: DDAL_TNSF_PRIVS Table

Column Name	Data Type/Size	Description
TRANSFER_NAME	CHAR(31)	Name of transfer
SEG_NUMBER	NUMBER(9)	Integer value indicating position of the segment in entire sequence of segments
TNSF_PRIVS_SEG	VARCHAR(512)	One segment of the transfer privileges access control list

A.1.17 DDAL_TNSF_SEL_EXPR Transfer Database Table

Table A–15 shows the columns in the DDAL_TNSF_SEL_EXPR table. See Section A.1.2 for an explanation of the origins of this table.

Table A–15 Replication Option Transfer Database: DDAL_TNSF_SEL_EXPR Table

Column Name	Data Type/Size	Description
TRANSFER_NAME	CHAR(31)	Name of transfer

(continued on next page)

Table A–15 (Cont.) Replication Option Transfer Database: DDAL_TNSF_SEL_EXPR Table

Column Name	Data Type/Size	Description
SELECTION_EXPR_ID	INTEGER	Uniquely identifies a selection expression stored in the transfer definition under a given transfer
SEG_NUMBER	NUMBER(9)	Integer value indicating position of the segment in entire sequence of segments
SEL_EXPR_SEG	VARCHAR(512)	One segment of the table select expression, in binary language representation form

A.2 Replication Option System Tables in Source Databases

This section describes the special-purpose system tables created in a source database by Replication Option when a replication database is initialized.

Refer to Section C.6 for information about the access rights granted when these tables are created. In certain cases, access is limited unless you are the system manager or you arrange with the system manager to change the access rights.

A.2.1 RDB$TRANSFERS System Table

The RDB$TRANSFERS system table is shown in Table A–16. A row for each transfer is stored in this table. As each transfer is executed, the corresponding RDB$VINTAGE_TSER column is updated to reflect that RDB$CHANGES rows up to and including this transaction serial number (TSER) have been processed with this execution of the transfer and that these have been applied to the target database.

Table A–16 Replication Option Source Database: RDB$TRANSFERS System Table

Column Name	Data Type/Size	Description
RDB$TRANSFER_NAME	CHAR(31)	Name of transfer

(continued on next page)

Table A–16 (Cont.) Replication Option Source Database: RDB$TRANSFERS System Table

Column Name	Data Type/Size	Description
RDB$VINTAGE_TSER	INTEGER	Transaction serial number used to organize transactions in a sequence to be applied to the target database

A.2.2 RDB$TRANSFER_RELATIONS System Table

The RDB$TRANSFER_RELATIONS system table is shown in Table A–17. For each transfer, a row is stored for each table to be transferred. The rows associated with a particular transfer are deleted when that transfer is deleted.

Table A–17 Replication Option Source Database: RDB$TRANSFER_RELATIONS System Table

Column Name	Data Type/Size	Description
RDB$RELATION_NAME	CHAR(31)	Name of table to be replicated
RDB$TRANSFER_NAME	CHAR(31)	Name of transfer
RDB$TRANSFER_SELECTION_BLR	Varbyte List/ BLR/segment length 512	Row selection criteria in binary language representation

To enhance performance of data lookup in the RDB$TRANSFER_RELATIONS table, an index is used. The index, RDB$TRAN_RELS_REL_NAME_NDX, is a sorted index on the RDB$RELATION_NAME column with duplicate values allowed. This is beneficial when you replicate many tables and have numerous replication transfers on a single source database.

A.2.3 RDB$CHANGES System Table

The RDB$CHANGES system table, shown in Table A–18, contains user updates to the selected source tables. Rows stored in this table are created by Rdb and read by the copy process. Obsolete rows are normally purged by the copy process at the end of each replication transfer. You can use the environment variable DDAL_PURGE to control when such purging is done.

Table A–18 Replication Option Source Database: RDB$CHANGES System Table

Column Name	Data Type/Size	Description
RDB$TRANSACTION_TID	INTEGER	Transaction ID number assigned by Rdb[1]
RDB$TRANSACTION_ SEQUENCE	INTEGER	Replication Option transaction sequence number[2]
RDB$TRANSACTION_TSER	INTEGER	Transaction serial number[3]
RDB$TRANSACTION_ CHANGES	VARCHAR(10000)	Stores changes to be forwarded to the replication database[4]

[1]The transaction ID allows Replication Option to identify all changes made within a single transaction. Rdb assigns the transaction ID when the transaction is started.

[2]The transaction sequence number is used to order RDB$CHANGES rows within the same transaction.

[3]Rdb assigns the transaction serial number when the transaction is committed. The transaction serial number is used to apply transactions to the target database in the order they are committed in the source database. Transaction serial numbers are assigned in ascending order, although the full range of numbers is not always used. Gaps in the numeric sequence typically occur after all users detach from the source database.

[4]The RDB$TRANSACTION_CHANGES column stores in detailed, encoded form the rows added to, changed in, or deleted from tables in the source database. Rdb encodes the changes stored in this column in a form similar to binary language representation (BLR) but in a form specific to this column.

A.2.4 RDB$CHANGES_MAX_TSER System Table

The RDB$CHANGES_MAX_TSER table, shown in Table A–19, is maintained and used by Rdb.

Table A–19 Replication Option Source Database: RDB$CHANGES_MAX_TSER System Table

Column Name	Data Type/Size	Description
RDB$TRANSACTION_TSER	INTEGER	Transaction serial number

A.3 Replication Option System Tables in Target Databases

To manage transfers, Replication Option creates one, two, or three tables of its own in a target database. The tables used for Rdb have different names than those used for all other target databases.

- Tables created for all transfers are the RDB$VINTAGE table for Rdb and the DDAL$TRANSFER_INFO table for all other target databases.

- Tables created for transfers to existing databases are the RDBVMS$ TRANSFER_OWNER table for Rdb, and the DDAL$TRANSFER_OWNR table for all other target databases.
- A table created for transfers to existing databases in which shared writing of target tables will be allowed is the DDAL$INDEXES table.

Table A–20 shows which tables are added to a target database by Replication Option to manage the transfer process.

Table A–20 Replication Option Tables in a Target Database

Rdb Databases	Non-Rdb Databases	Circumstances
RDB$VINTAGE	DDAL$TRANSFER_INFO	All transfer types
RDBVMS$TRANSFER_OWNER	DDAL$TRANSFER_OWNR	Transfers to existing databases
DDAL$INDEXES	DDAL$INDEXES	Shared-write transfers to existing databases

None of these tables is truly a system table. However, Replication Option treats them as system tables to distinguish them from customer application tables.

A.3.1 RDB$VINTAGE and DDAL$TRANSFER_INFO Tables

The RDB$VINTAGE or DDAL$TRANSFER_INFO table is created in each extraction, extraction rollup, and replication database. This table behaves like a user table rather than a system table. When you issue the SHOW TABLES statement on the target database, RDB$VINTAGE or DDAL$TRANSFER_ INFO and its columns are listed with the user tables and columns rather than with the system tables and columns.

The RDB$VINTAGE_TSER or DDAL$TRANSFER_TSER column exists in all target databases but is used only in replication transfers. The other columns exist in all target databases and are used for all transfer types.

In a replication transfer, the copy process updates the time and date stored in this table after each group of rows associated with a user transaction in the source database is applied to the target database. In extraction, extraction rollup, and initial replication transfers, the copy process updates the time and date stored in this table, giving the time when the transfer began.

With the ability to target several transfers to a single database, the RDB$VINTAGE or DDAL$TRANSFER_INFO table has multiple rows of data, one row per transfer. Because transfers can originate from different VMScluster systems or standalone systems, it is possible for each system to have duplicate transfer names. Such transfers can be distinguished from each other by the node/host name stored in one of the RDB$VINTAGE or DDAL$TRANSFER_INFO table columns.

Table A–21 lists the columns found in the RDB$VINTAGE or DDAL$TRANSFER_INFO table, shows the data type, and provides a description.

Table A–21 Replication Option Target Database: RDB$VINTAGE or DDAL$TRANSFER_INFO System Table

Column Name	Data Type/Size	Description
RDB$VINTAGE_TSER / DDAL$TRANSFER_TSER	INTEGER	Transaction serial number[1]
RDB$VINTAGE_TIMESTAMP / DDAL$TRANSFER_TIME	Date	The date and time for transfers [2]
RDBVMS$VINTAGE_TRANSFER_NODE / DDAL$TRANSFER_NODE	CHAR(255)[7]	Name of the node on which the transfer was first run[3]
RDBVMS$VINTAGE_TRANSFER_ADDR [6]	VARCHAR(402)	Reserved for future use
RDBVMS$VINTAGE_TRANSFER_NAME / DDAL$TRANSFER_NAME	CHAR(31)	Name of the transfer

[1] Replication Option uses this column to determine which updated rows in the RDB$CHANGES table in the source database are to be forwarded to the replication database.

[2] Replication Option stores in this column either the date and time at the source node of the most recent update by Replication Option to replicated tables, or the date and time that the extraction, extraction rollup, or initial replication transfer was begun. Each subsequent update to replicated tables causes the date and time in this column to change.

[3] The name is filled to the end with trailing space characters. The length of this column is 255 characters. This column will not contain a cluster alias name. A transfer from any node in a cluster is treated as if it were executing on the same node as the original transfer.

[6]There is no counterpart to this column in the DDAL$TRANSFER_INFO table.

[7]The size of this column has been increased from 20 (effective with version 7.0). However it is not automatically converted in existing target databases to this longer length. You must do so manually if you intend to transfer using TCP/IP.

(continued on next page)

Table A–21 (Cont.) Replication Option Target Database: RDB$VINTAGE or DDAL$TRANSFER_INFO System Table

Column Name	Data Type/Size	Description
RDBVMS$VINTAGE_TRANSFER_ID / DDAL$TRANSFER_ID	NUMBER(9)	Abbreviated transfer identifier[4]
RDBVMS$VINTAGE_TRANSFER_BUSY / DDAL$TRANSFER_BUSY	CHAR	Indicates a transfer is in progress[5]

[4] Identifies the transfer in abbreviated form. It must be unique within a given target database. Copy processes that happen to be executing at the same time take steps to ensure the uniqueness of the transfer ID they assign.

[5] Flags the fact that a transfer is in progress. With earlier versions of Replication Option, applications wishing to use a target database could test the value of the RDB$VINTAGE_TIMESTAMP column to find out if the transfer had completed. When a transfer into an existing database is re-executed, this test is no longer possible. A flag column for this purpose has been added. The values stored in this column are either "T" or "F", meaning TRUE or FALSE. The busy flag is set to "T" only during transfers where Replication Option performs a complete replacement of the data. In particular, this flag is not set when Replication Option performs a replication update.

A.3.2 RDBVMS$TRANSFER_OWNER and DDAL$TRANSFER_OWNR Table

The RDBVMS$TRANSFER_OWNER or DDAL$TRANSFER_OWNR table keeps track of which transfers own which objects in the target database. Although one name begins with RDBVMS$, it is a system table in name only. Rdb knows it as a user table, as is the case now with RDB$VINTAGE. Table A–22 lists the columns found in the RDBVMS$TRANSFER_OWNER or DDAL$TRANSFER_OWNR table, shows the data type, and provides a description. This table appears in a target database if only that database is the target of a TO EXISTING transfer.

Table A–22 Replication Option Target Database: RDBVMS$TRANSFER_ OWNER or DDAL$TRANSFER_OWNR Table

Column Name	Data Type/Size	Description
RDBVMS$TRANSFER_ OWNER_ID / DDAL$TRANSFER_ ID	NUMBER(9)	Short form of the full transfer identifier[1]
RDBVMS$TRANSFER_ OBJECT_TYPE / DDAL$OBJECT_ TYPE	NUMBER(9)	Identifies the type of object owned by the transfer[2]
RDBVMS$TRANSFER_ OBJECT_NAME / DDAL$OBJECT_ NAME	CHAR(31)	Names the object whose type is specified in the RDBVMS$TRANSFER_ OBJECT_TYPE or DDAL$OBJECT_TYPE column[3]

[1]This is the short form of the full transfer identifier corresponding to the column RDBVMS$VINTAGE_TRANSFER_ID in RDB$VINTAGE or to the DDAL$TRANSFER_ID column in the DDAL$TRANSFER_INFO table. Rows appear in this table for as many times as there are objects that a given transfer owns.

[2]Valid object types and their binary numeric codes include:

Domain: 0
Table: 1
View: 2
Index: 3

[3]Because an object might have a different name in the target database than its counterpart in the source database, the object name here is the target name.

A.3.3 DDAL$INDEXES Table

The DDAL$INDEXES table is used for shared writing of target tables when the transfer method used is REPLICATION. It lists the names of indexes on the target tables. Table A–23 lists the columns in the DDAL$INDEXES table.

Table A–23 Replication Option Target Database: DDAL$INDEXES Table

Column Name	Data Type/Size	Description
DDAL$TABLE_NAME	CHAR(31)	Name of the table
DDAL$INDEX_NAME	CHAR(31)	Name of the index
DDAL$INDEX_ NUMBER	INTEGER	A unique numeric value assigned to each index

B

Replication Option Environment Variables

This appendix lists and describes the environment variables included with Replication Option software that you can define. It does not include environment variables whose definitions are established by Replication Option and that you should not change.

B.1 Customizing Replication Option Using Environment Variables

Apart from the SQL language you use to create and execute Replication Option transfers, you can control the operational behavior of Replication Option by defining environment variables.

There are two classes of environment variables used by Replication Option: system environment variables and user environment variables.

- System environment variables

  System environment variables are those intended to apply to all of Replication Option. They are normally system logical names and are defined by executing DDAL$START_TR_MON.COM, the Replication Option startup procedure. Replication Option does not use a system configuration file for system logical names.

- User environment variables

  User environment variables are variables that can be applied to individual user accounts under which specific Replication Option transfers are run. User environment variables are typically designed to alter the behavior of a copy process during transfer execution.

  You can use either of the following two methods to define user-level environment variables:

  – You can define OpenVMS environment variables, commonly known as logical names. If the names are to apply only for a particular OpenVMS account, they should be defined in that account's LOGIN.COM file as process-level or job-level logical names. If they are to apply to more than 1 account, they can be defined as group-level or system-level

logical names. In this case, they should be defined in a procedure that is invoked on system startup instead of defining them in an individual user's LOGIN.COM file.

- You can define environment variables within a user-level configuration file named DDAL.RC. This file is used only by the copy process executable image. The configuration file is located in the user's login directory, SYS$LOGIN. Environment variables defined in the DDAL.RC file affect only those transfers that are executed under an account that uses that DDAL.RC file. Transfers can run under different user accounts, each with its own DDAL.RC file.

  See Section B.20 for rules regarding resource configuration files.

B.1.1 Environment Variables for Replication Option

You can define the following environment variables to affect how a transfer executes. Each of them is described in its own section later in this appendix.

- DDAL$CHANGES_BUFFER_SIZE
- DDAL$CHANGES_INDEX
- DDAL$CHECK_TARGET_DEFINITIONS
- DDAL$CHECKPOINT_DELAY_IN_MINUTES
- DDAL$CHECKPOINT_MAX_RETRIES
- DDAL$CP_CONTINUE
- DDAL$CP_LOG_OPTIONS
- DDAL$CP_READ_CHANGES
- DDAL$EXTRACT_DATA
- DDAL$LOG_TRMON_CLIENT_EVENTS
- DDAL$MAX_COPY_PROCESS
- DDAL_NUMBER_OF_DATABASES
  - DDAL_USERNAME_n "<username>"
  - DDAL_PASSWORD_n "<password>"
  - DDAL_DATABASE_n "<attach string>"
- DDAL$PURGE
- DDAL$RW_ROW_COUNT
- DDAL$SHARED_WRITE

- DDAL$SINGLE_DATA_COMMIT
- DDAL$START_WITH_SOURCE_TABLE
- DDAL$STORE_DEF_SEPARATE
- DDAL$TARGET_INDEX_OPTIONS

B.1.2 General Usage Guidelines for Replication Option Environment Variables

The following guidelines generally apply to all Replication Option environment variables; any exceptions are noted in the individual descriptions of the variables.

- Most of the environment variables have a generic name and a transfer-specific name (formed by appending an underscore and a transfer name to the generic name).
- If you supply the name of a specific transfer, the environment variable will affect only that transfer.
- If you define both the generic form and the transfer-specific form, the transfer-specific form takes precedence for that specific transfer.
- On an OpenVMS system, some environment variables can only be defined as logical names; others can be either defined as logical names or entered in a configuration file named DDAL.RC. For information about this for a specific variable, refer to the individual description of that variable later in this appendix.
- If you define the generic form of the environment variable as an OpenVMS logical name, the value you specify applies to all transfers executed on that processor.
- If you define the generic form in a DDAL.RC file, the value you supply applies only to those transfers that use that DDAL.RC file.
- If you define an environment variable both as a logical name and as an entry in the configuration file, the configuration file entry takes precedence over the logical name definition.
- If you plan to use environment variables as logical names in a cluster environment, you must define them on each processor that has Replication Option installed. Because the copy process can run on any machine in a cluster, defining logical names on each machine in a cluster ensures that the copy process has access to the definitions.

- If you define environment variables as logical names, you can type SHOW LOGICAL DDAL* at the OpenVMS system prompt to display a list of all Replication Option logical names.
- In most Replication Option environment variable names, an underscore can be substituted for the dollar sign. Replication Option first looks for names beginning with DDAL$. If that fails, it then looks for names beginning with DDAL_. For example, although Table B–1 shows DDAL$CHANGES_INDEX definition as the format for that variable, you could use DDAL_CHANGES_INDEX definition instead.

  The environment variables that are exceptions, in that they have required formats, are listed in Table B–2.
- Environment variable names and definitions are generally not case-sensitive; exceptions are noted in individual cases. They can be entered in uppercase, mixed-case, or in lowercase. They will be converted by Replication Option to uppercase.
- For environment variables that take a yes or no definition, Replication Option converts any combination of uppercase or lowercase yes or no responses to uppercase.[1]

  For example, strings such as the following are interpreted as a YES:

  ```
  Yes
  YES
  yes
  ```

  Similarly, all of the following strings will be interpreted as a NO:

  ```
  No
  NO
  no
  ```

Table B–1 shows the functions and the valid formats for the Replication Option environment variables. If the environment variable is shown with *definition* following the environment variable, Replication Option provides a set of valid definitions; you enter only 1 of those definitions when you define the environment variable. See the section on the specific environment variable for a list and discussion of definitions supplied by Replication Option. You can enter an environment variable definition of YES or NO for environment variables

[1] The one environment variable with different behavior is the DDAL$PURGE environment variable. This is the only environment variable where Replication Option parses only the first character of the variable definition. If the first character is either an uppercase or lowercase "y", Replication Option interprets the response as "Yes". If the first character is either an uppercase or lowercase "n", Replication Option interprets the response as "No". Replication Option does not read beyond the first character.

shown with yes/no following their name in Table B–1. If the environment variable name is followed by n, you enter a numeric value. You would replace transfername with the name of an actual Replication Option transfer.

Table B–1 Functions and Formats for Replication Option Environment Variables

Function	Valid Formats
Changing the size of the virtual memory buffer used for checkpoint restart of replication transfers	• DDAL$CHANGES_BUFFER_SIZE n • DDAL$CHANGES_BUFFER_SIZE_transfername n
Adding indexes to improve performance of replication transfers	• DDAL$CHANGES_INDEX definition • DDAL$CHANGES_INDEX_transfername definition
Suppressing comparison of data definitions between source and target tables	• DDAL$CHECK_TARGET_DEFINITIONS yes/no • DDAL$CHECK_TARGET_DEFINITIONS_transfername yes/no
Setting a delay to be used by the copy process (thereby overriding the default of 1 minute) during checkpoint restart	• DDAL$CHECKPOINT_DELAY_IN_MINUTES n • DDAL$CHECKPOINT_DELAY_IN_MINUTES_ transfername n
Setting the number of retries to be used by the copy process (thereby overriding the default of 10 retries) during checkpoint restart	• DDAL$CHECKPOINT_MAX_RETRIES n • DDAL$CHECKPOINT_MAX_RETRIES_transfername n
Logging information about source database rows during a replication update transfer	• DDAL$CP_LOG_OPTIONS definition • DDAL$CP_LOG_OPTIONS_transfername definition

(continued on next page)

Table B–1 (Cont.) Functions and Formats for Replication Option Environment Variables

Function	Valid Formats
Avoiding stalls for protected read operations on the source database during replication transfers	• DDAL$CP_READ_CHANGES definition • DDAL$CP_READ_CHANGES_*transfername* definition
Disabling the extraction of data during a transfer	• DDAL$EXTRACT_DATA yes/no • DDAL$EXTRACT_DATA_*transfername* yes/no
Logging events used to diagnose communication problems between the transfer monitor and user interface	• DDAL$LOG_TRMON_CLIENT_EVENTS definition • DDAL$LOG_TRMON_CLIENT_EVENTS_*transfername* definition
Enabling purging rows no longer needed in the RDB$CHANGES table by the copy process at the end of a replication transfer	• DDAL$PURGE yes/no [1] • DDAL$PURGE_*transfername* yes/no [1]
Setting the number of rows to be read and written at a time during a transfer	• DDAL$RW_ROW_COUNT n • DDAL$RW_ROW_COUNT_*transfername* n
Enabling writing of data to tables owned by other databases or transfers	• DDAL$SHARED_WRITE yes/no • DDAL$SHARED_WRITE_*transfername* yes/no

[1]The DDAL$PURGE environment variable is the one environment variable where Replication Option parses only the first character of the variable definitions. If the first character is either an uppercase or lowercase "y", Replication Option interprets the response as "Yes". If the first character is either an uppercase or lowercase "n", Replication Option interprets the response as "No". Replication Option does not read beyond the first character.

(continued on next page)

Table B–1 (Cont.) Functions and Formats for Replication Option Environment Variables

Function	Valid Formats
Enabling deleting and writing of data to target tables in one transaction without having to drop constraints for transfers to existing databases	• DDAL$SINGLE_DATA_COMMIT yes/no • DDAL$SINGLE_DATA_COMMIT_*transfername* yes/no
Improving performance for concurrently executing extraction or initial replication transfers	• DDAL$START_WITH_SOURCE_TABLE definition • DDAL$START_WITH_SOURCE_TABLE_*transfername* definition
Performing concurrent initial replication transfers using the same source database	• DDAL$STORE_DEF_SEPARATE yes/no • DDAL$STORE_DEF_SEPARATE_*transfername* yes/no
Allowing customer applications and replication transfers to write data into the same tables	• DDAL$TARGET_INDEX_OPTIONS n • DDAL$TARGET_INDEX_OPTIONS_*transfername* n

Table B–2 shows the environment variables for Replication Option that have only one valid format.

Table B–2 Environment Variables with Specific Formats

Function	Valid Format
Checking status of transfers during execution	DDAL$CP_CONTINUE definition
Setting the maximum number of concurrent transfers Replication Option will attempt to execute	DDAL$MAX_COPY_PROCESS n

(continued on next page)

Table B–2 (Cont.) Environment Variables with Specific Formats

Function	Valid Format
Specifying the maximum number of databases you can specify to be attached to	DDAL_NUMBER_OF_DATABASES n • DDAL_USERNAME_n—Specifies the user name used in attaching to the database • DDAL_PASSWORD_n—Specifies the password used in attaching to the database • DDAL_DATABASE_n—Specifies a database attach string

These environment variables are described in more detail in the rest of this appendix.

B.2 DDAL$CHANGES_BUFFER_SIZE Environment Variable

When an error accessing the target database occurs, such as a temporary network failure, checkpoint restart provides a mechanism for recovering quickly once the error is corrected. This is true for any type of transfer. However, for a copy process to be able to do this during a replication update, the copy process must be able to store all the data for the current transaction in virtual memory so that it can restart the transaction without also having to restart the source transaction. See Section 6.11 for more information about using checkpoints.

The default size of the virtual memory buffer is 10,000 bytes, enough to hold relatively small transactions.

You can change the default size for the virtual memory buffer, by defining the DDAL$CHANGES_BUFFER_SIZE environment variable as an OpenVMS logical name. Use one of the following formats:

```
$ DEFINE DDAL$CHANGES_BUFFER_SIZE n
$ DEFINE DDAL$CHANGES_BUFFER_SIZE_transfername n
```

Alternatively, you can define the DDAL$CHANGES_BUFFER_SIZE environment variable by including a line similar to one of the following lines in your resource configuration file, DDAL.RC:

```
DDAL$CHANGES_BUFFER_SIZE n
DDAL$CHANGES_BUFFER_SIZE_transfername n
```

Replace `n` with a simple, non-0 decimal string, consisting only of the characters 0 through 9. It must be a value between 1000 and 8388000.

Replace transfername with the name of a specific transfer to which the environment variable applies.

For general information about defining this environment variable, including the effects of using generic and transfer-specific forms, refer to Section B.1.2.

B.3 DDAL$CHANGES_INDEX Environment Variable

Adding indexes on the RDB$CHANGES table can improve performance of replication transfers.

See Section 6.17.4 for a description of indexing benefits and applications considerations.

By default, RDB$CHANGES is not indexed. You can override this default by defining DDAL$CHANGES_INDEX as an OpenVMS logical name using one of the following formats:

```
$ DEFINE DDAL$CHANGES_INDEX definition
$ DEFINE DDAL$CHANGES_INDEX_transfername definition
```

Alternatively, you can define the DDAL$CHANGES_INDEX environment variable by including a line similar to one of the following lines in your resource configuration file, DDAL.RC:

```
DDAL$CHANGES_INDEX definition
DDAL$CHANGES_INDEX_transfername definition
```

Replace definition with one of the definitions shown in Table B–3. The definition is not case-sensitive.

Replace transfername with the name of a specific transfer to which the environment variable applies.

For general information about defining this environment variable, including the effects of using generic and transfer-specific forms, refer to Section B.1.2.

Table B–3 shows the valid definitions for the environment variable DDAL$CHANGES_INDEX.

Table B–3 Definitions for DDAL$CHANGES_INDEX

String	Meaning
CREATE_ON_UPDATE	Create indexes during the replication update cycle, if necessary.
DROP	Drop indexes during replication initialization, reinitialization, update, or deletion.
NONE	No indexing desired. Queries will assume indexes do not exist.
USE	Create indexes during replication initialization or reinitialization, if necessary. During replication update, assume RDB$CHANGES is indexed.

B.4 DDAL$CHECK_TARGET_DEFINITIONS Environment Variable

The DDAL$CHECK_TARGET_DEFINITIONS environment variable pertains to transfers to existing databases. It is meaningful any time an extraction transfer is re-executed or a replication transfer is reinitialized and re-executed. Normally, during such transfers Replication Option compares source and target table definitions to ensure they match. This comparison can be suppressed by using DDAL$CHECK_TARGET_DEFINITIONS.

The following scenario is one example of why you might want to suppress comparison of source and target table definitions. Suppose you need to convert an Rdb date column into a format needed by a particular target system. You could do so by adding a second date column and a trigger definition to the target table after the first transfer has executed. The trigger would read the value of the first date column and update the row by storing a date in the new format into the second date column. If you did not use DDAL$CHECK_ TARGET_DEFINITIONS to suppress comparison, then upon reinitialization and re-execution of the transfer, Replication Option would see an extra column in the target table, assume it had been deleted from the source table, and so delete it from the target table as well.

Note

If you suppress comparison of data definitions, you must remember to manually make any changes to the target tables that you make to the source tables. Otherwise, the transfer might fail.

Refer to Chapter 8 for details about transferring to existing databases.

Use one of the following formats to define this environment variable as an OpenVMS logical name:

```
$ DEFINE DDAL$CHECK_TARGET_DEFINITIONS Yes
$ DEFINE DDAL$CHECK_TARGET_DEFINITIONS_transfername Yes
```

Alternatively, use one of the following formats to define this environment variable in your resource configuration file, DDAL.RC:

```
DDAL$CHECK_TARGET_DEFINITIONS yes
DDAL$CHECK_TARGET_DEFINITIONS_transfername yes
```

Replace `transfername` with the name of a specific transfer to which the environment variable applies.

For general information about defining this environment variable, including the effects of using generic and transfer-specific forms, refer to Section B.1.2.

B.5 DDAL$CHECKPOINT_DELAY_IN_MINUTES and DDAL$CHECKPOINT_MAX_RETRIES Environment Variables

The use of these environment variables is related to the CHECKPOINT RESTART option in the CREATE TRANSFER statement. For example, the following statement allows you to specify a recovery time interval following an error accessing the target database.

```
CHECKPOINT RECOVER WITHIN 45 MINUTES
```

The original purpose for implementing checkpoint restart was to deal with unreliable network links. If a network link went down in the middle of a transfer, the transfer would enter a recovery mode. The copy process would delay for 1 minute and then attempt to reattach to the target database to restart the failed transaction. The checkpoint clause in the preceding example instructs Replication Option to repeatedly attempt to reattach until either it successfully restarts the failed transaction or the 45 minute grace period has expired.

Because restarting a failed transaction is no guarantee that it will ever complete successfully, it would be possible for the operation to be repeated forever without success. To prevent this, the default maximum number of transaction retries is 10.

Because the default maximum number of transaction retries is 10 and the inter-recovery delay period is 1 minute, the checkpoint restart feature will not help in a deadlock or lock-conflict situation that lasts more than 10 minutes.

See Section 6.11 for more information about using checkpoints.

To give you some flexibility in this situation, there are two environment variables that the copy process looks for:

- DDAL$CHECKPOINT_DELAY_IN_MINUTES m
- DDAL$CHECKPOINT_MAX_RETRIES n

DDAL$CHECKPOINT_DELAY_IN_MINUTES m

The DDAL$CHECKPOINT_DELAY_IN_MINUTES variable is optional. If the variable is not defined, the default delay used by a copy process is 1 minute.

You can change the default delay time by defining the DDAL$CHECKPOINT_DELAY_IN_MINUTES environment variable as an OpenVMS logical name. Use one of the following formats:

```
$ DEFINE DDAL$CHECKPOINT_DELAY_IN_MINUTES m
$ DEFINE DDAL$CHECKPOINT_DELAY_IN_MINUTES_transfername m
```

Alternatively, you can define the DDAL$CHECKPOINT_DELAY_IN_MINUTES environment variable by including a line similar to one of the following lines in your resource configuration file, DDAL.RC:

```
DDAL$CHECKPOINT_DELAY_IN_MINUTES m
DDAL$CHECKPOINT_DELAY_IN_MINUTES_transfername m
```

Replace `m` with a value between 1 and 60. Any other value will result in a warning message in the copy process log file and the default value being used.

Although not a requirement, it is best to set the delay in minutes to be a number less than the recovery period specified in the CREATE TRANSFER statement. For example, if the delay is 10 minutes and the CREATE TRANSFER statement specifies CHECKPOINT RECOVER WITHIN 5 MINUTES, a recovery will delay 10 minutes. If the transaction then fails, an error message will be logged showing that the recovery period of 5 minutes has been exceeded.

Replace `transfername` with the name of the transfer to which the environment variable applies.

For general information about defining this environment variable, including the effects of using generic and transfer-specific forms, refer to Section B.1.2.

DDAL$CHECKPOINT_MAX_RETRIES n

The DDAL$CHECKPOINT_MAX_RETRIES variable is optional. If the variable is undefined, the default number of retries used by a copy process is 10.

You can change the default delay time by defining the DDAL$CHECKPOINT_MAX_RETRIES environment variable as an OpenVMS logical name. Use one of the following formats:

```
$ DEFINE DDAL$CHECKPOINT_MAX_RETRIES n
$ DEFINE DDAL$CHECKPOINT_MAX_RETRIES_transfername n
```

Alternatively, you can define the DDAL$CHECKPOINT_MAX_RETRIES environment variable by including a line similar to one of the following lines in your resource configuration file, DDAL.RC:

```
DDAL$CHECKPOINT_MAX_RETRIES n
DDAL$CHECKPOINT_MAX_RETRIES_transfername n
```

Replace `n` with a value between 1 and 999. Any other value will result in a warning message in the copy process log file and in the default value being used instead.

Replace `transfername` with the name of a specific transfer to which the environment variable applies.

For general information about defining this environment variable, including the effects of using generic and transfer-specific forms, refer to Section B.1.2.

B.6 DDAL$CP_CONTINUE Environment Variable

You can use the definitions associated with the environment variable DDAL$CP_CONTINUE for the following purposes:

- Check the status of your prologue or epilogue command procedures
- Check the status of your transfer during execution
- Affect your transfer during execution

See Section 7.2 for more information on the appropriate use of DDAL$CP_CONTINUE.

Note

The DDAL$CP_CONTINUE environment variable should not be defined in a resource configuration file. It is a variable that needs to be dynamically defined, first by the transfer monitor to establish its initial value, then by the user prologue procedure (if a prologue procedure is named for the transfer), then by the copy process itself, and finally by the user epilogue procedure (if an epilogue procedure is named for the transfer).

You can define DDAL$CP_CONTINUE as an OpenVMS logical name in your prologue or epilogue procedures using the following format:

```
$ DEFINE DDAL$CP_CONTINUE definition
```

Replace `definition` with one of the definitions shown in Table B–4.

You can check for these definitions in your prologue or epilogue files and assign a different string name to affect how a transfer executes.

Table B–4 shows the valid return strings and exit status values for the DDAL$CP_CONTINUE environment variable and their meaning.

Table B–4 Definitions for DDAL$CP_CONTINUE

Return String	Meaning
SUCCESS	The copy process image performed the transfer successfully and either no prologue existed or one of the following actions occurred: the prologue succeeded or the prologue did not cause DDAL$CP_CONTINUE to change (that is, the prologue did no error handling of its own). Replication Option read the return string of SUCCESS in DDAL$CP_CONTINUE. The copy process lets the transfer continue.
PRO_FAIL	The prologue failed and caused DDAL$CP_CONTINUE to change to reflect that failure. As a result, the copy process image intentionally exited without performing the transfer. Replication Option read the return string of PRO_FAIL in DDAL$CP_CONTINUE. The prologue procedure decided that the transfer cannot be automatically retried. The epilogue file executes, if one exists.
PRO_RETRY	The prologue failed and caused DDAL$CP_CONTINUE to change to reflect that failure. Replication Option read the return string of PRO_RETRY in DDAL$CP_CONTINUE. As a result, the copy process image intentionally exited without performing the transfer. However, the transfer can be retried automatically. The epilogue file executes, if one exists.
FAIL	The copy process image attempted the transfer upon seeing SUCCESS in DDAL$CP_CONTINUE but the copy process encountered a severe error or one that the transfer monitor does not recognize and understand. Such an error means that Replication Option cannot retry the transfer automatically. Replication Option read the return string of FAIL in DDAL$CP_CONTINUE.

(continued on next page)

Table B–4 (Cont.) Definitions for DDAL$CP_CONTINUE

Return String	Meaning
RETRY	The copy process image attempted the transfer upon seeing SUCCESS in DDAL$CP_CONTINUE but the transfer failed. Replication Option read the return string of RETRY in DDAL$CP_ CONTINUE. The transfer monitor recognized the cause of the failure as recoverable and initiates the retry schedule.
EPI_FAIL	The epilogue procedure was run, and either the epilogue procedure or DDAL$EPILOGUE.COM (if the epilogue procedure did not work as expected) detected a fatal error. Replication Option read the return string of EPI_FAIL in DDAL$CP_CONTINUE. This code can also mean that an unknown value appeared in DDAL$CP_ CONTINUE upon completion of the epilogue.
EPI_RETRY	The epilogue failed. The epilogue procedure was run and either the epilogue procedure or DDAL$EPILOGUE.COM (if the epilogue procedure did not work as expected) detected an error. Replication Option read the return string of EPI_RETRY in DDAL$CP_ CONTINUE. However, the transfer can be automatically retried.

B.7 DDAL$CP_LOG_OPTIONS Environment Variable

The copy process can log information about source database rows during a replication update transfer. See Section 6.10.2 for detailed information. By default, this information is not logged.

You can override this default by defining the DDAL$CP_LOG_OPTIONS environment variable as an OpenVMS logical name using one of the following formats:

```
$ DEFINE DDAL$CP_LOG_OPTIONS keyword
$ DEFINE DDAL$CP_LOG_OPTIONS_transfername keyword
```

Alternatively, you can define the DDAL$CP_LOG_OPTIONS environment variable by including a line similar to one of the following in your resource configuration file, DDAL.RC:

```
DDAL$CP_LOG_OPTIONS keyword
DDAL$CP_LOG_OPTIONS_transfername keyword
```

Replace *keyword* with one of the definitions shown in Table B–5. The definitions are not case-sensitive.

Replace *transfername* with the name of a specific transfer to which the environment variable applies.

For general information about defining this environment variable, including the effects of using generic and transfer-specific forms, refer to Section B.1.2.

Table B–5 shows the valid keywords for the environment variable DDAL$CP_LOG_OPTIONS.

Table B–5 Definitions for DDAL$CP_LOG_OPTIONS

Keyword	Meaning
SRC_DB_KEYS	Enables logging
NO_SRC_DB_KEYS	Disables logging

B.8 DDAL$CP_READ_CHANGES Environment Variable

Normally, while there are active applications writing in RDB$CHANGES, a copy process does not read the RDB$CHANGES table. Instead, the copy process forces a quiet point where it can start a protected read transaction. Although this method does have advantages, it can cause the copy process to stall for a time until there are no writers in RDB$CHANGES. By defining DDAL$CP_READ_CHANGES you can control whether or not a copy process protects against concurrent writers while reading RDB$CHANGES. The default is to protect against concurrent writers. However, if you are going to change the default. one of the following two cases must be true:

- Ensure that update transactions on the source database commit far apart. Running them serially will achieve this goal.
- Ensure the copy process runs when there are no applications writing to the source database.

Refer to Section 6.17.3 for more information about this copy process behavior.

You can define the DDAL$CP_READ_CHANGES environment variable as an OpenVMS logical name using one of the following formats:

```
$ DEFINE DDAL$CP_READ_CHANGES keyword
$ DEFINE DDAL$CP_READ_CHANGES_transfername keyword
```

Alternatively, you can define the DDAL$CP_READ_CHANGES environment variable by including a line similar to one of the following in your resource configuration file, DDAL.RC:

```
DDAL$CP_READ_CHANGES keyword
DDAL$CP_READ_CHANGES_transfername keyword
```

Replace keyword with one of the valid definitions shown in Table B–6. The definition is not case-sensitive.

Replace transfername with the name of a specific transfer to which the environment variable applies.

For general information about defining this environment variable, including the effects of using generic and transfer-specific forms, refer to Section B.1.2.

Table B–6 shows the valid definitions for the environment variable DDAL$CP_READ_CHANGES.

Table B–6 Definitions for DDAL$CP_READ_CHANGES

Keyword	Meaning
CONCURRENT	Requests that the copy process read RDB$CHANGES without protecting against concurrent writers.
SERIAL	Requests the copy process to protect against concurrent writers. This is the default.

B.9 DDAL$EXTRACT_DATA Environment Variable

During extraction and initial replication transfers, Replication Option creates table definitions in the target database and then transfers data into those tables from the source database. The DDAL$EXTRACT_DATA variable allows you to disable the extraction of the data itself and to transfer only the data definitions.

See Section 4.4.2 for an example of how this environment variable can be used. Also, see Appendix C for a description of the steps that occur during extraction and initial replication transfers.

Use one of the following formats to define this environment variable as an OpenVMS logical name:

```
$ DEFINE DDAL$EXTRACT_DATA yes/no
$ DEFINE DDAL$EXTRACT_DATA_transfername yes/no
```

Alternatively, use one of the following formats to define this environment variable in your resource configuration file, DDAL.RC:

```
DDAL$EXTRACT_DATA yes/no
DDAL$EXTRACT_DATA_transfername yes/no
```

Replace transfername with the name of a specific transfer to which the environment variable applies.

For general information about defining this environment variable, including the effects of using generic and transfer-specific forms, refer to Section B.1.2.

B.10 DDAL$LOG_TRMON_CLIENT_EVENTS Environment Variable

This environment variable enables the logging of events used to help diagnose communication problems between the transfer monitor and the user interface (for example, interactive SQL), and an intermediary called DDAL$SHR.EXE. Enabling event logging allows you to see:

- When messages are received by DDAL$SHR.EXE from the user interface (SQL)
- When DDAL$SHR.EXE sends and receives messages to/from the transfer monitor
- When DDAL$SHR.EXE replies to the user interface

You can define the DDAL$LOG_TRMON_CLIENT_EVENTS environment variable as an OpenVMS logical name using one of the following formats:

```
$ DEFINE DDAL$LOG_TRMON_CLIENT_EVENTS keyword
$ DEFINE DDAL$LOG_TRMON_CLIENT_EVENTS_transfername keyword
```

Alternatively, you can define the DDAL$LOG_TRMON_CLIENT_EVENTS environment variable by including a line similar to one of the following in your resource configuration file, DDAL.RC:

```
DDAL$LOG_TRMON_CLIENT_EVENTS keyword
DDAL$LOG_TRMON_CLIENT_EVENTS_transfername keyword
```

Replace *keyword* with one or more of the valid definitions listed later in this section. The keywords are not case-sensitive. When multiple keywords are used, separate them with spaces or commas.

Replace *transfername* with the name of a specific transfer to which the environment variable applies.

For general information about defining this environment variable, including the effects of using generic and transfer-specific forms, refer to Section B.1.2.

The valid definitions for the environment variable DDAL$LOG_TRMON_CLIENT_EVENTS are the following:

ALL
ALTER
CREATE
DROP
GRANT
MESSAGE
REINITIALIZE
REVOKE

SHOW
START
STOP

Most of these keywords (except ALL and MESSAGE) are SQL statements to Replication Option. Keywords can be abbreviated. For example, the following two definitions are equivalent:

```
DDAL$LOG_TRMON_CLIENT_EVENTS "C,D"

DDAL$LOG_TRMON_CLIENT_EVENTS "CREATE,DROP"
```

If an abbreviation matches more than one keyword, all such keywords are assumed to have been entered. For example, the abbreviation "S" is equivalent to entering "SHOW,START,STOP".

The keywords ALL and MESSAGE are not SQL statements to Replication Option. ALL enables all event classes. MESSAGE enables the logging of messages between SQL and DDAL$SHR.EXE and between DDAL$SHR.EXE and the transfer monitor. For instance:

```
DDAL Event:  10:32:13.45  message received from the user interface
DDAL Event:  10:32:13.46  message sent to the transfer monitor
```

If the keyword is defined to be "START,STOP", only events involving the START and STOP TRANSFER statements will be logged. When an event is logged, DDAL$SHR.EXE writes a message to standard output (SYS$OUTPUT), for example:

```
DDAL Event:  10:32:14.07  request to START TRANSFER has been received
```

When using this option, it would be most useful for you to write the output messages to SYS$OUTPUT in a log file for later review.

B.11 DDAL$MAX_COPY_PROCESS Environment Variable

Replication Option has a built-in limit to the number of concurrent transfers it will attempt to execute. The limit is 20. You may want to reduce this default limit to maintain system performance.

See Section 6.16.4 for information on system considerations.

Use the following format to define the DDAL$MAX_COPY_PROCESS environment variable as an OpenVMS logical name:

```
$ DEFINE DDAL$MAX_COPY_PROCESS n
```

Alternatively, use the following format to define this environment variable in the DDAL.CONF resource configuration file:

```
DDAL$MAX_COPY_PROCESS n
```

Replace n with a value between 1 and 20.

The value for n must be at least 1, and no more than the built-in limit of 20.

The transfer monitor selects the lesser of the following two values:

- The value translated from DDAL$MAX_COPY_PROCESS
- The transfer monitor's process quota called MAXDETACH minus 1

If DDAL$MAX_COPY_PROCESS is not defined, the transfer monitor selects the lesser of the following two values:

- The default limit of 20
- The transfer monitor's process quota called MAXDETACH minus 1

In order for a transfer monitor to do any meaningful work, MAXDETACH must have a value of 2 or more: one process to run a normal transfer and the other to perform a replication deletion, if necessary.

Defining or redefining this environment variable while Replication Option is running has no immediate effect. In order for a transfer monitor to recognize the current setting, you must stop Replication Option and restart it.

If Replication Option is running on several or all nodes in a cluster, you must define this environment variable on each CPU. The value defined for each processor can vary. This is a potential benefit in heterogeneous VMScluster systems where processor power varies.

B.12 DDAL_NUMBER_OF_DATABASES, DDAL_DATABASE_n, DDAL_USERNAME_n, and DDAL_PASSWORD_n Environment Variables

As a result of changes to the Rdb/Dispatch component of Oracle Rdb version 7.0, you cannot access databases at multiple remote Digital UNIX sites without making special provisions for listing user names and passwords. In Version 7.0, the user must have the same UID as the database owner at the remote site to access a remote database unless the correct account information (that is, the user name and password) for the remote database is provided.

See Section 4.8.5 for more information about remotely accessing multiple Rdb databases.

As a workaround to the single account restriction, the following user interface is implemented in Replication Option Version 7.0 so multiple accounts can be specified in the DDAL.RC configuration file. These multiple account names are read only by Replication Option.

In the DDAL.RC configuration file, the account information for accessing a source or target database is specified with the following three environment variables:

- DDAL_DATABASE_n—Specifies a database attach string
- DDAL_USERNAME_n—Specifies the user name used in attaching to the database
- DDAL_PASSWORD_n—Specifies the password used in attaching to the database

These environment variables must be specified in three separate lines as follows:

```
DDAL_DATABASE_n  "attach string"
DDAL_USERNAME_n  "user name"
DDAL_PASSWORD_n  "password"
```

These environment variable names are case-sensitive and must be entered in uppercase with underscores. Each group of three variables is distinguished from the others by a numeric suffix in the names, shown in the previous example as _n. Replace the letter *n*, with an integer between 1 and 20.

The following example shows a sample entry for the user name, password, and attach string in the DDAL.RC configuration file:

```
DDAL_DATABASE_1  "file osfit::/usr/users/admin/division4db"
DDAL_USERNAME_1  "admin"
DDAL_PASSWORD_1  "user_with_privs"
```

Repeat these three entries, using different definitions, for each account and database you need to mention. There is no requirement that the numbers be assigned consecutively starting with 1, but numbers chosen must normally be between 1 and 20.

If you need to specify more than 20 accounts and databases, insert an additional environment variable, DDAL_NUMBER_OF_DATABASES, to indicate the new maximum. For example, the following line changes the range of valid numeric suffixes from 1 to 20 to 1 to 30.

```
DDAL_NUMBER_OF_DATABASES  30
```

Remember, this only specifies the maximum number of account/database entries you may have. You may in fact have fewer entries than the maximum. As with the other variables named in this section, DDAL_NUMBER_OF_ DATABASES must be entered only in uppercase and only using the underscore character.

At run-time, before it attaches to a source or target database, a copy process will look in the DDAL.RC file for environment variables of the form DDAL_ DATABASE_n, where n is an integer from 1 to the designated maximum. First it will look for DDAL_DATABASE_1, then DDAL_DATABASE_2, and so on. For each such variable found, the corresponding attach string is compared with the attach string from the transfer definition. If a match is found, the corresponding user name and password will be used to attach to the Rdb database. This takes precedence over any of the other methods of attaching to a database described in this general section.

Note

If you create the transfer specifying the physical file path name but use a link to point to it in your configuration file, a match will not be found.

If a matching text string is found, Replication Option will look for the corresponding user name and password information. If a match is not found, the copy process searches for the next environment variable, DDAL_ DATABASE_n+1. This continues, as needed, until the next higher suffix is not found in the DDAL.RC file. If a match is found, Replication Option will attach to the database under the matching account. If a match is not found, or if there are no DDAL_DATABASE_n entries, or if there is no DDAL.RC file, Replication Option attaches using the attach string stored in the transfer definition. Otherwise, Replication Option attaches to the database using the user's UID or using the account specified in the SQL_USERNAME and SQL_PASSWORD environment variables included in the RDB$CLIENT_ DEFAULTS.DAT configuration file. However, this may create a problem when the copy process, in its startup phase, attempts to access the transfer database to retrieve information about the transfer. This is because the account specified in the SQL_USERNAME and SQL_PASSWORD environment variables is used to attach to every database, including the transfer database.

B.13 DDAL$PURGE Environment Variable

Deadlocks can occur when several copy processes attempt to purge rows no longer needed in the RDB$CHANGES table. You can avoid these deadlocks by controlling when, and if, a given transfer attempts to purge rows from the RDB$CHANGES table. The default is to purge.

Refer to Section 6.17.1 for more information on using controlled purging of RDB$CHANGES to improve performance.

Use one of the following formats to define the DDAL$PURGE environment variable as an OpenVMS logical name:

```
$ DEFINE DDAL$PURGE yes/no
$ DEFINE DDAL$PURGE_transfername yes/no
```

Alternatively, use one of the following formats to define the environment variable in a resource configuration file, DDAL.RC:

```
DDAL$PURGE yes/no
DDAL$PURGE_transfername yes/no
```

Replace `transfername` with the name of a specific transfer to which the environment variable applies.

If you use the environment variable DDAL$PURGE_transfername with a definition of No, Replication Option does not purge for the specified replication update transfer. Defining it to be Yes results in the copy process attempting to purge.[1]

For general information about defining this environment variable, including the effects of using generic and transfer-specific forms, refer to Section B.1.2.

To control when purging is done and by which transfer process, you must change the environment variable definitions at the appropriate time.

One possible method is to use the SUBMIT command to submit batch jobs that will execute at any time you want.

Messages appear in the copy process log file that indicate whether or not the copy process is attempting to purge. If either DDAL$PURGE or DDAL$PURGE_transfername is defined, the message indicates which environment variable applies and specifies its definition. If the exclusive purge lock is not obtained immediately, that fact is logged instead.

---

[1] In actuality, Replication Option parses only the first character in the DDAL$PURGE definition. If the first character is either an uppercase or lowercase “y”, Replication Option interprets the response as “Yes”. If the first character is either an uppercase or lowercase “n”, Replication Option interprets the response as “No”. Replication Option does not read beyond the first character.

Before trying to purge rows from RDB$CHANGES, a copy process first attempts to get a purge lock for exclusive access. If it cannot obtain the lock immediately, it is assumed that some other copy process has obtained the exclusive lock for itself and is in the process of doing the purge operation. Because there is no need for more than one copy process to do this, that copy process simply skips this step and completes the transfer process.

B.14 DDAL$RW_ROW_COUNT Environment Variable

You can specify how many data rows are read and written at a time by Replication Option during an extraction, extraction rollup, or initial replication transfer with the environment variable DDAL$RW_ROW_COUNT. The default is 16.

Refer to Section 6.16.1 for information on how you can use DDAL$RW_ROW_COUNT to tune execution speed.

By defining DDAL$RW_ROW_COUNT, you specify the number of rows to be read and written at a time by Replication Option.

Use one of the following formats to define this environment variable as an OpenVMS logical name:

```
$ DEFINE DDAL$RW_ROW_COUNT n
$ DEFINE DDAL$RW_ROW_COUNT_transfername n
```

Alternatively, use one of the following formats to define this environment variable in your resource configuration file, DDAL.RC:

```
DDAL$RW_ROW_COUNT n
DDAL$RW_ROW_COUNT_transfername n
```

Replace `n` with a value between 1 and 99999 when you define the environment variable DDAL$RW_ROW_COUNT.

Replace `transfername` with the name of a specific transfer to which the environment variable applies.

For general information about defining this environment variable, including the effects of using generic and transfer-specific forms, refer to Section B.1.2.

B.15 DDAL$SHARED_WRITE Environment Variable

By default, Replication Option prevents you from writing to tables owned by another database or transfer when transferring data to an existing database. For example, if there are two identically structured databases, Replication Option will not allow you to transfer data from the EMPLOYEES table in the first database into the EMPLOYEES table in the other database and vice versa. Replication Option will allow you to create an EMPLOYEES1 table in one of the databases and let you store your data there. This restriction means you can write to the same database but not normally to the same tables.

Replication Option will allow different writers to share the same database and write to the same tables. However, they still cannot write the same data. That is, if one transfer inserts a row into a table, that row should not be updated or deleted by another transfer or customer application. In order to share tables for writing, you must define the environment variable DDAL$SHARED_WRITE with the value YES.

See Chapter 8 for more information on using the shared-write feature.

Use one of the following formats to define this environment variable as an OpenVMS logical name:

```
$ DEFINE DDAL$SHARED_WRITE yes/no
$ DEFINE DDAL$SHARED_WRITE_transfername yes/no
```

Alternatively, use one of the following formats to define this environment variable in a resource configuration file, DDAL.RC:

```
DDAL$SHARED_WRITE yes/no
DDAL$SHARED_WRITE_transfername yes/no
```

Replace transfername with the name of a specific transfer to which the environment variable applies.

For general information about defining this environment variable, including the effects of using generic and transfer-specific forms, refer to Section B.1.2.

B.16 DDAL$SINGLE_DATA_COMMIT Environment Variable

You can direct Replication Option to delete data and write data that is transferred into the target tables all in one transaction with the environment variable DDAL$SINGLE_DATA_COMMIT.

This option can be used for any transfer to an existing database; that is, any transfer where the TO EXISTING FILENAME clause appears in the transfer definition. If the option is defined and the transfer definition contains the TO NEW FILENAME clause, the option is ignored.

By defining DDAL$SINGLE_DATA_COMMIT you can specify whether or not data transferred into the target tables is deleted and written in one transaction.

The default behavior is no single-data-commit.

Refer to Section 6.16.2 for information about using this option to improve transfer performance.

Use one of the following formats to define this environment variable as an OpenVMS logical name:

```
$ DEFINE DDAL$SINGLE_DATA_COMMIT yes/no
$ DEFINE DDAL$SINGLE_DATA_COMMIT_transfername yes/no
```

Alternatively, use one of the following formats to define this environment variable in your resource configuration file, DDAL.RC:

```
DDAL$SINGLE_DATA_COMMIT yes/no
DDAL$SINGLE_DATA_COMMIT_transfername yes/no
```

Replace *transfername* with the name of a specific transfer to which the environment variable applies.

For general information about defining this environment variable, including the effects of using generic and transfer-specific forms, refer to Section B.1.2.

B.17 DDAL$START_WITH_SOURCE_TABLE Environment Variable

Use of this environment variable can improve overall performance (decrease elapsed time) for concurrently executing extraction or initial replication transfers. When this option is enabled, it instructs a copy process to transfer data starting with a particular table, thus forcing execution of concurrent transfers to start working on different areas of the source database. Normally, a Replication Option copy process begins transferring data from the first table in its list.

The default (when this option is disabled), is for a copy process to begin transferring data from the first table in its list. You can override this default by defining an environment variable as an OpenVMS logical name using one of the following formats:

```
$ DEFINE DDAL$START_WITH_SOURCE_TABLE definition
$ DEFINE DDAL$START_WITH_SOURCE_TABLE_transfername definition
```

Alternatively, you can define the DDAL$START_WITH_SOURCE_TABLE environment variable by including a line similar to one of the following lines in your resource configuration file, DDAL.RC:

```
DDAL$START_WITH_SOURCE_TABLE definition
DDAL$START_WITH_SOURCE_TABLE_transfername definition
```

Replace transfername with the name of a specific transfer to which the environment variable applies.

For general information about defining this environment variable, including the effects of using generic and transfer-specific forms, refer to Section B.1.2.

The following example specifies a transfer name and instructs a copy process to start transferring data from source table Table1 for the TO_NJ transfer.

```
DDAL$START_WITH_SOURCE_TABLE_TO_NJ "Table1"
```

Table names must be entered with the correct alphabetic case. They are not automatically changed to uppercase because some database systems support mixed-case names.

Although you can use the generic form of the DDAL$START_WITH_SOURCE_ TABLE environment variable, it is unlikely to be of much use.

For example, the following definition does not name a specific transfer, so the definition applies to all other transfers as well.

```
DDAL$START_WITH_SOURCE_TABLE "Table1"
```

B.18 DDAL$STORE_DEF_SEPARATE Environment Variable

When several replication transfers are created to copy data from the same source database, and when those transfers are started at the same time, the normal (default) behavior is for the transfers to execute one after the other, not concurrently. This can have an adverse effect on overall replication performance. You can improve it by using the DDAL$STORE_DEF_ SEPARATE environment variable.

See Section 6.17.2 for more information about improving performance using this option.

Use one of the following formats to define this environment variable as an OpenVMS logical name:

```
$ DEFINE DDAL$STORE_DEF_SEPARATE yes/no
$ DEFINE DDAL$STORE_DEF_SEPARATE_transfername yes/no
```

Alternatively, use one of the following formats to define this environment variable in a resource configuration file, DDAL.RC:

```
DDAL$STORE_DEF_SEPARATE yes/no
DDAL$STORE_DEF_SEPARATE_transfername yes/no
```

Replace `transfername` with the name of a specific transfer to which the environment variable applies.

For general information about defining this environment variable, including the effects of using generic and transfer-specific forms, refer to Section B.1.2.

B.19 DDAL$TARGET_INDEX_OPTIONS Environment Variable

The DDAL$TARGET_INDEX_OPTIONS variable is useful in limited circumstances involving shared-write replication transfers to Rdb or Oracle target tables. It permits customer applications and replication transfers to write data into the same tables, under limited circumstances. See Section 8.2.5 for a discussion of the restrictions applied in such a situation.

Normally, Replication Option creates a unique index on each target table (or requires you to create a column constraint to require uniqueness) using the row owner and database key columns. These indexes are dropped before the initial transfer is re-executed and then re-created after the initial transfer has completed. During replication update transfers, the indexes are left in place.

It is the unique indexes or column constraints that make it normally not possible for local customer applications to also write data into these target tables. These applications would normally leave the row owner and database key columns alone; that is, the values in those columns would be null. Two such rows would generate duplicate index values, which would violate the uniqueness constraint.

You can use the DDAL$TARGET_INDEX_OPTIONS environment variable to prevent the copy process from creating and dropping these target indexes. However, if you do so, you must remember that once the target tables are in place, it is your responsibility to create the indexes (and constraints, if necessary) on the target tables.

See Section 8.2.5.1 for more information about multiple transfers writing to the same tables.

Use one of the following formats to define this environment variable as an OpenVMS logical name:

```
$ DEFINE DDAL$TARGET_INDEX_OPTIONS n
$ DEFINE DDAL$TARGET_INDEX_OPTIONS_transfername n
```

Alternatively, use one of the following formats to define this environment variable in a resource configuration file, DDAL.RC:

```
DDAL$TARGET_INDEX_OPTIONS n
DDAL$TARGET_INDEX_OPTIONS_transfername n
```

Replace `n` with either 1 or 2. A value of 1 is equivalent to not defining the variable at all; a copy process assumes its default behavior. A value of 2 results in the copy process neither creating nor dropping such target indexes.

Replace `transfername` with the name of a specific transfer to which the environment variable applies.

For general information about defining this environment variable, including the effects of using generic and transfer-specific forms, refer to Section B.1.2.

B.20 Rules for Setting Up a Configuration File

Configuration files are ASCII text files. They can be created using any text editor. The following rules apply to the format of such a file:

- Comments can begin with an exclamation point (!) or they can follow the definition after a space character ( ). For consistency, Oracle Corporation recommends you begin comments with an exclamation point after a definition.
- Blank lines are permitted.
- Each entry must fit on a single line (that is, no line continuation is allowed).
- A definition consists of the environment variable name followed by one or more space or tab characters followed by the variable's definition and optionally followed by one or more spaces or tabs and a comment.
- If a variable's definition must contain spaces, the definition must be enclosed in double quotation marks (" ").
- Environment variable names and definitions must use the valid formats described in this appendix.
- Names and definitions can be entered either in uppercase, mixed-case, or lowercase. Replication Option converts them to uppercase unless stated otherwise for a specific environment variable.

Example B–1 shows a sample user's resource configuration file.

Example B–1 Sample User Configuration File

```
! Replication Option user configuration file (ddal.rc)
!
! For replication transfers, unless there is a transfer-specific variable
! to override this, do not purge rows from the source "changes" table after
! transfer execution.

ddal_purge no

! For the monthly_sales replication transfer, after the transfer has
! completed, purge "changes" from the source database. This has precedence
! over the ddal_purge definition.

ddal_purge_monthly_sales yes

! Set the count of the maximum number of data rows to be read and written
! at a time.

DDAL$RW_ROW_COUNT 24

! Enable shared writing in target database tables for the DISTRICTS_ROLLUP
! transfer.

DDAL$SHARED_WRITE_DISTRICTS_ROLLUP  Yes
```

C

Replication Option Operational Information

This appendix includes the following information about Replication Option:

- Operational procedures
- Transfer states
- Special system tables for replications
- Access rights for the transfer database and special system tables
- Transaction activities during data transfers
- A table listing recoverable errors

C.1 Operational Procedures

Following the installation of Replication Option, you create a transfer database or convert an existing transfer database to the latest format. Also, you edit the system startup and shutdown procedures to include the automatic startup and shutdown of Replication Option. Information on these procedures appears in the *Replication Option for Rdb Installation Guide* for OpenVMS.

At some later date, you may find it necessary to manually re-execute one or more of these procedures. For example, if you have installed Replication Option for testing purposes and you now want to use it for production, you might find it simpler to delete the transfer database files you used for testing and re-create them. This section explains how to re-create a transfer database and how to manually stop and start a transfer monitor.

C.1.1 Creating the Transfer Database on OpenVMS

Replication Option provides command procedures for creating both Rdb and Oracle transfer databases. You must run the command files from a privileged account with a system UIC.

C.1.1.1 Using Rdb as the Transfer Database

To create an Rdb transfer database, enter the following command at the DCL prompt to invoke the command procedure:

```
$ @SYS$MANAGER:DDAL$CREATE_TR_DB.COM
```

The parameters for this procedure are described in the following sections.

P1—The disk and directory where the transfer database is to be created

This parameter is optional. The default is SYS$COMMON:[SYSEXE]. Ensure that there are no existing transfer databases in that location. You can specify a different device and directory as long as both currently exist. In a VMScluster environment, the location you specify for the transfer database should be accessible from all nodes in the cluster.

The file name is automatically set to DDAL$TR_DB.RDB. If this procedure is being executed interactively and this parameter is unspecified, you will be prompted for the information (unless P4 specifies not to).

P2—Rdb version number

This parameter is optional. If supplied, it takes one of the following formats: `m.n` or `mn` or `null`.

If this parameter is omitted (is null), the Rdb version defaults to the standard version on the system. The Rdb version can be specified with or without a period separating the major and minor versions, such as 70 or 7.0. The last digit is the minor version number and the preceding digits form the major version number. Specifying the Rdb version number is important when you want to run under a version that is different from the standard system version.

If this procedure is being executed interactively and this parameter is unspecified, you will be prompted for the information (unless P4 specifies not to). If it is still not specified, the default Rdb version on the system is used.

Note

If the Rdb environment for your current process does not match what you specify in P2, your process environment will be changed to the version of Rdb you selected.

P3—Log file to write to on error

This parameter is optional. If a log file is not specified, errors go to SYS$OUTPUT. If a log file is specified, the file is created, written to, and closed.

P4—Suppress prompting for parameters

This parameter is optional. If it is specified as the string NOINQUIRE or if this procedure is being run interactively, you will not be prompted for information. This parameter is used by the kit installation procedure to prevent the installation verification procedure (IVP) from asking questions of the installer. All installation questions, including those for running the IVP, are supposed to be asked first, before the installation is performed.

P5—Suppress output of informational messages

This parameter is optional. If it is defined as the string NOINFO, informational messages will not be produced.

C.1.1.2 Using Oracle as the Transfer Database

To create an Oracle transfer database, take the following steps:

1. Create an Oracle database. It is recommended that it be a local database.
2. Use SQL*Plus to create Replication Option transfer tables in the Oracle database. To do this, execute the SYS$MANAGER:DDAL$CREATE_TR_DB_ORA.SQL procedure.

C.1.2 Starting the Transfer Monitor on OpenVMS

Replication Option provides the DDAL$START_TR_MON.COM command procedure to start the transfer monitor.

The *Replication Option for Rdb Installation Guide* advises invoking the DDAL$START_TR_MON.COM procedure automatically as part of the system startup procedure.

Replication Option sets the name of the transfer database file to DDAL$TR_DB. The log file name includes the node on which the log file is located. The format is:

```
DDAL$node-name_TR_MON.LOG
```

The following system logical names are defined in DDAL$START_TR_MON.COM when a transfer monitor is started:

```
DDAL$TRANSFER_DATABASE -- clusterwide lock file
DDAL_TRANSFER_DATABASE -- transfer database attach string
DDAL_LOCK_FILENAME     -- cluster-wide lock file
DDAL_TM_START_INPUT    -- contains input DCL to transfer monitor process
DDAL_START_LOG         -- the startup log for the transfer monitor
DDAL_START_ERR         -- the transfer monitor error file if it fails to start
DDAL$TM_LOG_FILE       -- site-specified or default assignment
DDAL$COPY_PROCESS      -- points to the file specification for the copy
                          process image
DDAL$TM_LOG_ENABLE     -- indicates whether the transfer monitor log file is
                          enabled
DDAL$EPILOGUE          -- points to the file specification for the systemwide
                          Replication Option epilogue file
```

Note

To provide compatibility with prior versions of Replication Option, the value specified for the DDAL$TRANSFER_DATABASE logical name is also the attach string for the transfer database, if it is an Rdb database.

The DDAL$START_TR_MON.COM takes parameters P1 through P7 as described in the following sections:

P1—Transfer database attach string

This parameter is either optional or required, as follows:

- Optional—if the transfer database is Rdb.

  The default location is SYS$COMMON:[SYSEXE].

  Specify only the device and directory where the transfer database is located. DDAL$TR_DB.RDB is the fixed file name, which is why this procedure looks only for the device name and directory.

- Required—if the transfer database is Oracle.

  There are no defaults. The full attach string is required so that the Rdb Transparent Gateway to Oracle may access the transfer database through Oracle.

If this procedure is executed interactively and this parameter is unspecified, you will be prompted for the information (unless P6 specifies not to).

P2—File specification for the transfer monitor log

This parameter is optional. The default is SYS$SYSTEM:DDAL$nodename_TR_MON.LOG.

Specify the device and directory (and, optionally, the file name and extension) where the transfer monitor log file is located. If this procedure is executed interactively and this parameter is unspecified, you will be prompted for the log file's location (unless P6 specifies not to).

Note

The DDAL$START_TR_MON.COM procedure must be run from a privileged OpenVMS account. Also, there are certain privileges required in order to create log files. In OpenVMS, an account is assigned default OpenVMS privileges at login. They are defined in the system authorization file (SYSUAF) as part of the account record. Each account also has a set of authorized privileges that are not necessarily enabled by default, but which you have the authority to enable.

When you execute the Replication Option startup procedure, log files must be created. Typically, the SYSPRV privilege is required in order to create these log files in the designated log file directory. However, other privileges may suffice. These privileges must be enabled as default privileges; enabling them in your LOGIN.COM will not work.

The log file directory used for startup of the transfer monitor depends on a parameter that you pass to the startup procedure, SYS$STARTUP:DDAL$START_TR_MON.COM.

P3— Transfer database type and version number

This parameter takes the following format (in which braces and brackets indicate optional elements):

```
{R[DB] or O[RACLE]}{=}{m[.]n or m.n.o}
```

This parameter is either optional or required, as follows:

- Optional—if the transfer database is Rdb.

  If this parameter is omitted (is null), the transfer database type and version defaults to the standard version of Rdb on the system.

  You can specify the database type as RDB (case does not matter). RDB can be abbreviated or omitted altogether.

If the database type is Rdb, you can also specify a database version, such as 70 or 7.0., for the m[.]n element. The last digit is the minor version number and the preceding digits form the major version number. Specifying the Rdb version number is important when you want to run under a version that is different from the standard system version.

- Required—if the transfer database is Oracle.

  If the transfer database is Oracle you must specify the database type as Oracle (case does not matter). The word Oracle can be abbreviated.

  If the database type is Oracle, you can also specify a database version, such as 7.3.0, for the m.n.o element. However, the Oracle version number is ignored by this command procedure.

If you supply only the version number in this parameter, the database type is assumed to be Rdb and you must follow the Rdb version number format.

Otherwise, specify both the product name and version and separate the two using an equal sign with no intervening spaces, for example, RDB=7.0 or O=7.3.0.

If this procedure is executed interactively and this parameter is unspecified, you will be prompted for the information (unless P6 specifies not to).

If P3 is omitted completely, the transfer monitor is started using the standard system-default version of Rdb. If you have variant Rdb versions on your system and you wanted to run the transfer monitor with one of those variants, you would first have to make the variant version the standard version. To do that you would have to invoke the DECRDB$SETVER.COM command procedure to set up an environment in which you could use a multiversion variant of Rdb.

If a value is supplied for P3, Replication Option allows you to run the transfer monitor with a variant version of Rdb without having to make the variant version the standard version.

P4— Cluster lock file

This parameter specifies an existing file accessible to all transfer monitors on the same cluster. It is either ignored or required, as follows:

- Ignored—if the transfer database is Rdb.

  When the transfer database is an Rdb database, the transfer database file itself is used as the cluster lock file. An additional file is not required.

- Required— if the transfer database is Oracle.

The file named in this parameter is used as a cluster lock file to formulate a systemwide lock name so that each Replication Option process can detect the termination of any other cooperating process. To ensure that the same file is used by each transfer monitor, you should include the file version number as part of the file specification.

Note

Replication Option uses the VMS lock manager to synchronize the activities of cooperating transfer monitors on the same VMScluster. If an Rdb database is used as the common transfer database, the same database file number (obtained from VMS System Service) is used by all participating transfer monitors in deriving lock names. But if the transfer database is an Oracle database, it may be physically located on a system remote to the transfer monitor and may also have been installed on a platform different from OpenVMS. Thus, the need for a cluster lock file for use in deriving lock names.

Do not include a node name in the file specification. Remote access to the lock file might not work properly. Also, if the file specification is entered using a logical name, the definition of the logical name must be in the SYSTEM logical name table. That is because the transfer monitor is run as a detached process.

If this procedure is being executed interactively and this parameter is unspecified, you will be prompted for the file specification (unless P6 specifies not to).

P5—Log file to write to on error

This parameter is optional. If a log file is not specified, errors are written to SYS$OUTPUT. If a log file is specified, the file is created, written to, and closed.

P6—Suppress prompting for parameters

This parameter is optional. If it is specified as the string NOINQUIRE or if this procedure is being run interactively, you will not be prompted for information. This parameter is used by the kit installation procedure to prevent the installation verification procedure (IVP) from asking questions of the installer. All installation questions, including those for running the IVP, are supposed to be asked first, before the installation is performed.

P7—Suppress output of informational messages

This parameter is optional. If it is defined as the string NOINFO, informational messages will not be produced.

C.1.2.1 Additional Files Created When Transfer Monitor Is Started

When starting up the transfer monitor, there may be up to three other files created in the directory where the transfer monitor log file is located.

- DDAL_<nodename>_START_TM_SETVER.COM is a DCL command procedure for setting up the run-time environment (including Rdb and SQL) environment for the transfer monitor process.
- DDAL_<nodename>_START_TM.LOG, contains the log of starting up the transfer monitor process.
- A file that, if ever created, contains the error messages indicating the reason for failing to start up the transfer monitor process.

Only up to five versions of these three files are kept, and older versions are purged.

C.2 Stopping the Transfer Monitor on OpenVMS

Replication Option provides the DDAL$STOP_TR_MON.COM command procedure to stop the transfer monitor. The *Replication Option for Rdb Installation Guide* advises invoking the DDAL$STOP_TR_MON.COM procedure automatically as part of the system shutdown procedure. This procedure runs every time the system shuts down if you have added the line that invokes it to the SYS$MANAGER:SYSHUTDWN.COM file. The procedure should be run from a privileged OpenVMS account. The procedure deinstalls the SYS$SYSTEM:DDAL$COPY_PROCESS executable image and the system logical names that were assigned when you ran the DDAL$START_TR_MON.COM command procedure.

The DDAL$STOP_TR_MON.COM command procedure requires no parameters. To stop the transfer monitor, simply invoke the command procedure, as follows:

```
$ @SYS$MANAGER:DDAL$STOP_TR_MON.COM
```

This procedure only shuts down Replication Option on the machine on which the command procedure is run.

C.3 Transfer States and Transfer Status Codes

The Replication Option statements you enter can affect the state of a transfer. This section describes the transfer states, the events that change a transfer state, and the status codes of transfers after they have run. Table C–1 lists the possible transfer states, an explanation of the states, and the decimal value that each code represents in the DDAL$TRANSFER_STATE column of the DDAL$TRANSFERS_STATUS table. For information about the columns in the DDAL$TRANSFERS_STATUS table, refer to Table A–8.

Table C–1 Transfer States

Transfer State	Explanation	Decimal Value
Active	The transfer is currently executing. Either a user issued a START TRANSFER NOW statement or the transfer time specified in the transfer schedule has arrived.	1
Retrying	Replication Option is currently making another attempt to execute the transfer. This state is the same as the active state, except that the transfer is being retried following an earlier failure.	2
Scheduled	A transfer schedule exists. The transfer executes when the schedule time arrives.	3
Suspended	The user issued a STOP TRANSFER statement or the transfer monitor placed the transfer in a suspended state because of a fatal error.	4
Unscheduled	No schedule exists for the transfer and the transfer is not currently executing.	5
Waiting to Retry	Replication Option attempted a transfer, but the execution failed. The transfer monitor waits until the retry time specified in the schedule definition arrives. Then Replication Option attempts another transfer. This state is similar to the scheduled state.	6

Table C–2 shows the transfer state change resulting from either a user action or from an action taken by Replication Option.

Table C–2 Events Affecting Transfer States

Event	Transfer State
CREATE TRANSFER	Begins in the unscheduled state.
DROP TRANSFER	Deletes transfer and corresponding schedule.
CREATE SCHEDULE	If unscheduled, changes to scheduled state; if suspended, remains in suspended state.

(continued on next page)

Table C–2 (Cont.) Events Affecting Transfer States

Event	Transfer State
ALTER TRANSFER	If original transfer was unscheduled, remains in unscheduled state; if original transfer was scheduled, it was suspended to execute the ALTER TRANSFER statement, and it remains in the suspended state after the ALTER TRANSFER statement executes.
ALTER SCHEDULE	If original transfer was scheduled, it was suspended to execute the ALTER SCHEDULE statement, and it remains in the suspended state after the ALTER SCHEDULE statement executes.
START TRANSFER	Changes to scheduled or unscheduled state depending on whether or not the transfer has a schedule.
START TRANSFER NOW	Changes to active state.
STOP TRANSFER	Changes to suspended state.
REINITIALIZE TRANSFER	Remains suspended until a START TRANSFER statement is issued.
Schedule or retry time arrives	Changes to active or retrying state.
Transfer attempt fails with a nonfatal error	Changes to waiting to retry state.
Transfer attempt fails with a fatal error	Changes to suspended state.
Transfer is successful or the maximum number of retries is reached	Changes to scheduled or unscheduled state depending on whether or not the transfer has a schedule.

The DDAL$LAST_COMPLETION_STATUS column in the DDAL$TRANSFERS_STATUS table contains a status code to indicate what occurred the last time a transfer was run. Table C–3 lists and describes each status code. Refer to Table A–8 for information about the other columns in the DDAL$TRANSFERS_STATUS table.

Table C–3 Status Codes for the DDAL$LAST_COMPLETION_STATUS Column

Hexadecimal Status Code	Explanation
0126839B	Severe transfer failure and not retrying
012683A3	Transfer failed but retrying
012683AB	Transfer was successful
012683B3	Transfer is stopped and not retrying
012683BA	Transfer failed; maximum retry count has been reached
0126C44B	Prologue failed, copy aborted, but retrying
0126C453	Prologue failed, copy aborted, and not retrying
0126C45B	Epilogue failed but retrying
0126C463	Epilogue failed and not retrying

C.4 Source Database Tables Used for Replication Option

When a replication transfer executes for the first time or when a transfer is reinitialized and the transfer definition has changed, Replication Option stores the transfer definition in the source database.

The first time a replication transfer is executed, Replication Option creates several special tables in the source database, among which are the RDB$TRANSFERS and RDB$TRANSFER_RELATIONS tables. When these tables are stored in the source database, a message similar to the following is sent to the copy process log file. The file-specification will differ.

```
14:26:52  %DDAL-I-STOTRADEF, storing transfer definition in
           LOCNOD::DISK$1:[DBA.PERSONNEL]PERSONNEL.RDB
```

See Appendix A for complete details about data definitions created by Replication Option in the transfer database, source database, and the target database.

The following SQL SHOW statements display individual column names for two of the special system tables defined in the source database PERSONNEL by Replication Option:

```
SQL> SHOW TABLE RDB$TRANSFERS;
```

```
Columns for table RDB$TRANSFERS:
Column Name                     Data Type        Domain
-----------                     ---------        ------
RDB$TRANSFER_NAME               CHAR(31)         RDB$TRANSFER_NAME
 Missing Value: Not Available
RDB$VINTAGE_TSER                INTEGER          RDB$VINTAGE_TSER
 Missing Value: -1
   .
   .
   .
SQL> SHOW TABLE RDB$TRANSFER_RELATIONS;

Columns for table RDB$TRANSFER_RELATIONS:
Column Name                     Data Type        Domain
-----------                     ---------        ------
RDB$RELATION_NAME               CHAR(31)         RDB$RELATION_NAME
RDB$TRANSFER_NAME               CHAR(31)         RDB$TRANSFER_NAME
 Missing Value: Not Available
RDB$TRANSFER_SELECTION_BLR      VARBYTE LIST RDB$TRANSFER_SELECTION_BLR
                                         Subtype: BLR
                                         Segment Length: 512
   .
   .
   .
```

C.5 Target Database Tables and Indexes Used for Replication Option

When a transfer executes for the first time, the copy process creates new tables and columns in the target database. For shared-write transfers, an additional column, DDAL$ROW_OWNER, is included in each user table transferred to the target database. For replication transfers only, an additional column, DDAL$DBKEY, is included in every transferred user table in the target database. Database key values contained in the DDAL$DBKEY column provide Replication Option with information about the version of the corresponding row in the source database. If your host language programs manipulate entire rows at a time, rather than individual columns, your programs need to accommodate these additional columns. In addition, for replication transfers a sorted index is created for each user table in the target database on the DDAL$DBKEY and DDAL$ROW_OWNER columns.

An extra table, RDB$VINTAGE or DDAL$TRANSFER_INFO, is also created in the target database. Although created by the copy process, this table is considered a user table. When performing a SHOW TABLES statement, SQL filters out names that begin with Rdb, but will not filter out the DDAL$TRANSFER_INFO name. The following example lists tables in a replication target database created during the initial transfer to an Rdb database:

```
SQL> ATTACH 'FILENAME EMP_COLLEGE';
SQL> SHOW TABLES;
User tables in database with filename EMP_COLLEGE
     COLLEGES
     DEGREES
SQL> SHOW TABLES COLLEGES;

Columns for table COLLEGES:
Column Name                    Data Type        Domain
-----------                    ---------        ------
DDAL$DBKEY                     CHAR(8)          DDAL$DBKEY
COLLEGE_CODE                   CHAR(4)          COLLEGE_CODE
COLLEGE_NAME                   CHAR(25)         COLLEGE_NAME
CITY                           CHAR(20)         CITY
STATE                          CHAR(2)          STATE
POSTAL_CODE                    CHAR(5)          POSTAL_CODE

Constraints for table COLLEGES:
No constraints found

Indexes on table COLLEGES:
DDAL$DBKEY_INDEX1              with column DDAL$DBKEY
                               no duplicates allowed
                               type is sorted

Storage Map for table COLLEGES:
No Storage Map found

SQL> SHOW TABLES DEGREES;

Columns for table DEGREES:
Column Name                    Data Type        Domain
-----------                    ---------        ------
DDAL$DBKEY                     CHAR(8)          DDAL$DBKEY
EMPLOYEE_ID                    CHAR(5)          ID_NUMBER
COLLEGE_CODE                   CHAR(4)          COLLEGE_CODE
YEAR_GIVEN                     SMALLINT         YEAR_GIVEN
DEGREE                         CHAR(3)          DEGREE
DEGREE_FIELD                   CHAR(15)         DEGREE_FIELD

Constraints for table DEGREES:
No constraints found

Indexes on table DEGREES:
DDAL$DBKEY_INDEX2              with column DDAL$DBKEY
                               no duplicates allowed
                               type is sorted

Storage Map for table DEGREES:
No Storage Map found
```

There are three formats to index names for replication transfers:

- Transfers TO NEW FILENAME—For Rdb target databases, the format is DDAL$DBKEY_INDEX<index_number> where index_number is some integer. Because you can only use the TO NEW FILENAME mode with Rdb databases, there is no other form for these index names.
- Transfers TO EXISTING FILENAME with shared writing disabled—If shared writing to the target tables is disabled, the format is DDAL$DBKEY_INDEX<index_number>_<transfer_id> for Rdb target databases. The format is DDAL$KEY_IDX<index_number>_<transfer_id> for all other target databases. The <transfer_id> indicates the numeric transfer identifier (see Table A–21). With any database, <index_number> and <transfer_id> will be some integer.
- Transfers TO EXISTING FILENAME with shared writing enabled—If shared writing to the target tables is enabled, the format is DDAL$INDEX<index_number> where index_number is some integer.

Transfers to existing databases also create the table RDBVMS$TRANSFER_OWNER or DDAL$TRANSFER_OWNR in a target database. For information on this table, refer to Table A–22. Transfers with shared writing enabled also create the table DDAL$INDEXES. For more information on DDAL$INDEXES, refer to Section A.3.3.

C.6 Access Rights for Transfer Database and Special-Purpose System Tables

Replication Option creates special-purpose system tables in databases that are the sources for replication transfers. Table C–4 shows the access rights that are assigned by Replication Option to each of these special-purpose system tables in those source databases.

Table C–4 Special-Purpose System Tables Access Rights

System Table	System UIC [1,*]	PUBLIC UIC [*,*]
RDB$TRANSFERS	All rights	SELECT+SHOW
RDB$TRANSFER_RELATIONS	All rights	SELECT+SHOW
RDB$CHANGES	All rights	SELECT+INSERT+SHOW[1]
RDB$CHANGES_MAX_TSER	All rights	SELECT+SHOW

[1]The privileges displayed by the SHOW PRIVILEGES statement for the RDB$CHANGES system table does not indicate the privileges actually allowed by Rdb to a user with the UIC [*,*]. Rdb issues the error message, %RDB-E-NO_PRIV, privilege denied by database facility, when such a user tries to read or change the RDB$CHANGES system table.

The transfer monitor creates the transfer database and assigns access rights to the tables defined in it. Routine access to these tables is denied, but if you have special OpenVMS privileges, you, as database administrator, can query the tables when specific information is required. Table C–5 lists the access rights for transfer database tables.

Table C–5 Access Rights for Transfer Database Tables

Transfer Database Table	System UIC [1,*]	PUBLIC UIC [*,*]
DDAL$INVOKE_TIME	All rights	No access
DDAL$TRANSFERS	All rights	No access
DDAL$TRANSFER_SCHEDULE	All rights	No access
DDAL$TRANSFER_FIELDS	All rights	No access
DDAL$TRANSFER_RELATIONS	All rights	No access
DDAL$TRANSFERS_STATUS	All rights	No access
DDAL$ROLLUP_DATABASES	All rights	No access
DDAL$TRANSFER_VIEWS	All rights	No access
DDAL$TRANSFER_DATABASE	All rights	No access

C.7 Transaction Activity During Transfer Execution

When you design your database applications and schedule the execution of Replication Option transfers, you can benefit from understanding what takes place during a transfer. Knowing how a copy process accesses a particular database and knowing what transaction modes are used can alert you to potential sources of access conflict between a copy process and your applications or interactive users.

This section describes what happens during the execution of a copy process image, which is the program that does the actual copying of data. However, a transfer can also include a prologue procedure, which executes prior to the data transfer, and an epilogue procedure, which executes after the transfer. These procedures can also be the source of access conflicts, but are not described in this section.

This section describes the operations performed by a copy process for the following types of transfer:

- Extraction or extraction rollup transfer
- Initial or reinitialized replication transfer
- Replication update transfer

Unless stated otherwise, all copy process transactions specify that the copy process will wait for locks to be assigned. However, implicit transactions performed by some database managers can specify no wait. If a transfer should fail with a lock conflict error, because no wait was specified, this is most likely the result of such an implicit transaction.

C.7.1 Transfer Retries After Recoverable Errors

In the retry clause of a schedule definition, you can optionally specify how many times Replication Option should retry the transfer if that transfer fails due to a recoverable error. If the transfer fails due to a nonrecoverable error, the transfer is suspended. Nonrecoverable errors are those that require user intervention for correction.

Table C–6 lists the recoverable errors recognized by Replication Option.

Table C–6 Replication Option Recoverable Errors

Error Condition	Description
DDAL$_RECCNTEXC	Exceeded allowable number of checkpoint recovery attempts.
DDAL$_RECTLMEXC	Checkpoint recovery time limit exceeded.
RDB$_DEADLOCK	Data you need is locked by another user and you have locked data the other user needs.
RDB$_LOCK_CONFLICT	You have specified the no-wait option and the data your program needs is locked by another user's transaction.
SS$_CONNECFAIL	Connection to a network object timed out or failed.
SS$_LINKABORT	Network partner task aborted logical link.
SS$_LINKDISCON	Network partner task disconnected logical link.
SS$_LINKEXIT	Network partner task exited before confirming logical link.
SS$_PATHLOST	Path to network partner task node was lost.
SS$_PROTOCOL	Network protocol error occurred.
SS$_REJECT	Network object rejected the connection.
SS$_REMRSRC	Insufficient system resources at remote node.

(continued on next page)

Table C–6 (Cont.) Replication Option Recoverable Errors

Error Condition	Description
SS$_SHUT	Remote node is no longer accepting connections.
SS$_THIRDPARTY	Network logical link disconnected by a third party.
SS$_UNREACHABLE	Remote node currently unreachable.

Beware of timing errors after system startup. Sometimes a transfer will fail if information about the remote system has not been properly established. The following example uses DECnet as the communications software, but the problem it illustrates could occur with other software that requires setup at system startup. The nonrecoverable DECnet error NETWORK OBJECT UNKNOWN AT REMOTE NODE means that the RDBSERVER object is not installed or is not properly installed at the remote site. This error aborts the transfer without retries. For example, if the system has recently been rebooted, and the NCP command SET KNOWN NODES ALL has not completed, the transfer fails. To avoid such failures, you can write a prologue file, similar to the following, that checks the system uptime and waits until the system has been up at least 30 minutes before starting the transfer.

```
$ say = "write sys$output"
$ half_hour_ago = F$CVTIME("-:30")
$ BOOT          = F%CVTIME(F$GETSYI("BOOTTIME"))
$ say "SYSTEM boot time was    ''boot'."
$ say "Current time - 30 min is ''half_hour_ago'."
$ IF half_hour_ago .LTS. boot THEN WAIT 00:30:00
```

C.7.2 Common Operations

When a copy process image, DDAL$COPY_PROCESS.EXE, runs, it performs the following initial steps regardless of the transfer type:

1. Prologue failure check

   Checks to see if a prologue procedure executed and failed. If prologue failure occurs, the copy process does not perform the transfer.

2. Read transfer definition

   Attaches to the transfer database, starts a read-only transaction, reads the transfer definition, commits the transaction, and detaches from the transfer database. At this point, the copy process makes no further references to the transfer database for the remainder of transfer execution.

C.7.3 Extraction or Extraction Rollup Transfer

This section describes the operation of extraction and extraction rollup transfers. Each of these transfer types can create a target database. Only extraction transfers are permitted to copy data into an existing target database.

C.7.3.1 Transfer to a New Target Database

The copy process performs the following steps for an extraction or extraction rollup transfer that creates a new target database. The steps listed assume the checkpoint option is not enabled. If the checkpoint option is enabled, read/write transactions are used when accessing the target database and these transactions are each committed at regular intervals, as specified in the checkpoint clause.

1. Attach to source database or databases

   For each source database involved in the transfer, the copy process attaches to the database and inquires about the type and version of the database management software used. Database type influences the transaction mode that can be used; database version determines what features are supported by the system.

2. Check that data definitions still exist

   For extraction transfers, but not extraction rollup, the copy process starts a read-only transaction on the source database, checks that each table and column referenced in the transfer definition still exists in the source database, and then rolls back the transaction.

3. Create target database

   The copy process creates the target database files. The addition of domain and table definitions occurs later.

4. Determine target database system and version

   The copy process starts a batch-update transaction on the target database, inquires about the type and version of the database management software used, and commits the transaction. Database software must be Rdb; the software version determines what features are supported by Rdb.

5. Start source read-only transaction

   The copy process starts a read-only transaction on the source database or databases. All the data definitions and data extracted from the source database are read within this same transaction so that a consistent state of the source data is seen.

6. Add Replication Option domains and table to the target database

   a. Start a batch-update transaction on the target database, add each domain needed to support the transfer type, and commit the transaction.

   b. Start a batch-update transaction on the target database, add the RDB$VINTAGE table, and commit the transaction.

   c. If data definitions were added or modified, detach from the target database and reattach. This ensures that the protected write transaction in step 7 forces concurrently executing transfers to perform their transactions one after the other so that a unique identification number can be assigned to each transfer. This may seem pointless given that the transfer that creates the target database files will be the only transfer dealing with that target database. However, this same logic is used for transfers to existing databases (see Section C.7.3.2), where concurrently executing transfers to the same target database are possible.

7. Mark the transfer busy

   The copy process starts a read/write transaction, reserving RDB$VINTAGE for protected write, stores a row in RDB$VINTAGE with the RDB$VINTAGE_TIMESTAMP column set to "17-NOV-1858" and the RDBVMS$VINTAGE_TRANSFER_BUSY column set to "T", and commits the transaction.

8. Transfer domain definitions

   The copy process starts a batch-update transaction on the target database, reads domain, table and view definitions from the source database or databases, creates domain definitions in the target database, and commits the target transaction.

9. Transfer table definitions

   The copy process starts a batch-update transaction on the target database, creates the table definition, and commits the transaction for each table to be transferred.

10. Transfer view definitions

    The following copy process steps apply to extraction transfers but not to extraction rollup transfers:

    a. Read the view definitions from the source database and sort them in order of dependence, as one view might be defined in terms of another.

    b. For each view, start a batch-update transaction on the target database, create the view definition, and commit the transaction.

11. Transfer data rows

    The copy process starts a batch-update transaction on the target database, transfers data from the source to the target table, and commits the transaction for each table to be transferred. You can use the DDAL$EXTRACT_DATA environment variable to disable the data transfer step. See Section B.9.

12. Mark the transfer done

    The copy process modifies the transfer's RDB$VINTAGE row, setting RDBVMS$VINTAGE_TRANSFER_BUSY to "F", meaning FALSE, and setting RDB$VINTAGE_TIMESTAMP to the date and time the read-only transaction on the source database or databases was started.

13. Commit and detach

    The copy process commits the source and target transactions and detaches from the source and target databases.

C.7.3.2 Transfer to an Existing Target Database

The copy process performs the following steps for an extraction transfer that copies data to an existing target database. Extraction rollup transfers cannot be defined to use existing databases as targets.

1. Attach to source database

   The copy process attaches to the source database and inquires about the type and version of the database management software used. Database type influences the transaction mode that can be used; database version determines what features are supported by the system.

2. Check that data definitions still exist

   The copy process starts a read-only transaction on the source database, checks that each table and column referenced in the transfer definition still exists in the source database, and then rolls back the transaction.

3. Attach to the target database

   The copy process attaches to the existing target database. The addition of domain and table definitions occurs later.

4. Determine target database system and version

   The copy process starts a read/write transaction on the target database, inquires about the type and version of the database management software used, and commits the transaction. The software version determines what features are supported by the target database system.

5. Start source read-only transaction

The copy process starts a read-only transaction on the source database. All the data definitions and data extracted from the source database are read within this same transaction so that a consistent state of the source data is seen.

6. Add Replication Option domains, tables, and index to the target database

   a. Start a read/write transaction on the target database. If missing, add each domain needed to support this type of transfer and commit the transaction.

   b. Start a read/write transaction on the target database. If it is an Rdb database, add the RDB$VINTAGE and RDBVMS$TRANSFER_ OWNER tables if not present; if RDB$VINTAGE is present and needs conversion, convert the RDB$VINTAGE table to the latest format. If it is a database other than Rdb, add the DDAL$TRANSFER_INFO and DDAL$TRANSFER_OWNR tables if not present. For a transfer in which shared writing is enabled, no matter what the database type is, add the DDAL$INDEXES table. Finally, commit the transaction.

   c. For target systems that support index creation, start a read/write transaction on the target database, add indexes as needed for the Replication Option tables, and commit the transaction.

      When adding domains, tables or indexes, lock conflict errors can occur, for example, when several transfers into the same target database are executed at the same time. If this happens, the transaction is rolled back, the copy process detaches from the target database and hibernates for about 30 seconds, reattaches, and then retries the failed transaction.

      If data definitions were added or modified, the copy process detaches from the target database and reattaches. For Rdb targets this ensures that the protected write transaction in step 7 forces concurrently executing transfers to perform their transactions one after the other so that a unique identification number can be assigned to each transfer. For all other database systems that do not support protected write transactions, unique identification numbers will be assigned but concurrent writers can lead to lock conflicts and transfers that fail and must retry.

7. Mark the transfer busy

   For Rdb target databases, the copy process starts a read/write transaction reserving RDB$VINTAGE for protected write; checks if this transfer has a row in RDB$VINTAGE; inserts or updates the row in RDB$VINTAGE, setting the RDB$VINTAGE_TIMESTAMP column to "17-NOV-1858" and

the RDBVMS$VINTAGE_TRANSFER_BUSY column to "T" for TRUE; and commits the transaction.

For all other target databases, the copy process starts a read/write transaction; checks if this transfer has a row in DDAL$TRANSFER_INFO; inserts or updates the row in DDAL$TRANSFER_INFO, setting the DDAL$TRANSFER_TIME column to "17-NOV-1858" and the DDAL$TRANSFER_BUSY column to "T" for TRUE; and commits the transaction.

8. Transfer domain definitions

   a. Start a read/write transaction on the target database.

   b. Read domain, table, and view definitions from the source database.

   c. For each domain, if it already exists in the target database, check ownership and check if the definition is the same as for the source domain. The checks are performed only for Rdb or Distributed Option targets, not for targets accessed directly through an Rdb Transparent Gateway. The latter does not store domain definitions in permanent storage.

   d. Create or modify domain definitions in the target database as needed, recording the ownership of each in RDBVMS$TRANSFER_OWNER for Rdb target databases or DDAL$TRANSFER_OWNR for all other target databases. Modification of domain definitions can only be done for Rdb target databases. Neither Distributed Option nor Rdb Transparent Gateways support altering domain definitions.

   e. Commit the target transaction.

9. Transfer table definitions

   Transfer or modify the definitions of tables in the target database. This is done using one read/write transaction per table.

   a. Start a read/write transaction on the target database.

   b. If the table already exists in the target database, check ownership and check if the definition is the same as the source table. Modify the table definition if it has changed. Modification of table definitions is done only for Rdb targets. Neither Distributed Option nor Rdb Transparent Gateways support altering table definitions.

      If the table does not exist in the target database, create the table definition and insert a row in RDBVMS$TRANSFER_OWNER (for Rdb targets) or DDAL$TRANSFER_OWNR (for all other targets) to record the fact that the table is owned by this transfer.

c. Commit the transaction.

10. Transfer view definitions

    a. Read the view definitions from the source database, sort them in order of dependence because one view might be defined in terms of another, and drop the view definitions from the target database.

    b. For each view, start a read/write transaction on the target database, create the view definition, and commit the transaction.

11. Delete old target data

    For each table to be transferred:

    a. Start a read/write transaction on the target database.

    b. If the table exists from a prior execution of the transfer, erase those rows in the table owned by this transfer.

    c. Commit the transaction.

    d. If data was deleted, detach from and reattach to the target database. For Rdb and perhaps for other database systems, this ensures that deleted rows can be replaced by new ones.

12. Transfer new target data

    For each table to be transferred:

    a. Start a read/write transaction on the target database.

    b. Transfer data from the source to the target table.

    c. Commit the transaction.

13. Mark the transfer done

    For Rdb target databases, modify the transfer's RDB$VINTAGE row: set RDBVMS$VINTAGE_TRANSFER_BUSY to "F", meaning FALSE, and set RDB$VINTAGE_TIMESTAMP to the date and time at which the read-only transaction on the source database was started.

    For all other target databases, modify the transfer's DDAL$TRANSFER_INFO row: set DDAL$TRANSFER_BUSY to "F", meaning FALSE, and set DDAL$TRANSFER_TIME to the date and time at which the read-only transaction on the source database was started.

14. Commit and detach

    Commit the source and target transactions, and detach from the source and target databases.

C.7.4 Initial or Reinitialized Replication Transfer

This section describes the operation of initial or reinitialized replication transfers; replication update transfers are covered in Section C.7.5. Replication transfers can either create a target database or copy data into an existing target database.

C.7.4.1 Transfer to a New Target Database

The copy process performs the following steps for an initial or reinitialized replication transfer that creates a new target database:

1. Create Replication Option system tables in source database
    a. Attach to the source database and start a read/write transaction.
    b. Create each Replication Option system domain that does not exist in the source database.
    c. Create each Replication Option system table that does not exist in the source database. These tables include: RDB$TRANSFERS, RDB$TRANSFER_RELATIONS, RDB$CHANGES, and RDB$CHANGES_MAX_TSER.
    d. Commit the transaction and detach from the source database. A data row must be inserted in one of these newly created system tables. This cannot be done within the same attach during which the table was created.
    e. Attach to the source database a second time, and start a read/write transaction.
    f. Store a data row in the RDB$CHANGES_MAX_TSER table if the row is not already there, and commit the transaction.
    g. An option exists to either add or drop indexes on the RDB$CHANGES table. If that option is invoked, start a read/write transaction on the source database, add or drop the indexes as needed, and commit the transaction.
    h. Detach from the source database and reattach.
    i. Start a read-only transaction on the source database, check that the account under which the transfer is being run has access to the database, and roll back the transaction.
    j. Inquire about the version of the database management software used. The software version determines what features are supported by the system.

2. Store transfer definition in source database

   a. Start a read/write transaction on the source database.

   b. Check the source database to see if the transfer definition is already stored there and determine if it is unchanged. This will usually be the case whenever a replication transfer has to be reinitialized and the initial transfer re-executed.

      1. If the transfer exists and has not changed, roll back the transaction.

      2. If the transfer definition has not been stored in the source database, store the new one.

      3. If the definition has changed, erase the existing definition and store the new one.

         Erasing and storing a transfer definition in the source database requires that the copy process have exclusive access to the database. Without exclusive access, the erase or store operation results in a lock-conflict error. If this happens, the copy process rolls back the transaction, detaches from the source database, hibernates for about 30 seconds, reattaches, restarts the transaction, and retries the failed operation.

   c. If a transfer definition has just been stored, you can make the transfer operation terminate at this point by enabling the DDAL$STORE_DEF_SEPARATE environment variable. This variable is useful when it is necessary to initialize several replication transfers at the same time.

   d. Once the transfer definition is stored, check that the account under which the copy process is running has SELECT privilege on each of the tables to be transferred. If not, roll back the transaction. If so, commit the transaction. Exclusive access to the source database is relinquished once the transaction terminates. At this point, user applications can attach to the source database and make data additions and modifications to the tables being transferred. These changes will be recorded in the RDB$CHANGES table, as well as in the normal database storage areas.

3. Create target database

   Create the target database files. The addition of domain and table definitions occurs later.

4. Determine target database system and version

Start a batch-update transaction on the target database, inquire about the type and version of the database management software used, and commit the transaction. The database type must be Rdb; the software version determines what features are supported.

5. Start source read-only transaction

   Start a read-only transaction on the source database. All the data definitions and data extracted from the source database will be read within this same transaction so that a consistent state of the source data is seen.

6. Add Replication Option domains and table to the target database

   a. Start a batch-update transaction on the target database, add each domain needed to support this type of transfer, and commit the transaction.

   b. Start a batch-update transaction on the target database, add the RDB$VINTAGE table, and commit the transaction.

   c. If data definitions were added or modified, detach from the target database and reattach. This ensures that the protected write transaction in step 7 forces concurrently executing transfers to perform their transactions one after the other so that a unique identification number can be assigned to each transfer. This may seem pointless given that the transfer that creates the target database files will be the only transfer dealing with that target database. However, this same logic is used for transfers to existing databases (see Section C.7.4.2), where concurrently executing transfers to the same target database are possible.

7. Mark the transfer busy

   a. Start a read/write transaction, reserving RDB$VINTAGE for protected write.

   b. Store a row in RDB$VINTAGE with the RDB$VINTAGE_TIMESTAMP column set to "17-NOV-1858", the RDBVMS$VINTAGE_TRANSFER_BUSY column set to "T", and the RDB$VINTAGE_TSER column set to the maximum transaction serial number currently in use in rows within the RDB$CHANGES table of the source database.

   c. Commit the transaction.

8. Transfer domain definitions

Start a batch-update transaction on the target database; read domain, table, and view definitions from the source database or databases; create domain definitions in the target database, adding one called DDAL$DBKEY; and commit the target transaction.

9. Transfer table definitions

   For each table to be transferred, the copy process starts a batch-update transaction on the target database, creates the table definition, adding to it the DDAL$DBKEY column, and commits the transaction.

10. Transfer view definitions

    a. Read the view definitions from the source database and sort them in order of dependence, because one view might be defined in terms of another.

    b. For each view, start a batch-update transaction on the target database, create the view definition, and commit the transaction.

11. Transfer data rows

    For each table to be transferred:

    The copy process starts a batch-update transaction on the target database, transfers data from the source to the target table, and commits the transaction. You can use the DDAL$EXTRACT_DATA environment variable to disable the data transfer step. See Section B.9.

12. Create indexes

    For each table to be transferred:

    Start a batch-update transaction on the target database, add a table index based on the DDAL$DBKEY column, and commit the transaction. For those target systems that support it, this will be defined as an index requiring unique values (but not for replication transfers WITH NO DELETE). For those target systems that do not support unique indexes or for replication transfers WITH NO DELETE, simple sorted indexes will be created. After the transfer completes, the customer will be required to add column constraints to enforce uniqueness of values in the DDAL$DBKEY columns (but not for replication transfers WITH NO DELETE).

13. Mark the transfer done

    Modify the transfer's RDB$VINTAGE row, setting RDBVMS$VINTAGE_TRANSFER_BUSY to "F" and setting RDB$VINTAGE_TIMESTAMP to the date and time at which the read-only transaction on the source database.

14. Commit and detach

Commit the source and target transactions and detach from the source and target databases.

C.7.4.2 Transfer to an Existing Target Database

The copy process performs the following steps for an initial or reinitialized replication transfer that copies data to an existing target database:

1. Create Replication Option system tables in source database
    a. Attach to the source database and start a read/write transaction.
    b. Create each Replication Option system domain that does not exist in the source database.
    c. Create each Replication Option system table that does not exist in the source database. These include: RDB$TRANSFERS, RDB$TRANSFER_RELATIONS, RDB$CHANGES, and RDB$CHANGES_MAX_TSER.
    d. Commit the transaction and detach from the source database. A data row must be inserted in one of these newly created system tables. This cannot be done within the same attach during which the table was created.
    e. Attach to the source database a second time and start a read/write transaction.
    f. Store a data row, if one does not exist, in the RDB$CHANGES_MAX_TSER table. Commit the transaction.
    g. An option exists to either add or drop indexes on the RDB$CHANGES table. If that option is invoked, start a read/write transaction on the source database, add or drop the indexes as needed, and commit the transaction.
    h. Detach from the source database and reattach.
    i. Start a read-only transaction on the source database, check that the account under which the transfer is being run has access to the database, and roll back the transaction.
    j. Inquire about the version of the database management software used. The software version determines what features are supported.
2. Store transfer definition in source database
    a. Start a read/write transaction on the source database.

b. Check the source database to see if the transfer definition is already stored there and determine if it is unchanged. This will usually be the case whenever a replication transfer has to be reinitialized and the initial transfer re-executed.

   1. If the transfer exists and has not changed, roll back the transaction.
   2. If the transfer definition has not been stored in the source database, store the new one.
   3. If the definition has changed, erase the existing definition and store the new one.

      Erasing and storing a transfer definition in the source database requires that the copy process have exclusive access to the database. Without exclusive access, the erase or store operation results in a lock-conflict error. If this happens, the copy process rolls back the transaction, detaches from the source database, hibernates for about 30 seconds, reattaches, restarts the transaction, and retries the failed operation.

c. If a transfer definition has just been stored, you can make the transfer operation terminate at this point by enabling the DDAL$STORE_DEF_SEPARATE environment variable. This variable is useful when it is necessary to initialize several replication transfers at the same time.

d. Once the transfer definition is stored, check that the account under which the copy process is being run has SELECT privilege on each of the tables to be transferred. If not, roll back the transaction. If so, commit the transaction. Exclusive access to the source database is relinquished once the transaction is terminated. At this point, user applications can attach to the source database and make data additions and modifications to the tables being transferred. These changes will be recorded in the RDB$CHANGES table, as well as in the normal database storage areas.

3. Attach to the target database

   The copy process attaches to the existing target database. The addition of domain and table definitions occurs later.

4. Determine target database system and version

   Start a read/write transaction on the target database, inquire about the type and version of the database management software used, and commit the transaction. The software version determines what features are supported by the target database system.

5. Start source read-only transaction

Start a read-only transaction on the source database. All the data definitions and data extracted from the source database will be read within this same transaction so that a consistent state of the source data is seen.

6. Add Replication Option domains, tables, and indexes to the target database

   a. Start a read/write transaction on the target database. If missing, add each domain needed to support this type of transfer, and commit the transaction.

   b. Start a read/write transaction on the target database; if it is an Rdb database, add the RDB$VINTAGE and RDBVMS$TRANSFER_OWNER tables if not present; if RDB$VINTAGE is present and needs conversion, convert the RDB$VINTAGE table to the latest format. If it is a database other than Rdb, add the DDAL$TRANSFER_INFO and DDAL$TRANSFER_OWNR tables if not present. For a transfer in which shared writing is enabled, add the DDAL$INDEXES table. Finally, commit the transaction.

   c. For target systems that support index creation, start a read/write transaction on the target database, add indexes as needed for the Replication Option tables, and commit the transaction. For those target systems that support it, this will be defined as an index requiring unique values. For those target systems that do not support unique indexes, simple sorted indexes will be created. After the transfer completes, the customer will be required to add column constraints to enforce uniqueness.

      When adding domains, tables, or indexes, lock conflict errors can occur, for example, when several transfers into the same target database are executed at the same time. If this happens, the transaction is rolled back, the copy process detaches from the target database and hibernates for about 30 seconds, reattaches, and then retries the failed transaction.

      If data definitions were added or modified, the copy process detaches from the target database and reattaches. For Rdb target databases, this ensures that the protected write transaction in step 7 forces concurrently executing transfers to perform their transactions one after the other so that a unique identification number can be assigned to each transfer.

      For all other database systems that do not support protected write transactions, unique identification numbers will be assigned, but concurrent writers can lead to lock conflicts and transfers that fail and must retry.

7. Mark the transfer busy

   a. Start a read/write transaction. For Rdb targets only, the transaction will reserve the RDB$VINTAGE table for protected write.

   b. For Rdb targets, insert a row in RDB$VINTAGE with the RDB$ VINTAGE_TIMESTAMP column set to "18-NOV-1858", the RDBVMS$ VINTAGE_TRANSFER_BUSY column set to "T", and the RDB$ VINTAGE_TSER column set to the maximum transaction serial number currently in use in rows within the RDB$CHANGES table of the source database.

      For all other target systems, insert a row in DDAL$TRANSFER_ INFO with the DDAL$TRANSFER_TIME column set to "17-NOV-1858", the DDAL$TRANSFER_BUSY column set to "T", and the DDAL$TRANSFER_TSER column set to the maximum transaction serial number currently in use in rows within the RDB$CHANGES table of the source database.

   c. Commit the transaction.

8. Transfer domain definitions

   a. Start a read/write transaction on the target database.

   b. Read domain, table, and view definitions from the source database.

   c. For each domain, if it already exists in the target database, check ownership and check if the definition is the same as for the source domain. The checks are performed only for Rdb or Distributed Option targets, not for targets accessed directly through an Rdb Transparent Gateway. The latter does not store domain definitions in permanent storage.

   d. Create or modify domain definitions in the target database as needed, including one called DDAL$DBKEY (and optionally, one called DDAL$ROW_OWNER). Record the ownership of each domain definition in RDBVMS$TRANSFER_OWNER (for Rdb targets) or DDAL$TRANSFER_OWNR (for all other targets). Modification of domain definitions can be done only for Rdb databases; neither Distributed Option nor Rdb Transparent Gateways support altering domain definitions.

   e. Commit the target transaction.

9. Drop indexes

For each replicated table in the target database, start a read/write transaction, drop the table index created by Replication Option based on the DDAL$DBKEY column if it exists, and commit the transaction. You can use the DDAL$TARGET_INDEX_OPTIONS environment variable to prevent this step from being executed, leaving existing indexes in place.

10. Alter table definitions

    For each table to be replicated where the table already exists in the target database, check ownership and check if the definition is the same as the source table. If the definition has changed and the target system is Rdb, start a read/write transaction, alter the target definition, and commit the transaction. If the definition has changed and the target system is not Rdb, there is no support for altering table definitions. In such a case signal an error. You can use the DDAL$CHECK_TARGET_DEFINITIONS option to suppress definition checking.

11. Create table definitions

    For each table to be replicated where it does not yet exist in the target database:

    a. Start a read/write transaction on the target database.

    b. Create the table definition, adding to it a DDAL$DBKEY column (and if shared writing is enabled, a DDAL$ROW_OWNER column).

    c. Insert a row in the RDBVMS$TRANSFER_OWNER table (for Rdb targets) or DDAL$TRANSFER_OWNR table (for all other targets) to record the fact that the table is owned by this transfer.

    d. Commit the transaction.

12. Transfer view definitions

    a. Read the view definitions from the source database; sort them in order of dependence because one view might be defined in terms of another; and drop the view definitions from the target database.

    b. For each view, start a read/write transaction on the target database, create the view definition, and commit the transaction.

13. Delete old data

    For each replicated table in the target database:

    a. Start a read/write transaction on the target database.

    b. Delete all data in the table.

    c. Commit the transaction.

Detach from and reattach to the target database. This ensures for Rdb (and possibly other) targets erased data rows can be replaced by new ones.

14. Insert new data

    For each replicated table, start a read/write transaction on the target database, copy the source data to the target table, and commit the transaction.

15. Create indexes

    For each replicated table in the target database:

    a. Start a read/write transaction.

    b. Create a table index based on the DDAL$DBKEY column. If shared writing is enabled, base the index on two columns, DDAL$DBKEY and DDAL$ROW_OWNER. For target systems that support it, make it an index that enforces unique column values. For those that do not support unique indexes, customers must add column constraints to achieve the same effect. Unique indexes are not created in the case of a replication transfer with NO DELETE. Insert a row in the RDBVMS$TRANSFER_OWNER table (for Rdb targets) or the DDAL$TRANSFER_OWNR table (for all other targets) to record the fact that the index is owned by this transfer.

    c. Commit the transaction.

    You can use the DDAL$TARGET_INDEX_OPTIONS environment variable to suppress the dropping and creating of these indexes.

16. Mark the transfer done

    For Rdb target databases, modify the transfer's RDB$VINTAGE row by setting RDBVMS$VINTAGE_TRANSFER_BUSY to "F", and setting RDB$VINTAGE_TIMESTAMP to the date and time at which the read-only transaction on the source database was started.

    For all other target databases, modify the transfer's DDAL$TRANSFER_INFO row by setting DDAL$TRANSFER_BUSY to "F", meaning FALSE, and setting DDAL$TRANSFER_TIME to the date and time at which the read-only transaction on the source database was started.

17. Commit and detach

    Commit the source and target transactions and detach from the source and target databases.

C.7.5 Replication Update Transfer

This section describes the copy process steps in a replication update transfer. The steps are the same, whether the initial replication transfer created the target database or the initial replication transfer was to an existing database.

1. Add or drop indexes

   Replication Option provides an option to add or drop indexes on the source RDB$CHANGES table. If that option is chosen, the copy process takes the following steps:

   a. Starts a read/write transaction on the source database, adds or drops indexes as necessary, commits the transaction, and detaches from and reattaches to the source database.

2. Check access privilege to the source database

   a. Check that the user account under which the copy process runs has SELECT privilege on the source database.

   b. Start a read-only transaction on the source database, check that the user account has SELECT privilege for each of the tables to be transferred, and roll back the transaction.

3. Start read-only transaction at a quiet point

   To ensure that a copy process reads transactions correctly from the RDB$CHANGES table, there must be no active transactions writing to RDB$CHANGES when the copy process starts its read-only transaction.

   a. Attach to the source database a second time (the copy process is now attached twice).

   b. Start a read/write transaction within the second attach, reserving RDB$CHANGES for protected read.

   c. Start a read-only transaction on the first attach to the source database. This can only happen after the protected read transaction has started. Therefore, with respect to RDB$CHANGES, a quiet point has been reached.

   d. Roll back the protected read transaction and detach from the source database (second attach).

4. Determine what was the last transaction applied to the target

Examine the information in the target database to identify the last source transaction applied to it. This indicates at what point the process should continue.

a. Attach to and start a read-only transaction on the target database.

b. If RDB$VINTAGE needs to be converted to the current format, roll back the read-only transaction, start a read/write transaction, convert RDB$VINTAGE, commit the transaction, and restart the read-only transaction.

c. Read this transfer's data row in the target RDB$VINTAGE or DDAL$TRANSFER_INFO table to find the transaction serialization number associated with the latest source transaction applied to the target database.

d. Roll back the read-only transaction.

5. Transfer changes from source to target

Changes are applied to the target database in transactions in the same order they were committed in the source database. The transaction modes used in the source transactions have no bearing on the transaction modes used in the target database.

a. Start a read/write transaction on the target database.

b. Read the changes from the RDB$CHANGES table in the source database for one source transaction.

c. Determine if the changes are to tables being replicated. The copy process ignores those that are not. If the transfer is defined with row selectivity, the copy process decides if the change should be applied. Refer to Section 5.2.2.1 for information about row selectivity. Finally, the copy process makes applicable data changes in the target table. If this is a REPLICATION TRANSFER, applicable changes are to insert, update, or delete each row that was inserted, updated, or deleted from the source table. If this is a REPLICATION TRANSFER WITH NO DELETE applicable changes are to insert and alter each row that was inserted or updated in the source database and to set the DDAL$DBKEY column to null for each row deleted from the source table.

d. Modify this transfer's row in the target RDB$VINTAGE or DDAL$TRANSFER_INFO table, storing the transaction serialization number of the source transaction and the date and time that the source transaction was committed.

e. Commit the target transaction.

One of the benefits of the replication method is in failure recovery. For example, suppose there are 100 transactions in the RDB$CHANGES table to be replicated and that a network failure occurs during the transfer of the 61st transaction. After the network failure is repaired and the transfer is restarted, Replication Option restarts the 61st transaction. It does not have to repeat the first 60 transactions because they will have already been committed in the target database.

6. Change target vintage time

   Start a read/write transaction on the target database, modify the transfer's row in the RDB$VINTAGE or DDAL$TRANSFER_INFO table with the current date and time, and commit the transaction. If there were no source transactions to be applied to the target database, the current date and time are recorded in the transfer's vintage row to show the most recent time that the transfer was run.

7. Commit the read-only source transaction

8. Record the transfer's high-water mark in the source database (The concept of a transfer's high-water mark is explained in the following paragraphs.)

   Start a read/write transaction on the source database reserving RDB$TRANSFERS for exclusive access, modify the transfer's high-water mark, and commit the transaction.

   Several transfers can be defined to replicate the same source database. Each transfer can execute on a different schedule. When one transfer completes, the same transactions it read from RDB$CHANGES might need to be read by other transfers as well. Eventually the rows in RDB$CHANGES need to be deleted. To determine which rows have been processed by all transfers that need them, a high-water mark is employed. Each transfer has a row in the RDB$TRANSFERS table in the source database. One of the columns in that table records the transaction serialization number of the last source transaction to be processed by a given transfer. This is the transfer's high-water mark.

9. Delete rows no longer needed in RDB$CHANGES

   Normal Replication Option behavior is for RDB$CHANGES to be purged of rows no longer needed after each update transfer executes. Replication Option provides an option to do this purging on a more selective basis. The following steps describe the purging process:

   a. Start a read/write transaction on the source database.

   b. Find the lowest high-water mark of all replication transfers on the source database.

c. Delete all rows in RDB$CHANGES belonging to transactions whose transaction serialization number is lower than the lowest high-water mark. At least one row is left in RDB$CHANGES after this procedure, and it is used as part of a validity test the next time an update transfer is started.

d. Commit the transaction.

10. Detach from source and target databases

D
Examples

This appendix provides examples and information on:

- Creating, compiling, and linking applications created using Replication Option statements in dynamic SQL and SQL module language programs
- Using Rdb Transparent Gateway to RMS to populate a target database

D.1 Using an SQL Application to Create and Control the Execution of Transfers

The following example shows an application program that issues commands to Replication Option to create and control the execution of transfers. The sample application program has two modules: a main module written in DEC C and another module written in the SQL module language.

Note

This application is suited for any source and target database supported by Replication Option.

Shown first is a DCL command procedure used to control execution of the application program. The command procedure is followed by the two test program modules.

```
$!++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
$!
$!      TEST_DYNAMIC_SQL.COM
$!
$!  This DCL procedure tests the ability of an application program to issue SQL
$!  Replication Option statements, using the dynamic SQL interface. The general
$!  method is as follows.
$!
$!  1. The main application code is written in some host language, such as
$!      DEC C. This program generates text strings which are complete
$!      Replication Option statements, such as CREATE TRANSFER  . . .  ;
```

```
$!
$!  2. These Replication Option commands are then passed to a procedure
$!      in another module, written in the SQL module language.
$!
$!  3. The SQLMOD module calls the dynamic SQL interface to interpret the
$!      Replication Option statement at runtime. For certain host languages, such as
$!      DEC C, one can avoid the use of SQLMOD procedures and include the
$!      calls to the dynamic SQL interface directly. This example shows the
$!      more general way to call Replication Option, which applies equally well
$!      for languages which do not have an SQL precompiler.
$!
$!  4. Status information returned from dynamic SQL indicates only whether or
$!      not SQL accepted the statement and was able to execute it. In the case
$!      of the Replication Option statement START TRANSFER NOW WAIT, a success
$!      status does not mean that the transfer executed successfully. It
$!      simply means that the transfer was started and that either (1) the
$!      transfer completed successfully, or (2) the transfer failed (after
$!      an appropriate number of retries were attempted, if so defined in
$!      a transfer schedule).
$!
$!-----------------------------------------------------------------------------
$
$!  Copy a test database to play with (one that already has data in it)
$
$       COPY/NOLOG original-area:original.RDB,.SNP test-area:test_source.*
$
$!  Define a foreign command for invoking the test image so that a command line
$!  argument can be passed to it.
$
$       DO_DDD_COMMAND := $ test-area:TEST_DYNAMIC_TRANSFERS.EXE
$
$!  Create an extraction transfer to a new database. The test program will
$!  print out what it is doing.
$
$       DO_DDD_COMMAND "create transfer"
$
$!  Examine the transfer definition for correctness. The SHOW command cannot
$!  be executed from an application program, only from interactive SQL.
$
$       SQL

        SHOW TRANSFER (DEFINITION) DYN_1;
        EXIT;
$
$!  Create a schedule for the transfer.
$
$       DO_DDD_COMMAND "create schedule"
$
$!  Examine the transfer schedule for correctness. The schedule is set far
$!  into the future because we intend to execute the transfer under application
$!  control by direct command farther on. All that matters for this test is to
$!  show that an application program can in fact create a transfer schedule.
$
$       SQL

        SHOW TRANSFER (SCHEDULE) DYN_1;
```

```
        EXIT;
$
$!  Execute the transfer NOW, waiting for it to complete (whether successfully
$!  or not). Don't be confused by the fact that we don't mention the NOW WAIT
$!  option here. That is embedded in the C program code.
$
$       DO_DDD_COMMAND "start transfer"
$
$!  Examine the transfer status to insure that the transfer actually ran.
$
$       SQL
        SHOW TRANSFER (STATUS) DYN_1;
        EXIT;
$
$!  Stop the transfer.
$
$       DO_DDD_COMMAND "stop transfer"
$
$!  Verify that the transfer state is suspended.
$
$       SQL
        SHOW TRANSFER (STATUS) DYN_1;
        EXIT;
$
$!  Drop the transfer schedule.
$
$       DO_DDD_COMMAND "drop schedule"
$
$!  Verify that the transfer no longer has a schedule but that it still has a
$!  status.
$
$       SQL
        SHOW TRANSFER (SCHEDULE) DYN_1;
        SHOW TRANSFER (STATUS) DYN_1;
        EXIT;
$
$!  Drop the transfer definition.
$
$       DO_DDD_COMMAND "drop transfer"
$
$!  Verify that the transfer no longer exists.
$
$       SQL
        SHOW TRANSFER (STATUS) DYN_1;
        EXIT;
$
```

The following is a C language module that is used with the DYNAMIC_ SQL.SQLMOD module in the sample application.

```
/*++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
**
**      TEST_DYNAMIC_TRANSFERS.C
**
**  This program module, in conjunction with DYNAMIC_SQL.SQLMOD, illustrates
**  how Replication Option statements in the SQL language can be executed
**  under application program control.
**
**  A practical application program (which this is not) would provide robust
**  error handling and a robust interface for users to describe what they want
**  to transfer, how, when, etc. This in turn would be converted by the appli-
**  cation code into valid SQL statements for managing Replication Option
**  transfers. These SQL statements are then passed to SQL using SQL's
**  dynamic interface.
**
**  This sample program lacks a user interface. Instead, sample SQL statements
**  are hard-coded into the several functions in this module.
**
**--------------------------------------------------------------------------*/

#include <stdio.h>

#define check {if (sqlcode<0) SQL$SIGNAL();}
/*      FUNCTION TEMPLATES      */

static void create_schedule ( void );
static void create_transfer ( void );
static void drop_schedule   ( void );
static void drop_transfer   ( void );
static void start_transfer  ( void );
static void stop_transfer   ( void );

int main
/*****************************************************************************

    This is the main function for this application program. A single character
    string argument is passed to indicate the type of operation to be performed.
    The legal arguments are:

        "create schedule"
        "create transfer"
        "drop schedule"
        "drop transfer"
        "start transfer"
        "stop transfer"

    Other SQL statements used with Data Distributor are not shown in this
    sample program, although they are allowed in dynamic SQL. What is NOT
    allowed is the SHOW TRANSFER statement.

*****************************************************************************/
( argc, argv )

int      argc;
char    *argv[];
```

```
{
    int          c;
    int          i;
    char        *args[] = { "create schedule",
                            "create transfer",
                            "drop schedule",
                            "drop transfer",
                            "start transfer",
                            "stop transfer" };
    /*  Search the list of expected arguments; convert to an integer token  */

    c = 999;

    for (i = 0; i < 6; i++)
        if (strcmp ( argv[1], args[i] ) == 0) c = i;

    /*  Dispatch to the appropriate function or issue an error message  */

    switch (c)
        {
        case 0 : create_schedule (); break;
        case 1 : create_transfer (); break;
        case 2 : drop_schedule   (); break;
        case 3 : drop_transfer   (); break;
        case 4 : start_transfer  (); break;
        case 5 : stop_transfer   (); break;

        default : printf ( "\n  Invalid argument <%s>\n\n", argv[1] );
        }
} /* end of function main */

/*****************************************************************************/
static void create_schedule ( void )
{
    char *create_schedule = "CREATE SCHEDULE FOR DYN_1\
                             START 11-JUL-2025 01:00\
                             EVERY DAY AT 01:00\
                             RETRY 5 TIMES\
                             RETRY EVERY 0 00:20:00";
    long sqlcode;

    printf ("\n*** Start of program\n");

    printf ("*** Creating transfer schedule:\n");
    printf ("***   CREATE SCHEDULE FOR DYN_1\n");
    printf ("***     START 11-JUL-2025 01:00\n");
    printf ("***     EVERY DAY AT 01:00\n");
    printf ("***     RETRY 5 TIMES\n");
    printf ("***     RETRY EVERY 0 00:20:00\n");
    interpret ( &sqlcode, create_schedule ); check

    printf ("*** End of program\n\n");

} /* end of function create_schedule */

/*****************************************************************************/

static void create_transfer ( void )
{
```

```
    char *attach_source   = "ATTACH 'FILENAME test-area:test_source'";
    char *create_transfer = "CREATE TRANSFER DYN_1\
                             TYPE IS EXTRACTION\
                             MOVE TABLES *\
                             TO NEW FILENAME test_area:test_target.RDB\
                             LOG FILE IS test_area:test.LOG";
    long sqlcode;

    printf ("\n*** Start of program\n");

    printf ("*** Attaching to source database\n");
    interpret ( &sqlcode, attach_source ); check

    printf ("*** Creating transfer definition:\n");
    printf ("***   CREATE TRANSFER DYN_1\n");
    printf ("***     TYPE IS EXTRACTION\n");
    printf ("***     MOVE TABLES *\n");
    printf ("***     TO NEW FILENAME test_area:test_target.RDB\n");
    printf ("***     LOG FILE IS test_area:test.LOG\n");
    interpret ( &sqlcode, create_transfer ); check

    printf ("*** Detaching from source database\n");
    detach_database ( &sqlcode ); check

    printf ("*** End of program\n\n");

} /* end of function create_transfer */

/*****************************************************************************/

static void drop_schedule ( void )
{
    char *drop_schedule = "DROP SCHEDULE FOR DYN_1";
    long sqlcode;

    printf ("\n*** Start of program\n");

    printf ("*** Dropping schedule\n");
    interpret ( &sqlcode, drop_schedule ); check

    printf ("*** End of program\n\n");

} /* end of function drop_schedule */

/*****************************************************************************/

static void drop_transfer ( void )
{
    char *drop_transfer = "DROP TRANSFER DYN_1";
    long sqlcode;

    printf ("\n*** Start of program\n");

    printf ("*** Dropping transfer\n");
    interpret ( &sqlcode, drop_transfer ); check

    printf ("*** End of program\n\n");

} /* end of function drop_transfer */

/*****************************************************************************/
```

```
static void start_transfer ( void )
{
    char *start_transfer_now_wait = "START TRANSFER DYN_1 NOW WAIT";
    long sqlcode;

    printf ("\n*** Start of program\n");

    printf ("*** Starting transfer NOW and waiting for completion\n");
    interpret ( &sqlcode, start_transfer_now_wait ); check

    printf ("*** End of program\n\n");

} /* end of function start_transfer */

/*****************************************************************************/

static void stop_transfer ( void )
{
    char *stop_transfer = "STOP TRANSFER DYN_1";
    long sqlcode;

    printf ("\n*** Start of program\n");

    printf ("*** Stopping (suspending) transfer\n");
    interpret ( &sqlcode, stop_transfer ); check

    printf ("*** End of program\n\n");

} /* end of function stop_transfer */
```

The following is an SQL module language module that is used with the TEST_DYNAMIC_TRANSFERS.C module in the sample application.

```
--+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
--
--      TEST_DYNAMIC_SQL.SQLMOD
--
--
--  ABSTRACT
--
--      This is a SQL module language module. It contains procedures (routines)
--      that can be called from host language modules. Each routine implements
--      a single SQL statement.
--
--      This module was written as part of a set of tests of applications that
--      can execute Replication Option SQL statements. The SQL module language
--      does not support the Data Distributor SQL statements explicitly.
--      Therefore, the Replication Option statements must be formed by the
--      host language modules and passed to these SQLMOD procedures as
--      complete text strings. These are passed through to SQL using SQL's
--      dynamic access capability.
--
--      The default parameter passing mechanism is set up here for the C lan-
--      guage. That means that text strings are expected to be terminated by a
--      null byte.
--
------------------------------------------------------------------------------
```

```
module dynamic_sql
language c
authorization rdb$dbhandle

procedure detach_database
    sqlcode;

    disconnect all;

--  This procedure passes character strings to dynamic SQL to be interpreted
--  at run time. The size of the 'statement' parameter determines the maximum
--  length of the statement. The maximum size allowed is 65535 characters.

procedure interpret
    sqlcode
    :statement char(65535);

    execute immediate :statement;

------------------------------------------------------------------------------
```

D.1.1 Compiling and Linking the Test Program

To compile and link the test program, enter the statements in the following list either interactively or as part of a command procedure:

- Set default to the appropriate directory:

  ```
  $ SET DEFAULT test-area
  ```

- Compile the C module used for this test:

  ```
  $ CC/STANDARD=PORTABLE/LIST/OBJECT DYNAMIC_TRANSFERS.C
  ```

- Compile the SQL module language module used for this test:

  ```
  $ SQLMOD/LIST/OBJECT/ANSI_PARAMETERS/SESSION DYNAMIC_SQL.SQLMOD
  ```

- Link the test image:

  ```
  $ DEFINE/USER LNK$LIBRARY   SYS$LIBRARY:SQL$USER.OLB
  $ DEFINE/USER LNK$LIBRARY_1 SYS$LIBRARY:VAXCRTL
  $ LINK/MAP DYNAMIC_TRANSFERS,DYNAMIC_SQL
  ```

D.1.2 Checking Transfer Execution Status

Note that Replication Option does not report the transfer execution status to the application, although it does report statement failure. To work around this problem, you can:

- Use an epilogue procedure to test and report the status of a transfer.
- Have your application look at the values stored in the RDB$VINTAGE target table (for Rdb targets) or DDAL$TRANSFER_INFO (for all other target databases).

Within the RDB$VINTAGE table, the RDB$VINTAGE_TIMESTAMP column indicates when the last execution of a transfer took place. The RDBVMS$TRANSFER_BUSY column contains a "T" or "F", indicating whether or not a transfer is executing. If the RDBVMS$VINTAGE_ TIMESTAMP column indicates that a transfer did not take place when scheduled, and the RDBVMS$TRANSFER_BUSY column value is "F", meaning a transfer is not executing, you know something is wrong with the execution of the transfer.

Within the DDAL$TRANSFER_INFO table, the DDAL$TRANSFER_ TIME column indicates when the last execution of a transfer took place. The DDAL$TRANSFER_BUSY column contains a "T" or "F", indicating whether or not a transfer is executing. If the DDAL$TRANSFER_TIME column indicates that a transfer did not take place when scheduled, and the DDAL$TRANSFER_BUSY column value is "F", meaning a transfer is not executing, you know something is wrong with the execution of the transfer.

When creating an extraction transfer to a new database, if the transfer fails, an attach to that database will also fail. A partial execution of the batch-update transaction started for the extraction transfer results in a corrupt database. (Failed batch-update transactions normally result in a corrupt database.)

For a replication transfer, use the RDB$VINTAGE_TIMESTAMP or DDAL$TRANSFER_TIME column to see if the replication occurred as scheduled. If the replication does not occur, the value in this column does not get updated.

If a transfer fails partway through, the RDBVMS$TRANSFER_BUSY or DDAL$TRANSFER_BUSY flag stays set to "T".

D.2 Using Rdb Transparent Gateway to RMS with Replication Option

The following example shows how you might populate a target database with a transfer from an RMS file using the gateway software. In this example, the target database is not created by Replication Option. When this is the case, use the TO EXISTING clause in the SQL CREATE TRANSFER statement. If the target system is Rdb and you intend to repeat the extraction transfer on a regular basis, this avoids having a target database created with every execution of the transfer.

```
$ DEFINE LABOR$TARGET DISK1:[ELLINGSWORTH.SQL]LABOR_REPORTS
$
$ SQL
SQL> ! Create an empty Rdb database with required attributes
SQL>
SQL> CREATE DATABASE FILENAME 'LABOR$TARGET'
cont> GLOBAL BUFFERS ARE ENABLED (NUMBER IS 110, USER LIMIT IS 80);
SQL> DISCONNECT DEFAULT;
SQL>
SQL> ! Attach to RMS files using the gateway software
SQL>
SQL> ATTACH 'FILENAME ''/TYPE=NSDS/PATHNAME=CDD$DISK:[WEEKLY_REPORTS]''';
SQL> CREATE TRANSFER NH_EMPLOYEES TYPE IS EXTRACTION
cont> MOVE TABLES
cont>   SELECT EMPLOYEE_ID,
cont>          LAST_NAME,
cont>          FIRST_NAME,
cont>          MIDDLE_INITIAL,
cont>          HOURS_WORKED,
cont>          COST_CENTER_CODE,
cont>          ACTIVITY_CODE,
cont>          WEEK_ENDING_DATE
cont> FROM EMPLOYEES WHERE STATE = 'NH'
cont> TO EXISTING FILENAME LABOR$TARGET
cont> LOG IS LABOR$LOGS:NH_REPORTS.LOG
cont> COMMENT IS 'Extraction of labor information';
SQL>
SQL> CREATE SCHEDULE FOR NH_EMPLOYEES
cont> START 1-NOV-1995 01:24:00.00
cont> EVERY DAY AT 17:30
cont> RETRY 3 TIMES
cont> RETRY EVERY 0 00:30:00;
```

Each time the transfer executes, Replication Option erases any data in the EMPLOYEES table in the target database prior to transferring a fresh set of data.

E

Replication Option Error Messages and Online Help

This appendix describes how to access the Replication Option error messages and online help.

E.1 Replication Option Error Messages

Replication Option error message information is available in the online help under the Errors topic. The same information is also contained in a file, DDAL$MSG.DOC, that you can view online or print. To view this information on OpenVMS, use the DCL TYPE command or print using the DCL PRINT command. The DDAL$MSG.DOC error message file is located in the SYS$HELP directory.

E.2 Online Help Files for OpenVMS

Type HELP REPLICATION_OPTION at the SQL prompt to access Replication Option help.

Glossary

This glossary contains the terminology commonly used with Replication Option. If you need access to terminology specific to your native database management system, refer to glossaries in the appropriate documentation.

alias

A name specified in an ATTACH, DECLARE ALIAS, or CREATE DATABASE statement that identifies that database in subsequent SQL statements.

archival transfer

A transfer method that can maintain a target database as an archive of data records that are current or that might since have been deleted from the production, source database. An archival data transfer is a variation of the replication transfer method.

cascaded transfer

A transfer method that uses intermediate databases as sources for transfers to multiple target databases. Also called a multistage transfer.

Cascaded transfers are useful when you have a single source database and many targets. In this situation, it may be impractical to directly transfer data from the single source to all the targets. With cascaded transfers, you first define transfers to copy the required data from the single source to a few intermediate targets. Next, you define transfers that use these intermediate targets as the source databases. When you use intermediate source databases, you reduce demand on the original source database.

copy process

An OpenVMS detached process created by a transfer monitor for the purpose of executing a data transfer. Within that detached process, a prologue command procedure (optional), a copy process image, and an epilogue command procedure (optional) are executed by commands issued from the transfer monitor.

Generally, the term copy process refers to the copy process image (program) rather than to the OpenVMS process in which that image runs. The copy process image retrieves information from the transfer database and source database in order to initiate and execute transfers. It creates a target database for extraction, extraction rollup, and initial replication transfers. It also transfers to a replication target database changes that have occurred to the source database following the initial replication transfer.

See also Replication Option, extraction, extraction rollup, replication, target database, transfer database, transfer definition, and transfer monitor.

copy process log file

An RMS disk file that is created each time a transfer executes, if you specify the LOG FILE IS clause in the transfer definition. The copy process log file records event and status information that can help you identify certain error conditions. In addition, the log file compiles a set of statistics that can provide valuable information about transfers.

DDAL

A prefix used with file names, environment variable names, table and column names, and message texts in Replication Option. DDAL identifies an element as part of Replication Option software. For example, DDAL$EPILOGUE.COM is an OpenVMS file name that identifies a Replication Option command procedure.

Distributed Option for Rdb

The Oracle database interconnect software product that provides access to relational and nonrelational databases. A Distributed Option database can be either or both the source and target of a data transfer. To be a target of a data transfer, the link database or databases that form the backbone of the Distributed Option logical database must be themselves valid targets of data transfers. For a list of target database systems, refer to Table 1–2. Distributed Option data can be transferred using either the extraction or extraction rollup method of transfer.

environment variable

A variable that can be defined to affect the operation of Replication Option. An environment variable can be defined as an OpenVMS logical name or, in most cases, as an entry in a Replication Option configuration file.

epilogue command procedure

A DCL command procedure that runs automatically after each execution of a transfer, if you include the EPILOGUE FILE IS clause in the transfer definition. You can use an epilogue file to provide indexes, constraints, access rights, and additional views for your target database.

See also prologue command procedure.

expression

Another term for value expression.

extraction

The method by which a set or subset of tables from a source database is transferred into a target database. Each time an extraction transfer is executed, a snapshot of the entire data set or subset is copied to the target database. The source and target databases can be located on the same system where Replication Option is installed or on remote systems in the network. A target database created by an extraction transfer is referred to as an extraction database.

See also Replication Option, extraction rollup, replication, and source node.

extraction rollup

The method by which a set or subset of tables from one or more source databases is transferred into a target database. Each time an extraction rollup transfer is executed, a snapshot of the entire data set or subset is copied to the target database. The source and target databases can be located on the same system as the system on which Replication Option is installed or on remote systems in the network. A target database created by an extraction rollup transfer is referred to as an extraction rollup database.

See also Replication Option, extraction, replication, and source node.

ID

An abbreviation used for the words identification and identifier.

materialize

The Replication Option process of transferring data contained in a view or in a COMPUTED_BY column in the source table to a real table or column in the target database.

For example, in the source database there could be a real table, VEHICLES, containing data rows about cars and trucks. You could define a view, called CARS, that would show only those rows containing cars. The CARS table (a view) would be considered immaterial; that is, it would not actually have any real stored data (the data is stored in the VEHICLES table).

However, Replication Option, in the process of transferring data from the source database, can optionally create a real table, CARS, in the target database, containing real data from the source table VEHICLES. The CARS table in the target database would then be a real table because it had data rows stored in it.

node

In Replication Option, the word node can either refer to a computer in a network (also called a host name) or it can be an element in a hierarchical path name, such as a UNIX file name.

OpenVMS

This virtual memory system is one of the major operating systems for Digital Equipment Corporation processors.

prologue command procedure

A DCL command procedure that runs automatically before each execution of a transfer, if you include the PROLOGUE FILE IS clause in the transfer definition. You might use a prologue file to establish a network link to a remote system or define logical names for the target database.

See also epilogue command procedure.

Rdb

A relational database management product that uses the relational model of database organization. Replication Option can use Rdb databases as sources and targets of data transfers.

See also target database.

Rdb Transparent Gateway to Custom Drivers

The Oracle database interconnect software product that provides access to nonrelational data files. Replication Option transfers data from custom data sources into target databases. For a list of target database systems, refer to Table 1–2. Custom data can be transferred using either the extraction or extraction rollup method of transfer.

See also Replication Option, extraction, and extraction rollup.

Rdb Transparent Gateway to DB2

The Oracle database interconnect software product that provides access to IBM DB2 is known as Rdb Transparent Gateway to DB2. A DB2 database can be either or both the source and target of a data transfer. DB2 data can be transferred using either the extraction or extraction rollup method of transfer.

See also Replication Option, extraction, and extraction rollup.

Rdb Transparent Gateway to Oracle

The Oracle database interconnect software product that provides access to data stored in the Oracle relational database management system and, through SQL*Net, to many other databases. Oracle data can be transferred using either the extraction or extraction rollup method of transfer.

When this gateway is used as a data source, Replication Option supports access to any of the databases available. When this gateway is used as a data target, Replication Option supports only Oracle databases.

See also Replication Option, extraction, and extraction rollup.

Rdb Transparent Gateway to Oracle CODASYL DBMS

The Oracle database interconnect software product that provides access to Oracle CODASYL DBMS databases. Replication Option transfers data from Oracle CODASYL DBMS sources into target databases. For a list of target database systems, refer to Table 1–2. Oracle CODASYL DBMS data can be transferred using either the extraction or extraction rollup method of transfer.

Rdb Transparent Gateway to PC Data

The Rdb Transparent Gateway to PC Data enables OpenVMS or Digital UNIX users to have access to data stored in data sources associated with PC Data servers. Programmers and end users can use one of many Oracle or third-party tools with the gateway to have access to the data. The gateway to PC Data client works with a gateway to PC Data server and associated data sources to provide access to data.

When this gateway is used as a data source, Replication Option supports access to any of the databases available. When this gateway is used as a data target, Replication Option supports only Microsoft Access or Microsoft SQL Server databases.

Rdb Transparent Gateway to RMS

The Oracle database interconnect software product that provides access to RMS data files. Replication Option transfers data from RMS data sources into target databases. For a list of target database systems, refer to Table 1–2. RMS data can be transferred using either the extraction or extraction rollup method of transfer.

See also Replication Option, extraction, and extraction rollup.

Rdb Transparent Gateway to Sybase

The Oracle database interconnect software product that provides access to Sybase databases, and to many other databases accessible to Sybase open servers. A Sybase database can be either or both the source and target of a data transfer. Sybase data can be transferred using either the extraction or extraction rollup method of transfer.

When this gateway is used as a data source, Replication Option supports access to any of the databases available. When this gateway is used as a data target, Replication Option supports only Sybase databases.

replication

The method by which a set or subset of tables from an Rdb source database is transferred into a target database. The first time a replication transfer is executed, a snapshot of the entire data set or subset is copied to the target database. On subsequent executions of the transfer, only inserts, updates and deletions of data rows are transferred to the target. The target database can be located on the same system on which Replication Option is installed or on a remote system in the network. A target database created by a replication transfer is referred to as a replication database.

See also Replication Option for Rdb, extraction, extraction rollup, source database, source node, and target database.

Replication Option for Rdb

An Oracle product that runs on OpenVMS that lets you make copies or partial copies of database tables and transfer them to other databases. Except for transfers using the replication method, source databases and target databases can be located anywhere within the computer network. You can use the SQL interface to enter and execute Replication Option statements.

See also extraction, extraction rollup, Rdb, replication, source database, source node, SQL, and target database.

rollup transfer

A transfer method that uses multiple source databases for transfers to a single target database.

schedule definition

A definition that specifies when a particular transfer is executed by the transfer monitor. You can also define the frequency of transfer if you need to transfer rows or updates to existing rows on, for example, a daily or weekly basis. You create schedule definitions with the SQL CREATE SCHEDULE statement. Defining schedules is an optional task. Replication Option stores schedule definitions in the transfer database.

See also Replication Option for Rdb, transfer database, transfer definition, and transfer monitor.

select expression

A phrase in SQL that specifies a result table to be retrieved from a database. A result table is a temporary table derived from a combination of the tables or views identified in the FROM clause of the select expression. The select expression that Replication Option allows in a transfer definition is more limited than the select expression permitted in SQL.

source database

A database that provides data definitions and data values that Replication Option transfers during extraction, extraction rollup, or replication transfers. An extraction rollup database can have several databases as its source, unlike an extraction or replication database which can have only one database source. Target databases can be located on the same node as the source database or on nodes different from the source database.

See also Replication Option, extraction, extraction rollup, replication, source database, and target database.

source node

The node in a network on which the source database resides.

See also Replication Option, extraction, extraction rollup, replication, and source database.

SQL

A data definition and data manipulation language that serves as the primary interface used by Replication Option to create, start, stop, and display transfers. Using SQL, you can issue schedule and transfer definitions and initiate extraction, extraction rollup, and replication transfers.

system tables

Special-purpose tables automatically created by Replication Option in the source database when a copy process starts a replication transfer for the first time. Updates to rows involved in replication transfers are stored in Replication Option system tables.

target database

A database into which Replication Option copies data. A target database can be created at the beginning of a transfer or it can be a database that already exists. The target database can be located on any system in the network. The target database is referred to as either an extraction database, extraction rollup database, or replication database, depending on the method of transfer used.

See also Replication Option, extraction, extraction rollup, replication, and source database.

target node

The receiving node for data transferred from the source database on the source node. The target node can be the same node as the source node or can be a remote node.

See also Replication Option, extraction, extraction rollup, replication, source node, and target database.

transfer database

An Rdb or Oracle database that stores information necessary to carry out data distribution operations in the network. On a standalone node or in a cluster, there is only one transfer database.

Among items stored in the transfer database are:

- Transfer definitions, including the name of the definer, the type of transfer, the names of the source and target databases, the selection criteria in the form of select expressions
- Schedule definitions, including the next computed time to transfer and the number of times that Replication Option should retry a transfer that fails

- Current status of all transfers, including the time of the last successful execution and the current state of the transfer, such as whether or not it is executing, successful, and so forth

Appendix A lists the names and descriptions of the tables and columns defined in the transfer database.

transfer definition

A definition of what data and data definitions are to be copied from a source database or databases into a target database. The transfer definition identifies the source and target databases, the tables and views to be copied, and the data rows to be copied. You can create transfer definitions with the SQL CREATE TRANSFER statement. Defining transfer definitions is a required task in Replication Option. The transfer monitor stores transfer definitions in the transfer database.

See also Replication Option, extraction, extraction rollup, select expression, replication, source database, target database, and transfer monitor.

transfer monitor

The OpenVMS process that oversees all of the functions provided by Replication Option. Replication Option provides statements to create, drop, and show transfer and schedule definitions and to start and stop transfer execution. These statements are entered using the interactive language SQL or under application program control using the SQL dynamic interface. These statements are sent to the Replication Option transfer monitor, one per standalone system or one per computer processor in a cluster. The transfer monitor stores transfer and schedule definitions and schedule execution status in a transfer database. The transfer monitor regulates the execution of data transfers from source to target databases. The transfers are actually performed by copy processes at the request of the transfer monitor.

See also Replication Option, extraction, extraction rollup, replication, schedule definition, and transfer definition.

transfer monitor log file

An RMS disk file that is created by the transfer monitor to record Replication Option events and status information.

transfer state

The condition of a transfer that is directly affected either by Replication Option statements you enter or by actions automatically undertaken by Replication Option. For example, the START TRANSFER NOW statement places a transfer in the active state; Replication Option can automatically retry a transfer if it has been scheduled for a retry. The transfer states include: unscheduled, scheduled, active, suspended, waiting to retry, retrying, prologue failure retry, prologue failure, epilogue failure retry, and epilogue failure. Refer to Appendix C for a list of Replication Option statements and the transfer states each statement invokes.

See also transfer database and transfer monitor.

value expression

Another term for scalar expression. A symbol or string of symbols used to calculate a value. If you use a value expression in a statement, SQL calculates the value associated with the expression and uses that value when executing the statement. Types of value expressions include literals, arithmetic expressions, built-in functions, external functions, and column select expressions.

view

A logical table whose data is not physically stored. Views refer to rows stored in other tables. You define a view to do the following:

- Access a subset of the columns stored in a row.
- Access a set of columns stored in different rows.
- Avoid creating a redundant copy of data that is already stored.

VMScluster

A VMScluster configuration is a highly integrated organization of OpenVMS Alpha systems, VAX systems, or a combination of Alpha and VAX systems, that communicate over a high-speed communications path. VMScluster configurations have all the functions of single-node systems, plus the ability to share CPU resources, queues, and disk storage. Like a single-node system, the VMScluster configuration provides a single security and management environment. Member nodes can share the same operating environment and serve specialized needs.

Index

A

Absolute time, 9–5, 9–21
  DAY, 9–5, 9–21
  MONTH, 9–5, 9–21
  WEEK, 9–5, 9–21
  weekday, 9–5, 9–21
Access control lists (ACLs), 7–18, 7–21, 9–62, 9–89
  adding entries to, 9–76
  ALTER privilege, 9–78
  changing, 9–76, 9–89
  creating entries, 9–76
  database, 9–76
  DBCTRL privilege, 9–78
  deleting entries from, 9–89
  DROP privilege, 9–78
  for target databases, 4–35, 6–43
  identifiers, 9–76, 9–89
  schedule, 9–89
  transfers, 9–89
  UIC identifiers, 9–79, 9–91
Access control strings, 9–45
Access privilege information
  in target databases, 4–35
Access privilege sets
  access control list (ACL) style, 9–76
Access rights, 9–62, C–14
  for transfer database, C–15
ACLs
  See Access control lists (ACLs)
Active transfer state, 6–5
Adding
  a checkpoint attribute to an existing transfer definition, 9–10
  a log file to an existing transfer definition, 9–10
  an epilogue file to an existing transfer definition, 9–10
  a prologue file to an existing transfer definition, 9–10
  a transfer description to an existing transfer definition, 9–10
  entries
    to ACLs, 9–76
  indexes, B–9
AFTER clause
  of GRANT statement, 9–79
  of REVOKE statement, 9–91
Aliases
  in rollup move clause, 5–4
  with extraction databases, 5–8
  with extraction rollup databases, 5–8
  with replication databases, 5–17
Altering transfers, 6–11
ALTER privilege, 9–78
ALTER SCHEDULE statement, 6–16, 9–3
  restrictions for, 9–7
  time formats in, 9–4
  UIC required with, 9–7
ALTER TRANSFER statement, 6–11, 9–10
  transfer file options clause in, 9–12
Applications
  analyzing requirements, 6–47
  classifying, 2–18
  database maintenance operations, 2–18
  data origination tasks, 2–18

Applications (cont'd)
decision support tasks, 2–19
evaluating, 2–18
evaluating transaction modes, 6–51
moving to target sites, 4–5
prototyping tasks, 2–19
query operations, 2–19
scheduling for extractions, 6–52
selecting a transfer type for, 2–19
Archiving data, 8–11
Authorize utility (OpenVMS), 4–17
modifying NETPROXY.DAT file, 4–16

B

Backup
using RMU Backup command, 4–24
using SQL EXPORT statement, 4–24
using SQL IMPORT statement, 4–24
with RMU Restore command, 4–24
Buffers
allocation of, 9–49

C

Cascaded transfers, 2–3
CDD/Plus
See Data dictionary
CDD/Repository
See Data dictionary
Central data
advantages of, 1–7
Central databases, 1–9, 6–47
disadvantages of, 1–7
problems with, 1–8
remote access to, 1–7
Changing
a checkpoint attribute for an existing transfer definition, 9–10
ACL entries, 9–76, 9–89
a log file for an existing transfer definition, 9–10
an epilogue file for an existing transfer definition, 9–10
Changing (cont'd)
a prologue file for an existing transfer definition, 9–10
a target file specification for an existing transfer definition, 9–10
a transfer description for an existing transfer definition, 9–10
Character set
database
default, 9–51
for database, 9–51
Character sets
and versions of Replication Option, 5–23
considerations with transfers, 5–23
Digital Equipment Corporation MCS, 5–23
MIA, 5–23
mismatches, 5–23
mixing, 5–23
SHOW TRANSFER statement, 9–101
supported by SQL, 5–23
terminal device restrictions, 5–23
Checking status of transfers, B–13
Checkpoint attribute
adding to an existing transfer definition, 9–10
changing for an existing transfer definition, 9–10
removing from a transfer definition, 9–10
Checkpoints, 6–37
Columns
adding to existing tables in source databases, 4–29
altering in source databases, 4–30
dropping in source databases, 4–30
requirements for unique values, 5–31
Command procedures
bringing up network with, 7–21
creating transfer databases, C–2
DDAL$CREATE_TR_DB.COM, C–2
DDAL$CREATE_TR_DB_ORA.SQL, C–3
DDAL$START_TR_MON.COM, C–3
DDAL$STOP_TR_MON.COM, C–8
default directory for, 7–1
defaults for, 7–1
defining, 7–1
defining constraints with, 7–17

Command procedures (cont'd)
defining indexes with, 7–17
defining multischema attributes with, 7–17
defining protection with, 7–17
determining status with, 7–11, 7–16
epilogue procedures, 5–8
EPILOGUE statement, 7–1
executing, 7–1
executing at target site, 7–8
execution sequence of, 7–2
file specifications for, 7–1
in copy process log file, 7–9, 7–16
logical symbol names for, 9–62, 9–63, 9–98
LOGOUT statements with, 7–2
multiple, 7–6
on local nodes, 7–3
on remote nodes, 7–3
parameters in, 7–7
prologue procedures, 5–8
PROLOGUE statement, 7–1
shell procedures, 7–6
uses of, 7–17
with CREATE TRANSFER statement, 7–1
with DDAL$COPY_PROCESS image, 7–3
with DDAL$CP_CONTINUE process logical name, 7–3
with extraction rollup transfers, 7–2
with extraction transfers, 7–2
with replication transfers, 7–2
Command procedure status
showing with copy process log file, 7–15
showing with DDAL$EPILOGUE.COM, 7–16
Committing data in one transaction, B–17, B–25
Components, 3–1
copy process, 3–1, 3–10
source database, 3–1, 3–6
system tables, 3–11
target database, 3–1, 3–6
transfer database, 3–1, 3–8
transfer definition, 3–9
transfer monitor, 3–1, 3–7, 3–8
transfer schedule, 3–9
COMPUTED BY columns
and aggregate expressions, 5–11
transferring views, 5–11
COMPUTED BY columns (cont'd)
transferring with views, 9–40
COMPUTED BY columns
limitations transferring views, 5–20
Concurrent transfers
limiting, 6–55
Constraints
adding to target tables, 6–42
defining on extraction databases, 9–35
defining on extraction rollup databases, 9–41
defining on replication databases, 9–34
defining with command procedures, 7–17, 8–7
dropping with prologue procedures, 8–10
in transfers to existing databases, 8–7
requirements for columns, 5–31
requirements for indexes, 5–31
Conversion
using RMU Convert command, 4–24, 4–26
Copy process, 3–1, 3–10
communication with transfer monitor, 3–10
execution of, 3–10
pushed and pulled transfers, 4–5, 5–2
reading LOGIN.COM file, 3–10
reading RDB$CHANGES, 6–60
starting of, 3–10
Copy process log files, 5–17, 6–22, 6–23, 9–14, 9–52
command procedure information in, 7–9, 7–16
default directory for, 5–17
logical symbol names for, 9–62, 9–63, 9–97
showing command procedure status with, 7–15
CREATE SCHEDULE statement, 6–1, 6–2, 9–19
changing parameters in, 9–23
restrictions for, 9–23
RETRY clause in, 9–7, 9–22
time formats in, 9–19
UIC required with, 9–23
with suspended transfers, 9–23
with unscheduled transfers, 9–23
CREATE TRANSFER statement, 5–3, 5–13, 5–27, 9–28
command procedures with, 7–1
creating select expressions in, 5–10
default character set, 9–51

CREATE TRANSFER statement (cont'd)
defaults in, 9–45
file specification for, 9–45
identifier character set, 9–51
identifying rows with, 5–8
logical symbol names in, 9–62
move tables clause in, 9–51
move views clause in, 9–40
national character set, 9–51
rollup move clause in, 9–42, 9–59
starting a transaction with, 5–21
SYS$LOGIN with, 9–45
transfer file options clause in, 9–51
transferring to a new database, 9–45
transferring to an existing database, 9–45
transferring views with, 5–10
Currency
of data values, 1–7

D

Data
centralized, 1–7
replicated, 1–7
Database access
using the RMU Open command, 4–22
Database access conflicts, 8–11
Database backup
using RMU Backup command, 4–24
with RMU Restore command, 4–24
Database conversion
using command procedure, 4–29
using RMU Convert command, 4–24, 4–26
Database files
relocating source, 4–30
relocating target, 4–35
restructuring the target, 4–36
Database migration
using SQL IMPORT and SQL EXPORT statements, 4–24
Database pages
allocation, 9–49
block size, 9–49
Database recovery
using RMU Recover command, 4–24
Database restore
using the RMU Restore command, 4–24
Databases
access conflicts, 8–11
advantages of local access, 1–9
central, 6–47
character set
default, 9–51
consistency with the data dictionary, 6–48
converting, 4–26
default character set, 9–51
definitions in, 6–48
different Rdb versions, 5–23
distributed, 2–9
Distributed Option, 1–6
distribution, 2–2
evaluating tasks, 2–18
extraction rollups, 2–9
extractions, 2–9
identifier character set, 9–51
intermediate, 2–3
local, 2–2
maintaining a consistent state, 8–32
maintaining local directories in, 5–5
maintaining storage space in, 5–5
multifile, 3–7
multischema, 5–24
national character set, 9–51
Rdb, 1–6
Rdb Transparent Gateway to Custom Drivers, 1–5
Rdb Transparent Gateway to DB2, 1–5
Rdb Transparent Gateway to Oracle, 1–5
Rdb Transparent Gateway to Oracle CODASYL DBMS, 1–5
Rdb Transparent Gateway to PC Data, 1–5
Rdb Transparent Gateway to RMS, 1–6
Rdb Transparent Gateway to Sybase, 1–6
remote, 2–2
replications, 2–9
size of, 5–16
source, 2–1, 3–6, 3–7
target, 2–1, 2–9, 3–6, 3–7, 4–5

Databases (cont'd)
transfer, 3–1
transferring to existing, 8–1
transfers between
data exchange, 2–4
many sources to one target, 2–3
one source to many targets, 2–3
types of source databases, 2–21
types of target databases, 2–21
Databases access
contention on, 1–8
Data currency, 1–7
Data definitions
at target sites, 4–5
changing, 6–17
checking for changes, 8–8
database access conflict due to changes, 8–11
integrating, 7–22
transfer restrictions, 6–40
transferring, 5–22, 6–39
Data dictionary
changing path names, 4–6
consistency with database, 6–48
integrating data definitions into, 7–22, 7–23, 7–25
storing data definitions in, 4–1
using INTEGRATE statement (SQL) with, 4–6
using with extractions, 4–5
Data subsets, 2–1
Data types
changing
effects on extraction transfers, 8–34
date-time, 9–55
INTERVAL, 9–65
Date formats
default, 9–7, 9–23
DB2 databases, 1–5
DBCTRL privilege, 9–78
DBI
See Distributed Option
DBMS databases, 1–5
DDAL, xxviii
DDAL$CHANGES_BUFFER_SIZE environment variable, B–8
DDAL$CHANGES_BUFFER_SIZE logical name, B–8
DDAL$CHANGES_INDEX environment variable, 6–62, B–9
DDAL$CHANGES_INDEX logical name, B–9
DDAL$CHANGES_INDEX_transfername environment variable, 6–62, B–9
DDAL$CHECKPOINT_DELAY_IN_MINUTES environment variable, B–11
DDAL$CHECKPOINT_DELAY_IN_MINUTES logical name, B–11
DDAL$CHECKPOINT_MAX_RETRIES environment variable, B–11
DDAL$CHECKPOINT_MAX_RETRIES logical name, B–11
DDAL$CHECK_TARGET_DEFINITIONS environment variable, B–10
DDAL$CHECK_TARGET_DEFINITIONS logical name, B–10
DDAL$COPY_PROCESS image
with command procedures, 7–3
DDAL$CP_ACTION symbol name, 7–9
DDAL$CP_CONTINUE environment variable, B–13
DDAL$CP_CONTINUE logical name, B–13
DDAL$CP_CONTINUE process logical name, 7–3, 7–11, 7–12, 7–14, 7–15
determining status of transfer, 7–3
DDAL$CP_LOG_OPTIONS environment variable, 6–28, B–15
DDAL$CP_LOG_OPTIONS logical name, B–15
DDAL$CP_READ_CHANGES environment variable, B–16
DDAL$CP_READ_CHANGES logical name, B–16
DDAL$CP_READ_CHANGES_transfername environment variable, B–16
DDAL$CP_RETRY_ITERATION symbol name, 7–9
DDAL$CREATE_TR_DB.COM, C–1
DDAL$CREATE_TR_DB_ORA.SQL, C–3

DDAL$EPILOGUE.COM epilogue command procedure, 7–4
  determining transfer status with, 7–4
  showing command procedure status with, 7–16
DDAL$EXTRACT_DATA environment variable, B–17
DDAL$EXTRACT_DATA logical name, B–17
DDAL$INDEXES table, A–18
DDAL$LOG_TRMON_CLIENT_EVENTS environment variable, B–18
DDAL$LOG_TRMON_CLIENT_EVENTS logical name, B–18
DDAL$MAX_COPY_PROCESS environment variable, B–19
DDAL$MAX_COPY_PROCESS logical name, 6–56, B–19
DDAL$PURGE environment variable, B–23
DDAL$PURGE logical name, B–23
DDAL$RW_ROW_COUNT environment variable, B–24
DDAL$RW_ROW_COUNT logical name, B–24
DDAL$SHARED_WRITE environment variable, B–25
DDAL$SHARED_WRITE logical name, B–25
DDAL$SINGLE_DATA_COMMIT environment variable, B–25
DDAL$SINGLE_DATA_COMMIT logical name, B–25
DDAL$START_TR_MON.COM command procedure, 4–29, C–3
  logical names defined in, C–3
DDAL$START_WITH_SOURCE_TABLE environment variable, B–26
DDAL$START_WITH_SOURCE_TABLE logical name, B–26
DDAL$STOP_TR_MON.COM command procedure, C–8
DDAL$STORE_DEF_SEPARATE environment variable, B–27
DDAL$STORE_DEF_SEPARATE logical name, B–27
DDAL$TARGET_INDEX_OPTIONS environment variable, B–28
DDAL$TARGET_INDEX_OPTIONS logical name, B–28
DDAL$TM_LOG_FILE logical name, 6–29
DDAL$TRANSFER_DATABASE system logical, 3–9
DDAL$TRANSFER_NAME process logical name, 7–8
DDAL.RC file, B–29
DDAL_COMMENT table, A–9
DDAL_DEST_DPB table, A–10
DDAL_INVOKE_TIME table, A–3
DDAL_MOVE_CLAUSES table, A–10
DDAL_SCHED_OFFSETS table, A–11
DDAL_SOURCE_DBS table, A–5
DDAL_TNSF_COLUMNS table, A–6
DDAL_TNSF_PRIVS table, A–11
DDAL_TNSF_SCHEDULE table, A–7, A–8
DDAL_TNSF_SEL_EXPR table, A–11, A–12
DDAL_TNSF_TABLES table, A–6, A–7
DDAL_TNSF_VIEWS table, A–9
DDAL_TRANSFERS table, A–3
DDAL_TRANSFER_DB table, A–6
DDAL_TRANSFER_STATUS table, A–8, A–9
Deadlocks, 5–15
Default character set
  of CREATE TRANSFER statement, 9–51
  of database, 9–51
Defining
  privileges, 9–76
Deleting
  ACL entries, 9–89
  privileges, 9–89
  schedule access, 9–89
  transfer access, 9–89
Delta time, 9–4, 9–20
Directories
  target, 3–6
Distributed Option, 1–6
  databases, 1–6
  options
    effect on Replication Option, 3–5

Domain definitions
changing, 8–9
Domains
adding to target database, 4–34
altering in target database, 4–34
dropping from target databases, 4–34
DROP privilege, 9–78
DROP SCHEDULE statement, 6–16, 9–72
DROP TRANSFER statement, 6–14, 9–74

E

Enabling snapshot file, 9–50
Environment variables
DDAL$CHANGES_BUFFER_SIZE, B–8
DDAL$CHANGES_INDEX, B–9
DDAL$CHECKPOINT_DELAY_IN_ MINUTES, B–11
DDAL$CHECKPOINT_MAX_RETRIES, B–11
DDAL$CHECK_TARGET_DEFINITIONS, B–10
DDAL$CP_CONTINUE, B–13
DDAL$CP_LOG_OPTIONS, B–15
DDAL$CP_READ_CHANGES, B–16
DDAL$EXTRACT_DATA, B–17
DDAL$LOG_TRMON_CLIENT_EVENTS, B–18
DDAL$MAX_COPY_PROCESS, B–19
DDAL$PURGE, B–23
DDAL$RW_ROW_COUNT, B–24
DDAL$SHARED_WRITE, B–25
DDAL$SINGLE_DATA_COMMIT, B–25
DDAL$START_WITH_SOURCE_TABLE, B–26
DDAL$STORE_DEF_SEPARATE, B–27
DDAL$TARGET_INDEX_OPTIONS, B–28
for indexing RDB$CHANGES, 6–62
DDAL$CHANGES_INDEX_transfername, 6–62
Epilogue command procedures, 7–4
DDAL$EPILOGUE.COM, 7–4
enabling privileges, 7–19
executing remote procedures, 7–22
integrating data definitions, 7–22
in transfers to existing databases, 8–7
Epilogue files
adding to an existing transfer definition, 9–10
changing for an existing transfer definition, 9–10
removing from a transfer definition, 9–10
specifying, 9–13, 9–52
suspending with DDAL$CP_CONTINUE, 7–15
EPILOGUE statement
running command procedures with, 7–1
EPI_FAIL transfer status, 7–4
EPI_RETRY transfer status, 7–4
Equivalence name
in transfer status, 7–4
Error messages, E–1
RDB-E-DEADLOCK, 5–15
RDB-E-NO_PRIV, C–14
RDMS-F-LCKCNFLCT, 4–22
SQL-F-TRANOTOWNER, 8–29
SYSTEM-F-FLTOVFLOW, 8–34
Errors
fatal, 9–5, 9–20
recovering from fatal, 9–6, 9–22
Errors messages
RDMS-F-DEADLOCK, 5–15
Event logging, 6–37
EVERY clause
in transfer schedule, 6–3
specifying time, 6–3
Extension, snapshot file, 9–50
Extraction rollup transfers, 2–9, 8–15
creating, 5–3, 5–11
database aliases in, 5–4
location of definition for, 3–9
operation, C–18
starting, 5–11
transferring data definitions, 6–40
transferring views, 5–11
Extraction transfers, 2–9, 2–10, 2–13
after changing data types, 8–34
as source databases, 2–13
creating, 5–3, 5–11
creating select expressions in, 5–10
defining, 9–34, 9–41
defining transfers on, 5–7

Extraction transfers (cont'd)
identifying rows for, 5–10
local control of, 2–10
local maintenance operations for, 5–5
location of definition for, 3–9
modifying, 2–10
operation, C–18
pulled, 4–5, 5–2
pushed, 4–5, 5–2
remote nodes with, 2–12
second generation, 3–6
starting, 5–11
to existing databases, 8–7
transferring data definitions, 6–40
using as source databases, 3–6
using the data dictionary with, 4–5

F

F$MODE lexical function
in LOGIN.COM, 6–9
Failover in VMScluster environment, 6–36
FAIL transfer status, 7–4
Failures
of transfers, 6–8
RETRY clause, 6–8
File specifications
duplicates, 9–45

G

GRANT statement, 9–76
AFTER clause, 9–79
ALTER access, 9–76
ALTER privilege, 9–78
creating new ACL entries, 9–81
DBCTRL access, 9–76
DBCTRL privilege, 9–78
DROP access, 9–76
DROP privilege, 9–78
modifying existing ACL entries, 9–80
POSITION clause, 9–79
SELECT access, 9–76
TO clause, 9–78

H

Help files for OpenVMS, E–1
HELP statement, 9–85
Historical data
storing in target database, 8–11

I

Identifier character set
of CREATE TRANSFER statement, 9–51
of database, 9–51
Identifiers in ACLs, 9–76, 9–89
in GRANT statement, 9–78
UIC, 9–79, 9–91
Index definitions
on extraction databases, 9–35
on extraction rollup databases, 9–41
on replication databases, 9–34
Indexes
adding to RDB$CHANGES table, 6–60, 6–61, B–9
adding to target tables, 5–26, 6–41
defining with command procedures, 7–17, 8–7
dropping with prologue procedures, 8–10
in target databases, 4–35
in transfers to existing databases, 8–7
requirements for unique values, 5–31
INTEGRATE statement (SQL)
database definitions, 6–48
data dictionary, 4–6
integrating data definitions, 7–25, 7–27

L

Local access
benefits of, 1–9
cost of, 1–9
Local databases, 2–2
Lock types, 6–51, 6–52
Log file
adding to an existing transfer definition, 9–10
changing for an existing transfer definition, 9–10
copy process, 9–14, 9–52

Log file (cont'd)
removing from a transfer definition, 9–10
Log files
copy process, 6–23, 6–27
event, 6–37
extractions, 5–8
for extraction transfers, 6–22, 6–29
in VMScluster environment, 6–36
replications, 5–17
transfer monitor, 6–29
Logical name
use with Rdb Transparent Gateway products, 9–63, 9–97
using in transfer definitions, 9–62
Logical names
See also Environment variables
DDAL$CHANGES_BUFFER_SIZE, B–8
DDAL$CHANGES_INDEX, B–9
DDAL$CHECKPOINT_DELAY_IN_MINUTES, B–11
DDAL$CHECKPOINT_MAX_RETRIES, B–11
DDAL$CHECK_TARGET_DEFINITIONS, B–10
DDAL$CP_CONTINUE, B–13
DDAL$CP_LOG_OPTIONS, B–15
DDAL$CP_READ_CHANGES, B–16
DDAL$EXTRACT_DATA, B–17
DDAL$LOG_TRMON_CLIENT_EVENTS, B–18
DDAL$MAX_COPY_PROCESS, B–19
DDAL$PURGE, B–23
DDAL$RW_ROW_COUNT, B–24
DDAL$SHARED_WRITE, B–25
DDAL$SINGLE_DATA_COMMIT, B–25
DDAL$START_WITH_SOURCE_TABLE, B–26
DDAL$STORE_DEF_SEPARATE, B–27
DDAL$TARGET_INDEX_OPTIONS, B–28
defined in DDAL$START_TR_MON.COM, C–3
for indexing RDB$CHANGES DDAL$CHANGES_INDEX, 6–63
using to limit concurrent transfers, 6–56
LOGIN.COM file, 6–9
used by copy process, 3–10
using F$MODE lexical function, 6–9
LOGOUT command
with command procedures, 7–2

M

Mail utility
indicating transfer status with, 7–13
MAXDETACH process quota, 6–56
Migration, 4–24
Monitor
See Transfer monitor
Move tables clause
aliases with, 5–8, 5–17
creating a transfer with, 9–51
Move views clause
creating a transfer with, 9–40
view name in, 9–41
Multifile databases
using as source databases, 3–7
Multischema attributes
adding to existing database, 8–33
adding to target database, 6–39, 6–43
defining with command procedures, 7–17
Multischema databases
using as source databases, 3–7
Multiversion Rdb environment
using with Replication Option, 4–1, 4–3

N

National character set
of CREATE TRANSFER statement, 9–51
of database, 9–51
NETPROXY.DAT file
establishing proxy accounts, 4–16
with Authorize utility, 4–16
Networks
bringing up with command procedures, 7–21
transfer failure, 4–23
NO DELETE attribute
of replication transfers, 8–11

Nodes
failure of source, 9–58
single transfer database serving, 9–32
target, 3–6
NO WAIT option
with START TRANSFER statement, 6–7
NOW option
with START TRANSFER statement, 6–5, 6–7
N-way transfers, 8–5

O

Operational procedures, C–1
Oracle SQL/Services, 3–4
using with Replication Option, 3–4

P

PERSONNEL database, examples, 1–8, 2–11
POSITION clause
of GRANT statement, 9–79
of REVOKE statement, 9–91
Predicate
supported by Replication Option, 9–39
Privileges
See also Protection
account, 4–1, 4–7
adding, 9–76
ALTER, 9–78
copy process, 4–16
database, 9–76
DBCTRL, 9–78
defining, 9–76
deleting, 9–89
DROP, 9–78
for accessing target tables, 7–19
for replications, 4–8
GRANT statement, 9–76
REVOKE statement, 9–89
schedule, 9–89
source account, 4–8
source database, 4–7
table, 9–76
target account, 4–9
to execute Replication Option statements, 9–2
Privileges (cont'd)
transfers, 9–89
Prologue command procedures
in transfers to existing databases, 8–8
Prologue file
adding to an existing transfer definition, 9–10
changing for an existing transfer definition, 9–10
removing from a transfer definition, 9–10
specifying, 9–13, 9–51
specifying transfer file options clause, 9–13, 9–51
Prologues
suspending with DDAL$CP_CONTINUE, 7–12
PROLOGUE statement
running command procedures with, 7–1
Protection
See also Privileges
defining on extraction databases, 9–35
defining on extraction rollup databases, 9–41
defining on replication databases, 9–34
defining with command procedures, 7–17, 7–19
Proxy accounts, 9–45
for multiple transfers to existing target database, 7–19
NETPROXY.DAT file, 4–16
setting up, 4–16
using Authorize utility, 4–17
PRO_FAIL transfer status, 7–4
PRO_RETRY transfer status, 7–4
Pull, 4–5, 5–2
Push, 4–5, 5–2

Q

Query outlines
in target databases, 4–35
Quotas
account, 4–1, 4–7
source account, 4–8
target account, 4–9

R

Rdb
  databases, 1–6
  options
    effect on Replication Option, 3–5
RDB$CHANGES
  growth of, 5–14
RDB$CHANGES system table, 3–13, 6–45, A–13, A–14
  indexing for replication transfers, 6–60, 6–61
    benefits, 6–61
    default, 6–62
RDB$CHANGES_MAX_TSER system table, A–14
RDB$TRANSFERS system table, A–12, A–13, C–11
RDB$TRANSFER_RELATIONS system table, A–13, C–11
RDB$VINTAGE system table, 6–19, 6–46, 6–50, A–15, A–16, A–17, C–12
  accessing, 7–19
RDB$VINTAGE_TIMESTAMP column, 6–19, 6–50
RDB$_IMP_EXC error message, 5–20
Rdb Transparent Gateway products
  access specifications for, 9–97
  attach strings for, 9–63
Rdb Transparent Gateway to Custom Drivers, 1–5
Rdb Transparent Gateway to DB2, 1–5
Rdb Transparent Gateway to Oracle, 1–5
Rdb Transparent Gateway to Oracle CODASYL DBMS, 1–5
Rdb Transparent Gateway to PC Data, 1–5
Rdb Transparent Gateway to RMS, 1–6
Rdb Transparent Gateway to Sybase, 1–6
RDBVMS$TRANSFER_OWNER system table, A–18
  accessing, 7–19
Reconfiguring transfer monitors, 3–8
Recovering from transfer failures
  and using checkpoints, 6–37
Recovery
  using RMU Recover command, 4–24
Reference data
  merging with existing data, 8–2
REINITIALIZE TRANSFER statement, 9–87
Remote access
  access control string, 9–45
  cost of, 1–8, 1–9
  proxy accounts, 9–45
  to central database, 1–8
Remote command procedures, 7–24, 7–27
  executed by SUBMIT/REMOTE command, 7–23
  executed by TYPE command, 7–25
  for integrating data definitions, 7–22
Remote databases, 2–2
Remote nodes, 2–12
  with extraction transfers, 2–12
Removing
  a checkpoint attribute from an existing transfer definition, 9–10
  a log file from an existing transfer definition, 9–10
  an epilogue file from an existing transfer definition, 9–10
  a prologue file from an existing transfer definition, 9–10
Replication extraction move clause
  MOVE TABLES * option, 9–36
Replication Option kit location, 2–22
Replication rollup transfers, 8–2
Replication transfers, 2–9, 2–14, 2–17
  changes transferred, 8–5, 9–33
  creating, 5–13
  defining, 9–33
  defining indexes for, 6–42
  defining transfers, 5–13
  features of, 2–17
  identifying rows in, 5–17
  indexing RDB$CHANGES, 6–60, 6–62, 6–63
    during initialization, 6–63
    during reinitialization, 6–63
    during updates, 6–64
  infrequently used, 5–14
  initialization mode, 2–15

Replication transfers (cont'd)
inserting, altering, and dropping rows, 5–19
location of definition for, 3–9
log files for, 5–17
NO DELETE attribute, 8–11
operation, C–24
performance option, 6–60
reinitialization mode, 2–16
select expressions in, 5–19
snapshot file for, 5–16
special-purpose tables in, 9–33
system tables in, 6–45
to existing databases, 8–11
transferring data definitions, 6–41
transferring views with, 5–20
update mode, 2–16
update operation, C–34
updates for, 9–33
using REINITIALIZE statement with, 9–87
using the data dictionary with, 4–5
REQUEST command
indicating transfer status with, 7–13
Requirements
on local and remote sites, 4–4
Resetting transaction numbers, 6–27
Resource configuration file, B–29
Restrictions
on transfer definitions, 6–40
on transfers to existing databases, 8–32
overriding with command procedures, 7–17
RETRY clause
after transfer failure, 6–8
in transfer schedule, 6–3
Retry count, 9–6, 9–22
RETRY EVERY clause
in transfer schedule, 6–3
RETRY transfer status, 7–4
REVOKE statement, 9–89
See also GRANT statement
AFTER clause, 9–91
POSITION clause, 9–91
schedule access, 9–89
transfer access, 9–89
RMS files
populating a database with, D–9
transferring, 8–29
RMU Backup command, 4–24
RMU Convert command, 4–24, 4–26
RMU Open command, 4–22
RMU Recover command, 4–24
RMU Restore command, 4–24
Rollup
See Extraction rollup transfers
See Replication rollup transfers
Rollup move clause, 9–59
defining a transfer with, 9–42
specifying database aliases in, 5–4
RTO option, 3–5
restrictions, 3–5
Run-time-only option
See RTO option

S

Schedule definitions
using access control lists with, 7–21
Scheduled transfer state, 6–5
Schedules
altering, 9–3
ALTER privilege, 9–78
changing, 6–16
creating, 6–1, 9–19
default access, 9–76
defining transfer start time, 9–4, 9–19
deleting, 6–16, 9–72
denying access, 9–89
DROP privilege, 9–78
granting access, 9–76
privileges, 9–89
RETRY clause, 9–7, 9–22
retry count, 9–6, 9–22
Security
access to remote nodes, 4–18
Select expressions
in extraction rollups, 5–10
in extractions, 5–10
in replications, 5–19

SET LOGICAL_NAME TRANSLATION FOR TRANSFER statement, 9–96
Share modes, 6–51, 6–52
SHOW TRANSFER statement, 6–18, 9–99
  DEFINITION option, 9–99
  displaying transfer state, 9–99
  PROTECTION option, 9–100
  SCHEDULE option, 9–99
  STATUS option, 9–99
  wildcard option, 9–100
Snapshot file
  defining, 5–16
  defining size, 9–50
  disabling, 9–50
  enabling deferred, 9–50
  enabling immediate, 9–49
  extension, 9–50
Source databases, 2–1, 3–1, 3–6, 6–47
  adding columns to existing tables, 4–29
  adding tables and views, 4–29
  altering columns in existing tables, 4–30
  and transfer method, 2–21
  availability, 2–24
  changing data definitions, 4–29
  conversion of, 4–26
  database version, 5–23
  disk storage space requirements, 2–24
  dropping columns in existing tables, 4–30
  dropping tables or views, 4–30
  effect of transaction rate, 2–24
  inconsistency with target database, 9–33
  logical symbol names for, 9–62
  maximum size, 6–66
  multischema, 5–24
  preparations, 4–7
  Rdb, 1–3
  Rdb Transparent Gateway to DB2, 1–3
  Rdb Transparent Gateway to Oracle, 1–3
  recovery of, 4–25
  relocating files, 4–30
  specifying, 5–4, 5–14
  types, 2–21
Source nodes
  failure of, 9–58
  for extraction rollup, 9–67
Specifying number of rows read and written during a transfer, B–24
Specifying the number of concurrent transfers to be executed, B–19
SQL
  compiling application programs, D–8
  dynamic
    using Replication Option statements, 3–3, 3–4
  interactive
    entering Replication Option statements, 3–3, 3–4
  linking application programs, D–8
  module language
    using Replication Option statements, 3–3, 3–4
  precompiled
    using Replication Option statements, 3–3, 3–4
SQL EXPORT statement, 4–24
SQL IMPORT statement, 4–24
START clause
  in transfer schedule, 6–2
START TRANSFER statement, 9–104
  NO WAIT option, 6–7, 9–104
  NOW option, 6–5, 6–7, 9–104, 9–105
  transfer state, 9–104
  WAIT option, 6–5, 6–7, 9–104
Statements
  definition, 9–1
  information, 9–1
  operation, 9–1
  UIC associated with, 9–1
  UID associated with, 9–1
States
  suspended, 6–13
Status codes
  for transfers last run, C–11
STOP TRANSFER statement, 6–7, 6–9, 9–108
  suspending a transfer, 6–13
Stored procedures
  in target databases, 4–35
SUBMIT/REMOTE command
  executing a remote procedure, 7–23
  for executing remote procedures, 7–22

SUBMIT/REMOTE command (cont'd)
restrictions with, 7–24, 7–25
Subsets
columns, 2–6
data, 2–1
rows, 2–8
tables, 2–5
SUCCESS transfer status, 7–4
Suspended transfer state, 6–5, 6–13
Sybase databases, 1–6
System tables, 3–11
creating, 6–45
RDB$CHANGES, 3–13, A–13, A–14
RDB$CHANGES_MAX_TSER, A–14
RDB$TRANSFERS, A–12, A–13
RDB$TRANSFER_RELATIONS, A–13
RDB$VINTAGE, A–15, A–16, A–17
RDBVMS$TRANSFER_OWNER, A–18

T

Table definitions
changing, 8–8
Tables
adding to source database, 4–29
adding to target database, 4–33, 6–44
altering in target database, 4–34
creating manually, 5–26
dropping in source database, 4–30
dropping in target database, 4–34
Target databases, 2–1, 2–9, 3–1, 3–6
adding domains, 4–34
adding multischema attributes, 6–39, 6–43
adding tables and views, 4–33
advantages of, 2–18
advantages of local access, 1–9
altering domain definitions, 4–34
altering tables and views, 4–34
always Rdb databases, 1–6
backup of, 4–28
changing data definitions, 4–32
conversion of, 4–28
database version, 5–23
defining transfers on, 5–7
disk storage space requirements, 2–22
dropping domains, 4–34
Target databases (cont'd)
dropping tables and views, 4–34
inconsistency with source database, 9–33
indexes, constraints, triggers, query outlines, stored procedures, or access privilege information, 4–35
integrating data definitions from, 7–22
logical symbol names for, 9–62, 9–63, 9–97
maintenance of, 4–27
multiple transfers to an existing target, 8–15
parameters, 5–6
preparations, 4–9
relocating files, 4–35
renaming target tables, 5–6
replications, 5–16
restructuring, 4–36
setting protection for, 7–18
size of, 5–16
transferring to existing, 8–1
types, 2–21
updating the dictionary, 8–8
using access control lists with, 7–18
using MOVE TABLES * with, 5–7
writing to, 8–30
Target directories, 3–6, 5–5
Target file specification
changing for an existing transfer definition, 9–10
Target nodes, 3–6, 5–5
Target tables
renaming, 5–6
Time formats
absolute, 9–4, 9–19
default, 9–7, 9–23
delta, 9–4, 9–19
TO clause
of GRANT statement, 9–78
Transaction numbers
resetting, 6–27
Transactions, creating transfer definitions, 5–21
Transfer database
in VMScluster environments, 9–32
Transfer databases, 3–1, 3–8
access rights for, C–15
conversion of, 4–29

Transfer databases (cont'd)
location of, 3–9
re-creating, C–1
Transfer database table
DDAL_COMMENT, A–9
DDAL_DEST_DPB, A–10
DDAL_INVOKE_TIME, A–3
DDAL_MOVE_CLAUSES, A–10
DDAL_SCHED_OFFSETS, A–11
DDAL_SOURCE_DBS, A–5
DDAL_TNSF_COLUMNS, A–6
DDAL_TNSF_PRIVS, A–11
DDAL_TNSF_SCHEDULE, A–7, A–8
DDAL_TNSF_SEL_EXPR, A–11, A–12
DDAL_TNSF_VIEWS, A–9
DDAL_TRANSFERS, A–3
DDAL_TRANSFER_DB, A–6
DDAL_TRANSFER_STATUS, A–8, A–9
Transfer definitions, 2–1, 3–9
deleting, 6–13, 6–14
using access control lists with, 7–21
using logical names in, 9–62
Transfer description
adding to an existing transfer definition, 9–10
changing for an existing transfer definition, 9–10
Transfer execution status
checking, D–8
Transfer failure during execution, 6–37
Transfer file options clause
ALTER TRANSFER statement, 9–12
CREATE TRANSFER statement, 9–51
epilogue file specification, 9–13, 9–52
prologue file specification, 9–13, 9–51
Transfer methods
and source database type, 2–21
data currency, 2–22
data transfer rate, 2–22
disk storage space requirements, 2–22
requirements, 2–20
Transfer monitor, 3–1, 3–7, 3–8
event enable status, 6–36
in VMScluster systems, 3–8
log files, 6–29, 6–36
logical names defined when starting, C–3
Transfer monitor (cont'd)
MAXDETACH process quota, 6–56
protection of, 6–36
reconfiguring, 3–8
Transfer names
identifying, 7–8
Transfers, 2–1, 3–9
altering, 6–11, 9–10, 9–12
ALTER privilege, 9–78
and database distribution, 2–2
cascaded, 2–3
changing, 6–13, 6–14
creating, 5–3, 9–28
default access, 9–76
defining, 3–9, 5–13, 9–32
defining an extraction rollup transfer, 9–41
defining an extraction transfer, 9–34
defining a replication transfer, 9–33
deleting, 6–14
denying access, 9–89
displaying, 6–18, 9–99
dropping, 6–13, 9–74
DROP privilege, 9–78
executing, 6–5
extraction, 3–10
extraction rollup, 3–10
failure due to network load, 4–23
failure during execution, 6–8
granting access, 9–76
limiting concurrent, 6–55
multiple transfers to an existing target, 8–15
of multiple RMS files, 8–29
of multischema attributes, 8–33
of objects with different character sets, 5–23
operation, C–17
preparations, 4–7
privileges, 9–89
providing remote copies of definitions, 4–6
reinitializing, 6–48
replication, 3–10
replication rollup, 8–2
retrying, 9–59
retrying with DDAL$CP_CONTINUE, 7–11
rollup, 2–3
selecting a type, 2–18

Transfers (cont'd)
starting, 9–104
states, 9–99, C–8, C–9
status codes for last, C–11
stopping, 9–108
suspended state, 6–13
suspending with DDAL$CP_CONTINUE, 7–11, 7–14
to existing databases, 8–1
benefits, 8–1
disk storage requirements, 2–23, 8–31
extraction transfers, 8–7
inconsistent states, 8–31
merging reference data, 8–2
replication transfers, 8–11
restrictions, 8–32
transaction steps, 8–31
updating target dictionaries, 8–8
using, 8–7
two-way, 2–4
when source and target database versions differ, 5–23
Transfer schedules, 2–2, 3–9
Transfer states, 6–4, C–8, C–9
active, 6–5, 9–55
affected by START TRANSFER, 9–104
changes, C–9
scheduled, 6–5, 9–55
suspended, 6–5
unscheduled, 6–5, 9–55
Transfer status
codes for, C–11
determining with command procedures, 7–4
equivalence name, 7–4
indicating with Mail utility, 7–13
indicating with REQUEST command, 7–13
Transfer time, 9–61
Triggers
adding to target tables, 6–42
in target databases, 4–35
Two-way transfers, 2–4, 8–5
TYPE command
advantages over SUBMIT/REMOTE, 7–26
for executing remote procedures, 7–22, 7–25
restriction with, 7–26

U

UIC
for issuing Replication Option statements, 9–1
UIC identifiers, 9–79, 9–91
UNION ALL clause, 5–6
rolling up multiple tables, 9–42
Unions
physical and virtual compared, 8–15
Updates
frequency of, 2–10
restrictions on, 2–10
Users, number allowed, 9–49

V

VAXcluster systems
See VMScluster systems
Versions
compatibility of, 4–2
View
defining on extraction rollup databases, 9–41
transferring, 9–56
Views
adding to source database, 4–29
adding to target database, 4–33, 6–44
altering in target database, 4–34
changing definitions, 8–9
defining in command procedures, 8–7
defining on extraction databases, 9–35
defining on replication databases, 9–34
defining with epilogue procedures, 8–10
dropping in prologue procedures, 8–10
dropping in source database, 4–30
dropping in target database, 4–34
in extraction databases, 5–10
in replication databases, 5–20
in transfers with existing databases, 8–7
limitations in transferring, 5–11, 5–20
READ privilege with, 5–11
transferring, 5–10, 8–9, 8–33

Views name
move views clause, 9–41
VMScluster systems
failover mechanism with, 6–36
monitor log files for, 6–36
single transfer database in, 9–32
source databases with, 3–7
transfer monitors with, 3–8
with failed node, 3–8

W

WAIT option
with START TRANSFER statement, 6–5, 6–7
Wildcard option
SHOW TRANSFER statement, 9–100
Writing data to the same tables, B–25