Replication Option for Rdb

Release Notes

Release 7.2

November 2006

This document contains the release notes for Replication Option for Rdb release 7.2 for HP OpenVMS Industry Standard 64 Integrity Servers and OpenVMS Alpha operating systems.

ORACLE®

Replication Option for Rdb Release Notes

Release 7.2

Contents

2 Problems Corrected

3 Known Problems, Restrictions, and Other Notes

4 Documentation Updates and Corrections

Examples

Tables

Send Us Your Comments

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

- 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?

If you find any errors or have any other suggestions for improvement, please indicate the document title, chapter, section, and page number (if available). You can send comments to us in the following ways:

- Electronic mail:nedc-doc_us@oracle.com
- FAX — 603-897-3825 Attn: Oracle Rdb Documentation
- Postal service:
  Oracle Corporation
  Oracle Rdb Documentation
  One Oracle Drive
  Nashua, NH 03062-2804
  USA

If you would like a reply, please give your name, address, telephone number, and (optionally) electronic mail address.

If you have problems with the software, please contact your local Oracle Support Services.

Preface

This manual describes software errors that have been corrected and current restrictions for Replication Option for Rdb release 7.2. It also describes new features and any changes that might require your immediate attention before you decide to install the product.

Refer to *Replication Option for Rdb Installation Guide* for installation instructions.

Intended Audience

This manual is intended for use by all Replication Option for Rdb users. Read this manual before you install, upgrade, or use Replication Option release 7.2.

Structure

This manual contains the following chapters:

Chapter 1	Describes new and changed features for Replication Option that were implemented since release 7.0.
Chapter 2	Describes software problems that have been corrected for Replication Option release 7.2.
Chapter 3	Describes known problems and restrictions. It also describes workarounds, if available, to problems.
Chapter 4	Describes corrections and additions to the Replication Option documentation.

Related Manuals

The other manuals in the Replication Option documentation set are:

- *Replication Option for Rdb Installation Guide*—Describes how to install Replication Option on OpenVMS systems.
- *Replication Option for Rdb Handbook*—Provides information, guidelines, and examples for distributing all or portions of Rdb databases in a network.

Conventions

Replication Option for Rdb software is referred to as Replication Option throughout this document.

Oracle Rdb is often referred to as Rdb.

HP OpenVMS Industry Standard 64 Integrity Servers is often referred to as OpenVMS I64.

In this manual, OpenVMS means both the OpenVMS Alpha operating system and the OpenVMS I64 operating system.

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

In this manual, the dollar sign represents the DIGITAL Command Language prompt. This symbol indicates that the DCL interpreter is ready for input.

1

Features Implemented Since Release 7.0

This chapter provides a description of new and changed features for Replication Option that have been added since release 7.0.

1.1 Transfers Fail from Databases Open on Single Cluster Node

A problem can sometimes occur where transfers intermittently fail to access the source database. A solution, made available in Replication Option for Rdb release 7.0-12, uses a feature termed "node affinity". You can use this feature to specify for each transfer the VMS cluster nodes where the transfer is to take place. The following subsections describe this feature.

1.1.1 Replication of Oracle Rdb Databases Open on a Single Node

The row caching feature in Oracle Rdb requires that the database be opened only on a single node in an OpenVMS cluster. Other Rdb features offer performance advantages for Rdb databases that are opened only on a single node. Under certain conditions in an OpenVMS cluster, restricting database acccess to a single node can cause transfers performed by the Replication Option for Rdb to fail.

The Replication Option is typically installed on each node in an OpenVMS cluster, though that is not a requirement. It is possible to install on only some of the cluster nodes. For the sake of simplicity, in the following discussions it is assumed that the product is installed and run on each node in a cluster.

The Replication Option was designed to work in the following way on an OpenVMS cluster. A transfer monitor is started on each node of the cluster. A monitor receives requests from an application, typically interactive SQL, to create transfer definitions and transfer schedules. The monitors are responsible for keeping track of the time that each transfer is due to be executed and to create a copy process to perform a given transfer at the appropriate time. The transfer definitions, schedules, and status are recorded in a single transfer database that services all transfer monitors on the OpenVMS cluster. Normally, a given transfer can be executed from any

node in the cluster. Each monitor vies for the opportunity to run the transfer on its node. One monitor will win and claim control of the transfer. It is not possible to accurately predict on which node a given transfer will run.

1.1.2 Error Accessing the Source Database

The *Replication Option for Rdb Handbook, Release 7.0*, fails to mention the following requirement for transfers using the replication method. The copy process image, which performs the actual data transfer, must be able to access the source Rdb database as a local database. That is necessary for security reasons. In the past this was never considered a problem because the source database was typically open for access on all nodes in an OpenVMS cluster. Now, to take advantage of row cache and other Rdb performance features and to also be able to replicate data from that database using the Replication Option, you must insure that transfers which replicate data from a given Rdb database be performed on the same node on which the database is open and not from any of the other cluster nodes. If a transfer process is created on the wrong node, the transfer will fail when the copy process attempts to attach to the source database. The requirement that the source database be attached as a local database is the reason for the change to the Replication Option described in the remainder of this section. Changing the transfer definition so that the source database is treated as a remote database is not an option when the replication method of transfer is used.

1.1.3 Node Affinity Option

A mechanism has been implemented to allow users to declare to the Replication Option the nodes in an OpenVMS cluster on which certain transfers are allowed to execute. The method involves creating a text file, called a configuration file, in which declarations are made in a fashion similar to the way OpenVMS logicals are defined. In fact, OpenVMS system logical names can be used as an alternative to a configuration file.

Perform the following steps to declare node affinity for specific transfers:

1. Choose Method (Configuration File Recommended)

   Create a text file with a name and location of your choosing. In a later step you will declare the location and name of that file by defining an OpenVMS system logical name. Check the resulting file protection of your file. The file must be readable by the transfer monitors on each node. Note that an optional resource configuration file, DDAL.RC, is already used to control transfer copy processes. When placed in the login default directory of the copy process, it is read at the beginning of the copy operation and changes the behavior of the copy process. The configuration files used by the copy

processes are separate from the configuration file used by the transfer monitors.

2. Formatting Rules

   The format of information in the monitor configuration file follows the same rules as for the DDAL.RC files for the copy processes. Those rules explain how to enter comments and how to enter the definitions of environment variables. The rules are found in section B.20 of the *Replication Option for Rdb Handbook, Release 7.0*.

3. Define Transfer Groups

   Define one or more variables to identify groups of transfers. Later, each group will be associated with one or more nodes in the cluster. Also, one or more transfers will be associated with a transfer group. In that way, each transfer will be bound to one or more nodes in the cluster. Binding transfers to named groups and then named groups to cluster nodes makes the process of handling node failure easier as will be explained later.

   Define a variable named DDAL$TRANSFER_GROUPS to identify the various transfer groups. For example,

   ```
   DDAL$TRANSFER_GROUPS G1,G2,G3
   ```

   In this example, three groups are declared: G1, G2, and G3. Group names can be from one to 31 characters long and may consist of the letters A through Z (diacritical marks are not permitted), the numerals 0 through 9, and the dollar sign ($). Underscore characters, which are permitted elsewhere in variable names, are not allowed in group names. This restriction is necessary to avoid the ambiguity that would arise with line continuation variables.

   A line continuation variable is a variable name with _1, _2, etc., appended. You can define line continuation variables if you find that you do not have enough room in the main variable definition to include the complete list of names to be entered, for example:

   ```
   DDAL$TRANSFER_GROUPS "NodeAlphaGroup1 NodeAlphaGroup2"
   DDAL$TRANSFER_GROUPS_1 "NodeBetaGroup1  NodeBetaGroup2"
   ```

   The second variable, DDAL$TRANSFER_GROUPS_1, is a line continuation variable for the main variable, DDAL$TRANSFER_GROUPS. In effect, the two variables are combined to form a list of four transfer groups.

4. Associate Transfers with Transfer Groups

   Define one or more variables to assign transfers to named groups. Define a variable named DDAL$TRANSFERS_IN_GROUP_<group_name> by naming each transfer assigned to the group, for example:

   ```
   DDAL$TRANSFERS_IN_GROUP_G1 Geneve,Bern
   ```

   In this example, two transfers, GENEVE and BERN, are declared to belong to group G1.

   You can define line continuation variables if you find that you do not have enough room in the main variable definition to include the complete list of names to be entered, for example:

   ```
   DDAL$TRANSFERS_IN_GROUP_G1  Geneve,Bern,Munchen,Stockholm,Helsinki
   DDAL$TRANSFERS_IN_GROUP_G1_1 Amsterdam,Bonn
   ```

   These two variables allow you to associate the seven named transfers with group G1.

   Transfers that can be run from any node in a cluster do not have to be associated with a transfer group and do not have to be listed in the configuration file. Any transfer not explicitly associated with a transfer group will be allowed to execute on any node on which the Replication Option is running.

5. Associate Nodes with Transfer Groups

   Define one or more variables to assign OpenVMS cluster nodes to named groups. Define a variable named DDAL$TRANSFER_NODES_IN_GROUP_<group_name> by naming each node on which the named transfer group is permitted to run. Do not use the DECnet cluster alias name. Use the specific OpenVMS cluster node names, for example:

   ```
   DDAL$TRANSFER_NODES_IN_GROUP_G1 DEBIT,CREDIT
   ```

   In this example, two nodes named DEBIT and CREDIT are declared to belong to group G1.

   Line continuation variables may be defined if you find that you do not have enough room in the main variable definition to include the complete list of names to be entered, for example:

   ```
   DDAL$TRANSFER_NODES_IN_GROUP_G1 DEDIT,CREDIT,LOAN,PRIME,RATE
   DDAL$TRANSFER_NODES_IN_GROUP_G1_1 FRANC,MARK,LIRA
   ```

   These two variables allow you to associate the eight named nodes with group G1.

6. Identify the Transfer Monitor Configuration File

   If you are using a transfer monitor configuration file (instead of the alternative, OpenVMS logical names), define DDAL$TRANSFER_MON_CONFIG_FILE in the system logical name table, for example:

   ```
   $ DEFINE/SYSTEM DDAL$TR_MON_CONFIG_FILE  DISK:[DIR]TR_MON_CONFIG_FILE.RC.
   ```

   Also, remember to add this statement to your system startup procedures.

7. Restart the Transfer Monitors

   The transfer monitors first look for the node affinity logical names in the SYSTEM logical name table. Any name not found there will be checked in the configuration file if one exists. This check is made only at the time the monitors are started. Therefore, for a change in the configuration file or logical name table to be effective, the transfer monitors must be restarted.

   The transfer monitor log files record the configuration information used for easy diagnosis of configuration problems. An example of such output is shown later.

Notes:

- When line continuation variables are used, the numeric suffixes must be assigned in order starting from 1 through a maximum of 99, such as DDAL$TRANSFER_GROUPS, DDAL$TRANSFER_GROUPS_1, DDAL$TRANSFER_GROUPS_2, and so on. If, for example, you were to add DDAL$TRANSFER_GROUPS_4 to this series with no definition for DDAL$TRANSFER_GROUPS_3, the values in the configuration variable DDAL$TRANSFER_GROUPS_4 would be ignored.
- If a named group is declared but no transfer-in-group variable is declared, the group name is ignored.
- If a named group is declared and one or more transfer-in-group variables are declared but no transfer-nodes-in-group variable is declared, all transfers associated with that group are defined to be disabled on all nodes in the cluster.
- Environment variable definitions in a configuration file may appear in any order.
- Watch for typographical errors. There is no check that named transfers and nodes actually exist. After a configuration change is made, Oracle recommends that you examine the beginning of each transfer monitor log file to confirm that the transfers you expect to be executable on a given node are listed as executable.

- Node affinity only applies to scheduled transfers. If you use the SQL statement START TRANSFER NOW or START TRANSFER NOW WAIT to immediately start a transfer, the transfer will be executed on the node from which that command was issued, even if the configuration file says to not do so. The transfer would fail, of course, if the database is not open on that node.
- Node affinity does not affect the execution of SQL statements used for the Replication Option. That is, you may execute SQL statements from any node on which the Replication Option is executing. For instance, even though transfer BERN might not be allowed to execute on node EURO, you can still execute on node EURO any SQL statements that affect the BERN transfer.

1.1.4 Sample Configuration File

Below is an example of a configuration file for an OpenVMS cluster with three nodes. Two transfer groups are defined for each node. If a node should become disabled, you could move half of the transfers to one of the remaining nodes and half to the other. Of course, this requires that you open the databases on those other nodes at the appropriate time.

```
! Transfer monitor configuration file for the Replication Option for Rdb

DDAL$TRANSFER_GROUPS  NODE1$GRP1,NODE1$GRP2
DDAL$TRANSFER_GROUPS_1  NODE2$GRP1,NODE2$GRP2
DDAL$TRANSFER_GROUPS_2  NODE3$GRP1,NODE3$GRP2

DDAL$TRANSFERS_IN_GROUP_NODE1$GRP1 Bonn,Munchen
DDAL$TRANSFERS_IN_GROUP_NODE1$GRP2 Geneve,Bern

DDAL$TRANSFERS_IN_GROUP_NODE2$GRP1 Stockholm
DDAL$TRANSFERS_IN_GROUP_NODE2$GRP2 Helsinki

DDAL$TRANSFERS_IN_GROUP_NODE3$GRP1 Amsterdam
DDAL$TRANSFERS_IN_GROUP_NODE3$GRP2 Paris

DDAL$TRANSFER_NODES_IN_GROUP_NODE1$GRP1 NODE1
DDAL$TRANSFER_NODES_IN_GROUP_NODE1$GRP2 NODE1

DDAL$TRANSFER_NODES_IN_GROUP_NODE2$GRP1 NODE2
DDAL$TRANSFER_NODES_IN_GROUP_NODE2$GRP2 NODE2

DDAL$TRANSFER_NODES_IN_GROUP_NODE3$GRP1 NODE3
DDAL$TRANSFER_NODES_IN_GROUP_NODE3$GRP2 NODE3

! End of configuration file
```

1.1.5 What to Do When a Node Fails

Assume from the preceding example that NODE1 becomes disabled. Follow these steps to recover:

- Stop the Replication Option for Rdb

  On each machine running the Replication Option for Rdb, stop the transfers and the transfer monitor by running SYS$MANAGER:DDAL$STOP_TR_MON.COM, as described in the documentation for the Replication Option for Rdb.

- Change the Configuration File

  Edit the configuration file and distribute the NODE1 transfers between NODE2 and NODE3. Do so by making the following changes in the configuration file:

```
DDAL$TRANSFER_NODES_IN_GROUP_NODE1$GRP1 NODE2
DDAL$TRANSFER_NODES_IN_GROUP_NODE1$GRP2 NODE3
```

- Re-open the NODE1 Databases

  Databases which were open only on NODE1 now need to be opened on NODE2 or NODE3.

- Modify RDB$VINTAGE Records

  Any transfers which were originally executed from NODE1, the node which has now failed, must be modified in the target database as follows. In the preceding sample configuration file, the following four transfers were run on NODE1 and are now being redistributed between NODE2 and NODE3: Bonn, Munchen, Geneve, Bern. Rdb target databases have the RDB$VINTAGE table. Examine the following two columns in that table for each of the four target databases of those transfers. Look for transfers which originated from NODE1. For example, you might see:

```
RDBVMS$VINTAGE_TRANSFER_NAME RDBVMS$VINTAGE_TRANSFER_NODE
BONN                         NODE1
```

  Since NODE1 has failed and is no longer a member of the VMS cluster, any attempt to execute the transfer from NODE2 or NODE3 will fail. It will fail because NODE1 is not recognized as a member of the NODE2, NODE3 cluster. To remedy this situation, update the RDBVMS$VINTAGE_TRANSFER_NODE column for such transfers changing the name from NODE1 to either NODE2 or NODE3.

- Restart the Replication Option for Rdb

Restart the Replication Option for Rdb on NODE2 and NODE3 using the file SYS$STARTUP:DDAL$START_TR_MON, as explained in the Replication Option for Rdb documentation.

1.1.6 Sample Transfer Monitor Log File

The following example shows what would appear near the beginning of the transfer monitor log file for node NODE1, given the monitor configuration file shown above before the change due to NODE1 failure:

```
Transfers explicitly allowed to be executed on node NODE1
  Bonn
  Munchen
  Geneve
  Bern

Transfers explicitly NOT allowed to be executed on node NODE1
  Stockholm
  Helsinki
  Amsterdam
  Paris
```

All other transfers are implicitly allowed to be executed on node NODE1

1.1.7 Monitor Log File Error Messages

A few simple checks on the names of transfer groups and transfers are made. A group name cannot contain underscore characters and must not be longer than 31 characters. A transfer name must not be longer than 31 characters. If any of those rules is violated, you will see an error message similar to one of the following in the monitor log file preceding the list of allowed transfers:

```
Error in definition of DDAL$TRANSFER_GROUPS
  Group name ignored because it contains underscore: SWISS_GROUP

Error in definition of DDAL$TRANSFER_GROUPS_1
  Name ignored because it is too long:  ThisGroupNameExceeds31Characters

Error in definition of DDAL$TRANSFERS_IN_GROUP_G1
  Name ignored because it is too long: ThisTransferNameExceeds31Characters
```

1.2 New Options for Purging Rows from the RDB$CHANGES Table

Rows no longer needed in the RDB$CHANGES table of a replicated source database are normally deleted by a copy process at completion of an update transfer cycle. Some customers find this can cause long delays for their online applications, may use up an inordinate number of OpenVMS locks, or create a very large run unit journal (RUJ) file, depending on the number of rows that exist in that table.

Options have been added to the DDAL$PURGE environment variable that let you specify the extent and duration of the purging process. In addition, it is now possible to perform the purging process upon completion of a reinitialized replication transfer.

1.2.1 The Old and New Purging Methods

The old method for removing rows from the RDB$CHANGES table involved the following steps.

1. Start a read-write transaction.
2. Read a few columns and the database keys of all rows from the RDB$CHANGES table, passing these to OpenVMS SORT.
3. Read the sorted information from OpenVMS SORT and delete rows from the RDB$CHANGES table by specifying each row's database key.
4. Commit the transaction.

The new method follows these steps:

1. Start a read-only transaction.
2. Read a few columns and the database keys of all rows from the RDB$CHANGES table, passing these to OpenVMS SORT.
3. Commit the read-only transaction.
4. Sort the information. If all purgeable rows will be erased in a single transaction, sort by database key. This groups rows to be deleted in database page order. If the purging will be done so that a subset of the rows will be deleted in one or more transactions, the rows are sorted by transaction number and then by database key.
5. Start a read-write transaction.
6. Read the sorted information from OpenVMS SORT and delete rows from the RDB$CHANGES table by specifying each row's database key.

7. You have three options for how purging is done. The option you pick affects the extent and duration of the purge operation. (See Table 1–1 for information on how to specify these options.)

   - Erase all purgeable rows in one transaction. This is equivalent to the original method used by Replication Option.
   - Erase all purgeable rows, up to N rows at a time, using one or more transactions.
   - Erase up to N rows in a single transaction. This method might not erase all the purgeable rows. To do so you would have to run some other transfer to continue the erasing process.

   The row limit is an approximation. Once deletion of rows for a given source transaction has started, the deletion must continue until all rows for that transaction have been deleted.

   The last option is an incremental method; the first two methods erase all that can be erased.

8. Commit the read-write transaction.
9. If the second method is used and there are more rows to purge, start another read-write transaction and repeat from step 6.

1.2.2 Purging After a Reinitialized Transfer Has Executed

By default, the purging of rows from the RDB$CHANGES table is performed only at the end of replication update transfers, not at the end of initial replication transfers. Now an option exists to allow purging at the end of initial transfers (more specifically, reinitialized replication transfers) as well.

The reason you might want to purge RDB$CHANGES after reinitialized transfers can best be explained by example. Suppose you execute 12 different transfers throughout the business day, but only purge rows from the RDB$CHANGES table once, sometime in the evening. After a typical day, there are 300,000 rows to be purged from the RDB$CHANGES table.

Suppose you then find it necessary to reinitialize and rerun all 12 transfers. After these transfers have executed, the RDB$CHANGES table still has the 300,000 rows that are now no longer needed. The next day, operation continues as normal with each transfer performing incremental updates by transferring only new and changed data. As a result, purging only takes place at the end of the second day, when the size of the RDB$CHANGES table has grown to 600,000 rows.

It is important for good performance to keep the RDB$CHANGES table from growing in size. The more space allocated to that table, the longer it takes for applications to insert data into the table. Hence, purging after reinitialized transfers is a recommended change, although it is optional.

1.2.3 Choosing New Purge Options

The DDAL$PURGE environment variable, which already exists in Replication Option, has new options that are specified by defining the DDAL$PURGE variable (or a transfer-specific variable with the same root name). The definition of the variable can contain one or more keywords separated by commas (see Table 1–1 for a list of the keywords). Some keywords require values. The keywords and values are separated by single colons or equal signs. Keywords can be entered in upper or lower case. For example,

```
DDAL$PURGE_ABC OPTION=3,ROWS=5000
```

This example indicates that the ABC transfer should purge rows from the RDB$CHANGES table using purge option 3 (see Table 1–1) with a transaction committed on or after 5000 rows have been deleted. Also, because the REINIT keyword has been omitted, by default purging will not take place at the end of reinitialized transfers.

Table 1–1 DDAL$PURGE Keywords

Keywords	Meaning
YES	This can be abbreviated as Y. This means that the copy process will attempt to purge the RDB$CHANGES table, provided there is no other copy process already doing so. Transaction option 2 will be used and the row limit per transaction will be 10000. Purging will not be done after reinitialized transfers.
NO	This can be abbreviated as N. This means that the copy process will not attempt to purge the RDB$CHANGES table.
OPTIONS=2	Transaction options. 1=Delete all purgeable rows in a single transaction; 2=Delete all purgeable rows in multiple transactions; 3=Delete a limited number of rows in a single transaction. The default, as shown in the Keywords column, is 2.
ROWS=10000	Row limit per delete transaction. The allowed range of values is 1 to 100000, the limits being arbitrary. The default, as shown in the Keywords column, is 10,000.

(continued on next page)

Table 1–1 (Cont.) DDAL$PURGE Keywords

Keywords	Meaning
REINIT=YES	Purging will be done after reinitialized transfer execution. Alternatively you can say REINIT=NO (the default).

If none of these variables is defined, or if its definition does not begin with N or n, the copy process will purge RDB$CHANGES, which is its original default behavior. This will only be done after replication update transfers, not after reinitialized transfers. If the OPTIONS or ROWS keyword is not specified, the default values will be option 2 and a row limit of 10000. This is a departure from the original Replication Option behavior, which was option 1.

1.2.4 Changes to the Transfer Log

Additional information now appears in transfer logs (also known as copy process logs). Below is an example, with an explanation following the example. The purging options used to generate this log file are not the same as those in the preceding example.

```
10:43:33  %DDAL-I-STADATTRM, starting data transfer/modification
10:43:42  %DDAL-I-PURGING_1, starting purge of RDB$CHANGES data rows because ...
          %DDAL-I-PURGING_3, ... DDAL$PURGE is defined to be "OPTION=2,ROWS=10"

          %DDAL-I-TOTPURGE1, There are 38 rows (21 transactions) in
          %DDAL-I-TOTPURGE2, the RDB$CHANGES table for transactions with
          %DDAL-I-TOTPURGE3, TSER numbers in the range 709 to 773.
          %DDAL-I-TOTPURGE4, Rows associated with transactions whose TSER
          %DDAL-I-TOTPURGE5, numbers are less than 773 can be deleted.

          %DDAL-I-PURGEROWS, 11 rows (6 transactions) deleted from RDB$CHANGES
          %DDAL-I-PURGEROWS, 10 rows (6 transactions) deleted from RDB$CHANGES
          %DDAL-I-PURGEROWS, 11 rows (6 transactions) deleted from RDB$CHANGES
          %DDAL-I-PURGEROWS, 4 rows (2 transactions) deleted from RDB$CHANGES

10:43:44  %DDAL-I-TOTPURGE6, total of 36 rows (20 transactions)
          %DDAL-I-TOTPURGE7, deleted from the RDB$CHANGES table
          %DDAL-I-TOTPURGE8, with TSER numbers in the range 709 to 772

10:43:44  %DDAL-I-ENDDATTRM, ending data transfer/modification
```

The messages %DDAL-I-TOTPURGE1 through %DDAL-I-TOTPURGE5 indicate which rows were in the RDB$CHANGES table. The message %DDAL-I-PURGEROWS is printed for each purge transaction whenever multiple transactions are used (option 2). The messages %DDAL-I-TOTPURGE6 through %DDAL-I-TOTPURGE8 give a summary of how many rows and transactions were deleted.

1.3 Improved Performance By Batching Update Transactions

An option has been added that may improve transfer performance and reduce the amount of I/O performed in the target database.

1.3.1 The Old and New Methods for Executing Update Transactions

The original and default behavior of Replication Option is to execute each source transaction that it reads from the RDB$CHANGES table as one target transaction. Some source transactions will not have any changes that apply to a given transfer. Nevertheless, such "null" transactions result in a transaction on the target database. In that target transaction, the source transaction's serial number (TSER) and its timestamp are updated in the transfer's row in the RDB$VINTAGE or DDAL$TRANSFER_INFO table of the target database.

You can now reduce the number of target transactions and the number of times the VINTAGE or INFO table is updated. For example, you can specify that for every ten source transactions read from the RDB$CHANGES table, only one target transaction is to be applied. This reduction in I/O can result in better transfer performance.

1.3.2 Specifying Values for Execution of Update Transactions

You can specify two values that affect the number of update transactions that get executed. One is the number of source transactions to be grouped into a single target transaction. The other is the number of null source transactions to be treated as if they were a single source transaction. This is best explained by example.

```
$ DEFINE  DDAL$UPDATE_TRANSAC_BATCH_SIZE  "SOURCE=2,NULL=5"
```

Consider the example definition above and the following sequence of transactions in the RDB$CHANGES table. Transactions represented by S are source transactions that have changes relevant to the given transfer. Transactions represented by N have no changes relevant to the given transfer; these are the null transactions. The numbers above the letters count the null source transactions.

The transactions have been grouped into 6 target transactions (under the old method, there would have been 33 separate transactions). The numbers below the letters indicate in which of the 6 target transactions each source transaction will appear.

```
  123 451 2345123451 1234512345
SSNNNSNNNSNNNNNNNNNSNNNNNNNNNSS
112222223333334444444555555555566
```

Batching source transactions reduces I/O to the target database. However, when network links fail and a transfer has to be re-executed, the number of source transactions that need to be reprocessed increases.

1.3.3 Choosing Batch Size Values

The new options for batching update transactions can be specified by defining the DDAL$UPDATE_TRANSAC_BATCH_SIZE variable (or a transfer-specific variable with the same root name). The definition of the variable can contain one or more keywords, separated by commas. (See Table 1–2 for a list of the keywords.) Each keyword requires a value. The keywords and values are separated by single colons or equal signs. Keywords can be entered in upper or lower case.

Table 1–2 DDAL$UPDATE_TRANSAC_BATCH_SIZE Keywords

Keywords	Meaning
SOURCE=10	Number of source transactions to apply as a single target transaction when performing replication update transfers. The allowed range of values is 1 to 100. The upper limit is arbitrary. The default is 10.
NULL=10	Number of null source transactions to treat as if they were a single source transaction for the purposes of batching. The allowed range of values is 1 to 100. The upper limit is arbitrary. The default is 10.

If none of these variables is defined, or if one of the keywords is not defined, the appropriate batching factor is set to 1, which corresponds with the original Replication Option behavior.

1.3.4 Changes to the Transfer Log

Additional information will now appear in transfer logs (also known as copy process logs). Below is an example, with an explanation following the example.

```
10:43:44  %DDAL-I-REPUPSTAT,    REPLICATION UPDATE STATISTICS

          %DDAL-I-NUMROWINS, number of rows inserted         =        692
          %DDAL-I-NUMROWUPD, number of rows updated          =          0
          %DDAL-I-NUMROWDEL, number of rows deleted          =          0
          %DDAL-I-TOTALIUDS, total inserts, updates, deletes =        692

          %DDAL-I-TOTALSRCT, source transactions applied =         15
          %DDAL-I-TOTALTGTT, target transactions applied =          5
```

The message %DDAL-I-TOTALSRCT tells you how many source transactions were processed from the RDB$CHANGES table. The message %DDAL-I-TOTALTGTT, which is new, tells you how many target transactions were used to do so. The batching factors for this example were not the same as for the earlier example. Here DDAL$UPDATE_TRANSAC_BATCH_SIZE Source=4,Null=100 was used.

1.4 Improved Replication Update Checkpointing Behavior

In Replication Option version 7.0, checkpoint restart behavior was extended to the execution of replication update transfers. The method by which this is accomplished has been completely changed to improve its usefulness. If you already use checkpointing with replication transfers, you might need to make a simple adjustment to accommodate the software change.

1.4.1 The Old and New Methods of Checkpointing for Replication Update Transfers

The original implementation relied on buffering in memory all the data for a single source transaction as read from the RDB$CHANGES table. Then, if the target transaction failed and needed to be restarted, the source information would be in memory and could be reprocessed. For long source transactions, running out of room in memory to store the transaction could limit the usefulness of the checkpointing option. Also, with the implementation of source transaction batching (see Section 1.3), it has become necessary to be able to keep track of more than just one source transaction for recovery purposes.

The new method does not save the entire source transaction in memory. Instead, each row read from the RDB$CHANGES table is remembered by saving its database key and the transaction serialization (TSER) number. This is a reduction from a typical 960 bytes per row to 12 bytes per row. For the same amount of memory, Replication Option can keep track of 80 times the information. In the event of a target transaction failure, a copy process retrieves the database key of some row in RDB$CHANGES from memory and then rereads that row.

1.4.2 Specifying How Many Keys to Buffer

When no environment variable is defined, a transfer process allocates enough memory to keep track of 100 rows from the RDB$CHANGES table. You can change this value by defining the environment variable, DDAL$CKPT_UPDATE_ROWS (or a transfer-specific variable with the same root name). The definition of the variable must be a decimal number in the range 10 to 10000. For example,

```
$ DEFINE  DDAL$CKPT_UPDATE_ROWS_ABC  1000
```

This definition means that for the ABC transfer 12000 bytes of memory will be allocated to keep track of up to 1000 rows from the RDB$CHANGES table. The variable is examined only if the transfer definition has checkpointing enabled.

1.5 Improved RDB$TRANSFERS Transaction Behavior

At the end of a replication update transfer, the transfer's row in the RDB$TRANSFERS table must be updated. The method used to do this has been improved to reduce contention on the database.

1.5.1 The Old and New Way of Updating the RDB$TRANSFERS Table

In the past, whenever the RDB$TRANSFERS table needed to be updated, a transaction was started to reserve the table for exclusive access. This sometimes led to unnecessary delays and possible contention for database access.

The new method of updating the RDB$TRANSFERS table no longer requires exclusive access to that table. First, the copy process uses a read-only transaction to locate the transfer's row in the RDB$TRANSFERS table. This transaction is committed and a second read-write transaction is started reserving RDB$TRANSFERS for concurrent write. The transfer's row is updated by referencing the row's database key.

This behavior is a change from the behavior described in item 8, section C.7.5 of the *Replication Option for Rdb Handbook*, release 7.0.

1.6 Improved Replication Update Failure Error Reporting

There are a series of errors (BADDBKINS, BADDBKUPD, BADDBKERA) that can be reported during update transfers to a target database. These errors indicate which type of operation failed (INSERT, UPDATE, and so forth) and give the database key of the row in the source database. The format of the logged information and the level of detail have in some cases been improved. This can help in diagnosing the problem.

1.6.1 The Old and New Way of Reporting Errors for Update Transfers

In the past, whenever one of these errors was reported, the source database key was printed in hexadecimal format. No information was printed about the target table row if one already existed.

The changes now print all Rdb database keys in two formats: hexadecimal and area:page:line. In addition, if the target is an Rdb database, a line is printed identifying a row in the target table, if one exists, that is associated with the error.

The following transfer log excerpt shows the new information.

```
--- 26-APR-1997 08:02:33.19 --- Error  -------------------
%DDAL-E-BADDBKINS, insert in target table "COMP_RATE" failed
-DDAL-I-SRCDBKEY, source dbkey "015700002FA50002 (343:12197:2)" row
 owner "0"
-DDAL-I-TGTDBKEY, target dbkey "00320000021E0003 (50:542:3)"
```

1.7 Improved Archive Transfer Performance

An archive transfer is the informal name for a REPLICATION transfer WITH NO DELETE. The target database of an archive transfer contains both “live” data and “deleted” data. The live data is data that also exists in the source database. The deleted target data is not actually deleted. It remains in the target database but is marked in a way that indicates the rows no longer exist in the source database.

If, for some reason an archive transfer fails and the transfer needs to be reinitialized, the existing “live” target data in the target tables and their associated indexes must be deleted before the current “live” source data is transferred. The default Replication Option behavior is to perform the following steps:

1. Drop the table indexes.
2. Delete the table data.
3. Transfer the new “live” data from the source database.
4. Recreate the target indexes.

This sequence works best when ALL the data in the target tables is to be erased. However, for archive transfers, only the “live” data is to be erased. Over time, the “deleted” data rows will likely outnumber the “live” rows. In such a situation, row deletion is faster if the indexes are left in place and dropped only after the rows have been deleted. Therefore an option has been added for transfer execution, enabled by defining the environment variable, DDAL$TARGET_INDEX_OPTIONS.

1.7.1 The DDAL$TARGET_INDEX_OPTIONS Environment Variable

A description of the DDAL$TARGET_INDEX_OPTIONS environment variable is in Appendix B of the *Replication Option for Rdb Handbook* for version 7.0. The options available have been extended, as described in the following list. Not defining this variable or defining it with a bad value is equivalent to choosing the default behavior.

- Indexing option 1

  The default behavior. The copy process drops an index if it exists prior to deleting and retransferring data. Then it creates the index based on its built-in definition of what the index should be.

- Indexing option 2

  If an index definition already exists, do not drop it. If it does not exist, signal an error.

- Indexing option 3

  The copy process drops an index after deleting data from a target table. Later, it recreates the index after transferring the data.

Indexing option 1, the default, is useful when all the data in the target tables is to be deleted. Deletion will go faster without the index. The situation is different for archive transfers, REPLICATION WITH NO DELETE. Initially, using option 1 gives the best performance. Eventually, the number of rows to be deleted will be a fraction of those in the tables. The rows that remain are those that have been marked as deleted in the source database. In this case, the index may help performance during row deletion.

1.8 Message DDAL$_INVLOGTRN Has Been Changed

The DDAL$_INVLOGTRN error message will no longer display. It was being used for two different error conditions. Each of those conditions now has its own error message: DDAL$_NOCOMMIT and DDAL$_AFTERCOMMIT. Both of these new messages signify that there has been a corruption of the RDB$CHANGES table. DDAL$_NOCOMMIT indicates that there is no commit record recorded as part of the transaction. DDAL$_AFTERCOMMIT indicates that a commit record was found but that more data for the transaction exists following the commit record.

Note

There is a possible problem associated with the DDAL$_ AFTERCOMMIT error. See Section 3.1 for more information.

1.9 The DDAL$ANALYZE Utility

DDAL$ANALYZE is a new tool introduced in Replication Option for Rdb 7.0.1. Its primary purpose is to help diagnose problems with replicated data stored in the RDB$CHANGES table of a source database. Analysis of the RDB$CHANGES table can be performed on a periodic basis to detect data corruption and to report statistics about the transactions recorded in that table. When replication transfers are executed, they will fail if any corruption exists. If such transfers are performed frequently, it is more likely that DDAL$ANALYZE will be used after an error has been reported rather than before.

1.9.1 A Few Examples

This section shows examples of typical ways in which the DDAL$ANALYZE utility can be employed. In later sections the full set of options will be explained in detail, and more examples will be given.

Suppose your production system has a master database, HQ_PROD_DB, whose tables are being replicated to several remote databases in your company. At some point the transfers begin failing one by one. When you investigate you find that all show the same error message code, DDAL$_INVJOUVER, in the transfer log files. The fact that all transfers fail with the same error code is indicative of a data corruption in the RDB$CHANGES table. There are a number of other such error codes, listed in Section 1.9.6.

What can cause data corruption in the RDB$CHANGES table? It could be due to a problem in software design, a hardware malfunction, or human error. If the problem is due to human error, reinitializing and re-executing the transfers will usually make the problem go away and not return. That is, human error is often not repeatable. If it is due to a hardware error, hardware diagnostic programs usually alert you to a problem in the making. The hardware must be repaired and the transfers must be reinitialized and re-executed. If the problem is due to an error in software design, it most likely is a problem in Oracle Rdb or in Replication Option itself. Typical customer applications do not try to access and write to the RDB$CHANGES table, a table that requires special access privilege.

If the problem seems to repeat, albeit on an unpredictable basis, and the hardware appears to be functioning correctly, you will want to report the problem to Oracle Corporation and provide as much evidence as you can to help in its diagnosis. The DDAL$ANALYZE utility is one of the tools you can use to gather information about the problem. If you are going to

analyze RDB$CHANGES, you must do so before you reinitialize and re-execute the failed transfers. Otherwise you risk deleting the evidence from the RDB$CHANGES table before you get to look at it.

1.9.1.1 Scan RDB$CHANGES for Errors

DDAL$ANALYZE can report information in varying amounts of detail. To begin with you can ask it to output information only when it encounters errors and to list a summary of statistics. Example 1–1 shows such a report. The example in Section 1.9.8.5 gives additional detail.

In Example 1–1, a scan of all the transactions in RDB$CHANGES turned up a single error. Transaction number 1962487 (the TID/TSN) is the transaction that has the error in it. TID means transaction identifier; TSN means transaction sequence number. These are two names for the same thing. The transaction is also known by its TSER, its transaction serialization number. The TSER in this example is 641106. The TID/TSN is assigned when a transaction starts; the TSER is assigned when a transaction is about to commit. This transaction takes up two rows in the RDB$CHANGES table. The database keys for the rows are listed after the row sequence numbers.

The TID/TSN is from the RDB$TRANSACTION_TID column of RDB$CHANGES. The TSER comes from RDB$TRANSACTION_TSER. Be aware that the TSER value, which is assigned to a transaction only when the transaction is ready to commit, is only non-zero in the final row, the one with the highest sequence number. The sequence numbers come from the RDB$TRANSACTION_ SEQUENCE column, and the changes are stored in the RDB$TRANSACTION_ CHANGES column.

Example 1–1 Summary Report with a Scan for Errors

```
Replication Option for Rdb V7.0-1         3-SEP-1997 11:42:23.80

Analysis of the RDB$CHANGES table of replicated source database:
DISK020:[ADMIN]HQ_PROD_DB.RDB

************************************************************************

Transaction: TSER = 641106  TID/TSN = 1962487

%DDAL-E-AFTERCOMMIT, commit record not end of transaction

         1 - 146:40224:0

************************************************************************

SUMMARY STATISTICS:
```

(continued on next page)

Example 1–1 (Cont.) Summary Report with a Scan for Errors

```
    Number of transactions in RDB$CHANGES ... = 100011
    Number of rows in RDB$CHANGES ........... = 157358

    Lowest  TSER analyzed ................... = 616863
    Highest TSER analyzed ................... = 721169
    Number of transactions analyzed ......... = 100011
    Number of rows analyzed ................. = 157358
    Most rows for a transaction ............. = 2504

    Average number of rows per transaction .. = 1.57
    Average TSER density .................... = 0.95

ERRORS REPORTED:  1         3-SEP-1997 12:00:54.06
```

The preceding example shows the results of scanning all transactions in the RDB$CHANGES table and reporting all errors found. In this case there was only one error. Example 1–2, and also the examples in Section 1.9.8.1 and Section 1.9.8.2 show how to examine RDB$CHANGES for a narrow range of transactions when the TSER of the failing transaction is known.

1.9.1.2 Analyze a Narrow Range of Transactions

It might prove useful to examine in more detail the activity that was going on around the time the failed transfer was executing. Run DDAL$ANALYZE again, this time asking it to output information for transactions in the range 641100-641110. The failing transaction, 641106, is in the middle of this range. This time you ask for full detailed transaction output. Example 1–2 is an excerpt from such a report:

Example 1–2 Full Report of a Small Range of Transactions

```
Replication Option for Rdb V7.0-1         3-SEP-1997 12:19:25.37

Analysis of the RDB$CHANGES table of replicated source database:
DISK020:[ADMIN]HQ_PROD_DB.RDB

********************************************************************

Transaction: TSER = 641100  TID/TSN = 1962480  format version = 2

Logical record type = INSERT     length = 81
```

(continued on next page)

Example 1–2 (Cont.) Full Report of a Small Range of Transactions

```
    Source table name = COMP_RATE
    Source row database key = 343:12197:2 (015700002FA50002)
    After image:
         <------- <------- <------- <------- ---------------->
         20202020 31303139 393901CA 3B400001 ..@;J.999101     0000
         00000000 0000009B 465F3B73 48402020   @Hs;_F........ 0010
         45DDB9F3 C0000000 00030000 01A40000 ..$........@s9]E 0020
                        08 FE4E0000 0001009B ......N~.        0030

Logical record type = END_INSERT

Logical record type = COMMIT    Commit time: 3-SEP-1997 10:25:41.08

RDB$CHANGES rows in this transaction = 1

         1 - 146:40218:0

*************************************************************************

... other transactions, up through and including the one with TSER = 641110

*************************************************************************

ERRORS REPORTED:  1         3-SEP-1997 12:29:09.68
```

Corruption problems are sometimes difficult to reproduce at will. When that is the case, looking for patterns of activity can provide clues about the conditions under which the error occurs. A report such as the one in Example 1–2 might help paint a picture of this activity.

1.9.1.3 Output Rows from RDB$CHANGES Unanalyzed

Data corruption means that the analyzer is unable to interpret some or all of the information about the transaction. Further investigation will have to be done by someone trained in deciphering a printout of the transaction's rows from the RDB$CHANGES table. Whereas Example 1–2 shows a transaction composed of logical records (INSERT, UPDATE, DELETE, etc.), Example 1–3 shows the raw data from the RDB$CHANGES rows.

Example 1–3 Report of Raw Data Rows for a Transaction

(continued on next page)

Example 1–3 (Cont.) Report of Raw Data Rows for a Transaction

```
Replication Option for Rdb V7.0-1         3-SEP-1997 12:43:46.53

Analysis of the RDB$CHANGES table of replicated source database:
DISK020:[ADMIN]HQ_PROD_DB.RDB

************************************************************************

Transaction: TSER = 641105  TID/TSN = 1962486

Row sequence number = 1
    Row contents:
         <------- <------- <------- <------- --------------->
         45544152 5F504D4F 430C0000 00510402 ..Q....COMP_RATE 0000
         00010039 01570000 2FA50002 08424141 AAB...%/..W.9... 0010
         20202020 20203130 31393939 01CA3B40 @;J.999101      0020
         00000000 00000000 009B465F 3B734840 @Hs;_F.......... 0030
         009B45DD B9F3C000 00000003 000001A4 $........@s9]E.. 0040
         9B465F49 2D0F9D09 0508FE4E 00000001 ....N~.....-I_F. 0050
                                         00 .                0060

************************************************************************

ERRORS REPORTED:  0         3-SEP-1997 12:53:30.76
```

Once the reports have been produced, you can reinitialize and re-execute failed transfers.

1.9.2 How to Invoke DDAL$ANALYZE

The RDB$CHANGES table contains data from all replicated tables in the database. Security is maintained on that table to restrict user access to its contents. DDAL$ANALYZE is intended to be run by database administrators or system managers who normally have sufficient privileges and system quotas to manage databases and in particular to be able to read from RDB$CHANGES.

Example 1–4 shows how to run DDAL$ANALYZE and shows prompts and responses for the three file specifications it needs.

Example 1–4 Invoking DDAL$ANALYZE

```
$ RUN SYS$SYSTEM:DDAL$ANALYZE
Source database to be analyzed: DISK020:[ADMIN]HQ_PROD_DB.RDB
Outfile file [SYS$OUTPUT] ... : HQ.LIS
Input parameters file ..... : HQ.PARAMS
```

The first prompt asks you to enter the location and name of the source database root file. This is a required parameter for which the only default is the filename extension, .RDB.

The second prompt asks for the location and name of the output report file to be created by DDAL$ANALYZE. If you simply press the ENTER key without giving a filename, the report by default will be sent to SYS$OUTPUT (normally your terminal screen or batch file). You can give a filename without naming a location and/or without specifying an extension. In that case the report file will be created in your working (default) directory and the file extension will be .LIS.

The third prompt asks for the location and name of a parameter file. A parameter file is a simple text file you create containing options that specify a range of transactions to be analyzed and the amount of detail to appear in the report. You are not required to name a parameter file. All options have default values. Details about the parameter file are given in Section 1.9.4.

1.9.2.1 Effect of DDAL$ANALYZE on Other Applications

DDAL$ANALYZE attaches to the source database without requiring exclusive access. It starts a read-only transaction and reads all the rows in RDB$CHANGES ordered by ascending TID value and by descending sequence number. As it does so, records are passed to SORT.

Once all the rows have been read, the transaction is committed and SORT orders the records by increasing TSER and increasing sequence number.

Finally, a second read-only transaction is started, information is retrieved from SORT, rows are read from RDB$CHANGES by database key, the report is generated, the transaction is committed, and a disconnect is performed on the database.

DDAL$ANALYZE can be run at the same time as customer applications on the source database with minimal effect on performance of those applications.

Alert!

While DDAL$ANALYZE is running it is recommended that no replication transfers be executing if those transfers could delete (purge) rows from RDB$CHANGES. Otherwise DDAL$ANALYZE might encounter an error when it tries to read a row by database key in the second of its transactions. Purging rows from RDB$CHANGES is the normal behavior of replication update transfers. Options exist to allow you to control when purging takes place. Refer to the description of the DDAL$PURGE environment variable in the *Replication Option*

for Rdb Handbook and a recent addition to that variable described in Section 1.2 of these release notes.

1.9.3 Analyzer Parameters

You can define parameters as OpenVMS logical names or you can define parameters within a text file of your own naming and location.

- If you define a parameter both as a logical name and as an entry in the parameter file, the file entry takes precedence over the logical name definition.
- If you define parameters as logical names, you can type SHOW LOGICAL DDAL$ANAL* and SHOW LOGICAL DDAL_ANAL* at the OpenVMS prompt to display a list of all analyzer logical names defined.
- In all analyzer parameter names, an underscore can be substituted for the dollar sign. The analyzer first looks for a parameter whose name begins with DDAL$. If that fails, it then looks for the name beginning with DDAL_.
- Parameter names are not case-sensitive. They can be entered in upper case, mixed case, or lower case. They will be converted by the analyzer to upper case.

1.9.4 Rules for Setting Up a Parameter File

Parameter 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 a parameter's name and 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 parameter name followed by one or more space or tab characters followed by the parameter's definition and optionally followed by one or more spaces or tabs and a comment.
- If a parameter's definition must contain spaces, the definition must be enclosed within double quotation marks (" ").
- Valid parameter names and definitions are given in the following sections.

1.9.5 Parameters for DDAL$ANALYZE

You can define any or all of the following parameters in a DDAL$ANALYZE parameter file or as a logical name. Each parameter is described in its own section, which follows.

- DDAL$ANAL_DISK_BLOCKS_PER_TRANSACTION
- DDAL$ANAL_ENDING_TSER
- DDAL$ANAL_ERROR_LIMIT
- DDAL$ANAL_LOG_RESOURCE_USAGE
- DDAL$ANAL_LOG_STATISTICS
- DDAL$ANAL_LOG_TRANSACTIONS
- DDAL$ANAL_ROWS_PER_TRANSACTION
- DDAL$ANAL_STARTING_TSER
- DDAL$ANAL_TRANSACTION_LIMIT
- DDAL$ANAL_TSER_DENSITY

1.9.5.1 DDAL$ANAL_DISK_BLOCKS_PER_TRANSACTION

The analyzer can give you an estimate of the number of disk blocks that would be needed to generate a particular report (see DDAL$ANAL_LOG_ RESOURCE_USAGE). To perform the calculation the analyzer needs to know the average number of disk blocks that are used for each transaction in the report.

By default, the analyzer uses built-in values for each type of information to appear in the report. A transaction can be logged in varying degrees of detail (see DDAL$ANAL_LOG_TRANSACTIONS). The built-in values for all details to be logged are summed to yield an average number of disk blocks per transaction. These values do not necessarily reflect any one customer's database, but they are typical of actual customer situations. You can override the built-in value (the sum of the values for those details to appear in the report) by defining the DDAL$ANAL_DISK_BLOCKS_PER_TRANSACTION parameter.

If you produce the report in a disk file and include summary statistics in the report, the summary will tell you how many transactions were analyzed. Use that number and the number of disk blocks allocated for the report file to calculate the average value.

Use the following format to define the parameter as an OpenVMS logical name:

```
$ DEFINE   DDAL$ANAL_DISK_BLOCKS_PER_TRANSACTION   m.n
```

Alternatively, enter a definition in the parameter file:

```
DDAL$ANAL_DISK_BLOCKS_PER_TRANSACTION m.n
```

Here `m.n` represents an unsigned decimal number such as 20.15. Up to six fractional digits are accepted, though two digits should suffice for most needs.

Also see the DDAL$ANAL_ROWS_PER_TRANSACTION and DDAL$ANAL_TSER_DENSITY parameters. They also affect calculation of disk block usage.

1.9.5.2 DDAL$ANAL_ENDING_TSER

The parameters DDAL$ANAL_STARTING_TSER, DDAL$ANAL_ENDING_TSER, DDAL$ANAL_TRANSACTION_LIMIT and DDAL$ANAL_ERROR_LIMIT all serve to restrict the number of transactions to be analyzed. When you define DDAL$ANAL_ENDING_TSER you indicate that analysis should stop when a transaction is encountered with a higher TSER value. In other words, this is the highest numbered transaction that you want to appear in the report. It is not required that a transaction with the given number exist.

By default the analyzer processes all transactions.

You can define the ending TSER value in an OpenVMS logical name:

```
$ DEFINE   DDAL$ANAL_ENDING_TSER   n
```

Alternatively, enter a definition in the parameter file:

```
DDAL$ANAL_ENDING_TSER n
```

Here `n` represents an unsigned decimal integer from 1 to 4294967295. If you have defined a value in DDAL$ANAL_STARTING_TSER, the ending TSER must be the same value or higher.

Also see the DDAL$ANAL_ERROR_LIMIT and DDAL$ANAL_TRANSACTION_LIMIT parameters. They provide other reasons that the analysis stops before processing all transactions.

1.9.5.3 DDAL$ANAL_ERROR_LIMIT

When analyzing RDB$CHANGES to look for errors, you might be interested only in the first few errors that are found. If that is the case, define DDAL$ANAL_ERROR_LIMIT to reduce the number of errors to be reported. If the error limit is reached, analysis will stop.

By default, the analyzer processes all transactions no matter how many errors are found.

You can define the error limit in an OpenVMS logical name:

```
$ DEFINE   DDAL$ANAL_ERROR_LIMIT   n
```

Alternatively, enter a definition in the parameter file:

```
DDAL$ANAL_ERROR_LIMIT n
```

Here n represents an unsigned decimal integer from 0 (zero) to 4294967295. A value of zero is equivalent to placing no limit on the number of errors.

Also see the DDAL$ANAL_ENDING_TSER and DDAL$ANAL_TRANSACTION_LIMIT parameters. They provide other reasons that the analysis stops before processing all transactions.

1.9.5.4 DDAL$ANAL_LOG_RESOURCE_USAGE

If you generate a full report of all transactions in the RDB$CHANGES table, the output can consume a million disk blocks or more. You can reduce the amount of output by limiting the number of transactions to be analyzed (see DDAL$ANAL_STARTING_TSER, DDAL$ANAL_ENDING_TSER and DDAL$ANAL_TRANSACTION_LIMIT) and by decreasing the amount of detail in the report (see DDAL$ANAL_LOG_TRANSACTIONS). Still, if you know that the output might be lengthy, you may wish to get an estimate on the number of disk blocks before you actually perform the analysis. Do so by defining DDAL$ANAL_LOG_RESOURCE_USAGE.

By default the analyzer will output resource usage estimates.

You can define the resource usage parameter in an OpenVMS logical name:

```
$ DEFINE   DDAL$ANAL_LOG_RESOURCE_USAGE   x
```

Alternatively, enter a definition in the parameter file:

```
DDAL$ANAL_LOG_RESOURCE_USAGE x
```

The letter x stands for the keyword ONLY, YES, or NO. If you enter the word ONLY, the report will consist only of the resource estimate, even if you also specify that transaction and/or summary information should be logged. The reason for this is to provide an estimate of the size the report would be were you to actually produce the full report. With such information you can check first to see if you have a disk with sufficient free space on it. Since the size of the full report depends on the setting of the DDAL$ANAL_LOG_TRANSACTIONS parameter and the details chosen to appear, define that parameter as if you were producing the desired report.

If you enter the word YES, a resource estimate will appear in the report, along with transaction and summary information if those options are also selected. If you enter the word NO, no estimate will appear.

Be aware that the estimated number of disk blocks is for the generated report. It does not include disk space you might need for SORT files or for database snapshot pages.

1.9.5.5 DDAL$ANAL_LOG_STATISTICS

At the end of an analysis report you can get statistics about the transactions in the RDB$CHANGES table. Example 1–1 shows summary statistics. Note that unless the entire contents of RDB$CHANGES are analyzed, the statistics that apply to RDB$CHANGES as a whole are not included. See the example in Section 1.9.8.1 to see such an abbreviated summary.

By default, statistics will appear at the end of the report.

You can enable or disable the logging of a summary by defining an OpenVMS logical name:

```
$ DEFINE   DDAL$ANAL_LOG_STATISTICS   x
```

Alternatively, enter a definition in the parameter file:

```
DDAL$ANAL_LOG_STATISTICS   x
```

The letter x stands for the keyword YES or NO. If you enter the word YES, a summary of statistics will appear at the end of the report. If you enter the word NO, no summary will appear.

Also see the DDAL$ANAL_LOG_RESOURCE_USAGE parameter, which can override the output of statistical information.

1.9.5.6 DDAL$ANAL_LOG_TRANSACTIONS

The DDAL$ANAL_LOG_TRANSACTIONS parameter lets you choose whether or not to output information about RDB$CHANGES transactions and in what level of detail.

By default, the analyzer outputs all information available about transactions. You can enable or disable the logging of transactions by defining an OpenVMS logical name in one of two ways:

```
$ DEFINE   DDAL$ANAL_LOG_TRANSACTIONS   x
```

or

```
$ DEFINE   DDAL$ANAL_LOG_TRANSACTIONS   "a,b,c,d,e,f"
```

Alternatively, enter a definition in the parameter file:

```
DDAL$ANAL_LOG_TRANSACTIONS x
```

or

```
DDAL$ANAL_LOG_TRANSACTIONS a,b,c,d,e,f
```

Using the simple form, the letter x stands for one of the keywords, YES or NO. If you enter the word YES, all information about transactions will appear in the report. In such a case, the transaction changes will be output as logical records and data (see the RECS and DATA keywords) but not as physical records (see the ROWS keyword). If you enter the word NO, no information about transactions will appear, except when an error is detected.

You can use an alternate form, shown above as the list a,b,c,d,e,f, to choose which details you want in the report. The letters represent keywords, of which there are six. You are not required to list them all. Any keyword you omit is equivalent to saying keyword=NO. Separate keywords with a comma, and do not include spaces unless you enclose the list in quotation marks (" ").

Table 1–3 DDAL$ANAL_LOG_TRANSACTIONS Keywords

Keyword	Meaning
ID	Identification of the transaction (TSER and TID/TSN)
ROWS	Physical rows (raw, un-interpreted data)
RECS	Types of logical record (e.g., INSERT, DELETE)
DATA	Logical record data
COUNT	Number of rows in RDB$CHANGES for the transaction
DBKEYS	A list of sequence number, database key pairs for each row in RDB$CHANGES for the transaction

Some details make no sense in the absence of others. If ID is not printed, then neither will any of the other transaction details. The transaction ID will be printed when a transaction error is detected, even if you have said ID=NO. That means that COUNT and DBKEYS, if set to YES, will print in the event a transaction error is detected. ROWS, RECS, and DATA will not. If RECS are not printed, then neither will the data within the logical records. Changed data can be output in one of two ways: (1) as raw, un-interpreted data (ROWS) or (2) as logical records (RECS) and (optionally) logical record data (DATA). If you happen to list both keywords, the last one in the list overrides the other.

Example 1–5 Logging Transactions with Minimal Detail

(continued on next page)

Example 1–5 (Cont.) Logging Transactions with Minimal Detail

```
DDAL$ANAL_LOG_TRANSACTIONS      ID,COUNT
```

For each transaction list the transaction identifiers and a count of the number of rows.

Example 1–6 Logging Transactions with Moderate Detail

```
DDAL$ANAL_LOG_TRANSACTIONS      Id=Yes,Recs=Yes,Data=No,Count=No,Dbkeys=Yes
```

For each transaction list the transaction identifiers, the logical record types, and the pairs of sequence numbers and database keys.

Also see the DDAL$ANAL_LOG_RESOURCE_USAGE parameter, which can override the output of transaction information.

1.9.5.7 DDAL$ANAL_ROWS_PER_TRANSACTION

The analyzer can give you an estimate of the number of disk blocks that would be needed to generate a particular report (see DDAL$ANAL_LOG_ RESOURCE_USAGE). To perform the calculation the analyzer needs to know the average number of RDB$CHANGES rows per RDB$CHANGES transaction in the report.

By default, the analyzer uses a built-in value of 1.5 for the average. This average does not necessarily reflect the number for your particular database, but it is typical of what other customers see. You can override the built-in value by defining the DDAL$ANAL_ROWS _PER_TRANSACTION parameter.

If you produce the report in a disk file and include summary statistics in the report, the summary has a line that shows "Average number of rows per transaction." Use that number as the value for the DDAL$ANAL_ROWS _ PER_TRANSACTION parameter.

Use the following format to define the parameter as an OpenVMS logical name:

```
$ DEFINE   DDAL$ANAL_ROWS_PER_TRANSACTION   m.n
```

Alternatively, enter a definition in the parameter file:

```
DDAL$ANAL_ROWS_PER_TRANSACTION m.n
```

Here `m.n` represents an unsigned decimal number such as 2.14. Up to six fractional digits are accepted, though two digits should suffice for most needs.

Also see the DDAL$ANAL_DISK_BLOCKS_PER_TRANSACTION and DDAL$ANAL_TSER_DENSITY parameters. They can also affect calculation of disk block usage.

1.9.5.8 DDAL$ANAL_STARTING_TSER

The parameters DDAL$ANAL_STARTING_TSER, DDAL$ANAL_ENDING_TSER, DDAL$ANAL_TRANSACTION_LIMIT and DDAL$ANAL_ERROR_LIMIT all serve to restrict the number of transactions to be analyzed. When you define DDAL$ANAL_STARTING_TSER you indicate that analysis should begin with a transaction whose TSER number is equal to or greater than the specified value. It is not required that a transaction with the given number exist.

By default the analyzer processes all transactions.

You can define the starting TSER value in an OpenVMS logical name:

```
$ DEFINE   DDAL$ANAL_STARTING_TSER   n
```

Alternatively, enter a definition in the parameter file:

```
DDAL$ANAL_STARTING_TSER n
```

Here n represents an unsigned decimal integer from 1 to 4294967295.

1.9.5.9 DDAL$ANAL_TRANSACTION_LIMIT

The parameters DDAL$ANAL_STARTING_TSER, DDAL$ANAL_ENDING_TSER, DDAL$ANAL_TRANSACTION_LIMIT and DDAL$ANAL_ERROR_LIMIT all serve to restrict the number of transactions to be analyzed. When you define DDAL$ANAL_TRANSACTION_LIMIT you indicate that analysis should stop after the specified number of transactions have been analyzed. This does not include transactions that are skipped because their TSER numbers are less than the starting TSER number in DDAL$ANAL_STARTING_TSER.

By default the analyzer processes all transactions.

You can define the transaction limit in an OpenVMS logical name:

```
$ DEFINE   DDAL$ANAL_TRANSACTION_LIMIT   n
```

Alternatively, enter a definition in the parameter file:

```
DDAL$ANAL_TRANSACTION_LIMIT n
```

Here n represents an unsigned decimal integer from 1 to 4294967295.

Also see the DDAL$ANAL_ERROR_LIMIT and DDAL$ANAL_ENDING_TSER parameters. They contain other reasons that the analysis stops before processing all transactions.

1.9.5.10 DDAL$ANAL_TSER_DENSITY

The analyzer can give you an estimate of the number of disk blocks that would be needed to generate a particular report (see DDAL$ANAL_LOG_ RESOURCE_USAGE). To perform the calculation the analyzer needs to know the average density of TSER numbers used. If, out of every 10 possible TSER numbers, only nine are assigned to transactions, the TSER density is 0.9.

By default, the analyzer uses a built-in value of 0.9 for the average. This average does not necessarily reflect the number for your particular database, but it is typical of what other customers see. You can override the built-in value by defining the DDAL$ANAL_TSER_DENSITY parameter.

If you produce the report in a disk file and include summary statistics in the report, the summary has a line that shows "Average TSER density." Use that as the value for the DDAL$ANAL_TSER_DENSITY parameter.

Use the following format to define the parameter as an OpenVMS logical name:

```
$ DEFINE   DDAL$ANAL_TSER_DENSITY   m.n
```

Alternatively, enter a definition in the parameter file:

```
DDAL$ANAL_TSER_DENSITY  m.n
```

Here `m.n` represents an unsigned decimal number such as 0.86. Up to six fractional digits are accepted, though two digits should suffice for most needs. The density must be greater than 0.0 but not greater than 1.0.

Also see the DDAL$ANAL_DISK_BLOCKS_PER_TRANSACTION parameter, which affects calculation of disk block usage. See DDAL$ANAL_ROWS_PER_ TRANSACTION as well.

1.9.6 Types of Error Detected

DDAL$ANALYZE checks for a variety of possible data corruption errors in the RDB$CHANGES table. Checks are performed only for the range of transactions chosen for analysis. Listed below are the errors that are detected and the error codes reported.

Table 1–4 Data Corruption Error Codes Returned by DDAL$ANALYZE

Error Code	Classification	Meaning
DDAL$_AFTERCOMMIT	Illegal Value	Transaction does not end at the commit record; more data follows

(continued on next page)

Table 1–4 (Cont.) Data Corruption Error Codes Returned by DDAL$ANALYZE

Error Code	Classification	Meaning
DDAL$_DUPL_SEQ	Duplicate Value	Same sequence number is assigned to two or more rows in the same transaction
DDAL$_DUPL_TID	Duplicate Value	Same TID/TSN is assigned to two or more transactions
DDAL$_DUPL_TSER	Duplicate Value	Same TSER is assigned to two or more transactions
DDAL$_ERRLOGSIZ	Illegal Value	Error in logical record size
DDAL$_INVJOUVER	Illegal Value	Invalid journal version number
DDAL$_INVLOGCLA	Illegal Value	Invalid logical record class code
DDAL$_INVLOGREC	Illegal Value	Logical record length could not be computed
DDAL$_INVLOGTYP	Illegal Value	Invalid logical record type code
DDAL$_NOCOMMIT	Missing Value	Transaction is missing a commit record
DDAL$_NONZERO_ TSER	Illegal Value	A non-zero TSER value was found on a row which is not the final row for the transaction (the row with the highest sequence number)
DDAL$_PREMATEOT	Missing Value	End of transaction seen too soon
DDAL$_SEQ_MISSING	Missing Value	Intermediate rows (and therefore sequence numbers) are missing
DDAL$_SEQ1_ MISSING	Missing Value	Row with sequence number 1 is missing
DDAL$_TID_NEAR_ MAX	Illegal Value	TID value is nearing its maximum
DDAL$_TSER_NEAR_ MAX	Illegal Value	TSER value is nearing its maximum
DDAL$_ZERO_TID	Illegal Value	TID value is zero
DDAL$_ZERO_TSER	Illegal Value Missing Value	TSER value is zero, equivalent to saying final row for a transaction is missing

Note

The error code DDAL$_INVLOGTRN is being phased out and replaced by the two codes DDAL$_NOCOMMIT and DDAL$_AFTERCOMMIT. See Section 1.8 for more information.

DDAL$ANALYZE does not detect all errors that might possibly be received during transfer execution. For example, if a transaction in RDB$CHANGES contains an insert of a row with a database key that already exists in the target database, DDAL$ANALYZE will not report such an error. DDAL$ANALYZE does not access the target databases.

1.9.7 What to Do When Errors Are Detected

When data corruption errors are detected, such as those listed in the preceding section, report them to Oracle Corporation using the standard error reporting mechanism. When doing so:

1. Provide the version of Replication Option being used.
2. Show the version of Rdb being used to access the source, transfer, and target databases.
3. Provide a copy of the transfer log file (the log file named in the transfer definition).
4. List the TSER number(s) stored in the target database(s) for the failing transfer(s). See the example in Section 1.9.8.1 for a sample of how to query the target database(s).
5. Include an analyzer report for the failing RDB$CHANGES transaction, as shown in Section 1.9.8.2, but with DDAL$ANAL_LOG_TRANSACTIONS defined to be "YES", so that full detail is produced in the report.

   Also, define DDAL$ANAL_STARTING_TSER to be the TSER number obtained from the preceding step, and define DDAL$ANAL_ TRANSACTION_LIMIT to be "3". This will output the transaction preceding the one in error, the failing transaction, and the transaction that follows it.
6. Conclude with an analyzer report for the failing RDB$CHANGES transaction, as in the preceding step, but with DDAL$ANAL_LOG_ TRANSACTIONS defined to be "ID,ROWS", so that the data is output in un-interpreted form.

Such data corruption errors affect all replication transfers that use the same source database. To recover from such an error you should contact Oracle Corporation Technical Support. In some situations it might be possible for you to manually correct the problem so that transfers can continue. If you cannot afford the time to do this, the standard recovery method is to reinitialize all the replication transfers using that source database and re-execute each transfer. Once a transfer has gone through this process, the following update transfer will skip over the previous bad transaction(s) in RDB$CHANGES and continue

with changes that were written to RDB$CHANGES after the reinitialized transfer was executed.

1.9.8 Additional Examples of Using DDAL$ANALYZE

The following sections contain more examples of how to use DDAL$ANALYZE. With each example are shown the contents of the parameter file.

1.9.8.1 Search for the First Transaction Error Only

If you are interested in just the first error in the RDB$CHANGES table, you can limit your search and reduce processing time. When several update transfers have failed, each for the same reason, attach to each target database and perform the appropriate one of the following two queries:

If the target is an Oracle Rdb database:

```
SQL> SELECT RDB$VINTAGE_TSER FROM RDB$VINTAGE
cont> WHERE RDBVMS$TRANSFER_NAME = 'name';
```

For any other target database:

```
SQL> SELECT DDAL$TRANSFER_TSER FROM DDAL$TRANSFER_INFO
cont> WHERE DDAL$TRANSFER_NAME = 'name';
```

If each transfer failed because of the same corruption problem in RDB$CHANGES, they will all have the same TSER number shown from these SQL queries. These TSER numbers represent the last transaction from RDB$CHANGES that was successfully applied to the target databases.

Next, prepare a parameter file with the following definitions:

```
DDAL$ANAL_STARTING_TSER         641106
DDAL$ANAL_TRANSACTION_LIMIT     1
DDAL$ANAL_LOG_RESOURCE_USAGE    NO
DDAL$ANAL_LOG_TRANSACTIONS      DBKEYS
DDAL$ANAL_LOG_STATISTICS        NO
```

The last successful transaction had TSER number 641105, for example. Therefore you start the scan looking for a TSER of 641106 or higher, limiting the search to the first transaction that is found. It is not necessary that transaction 641106 exist. Gaps often occur in the range of assigned TSER numbers. This will find the transaction that is numerically the next higher one in RDB$CHANGES.

```
Replication Option for Rdb V7.0-1        3-SEP-1997 11:42:23.80

Analysis of the RDB$CHANGES table of replicated source database:
DISK020:[ADMIN]HQ_PROD_DB.RDB

*****************************************************************
```

```
Transaction: TSER = 641106  TID/TSN = 1962487

%DDAL-E-AFTERCOMMIT, commit record not end of transaction

         1 - 146:40224:0

*****************************************************************

ERRORS REPORTED:  1         3-SEP-1997 12:00:54.06
```

Depending on where the error occurs within the transaction you might see part of the transaction activity logged (such as INSERT, UPDATE, DELETE operations). In the example above, the error occurs at the beginning of the transaction, so no other data will be logged.

1.9.8.2 Report with Intermediate Level of Detail

Sometimes it is useful to analyze a range of transactions to see what types of operations (INSERT, UPDATE, DELETE) are being performed and how often. To produce such a report, prepare a parameter file with the following definitions:

```
DDAL$ANAL_LOG_RESOURCE_USAGE    NO
DDAL$ANAL_LOG_TRANSACTIONS      Id=Yes,Recs=Yes,Data=No
DDAL$ANAL_LOG_STATISTICS        YES
DDAL$ANAL_STARTING_TSER         641000
DDAL$ANAL_ENDING_TSER           641500
```

This allows you to see the types of database activity without being overwhelmed with details.

```
Replication Option for Rdb V7.0-1         3-SEP-1997 13:36:20.93

Analysis of the RDB$CHANGES table of replicated source database:
DISK020:[ADMIN]HQ_PROD_DB.RDB

*****************************************************************

Transaction: TSER = 641000  TID/TSN = 1962306  format version = 2

Logical record type = INSERT     length = 81

    Source table name = COMP_RATE
    Source row database key = 343:12195:11 (015700002FA3000B)
    After image: omitted

Logical record type = END_INSERT

Logical record type = INSERT     length = 81

    Source table name = COMP_RATE
    Source row database key = 343:12195:12 (015700002FA3000C)
    After image: omitted

Logical record type = END_INSERT

Logical record type = COMMIT    Commit time: 3-SEP-1997 10:23:34.18
```

```
*********************************************************************

... other transactions, up through and including the one with TSER = 641497

*********************************************************************

SUMMARY STATISTICS:

    Lowest  TSER analyzed ................... = 641000
    Highest TSER analyzed ................... = 641497
    Number of transactions analyzed ......... = 465
    Number of rows analyzed ................. = 922
    Most rows for a transaction ............. = 81

    Average number of rows per transaction .. = 1.98
    Average TSER density .................... = 0.93

ERRORS REPORTED:  0         3-SEP-1997 13:46:18.49
```

1.9.8.3 Resource Usage Estimate Report

If you plan to analyze a large number of transactions, the resultant size of the report could be quite large, taking up perhaps more than a million disk blocks. Will you have enough room on disk to accommodate the report? The first step should be to ask for a resource usage estimate. Such an estimate is made without actually reading through the transactions in the RDB$CHANGES table, so it only takes a few seconds to do.

To produce such a report, prepare a parameter file with the following definitions:

```
DDAL$ANAL_LOG_RESOURCE_USAGE    ONLY
DDAL$ANAL_LOG_TRANSACTIONS      ID,RECS,COUNT
```

The keyword ONLY tells the analyzer to take the other parameters into account when making the estimate but to not actually do an analysis. In the parameter file shown above, if the report were to be performed, all transactions in the RDB$CHANGES table would be analyzed (the default because you have not specified a range of transactions). For each transaction, the transaction identifier, the logical record types, and the number of rows in RDB$CHANGES would be shown. The transaction row count and the database keys would not be listed. The following are the resource usage estimates that would be given:

```
Replication Option for Rdb V7.0-1       20-AUG-1997 11:13:48.84

Analysis of the RDB$CHANGES table of replicated source database:
DISK020:[ADMIN]HQ_PROD_DB.RDB

RESOURCE USAGE ESTIMATES:

    Estimated RDB$CHANGES rows to be analyzed = 103070
    Estimated disk blocks for the report .... = 614050
```

1.9.8.4 Fine Tuning the Resource Estimates

The resource estimates shown in the example in Section 1.9.8.3 are exactly that—estimates. The actual values depend greatly on your particular database and how it is typically used. The estimates tend to be more accurate when used for a large range of transactions. After the first time you produce a report as a disk file and include summary statistics at the end of the report, there is information in the summary that will let you improve the accuracy of any subsequent resource estimate for the same type of report.

Here, for instance, is a typical summary:

```
SUMMARY STATISTICS:

    Number of transactions in RDB$CHANGES ... = 100011
    Number of rows in RDB$CHANGES ........... = 154533

    Lowest  TSER analyzed ................... = 616863
    Highest TSER analyzed ................... = 721169
    Number of transactions analyzed ......... = 100011
    Number of rows analyzed ................. = 154533
    Most rows for a transaction ............. = 2424

    Average number of rows per transaction .. = 1.54
    Average TSER density .................... = 0.95
```

The "Average number of rows per transaction" and the "Average TSER density" can be directly added to your parameter file, shown below. You must calculate the average number of disk blocks per transaction. First, use the OpenVMS DIRECTORY command to determine the number of disk blocks used for the report (disk blocks actually used, not blocks allocated). Divide that number by the "Number of transactions analyzed" from the summary. Add all three statistical values to your parameter file as follows:

```
DDAL$ANAL_DISK_BLOCKS_PER_TRANSACTION    3.95
DDAL$ANAL_ROWS_PER_TRANSACTION           1.54
DDAL$ANAL_TSER_DENSITY                   0.95
```

These statements are in addition to those you already had in your parameter file. The next time you ask for a resource estimate for the same report, the numbers will be based on these parameters rather than on the built-in values.

1.9.8.5 Summary Report with a Scan for Errors

The example shown in this section is a repeat of Example 1–1 with additional detail. Here you scan all the transactions in RDB$CHANGES looking for errors. Except for the summary at the end of the report, nothing is output unless an error is detected. If an error is found, the transaction ID is printed along with the error message and the database keys and row sequence numbers for each row in RDB$CHANGES for the bad transaction.

To produce such a report, prepare the following parameter file:

```
DDAL$ANAL_LOG_RESOURCE_USAGE    NO
DDAL$ANAL_LOG_TRANSACTIONS      DBKEYS
DDAL$ANAL_LOG_STATISTICS        YES
```

Note that for the DDAL$ANAL_LOG_TRANSACTIONS parameter, you did not list the ID keyword. Therefore the transaction ID will normally not be output. When the transaction ID is not output, neither will any of the other transaction details. Therefore, for transactions where no error is detected, nothing appears in the report. If an error is detected, the transaction ID is output first, even when you ask for the ID to not be shown. Since the ID is output when there is an error, so will the transaction sequence numbers and database keys since you asked to see them. The same would apply to the transaction row count, had you asked to see one; but logical/physical record data would not be output. By default, a summary will be included. See Example 1–1 for a listing of the report that would be produced.

2
Problems Corrected

The following sections describe problems that have been corrected in release 7.2.

2.1 Computed Column Restriction Lifted for CREATE TRANSFER

Until the release of Rdb 7.0.2, SQL had imposed a restriction on the definitions of computed columns used in CREATE TRANSFER statements. The computed column definitions were not permitted to refer to domain names. If such column definitions were encountered, SQL would issue a warning message, "SQL$_CMPBYWNRL, Invalid computed field <column-name> will not be transferred from relation <table-name>," and would have removed that column from the list of those to be transferred. That restriction was removed from SQL version 7.0.2.

Removal of this restriction in SQL does not completely solve the problem. If you attempt to create and execute a transfer without taking preparatory steps (see workaround below), execution of the transfer will fail. The Replication Option is not prepared to transfer the definitions of domains referenced only within computed columns.

The following example shows domain and table definitions and a CREATE TRANSFER statement which would have resulted in the SQL$_CMPBYWNRL warning message from SQL.

```
$ SQL

ATTACH 'FILE DISK:[DIR]SOURCE.RDB';

! Create a table with two columns, one of which is computed and whose
! definition references the name of a domain.

CREATE DOMAIN DOM1 SMALLINT;
```

```
CREATE TABLE TAB1 (
  COL1 INTEGER,
  COL2 COMPUTED BY CASE (SUBSTRING (CAST (COL1 AS CHAR(4))
                         FROM 1 FOR 2) AS DOM1));
COMMIT;

! Prior to lifting the restriction in SQL, the following transfer
! definition would have resulted in a SQL warning message:
! %SQL-W-COMBYWNRL, Invalid computed field COL2 will not be
! transferred from relation TAB1.

CREATE TRANSFER COMPUTED_DOMAIN_REF TYPE IS EXTRACTION
  MOVE TABLES TAB1
  TO EXISTING FILENAME DISK:[DIR]TARGET.RDB
  LOGFILE IS DISK:[DIR]COMPUTED_DOMAIN_REF.LOG;
```

To successfully perform this transfer, use the following workaround:

In the preceding example, the transfer definition resulting from the CREATE TRANSFER statement would include the COL2 column to be transferred. Since the DOM1 domain is only referenced within the definition of COL2, a computed column, the Replication Option does not recreate that DOM1 definition in the target database. Therefore, prior to the first execution of the transfer, you must add the DOM1 definition to the target database yourself, using a CREATE DOMAIN statement as shown in the preceding example. If you forget to do so, transfer execution will fail. A message in the transfer's log file will indicate -RDMS-F-FLDNOEXI, field COL2 does not exist in this database.

2.2 Purging RDB$CHANGES Table When Source File Extension Is Missing

It is possible to create a transfer definition where the filename for the Rdb source database is missing a filename extension. Because SQL normally supplies the missing filename extension, .RDB, it is common to attach to the database without specifying the .RDB extension. However, you can direct SQL to use the filename unchanged when a CREATE TRANSFER statement is issued. For example:

```
$ SQL

SQL> ATTACH 'FILENAME DISK2:[ADMIN]PERSONNEL';

SQL> SET LOGICAL_NAME TRANSLATION FOR TRANSFER OFF;

SQL> CREATE TRANSFER
   .
   .
   .
```

The SET LOGICAL_NAME OFF statement instructs SQL to leave the filenames in the CREATE TRANSFER statement and the source database filename from the ATTACH statement untranslated within the stored transfer definition. Logical names are not replaced in the file specifications with their equivalent text strings, and missing portions of the file specifications, such as the filename extension, are not supplied.

When this situation existed in earlier releases of the Replication Option, the initial transfer would be performed without error, but the replication update transfer would fail. The failure would occur after the changes had been transferred to the target database and just before rows were to be deleted (purged) from the RDB$CHANGES table. The following example shows what you would see in the transfer's log file:

```
12:58:15  %DDAL-I-LOATABDAT, loading data for table RDB$TRANSFERS
----- 11-SEP-1998 12:58:16.18 ----- Error           ---------------
%DDAL-E-RMSERR, error occurred using RMS PARSE or SEARCH on the file
DISK2:[ADMIN]PERSONNEL
-RMS-E-FNF, file not found
```

This problem has been corrected in Replication Option for Rdb release 7.2.

2.3 Monthly Transfer Executed Too Frequently or Not at All

Starting in Replication Option release 7.0, transfer schedules that repeated once a month stopped working correctly. Such schedules are those containing the clause EVERY MONTH, for example:

```
CREATE SCHEDULE FOR MONTHLY_AUDIT
  START 1-JAN-1999 00:01:00
  EVERY MONTH;
```

Two problems were reported:

- When a schedule was created, the first execution of the transfer would occur at the correct start time. However, the transfer would then immediately re-execute. This process would continue apparently indefinitely.
- After the first execution of the transfer, the transfer did not run again. Output from a SHOW SCHEDULE statement showed that the next execution of the transfer was due to occur sometime in the year 9194.

Both of these scenarios are the result of the same error. When the Replication Option code was converted to use a common operating system interface rather than using OpenVMS-specific system services, we failed to notice a change in the format of the data structures used for the calls in one particular instance: the monthly schedule logic.

This problem has been corrected in Replication Option for Rdb release 7.2.

2.4 "%DDAL-E-SQLCODERR, status code -1" Error Messages

When Replication Option for Rdb executable images encountered errors accessing the transfer database, errors would be reported with an error message similar to the following:

```
%DDAL-E-SQLCODERR, status code  -1  received, indicating an error in
accessing the transfer database
```

This error message does not give any details about the cause of the error, such as a protection violation or wrong version of Oracle Rdb being used.

There is no workaround.

This problem has been corrected in Replication Option for Rdb release 7.2.

2.5 Replication Transfer Failure, %RDB-E-NO_DUP and %DDAL-E-BADDBKINS Messages

Occasionally, a replication transfer would fail because the uniqueness of an index was being violated. The index was one of those added to tables in the target database, such as the DDAL$DBKEY_INDEX1_1 index in the error message, below:

```
----- 24-JUN-2005 10:18:12.41 ----- Error           -------------------------
%DDAL-E-ERRACCDSTDB, error occurred accessing the destination database
-DDAL-E-SUPP, %RDB-E-NO_DUP, index field value already exists; duplicates not
 allowed for DDAL$DBKEY_INDEX1_1
```

This message would appear in the transfer's log file, the log file named in the CREATE TRANSFER statement.

Once such an error occurs, there are two ways to resolve the problem. One is to stop and reinitialize the transfer and then to restart transfer execution. Doing so can be time-consuming, because it necessitates rebuilding the transferred tables in the target database. Another method is to locate the row in the target table that has the duplicate index key value and to delete that row. After the row has been deleted from the target table, the transfer can be rerun without requiring that it be reinitialized. The target database does not have to be rebuilt.

The second method, though it is the one that causes the least disruption to operations, was difficult to implement because the error message, shown earlier, did not identify the offending row in the target database.

This impediment has been removed. In release 7.2, if the error situation arises, the reported error message will be similar to the one in this example:

```
----- 24-JUN-2005 12:19:59.57 ----- Error           -------------------------
%DDAL-E-BADDBKINS, insert in target table "REGIONS" failed
-DDAL-I-SRCDBKEY, source dbkey "002E0000022A0001 (46:554:1)" row owner "0"
-DDAL-I-TGTDBKEY, target dbkey "0031000002540001 (49:596:1)"
```

The target database key shown in parentheses can be used to locate the offending row in the target table.

The workaround is to stop and reinitialize the transfer and then restart transfer execution.

This problem has been corrected in Replication Option for Rdb release 7.2.

3

Known Problems, Restrictions, and Other Notes

This chapter contains information about known problems, restrictions, and other information related to Replication Option for Rdb. For problems and restrictions, workarounds (if available) are described.

3.1 Workaround for the DDAL$_AFTERCOMMIT Error

Under rare circumstances transfers might fail with the following error message in the transfer log file: DDAL$_AFTERCOMMIT. In the past this error appeared as DDAL$_INVLOGTRN. Examination of several databases revealed a common thread when this error occurred. For a reason yet to be understood, two commit records were being written for the transaction in the RDB$CHANGES table.

To recover from such an error without requiring that you reinitialize your transfers, define an event variable, DDAL$ALLOW_SECOND_COMMIT, and re-execute the transfer. Define the variable either as an OpenVMS logical name or as an event variable in a DDAL.RC file. See the *Replication Option for Rdb Handbook* for a description of the DDAL.RC file and event variables. The variable can alternatively be named DDAL_ALLOW_SECOND_COMMIT. The variable's definition must be either the word Yes or the word No, in upper, lower, or mixed case. If you do not define one of these variables, the default is to not allow the second commit.

This workaround was deliberately designed as an option with the default to have it disabled. For those of you who want and can afford to investigate the problem, this gives you notification that such an error has occurred. Once you define the variable, this problem will no longer cause transfers to fail. A warning message will appear in the transfer log files if the error occurs, but other than that the transfers will continue to completion.

If you are experiencing this problem, please report it to Oracle so that a way to reproduce the problem can be found and the problem corrected. It is possible that the error might result indirectly from database applications that get deadlock errors. If you have a way to confirm this, it may help solve the problem. Also, assuming that the problem is related to application deadlocks, what do your applications do after a deadlock is detected? Do they roll back the deadlocked transaction or do they commit? When an application encounters a deadlock situation, the correct behavior is to roll back the transaction. Refer to the *Oracle Rdb7 Guide to SQL Programming* for more information.

Note that there is a new diagnostic utility provided with Replication Option for Rdb called DDAL$ANALYZE. You can use it to locate the bad transaction in the RDB$CHANGES table and to display information about it and its surrounding transactions. Perhaps by examining this output, you can correlate it with application activity that occurred around the time of those transactions.

3.2 Snapshots Deferred Fails

A replication transfer of an Rdb database with snapshots deferred may fail with the following error:

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

This is a Replication Option for Rdb restriction. It is suggested that while transfers are executing that snapshots are immediate. The snapshot immediate setting can be enabled by the following SQL command:

```
ALTER DATA FILE data_file_name SNAPSHOT IS ENABLED IMMEDIATE;
```

If the database to be transferred has snapshots deferred enabled, this command must be executed before the transfer is executed.

3.3 Restrictions on Replication Option Release 7.0 Software

The following sections describe restrictions for Replication Option release 7.0 that still exist in release 7.2.

3.3.1 Restriction on the Use of Temporary Tables

Rdb allows the creation of temporary tables, tables whose data are automatically deleted when an SQL session ends. The tables only materialize when you refer to them in an SQL session and the data does not persist beyond the session. Rdb distinguishes among three types of temporary tables: global temporary tables, local temporary tables, and declared local temporary tables.

Global and local temporary table definitions are stored permanently in the database; declared local temporary table definitions are not.

You can create and execute a transfer that includes global temporary table definitions. The temporary table definitions are reproduced in the target database; however, their temporary nature is lost. That is, they become permanent tables in the target. This behavior may be corrected in future releases of Replication Option so that the temporary attribute is preserved. Therefore, do not rely on the fact that temporary tables are transformed into permanent tables during a transfer.

You can create a transfer that includes local temporary tables. However, execution of such a transfer will fail when Replication Option tries to read data from the source. Replication Option has not been modified to recognize the difference between permanent and temporary tables; therefore, it always tries to read data from the source tables. The error you see is this:

```
RDMS-E-BAD_CODE, corruption in the query string
```

The only way to avoid this problem is to not transfer local temporary tables. In particular, be careful when using the * notation in a transfer definition, such as, MOVE TABLES *. The * in this case means "transfer all tables", which would include any temporary table definitions you might have.

This restriction applies to all methods of transfer.

3.3.2 Limiting Database Transaction Modes May Cause Failures

Rdb has an option that permits a database administrator to limit transaction modes that can be used in a particular database. Limiting transaction modes in a database might cause Replication Option transfers to fail. For extraction transfers, a copy process uses read-only transactions in source databases and both read-only and read/write transactions in a target database. For replication transfers, in addition to those transaction modes, read/write transactions are performed in a source database.

This restriction applies to all methods of transfer.

3.3.3 Manual Deletion of .TMP Files Required

The temporary files created by SQL to receive output from SHOW commands are sometimes not being deleted after use. The file names appear in your home directory (SYS$LOGIN), with the names SQL$SHOW_TRANSF_xxxxxx.TMP (where the xxxxxx varies for each file). You should periodically delete these files to prevent unneccesary use of disk space.

3.4 Restrictions on Replication Option Version 6.0 Software

The following sections describe restrictions from Version 6.0 that still exist in release 7.2.

3.4.1 Restriction on CREATE TRANSFER Syntax

The syntax for the CREATE TRANSFER statement indicates that you have the option of specifying NO PROLOGUE, NO EPILOGUE, and/or, NO LOG in the transfer-file-options-clause.

If you wish to create a transfer without a prologue, epilogue, and/or a log file, do not specify those files in the CREATE TRANSFER statement. By default, none of these files is created unless you specify them.

Note, that you can specify NO PROLOGUE, NO EPILOGUE, and/or, NO LOG in the ALTER TRANSFER statement.

3.4.2 Restrictions for Extraction, Extraction Rollup, and Replication Transfers

This section documents the restrictions that apply equally to extraction, extraction rollup, and replication transfers.

3.4.2.1 External Functions Not Supported

Rdb has a capability known as external functions. In an Rdb database, you can create definitions of external functions and functions implemented by customer application programs. Within SQL, you can use these external functions anywhere that a value expression is needed. For example, this might be in a COMPUTED BY column, in a CHECK clause, in a select expression within a view definition, and in a trigger definition.

Replication Option for Rdb cannot correctly transfer data definitions that contain external function references. If you want to transfer a table, but the table contains a COMPUTED BY column that references an external function, select all columns from the table except for the COMPUTED BY column. For example, if table HISTOGRAM(X, Y, Z, CB) contains a COMPUTED BY column, CB, that references an external function, do not use the following clause in the CREATE TRANSFER statement:

```
SELECT * FROM HISTOGRAM
```

Instead, use the following clause:

```
SELECT X, Y, Z from HISTOGRAM
```

3.4.2.2 Transfer Schedule Defined for Time Before Transfer Execution

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.

3.4.2.3 Transfer Completion Status After a Transfer Is Stopped

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.

3.4.2.4 Drop Replication Transfer Can Cause All of Replication Option to Hang

A situation exists in which it is possible for the deletion of a replication transfer definition to cause all of Replication Option to freeze on a given processor. A practical workaround for this problem follows.

The Replication Option transfer monitor is designed to handle a single replication DROP TRANSFER command at a time. While it is doing so, the transfer monitor does not service any other transfers that are happening, or whose schedules come due. The transfer monitor creates a copy process to perform deletion of the transfer definition from the source database, and waits for the copy process execution to complete.

If the transfer definition names an epilogue procedure, that procedure is executed once the copy process image exits after having performed (or failed to perform) the deletion of the transfer definition from the source database. If your epilogue procedure includes any Replication Option command, the epilogue procedure will hang after issuing that command. There are twelve Replication Option commands:

- ALTER SCHEDULE
- ALTER TRANSFER
- CREATE SCHEDULE
- CREATE TRANSFER
- DROP SCHEDULE
- DROP TRANSFER
- GRANT transfer-privs
- REINITIALIZE TRANSFER
- REVOKE transfer-privs

- SHOW TRANSFER
- START TRANSFER
- STOP TRANSFER

To prevent the freezing of Replication Option operation, design your replication epilogue procedures to test whether or not a drop transfer is being performed.

Check the definition of the DCL symbol DDAL$CP_ACTION. When a replication deletion is being performed, the copy process log file will contain the following line:

```
$ DDAL$CP_ACTION :== D
```

For example, if you wish to start some other transfer from within your epilogue procedure, include code similar to the following:

```
$ IF DDAL$CP_ACTION .NES. "D"
$ THEN
$       SQL := $SQL$
$       SQL START TRANSFER Daily_Invoices NOW NO WAIT;
$ ENDIF
```

You cannot use a SHOW TRANSFER (STATUS) command in an epilogue procedure to find out the status of the transfer being executed. This is because the new status is not recorded in the transfer database until the process executing the transfer terminates. Instead, your epilogue procedure should test the value of the DDAL$CP_CONTINUE environment variable. Examples of how to do so are in the *Replication Option for Rdb Handbook* in the chapter titled “Using Command Procedures with Transfers on OpenVMS”.

In general, you cannot execute Replication Option statements from within an epilogue procedure if the statements refer to the transfer being executed. If you need to do so, you must wait for the transfer monitor to record the status in the transfer database. For example, have your epilogue procedure submit a batch job with a suitable time delay before execution to allow for the transfer to complete, as shown in the following example:

```
$ SUBMIT/AFTER="+00:05" my-command-file
```

3.4.2.5 Restrictions on the CHECKPOINT Clause

The CHECKPOINT clause contains the following two optional subclauses:

- EVERY n MINUTES
- RECOVER WITHIN n MINUTES

The following section describes restrictions that apply to the EVERY n MINUTES subclause.

3.4.2.5.1 Restrictions on the EVERY n MINUTES Subclause

Although the EVERY n MINUTES clause is accepted by SQL as syntactically legal, it is ignored by Replication Option. It was our intent to use this transfer attribute to govern how often transactions applied at the target database would be committed. For tables with many data rows, which might take, for example, an hour or more to transfer, the intent was to transfer the data for the table in several transactions rather than one. In the event of a network failure during a transfer, recovery would not have to start at the beginning of the table. This CHECKPOINT EVERY n subclause of the checkpoint recovery is not implemented in Replication Option release 7.2.

In an extraction (or initial replication) transfer, data for a given target table are transferred within a single transaction. If a failure occurs while transferring data into a given table, that transaction will be effectively rolled back. When checkpoint recovery is employed and a new transaction is started, all the data for that table must be resent.

3.4.3 Restrictions for Extraction Rollup Transfers Only

This section describes restrictions that apply to extraction rollup transfers only.

3.4.3.1 Column Names for Like Tables Must Be Identical

A restriction that applies to extraction rollup transfers created using SQL states that column names of like tables must be identical.

SQL lets you create an extraction rollup transfer of like tables that differ only in the column names. For example, the following fragment shows a transfer definition of this type:

```
MOVE TABLES
    SELECT PLAYER_ID FROM DB1.PLAYERS
        UNION ALL
    SELECT PLAYER_CODE FROM DB2.PLAYERS
        INTO ALL_PLAYERS
```

The PLAYERS tables in the two source databases are identical in the following ways:

- The number of columns is the same.
- The columns are defined in the same order.
- The data definitions for each column (data type, length, scale, precision, and subtype) are the same.

They differ only in the two column names they each use, namely PLAYER_ID and PLAYER_CODE.

If you create such a transfer, the definition is accepted by SQL and stored in the transfer database. When a Replication Option copy process executes the transfer, the transfer fails. The following error messages will be written to the copy process log file.

- DDAL-E-RDBERR, RDB-E-OBSOLETE_METADATA
- RDMS-F-BAD_SYM

The messages indicate that the PLAYER_CODE column name caused the error.

Extraction rollup transfers must use identical column names for transfers to succeed.

3.4.3.2 SQL Privileges Can Restrict Access to Source Tables

Users with only the SQL SELECT privilege for source databases cannot execute extraction rollup transfers. The following example illustrates this restriction. Assume you want to define an extraction rollup transfer on two source databases, DB1 and DB2. Assume that you have the SELECT privilege on database DB2 and assume that you have the SELECT privilege for all the tables in DB2. If you enter the following transfer definition, you will receive the error message shown at the end of the example:

```
SQL> ATTACH 'ALIAS DB1 FILENAME PERS1';
SQL> ATTACH 'ALIAS DB2 FILENAME PERS2';
SQL> CREATE TRANSFER ERP_1 TYPE IS EXTRACTION ROLLUP
cont>   MOVE TABLES
cont>      SELECT FIRST.*
cont>      FROM DB1.EMPLOYEES FIRST WHERE FIRST.STATE = 'NH'
cont>   UNION ALL
cont>      SELECT SECOND.*
cont>      FROM DB2.EMPLOYEES SECOND WHERE SECOND.STATE = 'MA'
cont>   INTO NEW_MA_NH_EMPS
cont>   TO NEW FILENAME DISK1:[DIR1]PERS3
cont>   LOG FILE IS DISK1:[DIR1]PERS3.LOG
cont>   COMMENT IS 'Pull test all local'
cont> ;
%RDB-E-NO_PRIV, privilege denied by database facility
```

This is an SQL-enforced restriction.

You can use the RDO interface to work around this problem. Or you can use the shared-write feature with SQL extraction transfers to accomplish the equivalent of an extraction rollup transfer, but you will need to use multiple transfers to write the data to the target database.

3.4.4 Restrictions for Replication Transfers Only

This section contains restrictions for replication transfers only.

3.4.4.1 Restriction on Fields Named in an Rdb RDO RSE

All fields that you name in the record selection expression (RSE) of the DEFINE TRANSFER statement must also be included in the SELECT FIELDS clause. If you do not name the fields explicitly in the SELECT FIELDS select-field-name statement or implicitly in the select-field-name clause, Replication Option returns an error message when the transfer executes. The message indicates that the field name is not found in the symbol table.

3.4.4.2 Restriction on Columns Named in an SQL Select Expression

All columns that you name in a select expression of the CREATE TRANSFER statement must also be included in the SELECT column-name clause. If you do not name the columns explicitly in the SELECT column-name statement or implicitly in the SELECT ALL clause, Replication Option returns an error message when the transfer executes. The message indicates that the column name is not found in the symbol table.

3.4.4.3 Restriction on Wildcards in Select Expressions

When you port application programs compiled against a source database, wildcards in select expressions that access columns from a target replication database may generate the RDMS-F-BAD_SYM message.

Replication Option adds an extra column, DDAL$DBKEY, to all user tables that are transferred to a target replication database. If the application program was compiled against the target database or against a data dictionary record definition based on the target table's definition, wildcard queries will run correctly. However, if the application was developed against the source database and simply copied to the target site, a SELECT * FROM statement in a program fails. The number of columns in the source and target database table definitions is not consistent.

A program that makes reference to specific columns in a table will not generate this error, nor will interactive queries.

3.4.4.4 Restriction on COMPUTED BY Columns in a WHERE Clause

Do not use COMPUTED BY columns in a WHERE clause in SQL CREATE TRANSFER statements, or RDO DEFINE TRANSFER statements in replication transfers. Although you can transfer computed columns, you cannot select on a computed column. Rdb will not give you an error message if you try to execute one of these statements at run time. Although these statements will not work, Rdb will not inform you they did not work.

3.4.4.5 Dropping Transfer Definitions

When a replication transfer definition is created, Replication Option stores that definition in the transfer database. When the transfer runs for the first time, Replication Option also stores that definition in the source database.

When you ask to drop a replication transfer definition, Replication Option runs a copy process to delete the definition in the source database. While doing so, the copy process must have exclusive access to that database.

Because the copy process needs to have exclusive access to the source database, you must disconnect from the source database before you issue the DROP TRANSFER statement for a replication transfer.

If the source database is inaccessible or no longer exists and you need to drop the transfer, you can work around the problem as follows:

1. Create a new database with the same name and columns or tables as the inaccessible source database.
2. Locate this new database in the directory where the old one was.
3. Reinitialize and then start the transfer.

By following these steps, you will enter the transfer definition in the new database.

After the definition has been entered into the RDB$TRANSFER_RELATIONS table in the source database, stop the transfer. Then issue a DROP TRANSFER statement in SQL or a DELETE TRANSFER statement in RDO to remove the unwanted transfer definition.

3.4.4.6 RMU Copy_Database Propagates Replication Option Tables

The RMU Copy_Database command copies an Rdb database without weeding out Replication Option tables that may be present. This can be significant if the original database is the source of a replication transfer.

When a replication transfer is created and then executed, four system tables are added to the source database to help manage the replication process. The four tables are:

- RDB$TRANSFERS
- RDB$TRANSFER_RELATIONS
- RDB$CHANGES
- RDB$CHANGES_MAX_TSER

Using the RMU Copy_Database command on a replicated source database will produce another database that includes these four tables and their contents. Later as you use the database copy, you may encounter two side effects that you might not anticipate:

1. Data changes will be written to the RDB$CHANGES table just as they would in the original database. The existence of these four tables cues Rdb to do this, and information within them tells Rdb which changes to capture.
2. Rdb will prevent you from making data definition changes in the copied database if such changes would affect existing transfers.

You can correct this situation, after making the database copy, in one of two ways:

1. Drop the four tables and any indexes associated with them, if present.
2. Erase the contents of each of the four tables.

This behavior is not limited to the use of the RMU Copy_Database command. Any utility you use that copies the database, such as the RMU Backup command or the OpenVMS COPY utility, will produce similar results. SQL and RDO EXPORT statements do not produce a copy of a database. The exported file does not include system tables. For this reason, this problem does not apply to the export/import of databases.

3.4.5 Restrictions for Transferring Into an Existing Database

Transfers that target existing databases have the potential for experiencing lock conflict, deadlock, and other data access errors. Such errors can arise when two or more applications vie for a common database resource. The following examples illustrate situations in which such errors can occur.

- Parallel execution of shared-write transfers

  When you enable an environment variable that permits shared writing of target tables, two or more transfers might be executed concurrently, each attempting to write to the same target tables. This would result in a conflict. Conflicts can also arise between transfers and local customer applications writing to these same tables.

- Changes to data definitions

  For some target systems, the Replication Option can drop and add indexes as necessary and can alter target table definitions to match source definitions. Doing so during the execution of a transfer might conflict with table usage by other transfers or by customer applications.

If possible, schedule the execution of individual transfers to occur when other transfers and applications are not accessing the intended target tables. Doing so avoids the conflicts.

If you must execute the transfers concurrently with other transfers and/or customer applications, there are two things you can do to recover after transfer failures due to lock conflicts and deadlocks.

1. Schedule transfer retries

   Transfer schedules have an option to retry failed transfers a certain number of times after a customer-defined delay. This is useful for all types of transfers, though it works better for replication updates. Replication updates do not have to repeat target transactions that have successfully committed. Other types of transfer, following a failure, must restart from the beginning.

2. Enable the CHECKPOINT option

   The CREATE TRANSFER statement has a CHECKPOINT option. In Replication Option release 7.2, the checkpoint option has been extended to more fully cover extraction and initial replication transfers and also to work for replication updates. With the checkpoint option enabled, a target error does not necessarily result in the termination of the transfer. Instead, the transfer becomes temporarily suspended and in a short while is retried. The retry begins with the transaction that failed; it is not necessary to start over from the beginning.

3.4.6 VMScluster Restrictions

This section documents the restrictions that apply to VMScluster installations of Replication Option.

3.4.6.1 Load Balancing in a VMScluster

When there are multiple transfers defined in a VMScluster transfer database, each node's transfer monitor is supposed to participate in the scheduling activity for transfers. However, sometimes one node does all the work.

For example, assume that three transfers have been defined, but none has a schedule definition. Transfer monitors are running on Nodes A, B, and C, and each is hibernating. If, on Node A, you create schedules for the three transfers, only the transfer monitor on Node A becomes aware of the new schedules. Therefore, all transfer operations are done by Node A.

If, on Node B, you issue a START TRANSFER statement from SQL or RDO, the transfer monitor on Node B subsequently participates in the scheduling of existing transfers. Similarly, you can issue a START TRANSFER statement from Node C to further balance the workload.

3.4.6.2 Losing a Cluster Node Can Cause Later Transfer Failure

Transfers to existing databases are designed to work when executed from any node in a VMScluster. This assumes that Replication Option is installed and running on each cluster node.

One of the characteristics of such transfers is that the ownership of transferred objects such as tables, domains, and indexes is recorded in the target database in the RDBVMS$TRANSFER_OWNER table (for Rdb targets) or in DDAL$TRANSFER_OWNR (for all other targets). The transfer that adds such objects to the database is the owner of the objects.

Transfers created in separate VMSclusters can have the same name. To distinguish one transfer from another when both target the same database, the name of the originating node is included in the ownership information.

A transfer to an existing database can be executed on any of a cluster's nodes, not just the node on which the transfer was first executed.

A problem can arise, however, if the node on which the transfer first executes is dropped from the cluster. Because that node is no longer recognized as a cluster member, the objects that once belonged to the transfer are no longer recognized as owned by the transfer on your cluster. That is, the objects appear to be owned by a transfer of the same name as yours but from a node in some other part of the network.

One solution is to restore the node to cluster membership. Another is to change the originating node name. To do so, first examine the rows in the RDB$VINTAGE or DDAL$TRANSFER_OWNR table of the target database. One of the columns will have the transfer name. The originating node name will be in the RDBVMS$VINTAGE_TRANSFER_NODE or DDAL$TRANSFER_OWNR column. Alter that data row by substituting the name of some other node that is still a part of the cluster for the name of the dropped node. Having done this, you should be able to start the transfer and have it complete successfully.

4

Documentation Updates and Corrections

This chapter contains corrections and clarifications to the documentation that was provided for Replication Option release 7.2.

4.1 Products no Longer Supported

The Replication Option for Rdb no longer supports the following products:

- Distributed Option for Rdb
- Rdb Transparent Gateways
- Oracle Rdbms as the manager for the transfer database

4.2 Creating an Rdb Transfer Database

The following changes apply to the *Replication Option for Rdb Handbook Release 7.0*, Section C.1.1.1 "Using Rdb as the Transfer Database" for Replication Option release 7.2.

Add the following text at the end of the first paragraph under the description for parameter P1—The disk and directory where the transfer database is to be created:

"In a mixed-architecture cluster, each hardware architecture has a different definition of the logical name SYS$COMMON. In that case, you must choose a location for the transfer database that is different from the default of SYS$COMMON:[SYSEXE]."

Replace the first sentence in the second paragraph under the description for parameter P2—Rdb version number as follows:

"If this parameter is omitted (null), the Oracle Rdb version defaults to the version defined in the SYSTEM logical name table. For versions of Replication Option prior to release 7.0, if there were no SYSTEM version defined for Oracle Rdb, the standard version of Oracle Rdb would be assumed (a version in which the product version number does not appear in the file names). For Replication

Option release 7.2 and the minimum version of Oracle Rdb it requires (7.2), there is no standard version of Oracle Rdb."

Replace the last sentence in the second paragraph under the description for parameter P2—Rdb version number as follows:

"Specifying the Oracle Rdb version number is important when you want to run under a version that is different from the version defined in the SYSTEM logical name table or when no such system version is defined."

Replace the last sentence in the third paragraph under the description for parameter P2—Rdb version number as follows:

"If it is still not specified, the default Oracle Rdb version as explained above is used."

4.3 Location in Which to Create a Transfer Database

In the *Replication Option for Rdb Handbook Release 7.0*, Section C.1.1.1 "Using Rdb as the Transfer Database" should be renamed "Using Rdb as the Transfer Database Manager". This section explains how to create a transfer database using the command procedure SYS$MANAGER:DDAL$CREATE_TR_DB.COM.

Add the following paragraph after the first two paragraphs that describe the optional parameter P1.

"The transfer database must be visible to processes on all nodes in an OpenVMS cluster. In a mixed-architecture cluster, each hardware architecture has a different definition of the logical name SYS$COMMON. In that case, you must choose a location for the transfer database different from the default of SYS$COMMON:[SYSEXE]."

4.4 Starting the Transfer Monitor

The following changes apply to the *Replication Option for Rdb Handbook Release 7.0*, Section C.1.2 "Starting the Transfer Monitor on OpenVMS" for Replication Option release 7.2.

Add the following sentence at the end of the paragraph under the second bullet "Required—if the transfer database is Oracle." under the description of parameter P1—Transfer Database attach string:

"As of version 7.2 of Replication Option, use of Oracle Rdbms as a repository for the transfer database tables is no longer supported."

Add the following as a third paragraph under the second bullet "Required—if the transfer database is Oracle." under the description of parameter P3—Transfer Database type and version number:

"As of version 7.2 of Replication Option, use of Oracle Rdbms as a repository for the transfer database tables is no longer supported."

Replace the first paragraph under the first bullet, Optional—if the transfer database is Rdb, under the description for parameter P3—Transfer database type and version number, as follows:

"If this parameter is omitted (is null), the transfer database type is defaulted to Rdb and the Rdb version is defaulted to the version defined in the SYSTEM logical name table (if such a system-wide version is defined)."

Replace the last sentence in the last paragraph under the first bullet, Optional—if the transfer database is Rdb, under the description for parameter P3—Transfer database type and version number, as follows:

"Specifying the Rdb version number is important when you want to run under a version that is different from the version defined in the SYSTEM logical name table or when no such system version is defined."

4.5 Account Used to Run DDAL$START_TR_MON.COM

In the Replication Option for Rdb Handbook Release 7.0, Section C.1.2 Starting the Transfer Monitor on OpenVMS explains how to use the command procedure SYS$STARTUP:DDAL$START_TR_ON.COM to run a transfer monitor. Replace the first paragraph in that section with the following:

"Replication Option provides the DDAL$START_TR_MON.COM command procedure to start the transfer monitor. You must execute this procedure from an OpenVMS account whose default device upon logging into the system is accessible from all nodes in the OpenVMS cluster. This insures that the run unit journal (RUJ) file for each transfer monitor process in the OpenVMS cluster is visible to all other cluster nodes in case a node should become disabled and transfer database recovery should become necessary from another node."

4.6 Clarification on Parameter P5 of DDAL$START_TR_MON.COM

The *Replication Option for Rdb Handbook* Release 7.0, Section C.1.2 Starting the Transfer Monitor on OpenVMS, explains parameters to be passed when invoking the monitor startup procedure, DDAL$START_TR_MON.COM. The explanation of parameter P5 log file to write on error needs to be changed to differentiate the use of the P5 and P2 log files. Parameter P2 names the transfer monitor log file, the file to which the transfer monitor logs messages during normal operation, once the monitor has successfully started running.

Parameter P5 names a startup log file, one to which error messages are logged during the execution of DDAL$START_TR_MON.COM.

4.7 Stall on Reinitialized Transfer

When a replication transfer is reinitialized and rerun, some of the steps include the dropping of indexes on the target tables, deletion of data in the tables, transfer of new data, and re-creation of the target indexes. If this is done at a time that other applications are using the same target tables, the transfers might stall waiting for those applications to disconnect from the target database. This is normal behavior for Oracle Rdb databases, and perhaps it is also seen with other target systems. This additional information applies to section C.7.4.2 "Transfer to an Existing Target Database" list item 9 "Drop Indexes" in the *Replication Option for Rdb Handbook* version 7.0.

It is suggested that while transfers are executing again after being reinitialized, normal target database access be suspended to permit the retransfer process to complete as quickly as possible. Note that while the target tables are being repopulated, the database is not in a consistent state. The dropping of data and insertion of data is done one table at a time, each in a separate transaction. Any user trying to read from these tables might find a given table empty.

4.8 References to the OpenVMS DETACH Privilege

The *Replication Option for Rdb Installation Guide* lists process privileges you need to successfully execute the Installation Verification Procedure. One of the privileges listed is DETACH. As of OpenVMS version 6.2, the DETACH privilege was replaced by IMPERSONATE. Later versions of OpenVMS accept the keyword DETACH as a synonym for the IMPERSONATE privilege.

4.9 Datatype Depends on Transfer Database Type

Appendix A of the *Replication Option for Rdb Handbook* lists each release 7.0 system table as having columns with datatype NUMBER(9) instead of INTEGER.

It is important to note that NUMBER(9) is for Oracle transfer databases only; for Oracle Rdb databases, the datatype is INTEGER.

The use of Oracle Rdbms to manage the transfer database tables is no longer supported in Replication Option release 7.2.

4.10 Errors in Transfer Database Table Descriptions

Appendix A in the *Replication Option for Rdb Handbook* contains several errors pertaining to data types and sizes and column names.

- In Table A-10 the Data Type/Size of the COMMENT_SEG table is shown to be VARCHAR(512). This is true only for an Oracle Rdbms database. If the database is an Rdb database, the datatype and size is CHAR(512).

  The same is true for the following columns:

Column Name	In Table...	As Shown in...
DPB_SEG	DDAL_DEST_DPB	Table A-11
MOVE_CLAUSES_SEG	DDAL_MOVE_ CLAUSES	Table A-12
TNSF_PRIVS_SEG	DDAL_TNSF_PRIVS	Table A-14
SEL_EXPR_SEG	DDAL_TNSF_SEL_ EXPR	Table A-15

- Tables A-2 and A-8 contain footnotes that refer to columns using incorrect column names. In those footnotes, the following column names should not have the DDAL_ prefix.

  DDAL_USER_ID_CODE (in Table A-2)
  DDAL_TRANSFER_TYPE (in Table A-2)
  DDAL_REINI_FLAG (in Table A-8)
  DDAL_TRANSFERS_STATE (in Table A-8)
  DDAL_LAST_TRANSFER_TIME (in Table A-8)

4.11 The DDAL$ROW_OWNER column Is of Datatype Smallint

There is information missing in the *Replication Option for Rdb Handbook* release 7.0 in section 8.2.5 "Rules for Multiple Writers to the Same Tables", list item 11. The book explains that you might need to manually add the DDAL$ROW_OWNER column in order to use the “shared write” feature. The datatype for that column, which is smallint, was omitted from the handbook.

4.12 RDB$CHANGES Rows with "TSER" Values of Zero

There has been confusion about the significance of the RDB$TRANSACTION_TSER column in the RDB$CHANGES table. This note provides further clarification than is provided in the *Replication Option for Rdb Handbook*.

Since the RDB$TRANSACTION_TSER column contains a transaction serialization number, some people have wondered why a value of zero is shown for it in so many rows in the RDB$CHANGES table.

Some transactions require that more than one row be written to the RDB$CHANGES table. Each row in the RDB$CHANGES table has two transaction serial numbers. RDB$TRANSACTION_TID contains the Rdb transaction serial number (TSN). Since the TSN is assigned at the start of a transaction, its value will appear in each row in RDB$CHANGES associated with a given transaction. The transaction serialization number in RDB$TRANSACTION_TSER is only assigned when the transaction is about to commit. That means that the value can only be assigned to the final row in the RDB$CHANGES table for a given transaction.

You can demonstrate this by performing the following SQL query:

```
SELECT
RDB$TRANSACTION_TID,RDB$TRANSACTION_SEQUENCE,RDB$TRANSACTION_TSER
FROM RDB$CHANGES ORDER BY RDB$TRANSACTION_TID,RDB$TRANSACTION_SEQUENCE;
```

You might want to add a WHERE or LIMIT clause to this query in order to limit the number of rows that are printed out. The following is an example of what you might see:

```
RDB$TRANSACTION_TID    RDB$TRANSACTION_SEQUENCE    RDB$TRANSACTION_TSER

166023                          1                       0
166023                          2                       0
166023                          3                       103984
166024                          1                       0
166024                          2                       103985
166025                          1                       103986
```

This example shows three transactions:

1. TID 166023, with three rows in RDB$CHANGES
2. TID 166024, with two rows
3. TID 166025, with one row

RDB$TRANSACTION_TSER values in this case are correct. For example, consider the three row transaction whose RDB$TRANSACTION_TID value is 166023. The third row for the transaction has RDB$TRANSACTION_TSER as 103984. Rows one and two, which were written to the RDB$CHANGES table prior to the "TSER" number being assigned, show a value of zero. This is normal.

4.13 Mapping Replication Option Tables to Their Own Storage Area

Section 3.10 of the *Replication Option for Rdb Handbook* Release 7.0, Special Replication Option System Tables for Replications, has three errors in it:

- The second paragraph describes the RDB$CHANGES table as "This table is managed by both Rdb, which writes into this table, and Replication Option, which reads and deletes from this table." In reality all reading, writing and deleting are performed by Rdb. The writing is done implicitly whenever a customer application inserts, updates, or deletes rows from non-system tables. The reading and deleting are done explicitly by Rdb at the request of the Replication Option during transfer execution.
- Section 3.10 also explains how to move the system tables used for the Replication Option, one of which is the RDB$CHANGES table, into storage areas of their own. Step 3 states, "Create a storage map for RDB$CHANGES to map it to the RDB$SYSTEM storage area and commit." This sentence is missing a key requirement and it should be revised to read, "Create a storage map for RDB$CHANGES, which must be named RDB$CHANGES_MAP, to map it to the RDB$SYSTEM storage area. Commit."

  Following the list of steps, this example should be included:

```
ALTER DATABASE FILENAME SOURCE_DB
  RESERVE 1 STORAGE AREAS;

ALTER DATABASE FILENAME SOURCE_DB
  ADD STORAGE AREA RDB_CHANGES_AREA
    FILENAME DISK:[DIR]REPLICATION_AREA
    PAGE SIZE IS 5 BLOCKS;

ATTACH 'FILENAME SOURCE_DB';

CREATE STORAGE MAP RDB$CHANGES_MAP FOR RDB$CHANGES
  STORE IN RDB$SYSTEM;

COMMIT;

ALTER STORAGE MAP RDB$CHANGES_MAP
  STORE IN RDB_CHANGES_AREA;
```

```
COMMIT;
```

- In section 3.10, the last item in the bulleted list says, "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." This should be amended to say that for the RDB$CHANGES table, the area attribute UNIFORM should be chosen for the page type.

4.14 Do Not Use GRANT, REVOKE, or ALTER Within a Transaction

In Replication Option for Rdb, the following SQL statements are not permitted within a transaction:

- GRANT . . . ON TRANSFER
- REVOKE . . . ON TRANSFER
- ALTER TRANSFER
- ALTER SCHEDULE

This restriction is in keeping with behavior for the CREATE TRANSFER, CREATE SCHEDULE, DROP TRANSFER, and DROP SCHEDULE statements. If these statements were issued within a transaction and a ROLLBACK were performed, the rollback would have no effect on these statements. Not allowing these statements to be executed within a transaction removes any possible confusion about this.

This restriction should be added to the *Replication Option for Rdb Handbook*.

4.15 Ownership of Target Tables, Views and Domains

The following is a new section, 8.1.5, for the *Replication Option for Rdb Handbook* Release 7.0. It describes the concept of ownership of data definitions, such as tables, views, and domains in a database that is the target of a Replication Option transfer. For example, a target table is either locally owned, or it is owned by one of the transfers to the target database.

8.1.5 Ownership of Target Tables, Views and Domains

Table 8-1 introduced the notion of data sharing and how it is supported to varying degrees by the Replication Option. Mode 1 is the most restrictive. This mode applies to transfers whose definitions use the clause TO NEW filename. Whenever such a transfer is re-executed (or, in the case of a REPLICATION transfer, re-initialized and then re-executed), a new set of target database files is created, target domains, tables, and views are re-created, and data are re-copied from the source tables. Although customer applications can write

into the target tables given the necessary access permission, any additions or changes they might make are effectively ignored once the new set of target files is created. For this reason, we say that the only writer into such a target database should be the transfer which creates the target files. That transfer is said to be the owner of the entire database.

The Replication Option, via data sharing modes 2 and 3, supports multiple writers to the same target database. Modes 2 and 3 apply to transfers whose definitions use the TO EXISTING filename clause. For such transfers, the Replication Option does not create the target database files. Instead, you give it the names of some existing databases. When such transfers execute again, they do not create new database files. Normally, then, existing target tables remain in place.

For transfers to existing databases modes 2 and 3, the Replication Option has a notion of data definition ownership. This applies to domains, tables and views. For example, a table is either owned by a single transfer, or it is locally-owned. Who owns a domain, view, or table dictates whether or not a transfer is permitted to drop or alter that target definition. Also, if the shared write option is disabled, who owns a table determines who is permitted to write into the table. If the table is locally-owned, only local applications are allowed to write into the table; transfers are not. If the table is owned by a transfer, only that transfer is permitted to write into the table, not other transfers and not local applications. If the shared write option is enabled, local applications and transfers are permitted to write into the target tables . Even with shared writing, only the owner of the target table is permitted to drop or alter the table definition.

To keep track of who owns which target data definitions, the Replication Option uses the RDBVMS$TRANSFER_OWNER or DDAL$TRANSFER_OWNR table. If a domain, table or view is not listed in one of these tables, it is a locally-owned data definition. If the domain, table or view is listed, it has along with it a code which identifies the transfer which owns it.

The RDBVMS$TRANSFER_OWNER or DDAL$TRANSFER_OWNR table is not created in the target database by a mode 1 transfer to a new database. Therefore the transfer does not record ownership of any data definitions it creates. Such a transfer has no need to do so. Since it is a transfer TO NEW filename, it is implied that the transfer owns all data definitions in the target database.

Because of the difference in the way data definition ownership is managed in the case of a transfer TO NEW filename versus a transfer TO EXISTING filename, it is possible to make the following mistake when changing a transfer

definition from the one type to the other. The following steps illustrate the problem:

1. Create and execute a transfer named ABC TO NEW target database. This creates table XXX in the target database and copies data into that table from the source database.
2. STOP and DROP transfer ABC. Re-create transfer ABC, this time changing it TO EXISTING filename target database. Re-execute the transfer. The transfer will fail with an error message indicating that transfer ABC is not the owner of table XXX in the target database.

The following explains how this can arise. When the transfer executes TO EXISTING filename, the Replication Option creates the RDBVMS$TRANSFER_OWNER or DDAL$TRANSFER_OWNR table if such table does not yet exist in the target database. That is the situation here. Then the Replication Option tests to see if table XXX already exists. It does. Next the Replication Option checks the appropriate ownership table to see if some transfer owns the table. In this case, table XXX is not listed. The assumption is that XXX is a locally-owned table. With shared-writing disabled, the transfer is not permitted to write into a locally-owned table. In other words, since execution when the transfer definition uses TO NEW filename leaves no record of table ownership, subsequent execution using TO EXISTING filename has to assume that the table is owned by someone else.

If you find a need to change a transfer from TO NEW filename to TO EXISTING filename, the simplest thing to do is to drop the tables transferred originally by transfer ABC. When ABC is re-executed, using the TO EXISTING clause, the tables will not exist in the target database. In such case the transfer is free to create the tables in the target database and then to declare ownership by making entries in the appropriate ownership table.

4.16 Excessive Pages Checked in RDB$CHANGES

This section explains how Oracle Rdb checks RDB$CHANGES pages when applications do an insert into a replicated table.

By default, Oracle Rdb sizes records for the RDB$CHANGES table so that they just fit on a database page with very little space to spare. For information on how to improve RDB$CHANGES space utilization, see Section 4.17.

When a Replication Option transfer executes and purges a row from the RDB$CHANGES table, the record on the database page is deleted but the line and TSN index entries are retained. Normally, locked lines will be reclaimed. However, if one application uses DBKEY SCOPE IS ATTACH, the reclamation is not done. As a result, when storing a new row in RDB$CHANGES, Oracle

Rdb must allocate a new line and TSN index (a new database key). After a few database keys are locked on a page, there is no longer enough room to insert a new record. Consequently Oracle Rdb may have to scan many seemingly empty pages before it can find one with enough free space on it to insert a row. This behavior can cause the size of the storage area in which RDB$CHANGES resides to grow.

It is recommended that applications not use DBKEY SCOPE IS ATTACH if they remain attached to the source database for a long time.

4.17 Improving RDB$CHANGES Space Usage and Related Performance

Replicated data for use with the Replication Option is stored by Rdb in the RDB$CHANGES table. Disk space utilization for data stored in the RDB$CHANGES table can be improved by changing the storage threshold values for the table. This in turn can help improve application and transfer performance.

This section explains how to set thresholds for the RDB$CHANGES table for the case where the table has been mapped into its own storage area. By default, this table is created in the RDB$SYSTEM storage area. Refer to section 3.10 of the *Replication Option for Rdb Handbook* version 7.0 for information about mapping the RDB$CHANGES table into its own storage area. Setting the thresholds for the RDB$CHANGES table can be done whether or not the table is in the RDB$SYSTEM storage area or some other storage area. It is recommended that RDB$CHANGES be placed in a uniform storage area of its own, with no other tables or indexes.

Refer to the *Replication Option for Rdb Handbook* for information on setting thresholds for the optional indexes for the RDB$CHANGES table.

4.17.1 What Are Storage Thresholds?

Rdb supports the definition of thresholds for storage areas, storage maps and indices. Thresholds provide an efficient way for Rdb to determine where there is room in the database to insert new data rows. To understand the purpose of thresholds and how to define them appropriately requires an understanding of how Rdb manages space on database pages.

4.17.2 Space Management in Rdb Databases

The following is an abbreviated discussion of space management in Rdb databases. Only information relevant to setting thresholds for the RDB$CHANGES table is explained.

For Rdb databases, all data rows are stored on data pages in storage areas. As data are stored the page eventually fills up and can no longer be used to insert more data. Rdb uses SPAM pages to find a page with enough free space on it to handle the insertion of another row, as described in the following section.

4.17.2.1 SPAM Pages

To avoid the overhead of doing a sequential scan of database pages, Rdb uses a special page called a SPAM (SPAce Management) page. SPAM pages are placed at regular intervals in each storage area. The first page in each storage area is a SPAM page, and each SPAM page manages a range of pages known as the SPAM interval. For example, for a UNIFORM storage area, such as one for the RDB$CHANGES table with a page size of two disk blocks, the default SPAM interval is 216 pages. In this case it means that SPAM pages occur every 216 pages within the storage area.

Each SPAM page contains two bits to describe every page in its SPAM interval. These bits encode the values 0, 1, 2 and 3, and have the following meanings:

Threshold	Meaning
3, 2, and 1	The page is full, or at least cannot accommodate more data because the remaining free space is too small for any row type written to this page.
0	The page has room on it for at least one data row.

Even when a page is marked full, there might be a little free space which can be used to expand the size of an already-stored data row if necessary.

As rows are inserted on the page and the percentage at each level is exceeded, an indication is recorded in the SPAM bits for the corresponding page in the SPAM page. Likewise, as rows are deleted, the threshold bits are updated when the level changes past a set level.

As data is stored on a page, the page fills up. Consider the case of a page of size two blocks: this page will allow up to 964 bytes of data to be stored. If you were storing a row from the EMPLOYEES table of our sample PERSONNEL database, each row takes up 134 bytes: 126 bytes plus 8 bytes of overhead for a line index and a TSN index. The line index is used to locate a particular row on a database page. It is one of the components of a database key. The TSN index is information about the row related to transactions. After inserting

seven such rows, and taking into consideration page level overhead, 26 bytes of room would remain. If this is the only type of row being stored on the page, as would be the case for rows in the RDB$CHANGES table if they were stored in their own storage area, then the page is effectively full and no longer is a candidate for inserting new data.

The simplest use of the SPAM threshold bits is used for this case, which describes the mapping of a table to a UNIFORM area. In this case, the page is considered full when a new row of this type will no longer fit, or not full when space remains for one or more rows of that type.

4.17.2.2 AIP Pages

The algorithm uses the row length stored in the AIP (the Area Inventory Page), another type of structure page. The AIP record length is written to the AIP when a table, storage map or index is created.

The AIP record length for the RDB$CHANGES table is 10022 bytes. You can confirm this by issuing the command:

```
$ RMU/DUMP/LAREA=RDB$AIP/OUT=filename.out  root-file-spec
```

This command produces an output file named filename.out (substitute a filename of your own choosing). For root-file-spec, substitute the specification for the database root file.

In the output file, search for entries that say, "area name 'RDB$CHANGES..."'. If you have moved the RDB$CHANGES table into the RDB$SYSTEM storage area and then into its own storage area (per instructions in section 3.10 of the *Replication Option for Rdb Handbook* version 7.0), there will be two entries in the AIP.

The first entry will say:

```
entry is free
deleted by TID...
```

This was the entry when the table was first mapped into the RDB$SYSTEM storage area. The second entry will say:

```
record length 10022 bytes
entry is in use
thresholds are (0,0,0)
```

This shows the current AIP record length for the RDB$CHANGES table. Also, the value (0,0,0) means that the threshold values for this table have not been defined. As a result, Rdb will use default threshold values, which is discussed in Section 4.17.3.

4.17.2.3 Use of Multiple Thresholds Per Page

The multiple threshold values allow Rdb to handle the case where more than one type of data row is stored in the same storage area. It would allow Rdb to determine that there is not enough room on a page for a row of one type but there is enough room for a row of a different type.

When there is only one table in the storage area, multiple thresholds can be used to indicate the minimum, average, and maximum table length as a percentage of the maximum free space. For the purposes of this discussion, this will be simplified so that only the average length of an RDB$CHANGES row is considered.

Threshold specifications provide up to three percentage values which describe the high water mark for various row types. The analogy here is that of filling up a glass of water. Going back for a moment to our example using the PERSONNEL database, too full for an EMPLOYEES row means that the free space on the page falls below 134 bytes. This byte length is expressed as a percentage of the maximum free space on the page; the maximum free space depends on the size of the database page minus page management overhead information. To encode the threshold values in the SPAM page compactly, the percentages are recorded in the AIP page, and the current state of a page's free space is encoded in two bits within the SPAM page and indicates the relative fullness of the page with respect to the threshold values.

4.17.3 Initial RDB$CHANGES Thresholds Result in Excessive I/O

When the RDB$CHANGES table is created, its threshold values are not defined. You can see this in the AIP output (refer to Section 4.17.2.2) as (0,0,0). If there are no thresholds defined, Rdb calculates a default set of thresholds using the following formula:

```
T1 = T2 = T3 = 100 - (Max_New_Seg_Len * 100 / Max_Free_Len )
```

Here T1, T2, and T3 represent the three threshold values; and Rdb sets them all to the same value. How to determine the values of Max_New_Seg_Len and Max_Free_Len will be explained in Section 4.17.4. Example values for the RDB$CHANGES table are 2500 and 2518, respectively. Substitution of those values into the formula results in threshold values of (1,1,1). This means that once a database page becomes one percent full, it is considered to be completely full.

However, consider the case where the actual compressed length of the row is smaller than the Max_Free_Len. After the row is stored, Rdb detects that the page will no longer store a new row and marks the page full. Thus, for any extra long row which compresses well, space in the database will be wasted. Wasted space translates into excessive I/O to locate room for insertion of new

data rows. Many reports of excessive I/O to insert a row in RDB$CHANGES involve the incorrect or untuned SPAM thresholds. Correction of this type of problem is important in production applications. This problem can be overcome by specifying appropriate thresholds for the area.

The RDB$CHANGES table is defined with one VARCHAR column of 10,000 bytes. This table is a journal of user changes which are read by the replication transfers scheduled against the database. By default the table is mapped to the RDB$SYSTEM storage area, which often has a page size of two blocks. The RDB$CHANGES rows are always sized and compressed to fit the available free space. How well these rows compress is a function of the user data being journalled. Therefore, there might be wasted space on pages which store rows for the RDB$CHANGES table.

4.17.4 Changing the Thresholds and AIP Record Length

For the RDB$CHANGES table, because row lengths are based on the free space given a particular page size rather than on the maximum size of the VARCHAR 10,000 column, the formula for calculating the thresholds should be:

```
T1 = T2 = T3 = 100 - ( (Average_Row_Len + 8) * 100 / Max_Free_Len )
```

The following example shows the RDB$CHANGES table mapped to its own storage area given a page size of five blocks.

Step 1: Dump the Database Header

Dump out information from the database header file, as follows:

```
$ RMU/DUMP/HEADER/OPTIONS=DEBUG/OUT=filename.out   root-file-spec
```

This command produces an output file named filename.out (substitute a filename of your own choosing). For root-file-spec, substitute the specification for the database root file.

If the RDB$CHANGES table is in the RDB$SYSTEM storage area, look in the output file for: Storage area "RDB$SYSTEM". If the RDB$CHANGES table is in its own storage area, for example, the RDB_CHANGES_AREA, look in the output file for: Storage area "RDB_CHANGES_AREA". Under that heading you will see a variety of information, of which the following is useful or instructive:

```
Page format is uniform
```

Uniform is the recommended page format for the storage area for the RDB$CHANGES table.

```
Page size is 5 blocks
```

Page size affects MAX_FREE_LEN and MAX_NEW_SEG_LEN.

```
SPAMS are enabled
```

You should always see the message informing you that SPAMS are enabled.

```
MAX_FREE_LEN = 2518.
```

The value you see for MAX_FREE_LEN might be different from what is shown here.

```
MAX_NEW_SEG_LEN = 2500.
```

The value you see for MAX_NEW_SEG_LEN might be different from what is shown here.

```
THRESHOLD = (0., 0., 0.)
```

This indicates that threshold values have not been defined for the RDB$CHANGES table.

Step 2: Analyze the RDB$CHANGES Table

Next, determine the average length of a data row in the RDB$CHANGES table. When you do so, make sure that there is a fair sample of data rows in that table so that a reasonable average can be computed. Use the following command:

```
$ RMU/ANALYZE/LAREA=RDB$CHANGES/OUT=filename.out  root-file-spec
```

This command produces an output file named filename.out (substitute a filename of your own choosing). For root-file-spec, substitute the specification for the database root file. This will dump information about storage areas and also about the selected logical area, the RDB$CHANGES table. In our example, the RDB$CHANGES table has its own storage area; no other database object is stored in that same area. Consequently, the statistics for the table's storage area will be the same as those for the table itself. In our example, the storage area is named RDB_CHANGES_AREA. Look for the following lines in the report:

```
Logical area:   RDB$CHANGES for storage area : RDB_CHANGES_AREA
Record length:  10022, Compressed
Average length: 227, compression ratio:   .02
```

The AIP record length of 10022 is the default length for the RDB$CHANGES table. You will change this value farther on. The average record length, shown here as 227, will vary. Use the value you obtain from the report for your database.

Step 3: Compute the Threshold Values

Substitute the values you obtained for the two RMU reports into the formula:

```
T1 = T2 = T3 = 100 - ( (Average_Row_Len + 8) * 100 / Max_Free_Len )
= 100 - ( (227 + 8) * 100 / 2518 )
= 91
```

The value 8 is added to the average row length to account for eight bytes to record the line index and the TSN index entries on the page for the data row. What you end up with is a percent full factor for the page to tell Rdb at what point a row of this type can no longer fit on the remaining space on the page. In other words, when a page in the storage area for the RDB$CHANGES table becomes more than 91% full, there's no room on the page to insert another row.

In this note, it is proposed that this storage area be used exclusively for the RDB$CHANGES table and that storage calculations be simplified by using just the average row length. In such a case, you can set the threshold values to be the same, (91, 91, 91).

Step 4: Create an Initialization File for RMU

In step five you will execute an RMU command which references an initialization file. Create the initialization file with a name of your own choosing. Here we call it init.opt. This is a simple text file. Enter the following line:

```
RDB$CHANGES/AREA=RDB_CHANGES_AREA/THRESHOLDS=(91,91,91)/LENGTH=227
```

In the example, the area name is RDB_CHANGES_AREA. Substitute the actual name of the storage area for your RDB$CHANGES table. Use the threshold values you calculated in step 3, and use the average record length determined from step 2.

Step 5: Repair the SPAM Pages and Change the AIP Record Length

Now issue the following statement:

```
$ RMU/REPAIR/SPAMS/INIT=LAREA=init.opt   root-file-spec
```

The init.opt file is the file you created in step 4. For root-file-spec, substitute the specification for the database root file. This RMU/REPAIR statement, combined with the init.opt file, will cause the SPAM pages for the RDB_ CHANGES_AREA to be modified, based on the new threshold values and the new AIP record length for RDB$CHANGES.

Note

When you use RMU/REPAIR in this way, these options are not journalled to the AIJ (After Image Journal) file. Therefore you

must make a full backup of your database when performing database restructuring in this way. For further information, refer to the *Oracle Rdb RMU Reference Manual*.

Step 6: Confirm That the Changes Have Been Made

You can confirm that the threshold values and the AIP record length have been changed by reissuing the RMU/DUMP command shown in Section 4.17.2.2.

4.17.5 Verifying Changes in Space Utilization

Before the thresholds were changed, each row in the RDB$CHANGES table took up an entire database page. One way to see that this is so is to dump all the database keys for the existing rows in the RDB$CHANGES table. Database keys are printed in the format area:page:line. For the RDB$CHANGES rows inserted prior to the threshold change, all line numbers will be zero. This indicates that there is only one row stored per database page. You can output the database keys using an SQL SELECT statement or by using the DDAL$ANALYZE utility, which has an option for outputting the database keys.

After the thresholds have been changed, some pages which were formally marked as full will now be marked as having space available. After this point, some of the new rows inserted into the RDB$CHANGES table will be placed in pages already occupied by other RDB$CHANGES rows. If you then dump out the database keys of the RDB$CHANGES rows, you will begin to see some with line numbers greater than zero. This indicates that the thresholds are taking effect and that more rows are occupying the same database pages than before.