Oracle CDD/Repository™

User's Guide Supplement

Version 6.1

Part No. A31157-1

ORACLE®

Oracle CDD/Repository User's Guide Supplement

Version 6.1

Part No. A31157-1

This document is available on CD-ROM.

Contents

2 Enhancing Repository Performance

3 Upgrading a Dictionary or Repository

4 Using Oracle CDD/Repository with Oracle Rdb

Index

Examples

Figures

Tables

Send Us Your Comments

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

You can send comments to us in the following ways:

- Electronic mail — database_doc@weorg.enet.dec.com
- FAX — 603-897-3334 Attn: Oracle CDD/Repository Documentation
- Postal service

  Oracle Corporation
  Oracle CDD/Repository Documentation
  One Oracle Drive
  Nashua, NH 03062
  USA

If you like, you can use the following questionnaire to give us feedback.

Name _____ Title _____

Company _____ Department _____

Mailing Address _____ Telephone Number _____

Book Title _____ Version Number _____

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

- What features did you like most about this manual?

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

Preface

This document provides updated sections to *Using Oracle CDD/Repository on OpenVMS Systems*, Version 5.0, dated October 1991.

Document Structure

This document contains the following chapters:

- Chapter 1, Verifying and Rebuilding Repositories

  The information in this chapter supersedes Section 3.4 through Section 3.6 in *Using Oracle CDD/Repository on OpenVMS Systems*.

- Chapter 2, Enhancing Repository Performance

  The information in this chapter supersedes Section 3.7 through Section 3.9 in *Using Oracle CDD/Repository on OpenVMS Systems*.

- Chapter 3, Upgrading a Dictionary or Repository

  The information in this chapter supersedes Section 4.5 in *Using Oracle CDD/Repository on OpenVMS Systems*.

- Chapter 4, Using Oracle CDD/Repository with Oracle Rdb

  This chapter supersedes Chapter 7 in *Using Oracle CDD/Repository on OpenVMS Systems*.

Notice of Version Number Change

The version numbers have been synchronized with Oracle Rdb™ to reflect the tighter integration of the Oracle Rdb and Oracle CDD/Repository product set, and to simplify product referencing. Effective with this release, the Oracle CDD/Repository and Oracle CDD/Administrator™ version numbers are the same. They are both called Version 6.1, and they coincide with the current version number of Oracle Rdb (Version 6.1).

Version 5.3 and subsequent Engineering Change Orders (ECOs) make up the last release of Oracle CDD/Repository before the Version 6.1 release. Version 1.2 was the last release of Oracle CDD/Administrator before Version 6.1. Version 6.1 is *not* a major release; it represents the next scheduled maintenance release after Version 5.3 of Oracle CDD/Repository and Version 1.2 of Oracle CDD/Administrator.

Related Documents

Documents related to Oracle CDD/Repository include the following:

- *Oracle CDD/Repository CDO Reference Manual*
- *Using Oracle CDD/Repository on OpenVMS Systems*
- *Oracle CDD/Repository User's Guide Supplement*
- *Oracle CDD/Repository Architecture Manual*
- *Oracle CDD/Repository Callable Interface Manual*
- *Oracle CDD/Repository Information Model Volume I*
- *Oracle CDD/Repository Information Model Volume II*
- *Installing Oracle CDD/Repository for OpenVMS VAX Systems* or, depending on your system, *Installing Oracle CDD/Repository for OpenVMS Alpha Systems*.
- *Read Before Installing or Using Oracle CDD/Repository Version 6.1 on OpenVMS VAX Systems* or, depending on your system, *Read Before Installing or Using Oracle CDD/Repository Version 6.1 on OpenVMS Alpha Systems*

Conventions

Table 1 shows the conventions used in this manual.

Table 1 Documentation Conventions

Convention	Description
n	A lowercase italic *n* indicates the generic use of a number. For example, 19*nn* indicates a 4-digit number in which the last 2 digits are unknown.

(continued on next page)

Table 1 (Cont.) Documentation Conventions

Convention	Description
Return	A key name enclosed in a box indicates that you press that key. In examples, an implied carriage return occurs at the end of each line, unless otherwise noted. You must press the Return key at the end of a line of input.
Oracle CDD/Repository	This manual uses the name Oracle CDD/Repository to refer to all versions of this product. Prior to Version 5.0, this product was known as CDD and CDD/Plus.
( )	In format descriptions, parentheses delimit the parameter or argument list.
[ ]	In format descriptions, brackets indicate optional elements. You can choose none, one, or all of the options. (Brackets are not optional, however, in the syntax of a directory name in an OpenVMS file specification.)
" "	Quotation marks enclose system messages that are specified in text.
. . .	In format descriptions, horizontal ellipsis points indicate one of the following: • An item that is repeated • An omission, such as additional optional arguments • Additional parameters, values, or other information that you can enter
. . .	Vertical ellipsis points indicate the omission of information from an example or command format. The information is omitted because it is not important to the topic being discussed.
italic type	Italic type emphasizes important information, indicates variables, and indicates complete titles of manuals.

(continued on next page)

Table 1 (Cont.) Documentation Conventions

Convention	Description
UPPERCASE	Words in uppercase indicate a command, a property, a parameter, the name of a file, the name of a qualifier, the name of a file protection code, or an abbreviation for a system privilege.
$	A dollar sign ($) represents the OpenVMS DCL system prompt.
`monospaced`	This typeface is also used in interactive examples and other screen displays.

1
Verifying and Rebuilding Repositories

You should periodically monitor the condition of your repository. This chapter describes how to make backup copies of your repository and use the CDO VERIFY command to verify the structural condition of the repository.

1.1 Backing Up Repositories

Include a regular backup of the repository as part of repository maintenance, in addition to the normal system backup procedure established at your site. A full OpenVMS system backup and incremental backups do not capture metadata changes, and are not adequate for repository backup. Your repository backup schedule should be based on the number of users and their requirements, and the amount and frequency of changes to the repository.

You perform a repository backup from the OpenVMS environment, not the CDO environment. Backing up a repository involves using the OpenVMS Backup utility (BACKUP) to back up the nondatabase files in the anchor directory, and then using the RMU/BACKUP command to back up the database files.

An anchor specifies the OpenVMS directory where the repository hierarchy is stored. The anchor can consist of node, device, and directory components. It can also be described in a logical name format, such as CDD$COMPATIBILITY. The anchor directory contains the CDD$DATABASE.RDB relational database file, as well as several files and subdirectories that make up the repository hierarchy.

The instructions in the following sections describe how you back up a repository by using both the OpenVMS BACKUP and RMU/BACKUP commands. Both parts are critical for a valid backup. If you use only the OpenVMS BACKUP command for the anchor directory, including the database files, or only the RMU/BACKUP command, repository integrity is not guaranteed.

Caution

Do not use the CDO VERIFY/REBUILD_DIRECTORY command with the RMU/BACKUP and RMU/RESTORE commands to perform a repository backup. For information about using the VERIFY /REBUILD_DIRECTORY command, see Section 1.2.3.

Do not use the REPOSITORY EXPORT command to back up your files. See Chapter 3 for information on upgrading a dictionary or repository using REPOSITORY EXPORT.

1.1.1 Before You Begin a Backup

When possible, back up the repository during a time when your system is least used. You can also temporarily change the protection on the repository anchor by using the CDO CHANGE PROTECTION command and allow access to only the database administrator.

To prohibit the use of the repository by other users, follow these steps:

1. Broadcast a message to notify users that you are preparing to back up the repository. This command requires OPER (operator) privilege.

   For example:

   ```
   $ REPLY/ALL/BELL "Repository backup - please exit the repository"
   ```

2. Issue the following command on each cluster for each repository you are about to back up to be sure that no one is accessing the repository or database:

   ```
   $ RMU/DUMP/USERS disk:[anchor_dir]CDD$DATABASE.RDB
   ```

   If you have distributed repositories (linked repositories that share data), prohibit the use of each repository. Repeat this command for each linked repository to avoid inconsistencies between repositories.

   Refer to Section 1.1.2 for instructions on locating linked repositories.

3. Prevent users from starting up Oracle CDD/Repository by invoking the SYS$STARTUP:CDDSHUTDOWN.COM command procedure.

1.1.2 Locating Repositories

To locate the repositories on your system, use the DCL directory command and search each device for the CDD$DATABASE.RDB relational database file.

```
$  DIR disk:[000000...]CDD$DATABASE.RDB
```

If you have Oracle CDD/Repository Version 5.*n* or higher installed, enter the CDO SHOW REPOSITORIES command to locate linked repositories:

```
$  REPOSITORY OPERATOR
 Welcome to CDO V6.1
   The CDD/Repository V6.1 User Interface
   Type HELP for help
CDO>  SET DEFAULT disk:[anchor_dir]
CDO>  SHOW REPOSITORIES
CDO>
```

If you have Version 4.*n* installed, enter the CDO SHOW GENERIC command to locate linked dictionaries:

```
   .
   .
   .
CDO> SHOW GENERIC CDD$ANCHOR/FULL disk:[anchor_dir]CDD$PROTOCOLS.CDD$SELF
CDO>
```

1.1.3 Regular Repository Backup

To perform a regular backup of your repository, follow these steps:

1. Back up the nondatabase components by using the OpenVMS BACKUP command. Do not create the backup file (.BCK) in the OpenVMS directory that contains the repository you are backing up. Either specify a different directory for the .BCK file, or set default to a different directory. This command requires SYSPRV.

   ```
   $  BACKUP/VERIFY/EXCLUDE=(.RDA,.RDB,.SNP) -
   _$  disk:[anchor_dir...] disk:[different-dir]filename.BCK/SAVE
   ```

2. Back up your repository database using the RMU/BACKUP command. Do not create the backup file (.RBF) in the OpenVMS directory that contains the repository you are backing up. Either specify a different directory, or set default to a different directory.

   ```
   $ RMU/BACKUP disk:[anchor_dir]CDD$DATABASE -
   _$ disk:[different-dir]filename.RBF
   ```

If you have linked repositories, perform backups at the same time to avoid inconsistencies between repositories.

Note

If you are backing up more than one repository, it is important to give the .BCK and .RBF backup files meaningful names.

If you are using Oracle CDD/Repository for configuration management, you may want to perform an image backup rather than a regular backup. An image backup backs up the entire contents of the device that contains your repository. A regular backup makes extra copies of any files (binaries) that are opened in user contexts. As a result, more disk space is required during the restore operation. Refer to the OpenVMS documentation on the Backup utility for more information.

If all the components of the CDD$DATABASE files and the .RUJ files are not all on the same disk as the image backup, you must also perform the RMU/BACKUP operation.

1.1.4 Restoring a Repository from a Regular Backup

If you need to restore the repository from a backup, follow these steps:

1. Set default to an empty directory:

   ```
   $ SET DEFAULT disk:[new-anchor-dir]
   ```

   This will be your new anchor directory. The new anchor directory should not contain any files or subdirectories.

2. Restore the nondatabase components, as follows:

   ```
   $ BACKUP filename.BCK/SAVE_SET/SELECT=[anchor_dir...] [...]/LOG/VERIFY
   ```

3. Restore the database, as follows:

   ```
   $ RMU/RESTORE/NOCDD/DIRECTORY=[] filename.RBF
   ```

4. Verify each repository. Refer to Section 1.2 for how to perform a verify operation.

If the repository you are restoring has external references, you must restore each repository that is linked to that repository. Be sure to restore the linked repositories from backups that were made at the same time, to prevent inconsistencies between repositories. Perform the verify operation *after* you have restored all the linked repositories.

1.2 Using the CDO VERIFY Command

The CDO VERIFY command scans your repository files to confirm that the repository is structurally correct. Perform a verify operation in the following situations:

- Before and after you upgrade a repository
- After you restore a repository from a backup
- After you move a repository to a new location
- When an error message indicates the need for verification
- When you suspect a problem, such as after a system failure
- When you get errors from programs that access the repository
- When you notice unexpected behavior or missing information during repository activity

Use the CDO VERIFY/NOFIX command periodically to confirm the integrity of the repository. The frequency with which you verify repositories depends on your site's policies and the amount of repository activity at your site.

The CDO VERIFY/ALL/NOFIX command is the recommended method for verifying repositories on a regular basis.

Note

In prior versions of Oracle CDD/Repository, when you issued the VERIFY/ALL command, the /FIX qualifier was included by default. In Version 6.1, the default has been changed to /NOFIX. However, you can continue to use /FIX as the default by defining the CDD$VERIFY_ALL_FIX logical to be any value. Define the CDD$VERIFY_ALL_FIX logical at the process level or higher.

In addition, if you issue the VERIFY/ALL command without specifying a /FIX or a /NOFIX qualifier, and if you have not defined the CDD$VERIFY_ALL_FIX logical, a %CDO-I-VF_ALL_NOFIX informational message will be displayed immediately and the verify operation will continue.

If you issue the VERIFY command with the /FIX qualifier, make sure you have full backups of the repository and all other repositories that are linked to it *before* you attempt the fix.

To successfully run VERIFY/FIX you must have SYSPRV or BYPASS privilege.

For a full description of the VERIFY command and its qualifiers, see the *Oracle CDD/Repository CDO Reference Manual*.

1.2.1 Verifying Repository Condition

To verify the structural condition of a repository, perform the following steps:

1. Before you issue the VERIFY command, prohibit all use of the repository. Refer to Section 1.1.1 for details.
2. Invoke the CDO utility, and enter the VERIFY/ALL/NOFIX command at the CDO prompt:

```
$ REPOSITORY OPERATOR
   Welcome to CDO V6.1
   The CDD/Repository V6.1 User Interface
   Type HELP for help
CDO>  VERIFY/ALL/NOFIX disk:[anchor_dir]
CDO>  EXIT
```

   In some cases, it may be necessary to run VERIFY more than once. A VERIFY/ALL/FIX command should be followed by VERIFY/ALL/NOFIX to ensure that the repository is completely verified. If an error occurs after you use the /NOFIX qualifier, then run VERIFY/ALL/FIX again.
3. Review the results. If problems exist, correct them. For example, you may need to upgrade the repository if you have recently installed a new version of Oracle Rdb. Refer to Chapter 3 for upgrade instructions.

   Most other problems can be corrected with VERIFY/ALL/FIX, as follows:

   a. Confirm whether the repository you are verifying is linked to other repositories. Refer to Section 1.1.2 for information on how to locate linked repositories.

      Be sure that you upgrade all linked repositories.

   b. Issue VERIFY/ALL/FIX for each repository. Wait for each VERIFY command to complete before issuing the next VERIFY command.

      Do *not* issue concurrent VERIFY commands against linked repositories. By default, VERIFY locks the repository it is verifying. A concurrent VERIFY will report errors when it attempts to access a linked repository that is locked.

   c. After each VERIFY/ALL/FIX command completes, you can issue a VERIFY/ALL/NOFIX command to verify the structural condition of the repository. Again, wait for each VERIFY command to complete before issuing the next VERIFY command.

Restrictions

- The VERIFY operation needs unrestricted access to a number of internal objects and structures to successfully complete its job. You must have SYSPRV or BYPASS privilege to run VERIFY/FIX.
- To use the VERIFY command, you need SHOW access to the repository. You also need SHOW access to any definitions that you are verifying, such as cross-repository relationships.
- You cannot perform a remote verify operation.

1.2.2 Logging Repository Verification

During a verify operation, you can display information and error text on your default output device by including the /LOG qualifier when you issue the VERIFY command.

The following example shows the output from the VERIFY/LOG command. (The output from this example indicates that there are no problems.)

```
$  REPOSITORY OPERATOR
 Welcome to CDO V6.1
   The CDD/Repository V6.1 User Interface
   Type HELP for help
CDO> VERIFY /ALL/NOFIX/LOG SYS$COMMON:[CDDPLUS.PERSONNEL]
%CDD-I-VFRECVR, VERIFY recovering journalled changes...
%CDD-I-NORECOVER, dictionary does not need recovery
%CDD-I-VFALL, VERIFY checking location, internal, and external references...
%CDD-I-OKLOC, dictionary SYS$COMMON:[CDDPLUS.PERSONNEL] location self-reference is okay
%CDD-I-SKIPTYPE, skipping protocol type MCS_ELEMENT_TYPE
%CDD-I-SKIPTYPE, skipping protocol type MCS_RELATION_TYPE
%CDD-I-SKIPTYPE, skipping protocol type MCS_METHOD
%CDD-I-SKIPTYPE, skipping protocol type MCS_MESSAGE
%CDD-I-SKIPTYPE, skipping protocol type MCS_MSGARG
%CDD-I-SKIPTYPE, skipping protocol type MCS_DATA_TYPE
%CDD-I-SKIPTYPE, skipping protocol type MCS_PROPERTY_TYPE
%CDD-I-SKIPTYPE, skipping protocol type CDD$LINK_TYPE
%CDD-I-SKIPTYPE, skipping protocol type MCS_VALIDATION
%CDD-I-SKIPTYPE, skipping protocol type MCS_HAS_DEFAULT_METHOD
%CDD-I-SKIPTYPE, skipping protocol type MCS_HAS_MESSAGE
%CDD-I-SKIPTYPE, skipping protocol type MCS_HAS_MSGARG
%CDD-I-SKIPTYPE, skipping protocol type MCS_HAS_DATATYPE
%CDD-I-SKIPTYPE, skipping protocol type MCS_HAS_PREAMBLE
%CDD-I-SKIPTYPE, skipping protocol type MCS_HAS_POSTAMBLE
%CDD-I-SKIPTYPE, skipping protocol type MCS_HAS_RELATION
%CDD-I-SKIPTYPE, skipping protocol type MCS_RELATION_MEMBER
%CDD-I-SKIPTYPE, skipping protocol type MCS_HAS_PROPERTY
%CDD-I-SKIPTYPE, skipping protocol type MCS_HAS_RELATION_PROPERTY
%CDD-I-SKIPTYPE, skipping protocol type MCS_HAS_LINK_PROPERTY
%CDD-I-SKIPTYPE, skipping protocol type MCS_HAS_COMPUTED_PROPERTY
```

```
%CDD-I-SKIPTYPE, skipping protocol type CDD$HAS_LINK
%CDD-I-SKIPTYPE, skipping protocol type MCS_IMPLEMENTS_RELATION
%CDD-I-SKIPTYPE, skipping protocol type MCS_IMPLEMENTS_LINK
%CDD-I-SKIPTYPE, skipping protocol type MCS_IMPLEMENTS_METHOD
%CDD-I-SKIPTYPE, skipping protocol type MCS_HAS_SUPERTYPE
   .
   .
   .
CDO>
```

1.2.3 Recovering the Directory System

The purpose of the VERIFY/REBUILD_DIRECTORY command is to rebuild *only* the directory system in a repository. The directory system does not include any database files with the extension .RDA, .RDB, or .SNP. It also does not include the subdirectories used for configuration management, namely CONTEXTS.DIR, DELTAFILES.DIR, PARTITIONS.DIR, and their contents.

Use the CDO VERIFY/REBUILD_DIRECTORY command when the directory entries for the repository anchor are so corrupt that all other VERIFY qualifiers fail. For example:

```
   .
   .
   .
CDO> VERIFY/REBUILD_DIRECTORY/LOG SYS$COMMON:[CDDPLUS.PERSONNEL]
CDO>
```

VERIFY will ask if you are satisfied with your backup. Make sure you have an adequate backup of the repository before running VERIFY/REBUILD_ DIRECTORY. The default answer to this question is NO. See Section 1.1 for the backup procedure.

```
Are you satisfied with the backup of your repository? [Y/N] (N): y
```

The VERIFY/REBUILD_DIRECTORY command first deletes all files in the anchor except the three subdirectories, DELTAFILES.DIR, PARTITIONS.DIR, CONTEXTS.DIR, and the database files. It then rebuilds the deleted files from the contents of the database. If any directory or database files are missing or inaccessible, the VERIFY/REBUILD_DIRECTORY operation will fail.

If you issue the VERIFY command with /REBUILD_DIRECTORY and /LOG qualifiers, Oracle CDD/Repository provides numerous informational messages about the rebuilding process. For example:

```
%CDD-I-VFREBLD, VERIFY rebuilding directory system...
%CDD-I-REBUILD, rebuilding directory structure for
  SYS$COMMON:[CDDPLUS.PERSONNEL]
%CDD-I-OKLOC, dictionary location SYS$COMMON:[CDDPLUS.PERSONNEL]
  self-reference is okay
CDO>
```

Restriction

You must have SYSPRV or BYPASS privilege to successfully run VERIFY /REBUILD_DIRECTORY.

1.2.4 Reducing the Snapshot File

A snapshot is a picture of the database for read-only transactions. To improve performance, reduce the size of the snapshot file by using the /COMPRESS qualifier with the VERIFY command, as follows:

```
   .
   .
   .
CDO>  VERIFY/COMPRESS disk:[anchor_dir]
CDO>
```

Restrictions

The following restrictions apply to using the /COMPRESS qualifier:

- You must be the only user of the database at the time when you enter the command.
- VERIFY/COMPRESS must be the first command you enter after starting a CDO session; otherwise, CDO reports a conflict error with other users.

1.3 Moving a Repository

This section describes how to move a repository. Before you move a repository from one location to another, there are several things you need to consider:

- Repository locations

  Is the old location on the same cluster as the new location?

- Oracle Rdb and Oracle CDD/Repository versions

  Are the versions of Oracle Rdb and Oracle CDD/Repository for the repository you are moving compatible with the versions on the new system?

- Is it a distributed repository?

  Is this a standalone repository or a distributed repository (a repository that has external references to other repositories)?

If this repository has external references, are you also moving the other repositories and will the other repositories be accessible through the network during and after the move?

- Are you moving DMU dictionary files?
- Privileges

  Does the account from which you are performing the move have SYSPRV or BYPASS privileges?

The following are some general guidelines for moving a repository. These guidelines will help you determine the correct method to use for your situation, and will make the operation easier.

Repository Locations

If you are moving a repository from one location on a cluster to another location on the same cluster, then use the CDO MOVE REPOSITORY command. See Section 1.3.1 for instructions on using the CDO MOVE REPOSITORY command.

If you are moving a repository from one system to another system that is not on the same cluster, you are performing a remote move operation. You cannot use the CDO MOVE REPOSITORY command for a remote move operation. See Section 1.3.2 for instructions on performing a remote move operation.

Oracle Rdb and Oracle CDD/Repository Versions

It is important to consider the versions of Oracle CDD/Repository and Oracle Rdb at both the source and target locations before you move a repository. If you are moving a repository within the same cluster (using the CDO MOVE REPOSITORY command), this will not be a problem.

However, if you are moving a repository from one system to another system that is not on the same cluster (performing a remote move operation), it is important to consider the versions of Oracle Rdb and Oracle CDD/Repository.

You cannot move a repository to a system that is running a version of Oracle CDD/Repository that is lower than the version of Oracle CDD/Repository where the repository currently resides.

Likewise, you cannot move the repository to a system that is running a version of Oracle Rdb that is lower than the version of the repository database.

However, you can move a single repository that has no external references from a system that is running a lower version of Oracle Rdb and Oracle CDD/Repository to a system that is running a higher version. In that case, after you move the repository, you need to upgrade it. See the upgrade procedure in Chapter 3.

If you are performing a remote move operation and the repository has external references, version restrictions may apply. See the details for performing a remote move operation in Section 1.3.2.

Distributed Repositories

If you are moving one or more repositories that have external references to other repositories, there are some precautions and restrictions to consider. This information is provided in Section 1.3.2.

If you are not sure if the repository has external references, or if you want to display a list of all repositories that a specific repository references, use one of the following commands:

- In Version 5.0 and higher, use the CDO SHOW REPOSITORIES command:

  ```
  CDO> SET DEFAULT disk:[anchor_dir]
  CDO> SHOW REPOSITORIES
  ```

- In Version 4.*n* use the CDO SHOW GENERIC command:

  ```
  CDO> SHOW GENERIC CDD$ANCHOR disk:[anchor_dir]CDD$PROTOCOLS.CDD$SELF
  ```

DMU Dictionaries

The precautions and restrictions that apply to moving repositories described in Section 1.3.1 and Section 1.3.2 do not apply to moving DMU dictionary files. See Section 1.3.3 for instructions on how to move DMU dictionaries.

Privileges

With the exception of moving a DMU dictionary, most of the steps involved in either method for moving a repository require the account from which you perform the move operation to have SYSPRV or BYPASS privilege.

The CDO MOVE REPOSITORY command, the CDO VERIFY/LOCATION/FIX command, and both upgrade utilities (CDO CONVERT/REPOSITORY and CDDX) require SYSPRV or BYPASS privileges. Depending on the method you use, some of these commands will be executed during the move operation.

The following sections include detailed instructions for each method of moving a repository and the restrictions that apply to that method.

1.3.1 Moving a Repository to a Different Location on the Same Cluster

To move a repository to a different location on the same cluster, use the CDO MOVE REPOSITORY command. The CDO MOVE REPOSITORY command resolves all pointers to the old repository location to indicate the new location. The target location must be an OpenVMS directory that contains no files.

Note

When you use the MOVE REPOSITORY command, you must specify the entire name, *including the device and directory name*, of both location and target files. If you specify only directory names, the move may not be successful.

The following command moves the repository and the database root (.RDB) at SYS$COMMON:[CDDPLUS] to DISK$1:[CORPORATE.MIS]:

```
   .
   .
   .
CDO> MOVE REPOSITORY SYS$COMMON:[CDDPLUS]
cont> TO DISK$1:[CORPORATE.MIS].
CDO>
```

Restrictions

- When the repository you are moving contains definitions that are related to definitions in a remote repository, the MOVE REPOSITORY command can resolve these references only if the network link is viable at the time of execution. If the network link is not viable, the MOVE REPOSITORY command fails and you must try the operation again.
- The move repository operation needs unrestricted access to a repository because it will automatically perform a CDO VERIFY/LOCATION /FIX operation. The account from which you issue the CDO MOVE REPOSITORY command must have SYSPRV or BYPASS privilege, which is required for the verify operation; otherwise, CDO will issue an error message and abort the move repository operation.
- The CDO MOVE REPOSITORY command is not supported for moving a repository to a node that is not in the same cluster (a remote move operation).
- Both the source and the target disk must be accessible to the system you are logged in to for the move operation.

1.3.2 Moving a Repository from One System to Another System Not on the Same Cluster

This section describes how to move a repository from one system to another system that is not on the same cluster. This is considered a remote move operation. It is recommended that you review all the restrictions in Section 1.3.2.6 before you perform this operation.

How Distributed Repositories Are Affected by a Remote Move

There are basically four different ways to perform a remote move. Each way involves using the combined OpenVMS and RMU backup, copying or loading from media, and using OpenVMS and RMU restore operations. In some situations, these steps are followed by the CDO VERIFY command. In other situations, such as when you are moving distributed repositories, the CDO VERIFY command cannot be used and you must manually change the external references in the CDD$DATABASE.RDB file using SQL.

Oracle CDD/Repository stores the location of a repository's self-reference and any external repositories that the repository references, inside the CDD$DATABASE.RDB file. If all repositories linked to each other through external references are not accessible through the network during or after the move operation, or if linked repositories have been moved simultaneously, there is no way to resolve the external references between them through the CDO VERIFY command.

If the verify operation cannot find the referenced repositories where it expects to find them (because they have been moved, are not on the network, or network proxy accounts are not set up correctly), verify removes the external references. This makes objects that depended on these references unusable. There is no way to relink the objects after the link has been broken. Use caution when you move distributed repositories.

Version Number Requirements for a Remote Move

The version of Oracle Rdb on the target system must be at the same version or at a later version than that of the source system.

If you are moving distributed repositories, then all linked repositories must be at compatible versions of Oracle CDD/Repository.

Use Table 1–1 to determine the correct solution for your remote move operation.

Table 1–1 Solution Choices for Remote Move Operations

Distributed Repository Has External References?	Moving Multiple Repositories?	Move Solution	Proxy Account Required?
No	-	1	No
Yes	No	2	Yes
Yes	Yes (one at a time)	3	Yes

(continued on next page)

Table 1–1 (Cont.) Solution Choices for Remote Move Operations

Distributed Repository Has External References?	Moving Multiple Repositories?	Move Solution	Proxy Account Required?
Yes	Yes (simultaneously)	4	No

The following sections provide step-by-step instructions for each move solution.

1.3.2.1 Move Solution 1

Use this solution to move a single repository that has no external references to any other repositories.

1. Make a backup of the repository you are moving, using the combined OpenVMS and RMU backup procedure, as described in Section 1.1.
2. Move the two backup files (.RBF and .BCK) to the new system. If both the source and target systems are accessible on the network, use the DCL COPY command or the FTSV SPOOL COPY command. Otherwise, load the files onto media and unload them at the new system.
3. Restore the database backup file (.RBF) to the new location, which should be an empty directory. Include the /NOCDD and /DIRECTORY qualifiers.

   ```
   $ RMU/RESTORE/NOCDD/DIRECTORY=new_dev:[new_anchor_dir] -
   _$ backup_file.RBF
   ```

   Press the Return key in response to any prompts for storage_area information.
4. Restore the OpenVMS backup file (.BCK) to the new location, which should be the same directory into which you restored the .RBF file.

   ```
   $ BACKUP/LOG backup_file.BCK/SAVE new_dev:[new_anchor_dir]
   ```
5. Issue the CDO VERIFY/LOCATION/FIX command to verify the repository in its new location.
6. Run the SYS$LIBRARY:CDD$UPGRADE.COM command procedure if you need to upgrade the repository.

1.3.2.2 Move Solution 2

Use this solution to move a single, distributed repository.

Caution

Before you begin this procedure, carefully review the Oracle CDD/Repository Version 6.1 release notes Remote Access Corrections section to ensure that all proxy accounts are properly set up and functioning as expected, login command procedures on default proxy accounts are correct, the versions of Oracle CDD/Repository and Oracle Rdb are compatible, and the test case has run successfully. Otherwise, the verify operation could remove external references rather than update them.

To use this solution, all repositories must be at compatible versions of Oracle CDD/Repository. In addition, all repositories must be available through network proxy accounts during the move operation so that the verify operation can fix the external references. All repositories must be available through network proxy accounts after the move for the repositories to continue to share data. See Section 1.3.2.5 before you begin.

Perform the following steps:

1. Make a backup of the repository you are moving, using the combined OpenVMS and RMU backup procedure, as described in Section 1.1.

   It is recommended that you also make backups of all referenced repositories, as a precaution.

2. Move the two backup files (.RBF and .BCK) to the new system using the DCL COPY command or the FTSV SPOOL COPY command. Or, load the files onto media and unload them at the new system.

3. Restore the database backup file (.RBF) to the new location, which should be an empty directory. Include the /NOCDD and /DIRECTORY qualifiers.

   ```
   $ RMU/RESTORE/NOCDD/DIRECTORY=new_dev:[new_anchor_dir] -
   _$ backup_file.RBF
   ```

   Press the Return key in response to any prompts for storage_area information.

4. Restore the OpenVMS backup file (.BCK) to the new location, which should be the same directory into which you restored the .RBF file.

   ```
   $ BACKUP/LOG backup_file.BCK/SAVE new_dev:[new_anchor_dir]
   ```

5. Issue the CDO VERIFY/LOCATION/EXTERNAL_REFERENCES/FIX command to verify the repository in its new location.
6. Run the SYS$LIBRARY:CDD$UPGRADE.COM command procedure if you need to upgrade the repository.
7. From the system of the old location, use the DCL DELETE command to delete the repository that you moved. Do not use the CDO DELETE REPOSITORY command. Deleting the repository from the old location saves disk space and avoids accidental use of the invalid repository.

1.3.2.3 Move Solution 3

Use this solution to move multiple, distributed repositories. Move one repository at a time.

Caution

Before you begin this procedure, carefully review the Oracle CDD/Repository Version 6.1 release notes Remote Access Corrections section to ensure that all proxy accounts are properly set up and functioning as expected, login command procedures on default proxy accounts are correct, the versions of Oracle CDD/Repository and Oracle Rdb are compatible, and the test case has run successfully. Otherwise, the verify operation could remove external references rather than update them.

To use this solution, all repositories must be at compatible versions of Oracle CDD/Repository. In addition, all repositories must be available through network proxy accounts during the move operation so that the verify operation can fix the external references. All repositories must be available through network proxy accounts after the move for the repositories to continue to share data. See Section 1.3.2.5 before you begin.

If the repositories are not all available on the network during the move operation, you must use Move Solution 4.

Perform the following steps for each repository being moved, *one repository at at time.*

1. Make a backup of the first repository you are moving, using the combined OpenVMS and RMU backup procedure, as described in Section 1.1.
2. Move the two backup files (.RBF and .BCK) to the new system using the DCL COPY command or the FTSV SPOOL COPY command. Or, load the files onto media and unload them at the new system.

3. Restore the database backup file (.RBF) to the new location, which should be an empty directory. Include the /NOCDD and /DIRECTORY qualifiers.

   ```
   $ RMU/RESTORE/NOCDD/DIRECTORY=new_dev:[new_anchor_dir] -
   _$ backup_file.RBF
   ```

   Press the Return key in response to any prompts for storage_area information.

4. Restore the OpenVMS backup file (.BCK) to the new location, which should be the same directory into which you restored the .RBF file.

   ```
   $ BACKUP/LOG backup_file.BCK/SAVE new_dev:[new_anchor_dir]
   ```

5. Issue the CDO VERIFY/LOCATION/EXTERNAL_REFERENCES/FIX command to verify the repository in its new location.

6. Run the SYS$LIBRARY:CDD$UPGRADE.COM command procedure if you need to upgrade the repository.

7. From the system of the old location, use the DCL DELETE command to delete the repository that you moved. Do not use the CDO DELETE REPOSITORY command. Deleting the repository from the old location saves disk space and avoids accidental use of the invalid repository.

8. Perform steps 1 through 7 for each repository separately, repeating the steps for every repository that you move.

1.3.2.4 Move Solution 4

Use this solution to simultaneously move multiple, distributed repositories.

Caution

Use this solution with caution and only when necessary. Follow the steps in this procedure exactly; otherwise the repository will not be usable.

To use this solution, all repositories must be at compatible versions of Oracle CDD/Repository.

Perform the following steps:

1. Make a backup of the each repository you are moving, using the combined OpenVMS and RMU backup procedure, as described in Section 1.1.

   Be sure to give meaningful names to each pair of backup files (.BCK and .RBF), so you can identify them when you copy them to the new system.

2. Move all pairs of backup files (.RBF and .BCK) to the new system.

If both the source and target systems are accessible on the network, use the DCL COPY command or the FTSV SPOOL COPY command. Otherwise, load the files onto media and unload them at the new system.

3. Restore the database backup files (.RBF) to their new locations, which must be empty directories. Include the /NOCDD and /DIRECTORY qualifiers.

```
 $ RMU/RESTORE/NOCDD/DIRECTORY=new_dev:[new_anchor_dir] -
_$ backup_file.RBF
```

   Press the Return key in response to any prompts for storage_area information.

4. Restore the OpenVMS backup files (.BCK) to their new locations, which should be the same directories into which you restored the .RBF files.

```
 $ BACKUP/LOG backup_file.BCK/SAVE new_dev:[new_anchor_dir]
```

   Do not run any CDO VERIFY commands at this time; otherwise, all external references will be lost.

5. Manually modify each repository's self-reference and external references by directly accessing the CDD$DATABASE.RDB file through SQL, and changing the old node and anchor directory locations to the new node and anchor directory locations.

Note

The syntax shown in the following steps is critical for a successful modification and should be followed exactly.

Perform the following steps:

a. Invoke SQL and enter the following commands:

```
$ MCR SQL$
SQL> attach 'filename NEW_DISK:[NEW_ANCHOR_DIR_1]CDD$DATABASE';
SQL> set transaction read write;
```

b. Display a list all repositories referenced in this repository:

```
SQL> select cdd$$a_nad_node_name ,
cont>       cdd$$a_nad_anchor_name from cdd$$o_anchor ;
 CDD$$A_NAD_NODE_NAME
   CDD$$A_NAD_ANCHOR_NAME
 OLD_NODE::
   OLD_DISK:[OLD_ANCHOR_DIR_1]
   >>
   >>
   >>
 OLD_NODE::
   OLD_DISK:[OLD_ANCHOR_DIR_2]
   >>
   >>
   >>
 OLD_NODE::
   OLD_DISK:[OLD_ANCHOR_DIR_3]
   >>
   >>
   >>
3 rows selected
SQL>
```

The first repository listed is always this repository's self-reference. In this example it is:

```
 OLD_DISK:[OLD_ANCHOR_DIR_1]
```

c. Modify the repository's self-reference by carefully changing each location from the old location to the new location. Include the node and directory specification *exactly* as it appears in the listing. Use all uppercase letters, use the double colons (::) after the node name, and put all names (both the old and new) within single quotes, as shown.

```
SQL> update cdd$$o_anchor
SQL> set cdd$$a_nad_node_name = 'NEW_NODE::',
cont> cdd$$a_nad_anchor_name = 'NEW_DISK:[NEW_ANCHOR_DIR_1]'
cont> where cdd$$a_nad_anchor_name = 'OLD_DISK:[OLD_ANCHOR_DIR_1]';
```

d. Modify the external references. In this step, the two external references are being moved simultaneously from:

   `OLD_NODE::OLD_DISK:[OLD_ANCHOR_DIR_2]` and
   `OLD_NODE::OLD_DISK:[OLD_ANCHOR_DIR_3]` to
   `NEW_NODE::NEW_DISK:[NEW_ANCHOR_DIR_2]` and

NEW_NODE::NEW_DISK:[NEW_ANCHOR_DIR_3], respectively.

```
SQL> update cdd$$o_anchor
SQL> set cdd$$a_nad_node_name = 'NEW_NODE::',
cont>  cdd$$a_nad_anchor_name = 'NEW_DISK:[NEW_ANCHOR_DIR_2]'
cont> where cdd$$a_nad_anchor_name = 'OLD_DISK:[OLD_ANCHOR_DIR_2]';

SQL> update cdd$$o_anchor
SQL> set cdd$$a_nad_node_name = 'NEW_NODE::',
cont>  cdd$$a_nad_anchor_name = 'NEW_DISK:[NEW_ANCHOR_DIR_3]'
cont> where cdd$$a_nad_anchor_name = 'OLD_DISK:[OLD_ANCHOR_DIR_3]';
```

e. Display a list again of all the repositories referenced in this repository; check that they were updated correctly before you commit the transaction.

```
SQL> select cdd$$a_nad_node_name ,
cont> cdd$$a_nad_anchor_name from cdd$$o_anchor ;
 CDD$$A_NAD_NODE_NAME
   CDD$$A_NAD_ANCHOR_NAME
 NEW_NODE::
   NEW_DISK:[NEW_ANCHOR_DIR_1]
   >>
   >>
   >>
 NEW_NODE::
   NEW_DISK:[NEW_ANCHOR_DIR_2]
   >>
   >>
   >>
 NEW_NODE::
   NEW_DISK:[NEW_ANCHOR_DIR_3]
   >>
   >>
   >>
3 rows selected
SQL>
SQL> commit;
SQL> disconnect default;
SQL> exit;
```

6. Repeat steps a through e for all distributed repositories that you have moved.

   Do not issue any CDO VERIFY commands in any of these repositories until after all repositories have been moved and modified.

7. After you have modified the CDD$DATABASE.RDB file for each repository you have moved, issue the CDO VERIFY/LOCATION/EXTERNAL_REFERENCES/NOFIX command to check that you have correctly updated all the location pointers in all the repositories.

Do not use the /FIX qualifier if there are errors in the pointers. Rather, invoke SQL again and check your modifications. Be sure you have entered the names in uppercase letters, used the double colon (::), and single quotes where needed. Make any necessary corrections, exit SQL, and retry the CDO VERIFY/LOCATION/EXTERNAL_REFERENCES/NOFIX command.

1.3.2.5 Distributed Repository Move Operations Requiring Remote Network Proxy Access

This section applies if you plan to use Move Solution 2 and Move Solution 3. Both of these move solutions require remote access to external references during and, possibly after, you move one or more distributed repositories.

Before you use Move Solution 2 or Move Solution 3, there are steps you must perform to prepare for the move operation. The steps are provided in this section. After you perform these steps, run the remote move test sample in Example 1–1. Resolve all problems in the test move before you move your repositories.

1. Check version number compatibility

   The Oracle CDD/Repository Version 6.1 release notes describe several remote repository access corrections and enhancements that have been made in Version 6.1. These corrections and enhancements are also included in Oracle CDD/Repository for OpenVMS VAX Version 5.3-08 (ECO). These changes make Version 6.1 incompatible for remote access with versions of Oracle CDD/Repository that are lower than Version 5.3-08 (ECO). Therefore, if one of the distributed repositories involved in the move is at either Version 5.3-08 or later, then all the repositories must be at Version 5.3-08 or later.

   This is an important consideration if you are moving a repository from a system that is running a lower version of Oracle CDD/Repository (the source) than the version of Oracle CDD/Repository on the system to which you are moving the repository (the target). See Table 1–2 for the compatibility requirements for earlier versions of Oracle CDD/Repository.

Table 1–2 lists the minimum version of Oracle CDD/Repository that is required on the target, based on the version of Oracle CDD/Repository at the source.

Table 1–2 Oracle CDD/Repository Version Compatibility

Version at Source	Minimum Version Required at Target
V4.*n*	V4.*n*
V5.0	V5.0
V5.1 - V5.3-06 (ECO)	V5.1 - V5.3-06 (ECO)
V5.3-08 (ECO) or later	V5.3-08 (ECO) or later

2. Set up proxy accounts

   Default proxy accounts must be set up on each system involved in the remote access. Issue the CDO SHOW REPOSITORIES command from each of the distributed repositories.

   You must set up a proxy account for each of the referenced repositories that was displayed by the SHOW REPOSITORIES command. You will also need to set up proxy accounts for the system (or systems) to which you are moving the repository.

   For information on setting up proxy accounts, invoke the AUTHORIZE utility and type HELP ADD/PROXY. Be sure you use the /DEFAULT qualifier when you set up these proxy accounts.

3. Check LOGIN.COM files

   The LOGIN.COM command procedures for the default proxy accounts that you set up will be invoked during any remote access. Therefore, you must examine all of the LOGIN.COM files to ensure that adequate privileges and other setup requirements are run automatically. For example, if your remote system has an Oracle Rdb multiversion environment, a specific version is required to activate the proper RMU and Oracle Rdb images on that system.

Remote Move Operation Test Sample

Example 1–1 creates two empty repositories on a Version 5.3-09 source node (old node), links them by creating external references to each other, moves one of the repositories from the source node to a Version 6.1 target node (new node), verifies, and then upgrades the moved repository.

This example assumes that you have already set up your remote default proxy accounts and have verified that the LOGIN.COM files at these accounts are correct for remote access. See the Oracle CDD/Repository Version 6.1 release notes for more information.

Substitute old_node/old_anchor in the sample for the actual location and anchor name of the old node, and new_node/new_anchor in the sample for your new node location and anchor name.

It can be difficult to troubleshoot remote access failures, because the error messages do not always accurately reflect the problem. If errors occur when you perform the verify operation (when the remote access occurs), review the Oracle CDD/Repository Version 6.1 release notes section on Remote Access Corrections for troubleshooting tips.

It is recommended that you use the sample in Example 1–1 to test a remote move operation before you move your repositories.

This sample contains two repositories that are distributed, and shows how to move one of the repositories to a remote node. The systems are set up as follows:

Source node = OLD_NODE
Oracle CDD/Repository = Version 5.3-09
Oracle Rdb = Version 5.1

Remote target node = NEW_NODE
Oracle CDD/Repository = Version 6.1
Oracle Rdb = Version 6.1

Example 1–1 Remote Test Sample

```
OLD_NODE> crea/dir OLD_DISK:[OLD_ANCHOR_1]
OLD_NODE> crea/dir OLD_DISK:[OLD_ANCHOR_2]
OLD_NODE>

OLD_NODE> REPOSITORY OPERATOR
Welcome to CDO V2.3
The CDD/Repository V5.3 User Interface
Type HELP for help
CDO> define repos OLD_DISK:[OLD_ANCHOR_1].
CDO> define repos OLD_DISK:[OLD_ANCHOR_2].

CDO> set def OLD_DISK:[OLD_ANCHOR_1]
CDO> define field  fld_1 datatype text size 25.
```

(continued on next page)

Example 1–1 (Cont.) Remote Test Sample

```
CDO> set def OLD_DISK:[OLD_ANCHOR_2]
CDO> define record rec_1. OLD_DISK:[OLD_ANCHOR_1]fld_1. end.

CDO> set def OLD_DISK:[OLD_ANCHOR_1]
CDO> sho repos

 The following REPOSITORIES are referenced:

    OLD_NODE::OLD_DISK:[OLD_ANCHOR_2]

CDO> set def OLD_DISK:[OLD_ANCHOR_2]
CDO> sho repos

 The following REPOSITORIES are referenced:

    OLD_NODE::OLD_DISK:[OLD_ANCHOR_1]

CDO> exit
OLD_NODE>

! Move OLD_DISK:[OLD_ANCHOR_2] to remote node NEW_NODE.

! Make backups of OLD_DISK:[OLD_ANCHOR_2].

OLD_NODE> rmu/backup OLD_DISK:[OLD_ANCHOR_2]cdd$database v53_dict2.rbf
OLD_NODE> backup/exclude=(.snp,.rdb,.rda)
_From: OLD_DISK:[OLD_ANCHOR_2...]
_To:   v53_dict2.bck/sav

! Copy the two backup files to NEW_NODE.

NEW_NODE>  copy
_From: OLD_NODE::OLD_DISK:[OLD_LOCATION]v53_dict2.rbf,v53_dict2.bck
_To:   NEW_NODE::NEW_DISK:[NEW_LOCATION]*

! Restore the two backup files into an empty directory on NEW_NODE.

NEW_NODE> crea/dir NEW_DISK:[NEW_ANCHOR_2]

NEW_NODE> rmu/restore/nocdd/dir= NEW_DISK:[NEW_ANCHOR_2]
_Backup: V53_DICT2.RBF
_Storage_area:
%RMU-I-AIJRSTAVL, 0 after-image journals available for use
%RMU-I-AIJISOFF, after-image journaling has been disabled
%RMU-I-LOGCONVRT, database root converted to current structure level
%RMU-S-CVTDBSUC, database NEW_DISK:[NEW_ANCHOR_2]CDD$DATABASE.RDB;1
  successfully converted from version V5.1 to V6.1
%RMU-I-CVTCOMSUC, CONVERT committed for
  NEW_DISK:[NEW_ANCHOR_2]CDD$DATABASE.RDB;1 to version V6.1
%RMU-W-USERECCOM, Use the RMU RECOVER command. The journals are not
  available.
```

(continued on next page)

Example 1–1 (Cont.) Remote Test Sample

```
NEW_NODE> backup/log V53_DICT2.bck/sav NEW_DISK:[NEW_ANCHOR_2]
%BACKUP-S-CREATED, created NEW_DISK:[NEW_ANCHOR_2]00000000.30000000;1
%BACKUP-S-CREATED, created
   NEW_DISK:[NEW_ANCHOR_2]30000000CDD$PROTOCOLS.40000000;1
%BACKUP-S-CREATED, created
   NEW_DISK:[NEW_ANCHOR_2]CDD$DIRECTORY.CDD;1
%BACKUP-S-CREDIR, created directory
   SYS$SYSDEVICE:[CDDSYSTEM.UPG5361_DICT2.CONTEXTS]
%BACKUP-S-CREDIR, created directory
   SYS$SYSDEVICE:[CDDSYSTEM.UPG5361_DICT2.DELTAFILES]
%BACKUP-S-CREDIR, created directory
   SYS$SYSDEVICE:[CDDSYSTEM.UPG5361_DICT2.PARTITIONS]

! Now complete the move with a CDO verify and upgrade.

! Default proxy accounts MUST already be set up at this point for
! NEW_NODE on OLD_NODE and for OLD_NODE on NEW_NODE.  The default
! account's LOGIN.COM files should be correct for remote access.
! This point is critical - if the proxy accounts are not correctly
! in place, performing the verify operation can delete the external
! references rather than update them.

! Fix the location and external references with the CDO VERIFY command.

NEW_NODE> REPOSITORY OPERATOR
Welcome to CDO V6.1
The CDD/Repository V6.1 User Interface
Type HELP for help
CDO> verify/location/external_references/fix NEW_DISK:[NEW_ANCHOR_2]
CDO> EXIT

! Perform an upgrade from Version 5.3-09 to Version 6.1.

! If you use the CONVERT/REPOSITORY command rather than execute the
! SYS$LIBRARY:CDD$UPGRADE.COM command procedure, be sure that the
! CONVERT/REPOSITORY command is the first command you issue in the
! next CDO session.

NEW_NODE> @SYS$LIBRARY:CDD$UPGRADE

                    CDD$UPGRADE.COM

        This procedure is used to upgrade repositories that were created
        under a previous version of CDD/Plus or CDD/Repository.


        Please enter the OpenVMS anchor of the repository to be upgraded.

        DEVICE:[DIRECTORY] : NEW_DISK:[NEW_ANCHOR_2]

        Are you satisfied with the backup of your repository? [Y/N] (N): Y
```

(continued on next page)

Example 1–1 (Cont.) Remote Test Sample

```
        What version is this repository? [ V50 | V51 | J51 | V53 | V61]
        [V61] : V53
        There are no more questions.
        Converting repository NEW_DISK:[NEW_ANCHOR_2] to V6.1...
    %CDO-I-UPGRADE_SUCCEED, dictionary successfully upgrade to new protocols
        CDD$UPGRADE completed successfully.
```

1.3.2.6 Restrictions for Moving a Repository to Another System

The following restrictions apply if you are moving a repository from one system to another system that is not on the same cluster:

- Moving a repository to a different location that is not on the same cluster (performing a remote move operation) is not supported by the CDO MOVE REPOSITORY command.
- You need to perform additional steps if the repository you are moving contains the following:
  - Binary objects (instances of MCS_BINARY type and its subtypes), where the MCS_storeType property has a value of MCS_STORETYPE_EXTERNAL

    Use the CDO DIRECTORY/TYPE=MCS_BINARY command to determine if you have binary types in your repository.
  - User-defined methods (instances of the MCS_METHOD type), where the MCS_funcType property has a value of MCS_METHOD_EXTERNAL_CODE.

    Use the CDO commands SHOW GENERIC MCS_BINARY and SHOW GENERIC MCS_METHOD to locate binary and method objects in your repository.

  Perform these steps:

  1. Move the referenced files to the system where you moved the repository.
  2. If you are using a different directory hierarchy on your new system, it may be necessary to change the value of the MCS_storedIn property to point to the new file location. Use the CDO CHANGE FILE_ELEMENT command to change the values. In addition, if the referenced file is an executable, and you are moving the repository from an OpenVMS VAX system to an OpenVMS Alpha system, you

must port the image or use the VEST utility and vest the image. See the OpenVMS documentation for more information.

3. If you are moving a repository from an OpenVMS VAX system to an OpenVMS Alpha system, you must port or vest the image that contains the external method code. (The image that must be ported or vested is specified as the value of the MCS_application property.)

- You must have the same values for the rights identifiers CDD$SYSTEM and CDD$EXTENDER on both systems. In addition, the RIGHTSLIST.DAT files should contain the same values for all identifiers and owner UICs used on repository objects to ensure you have the same access on both systems.

  Check the value of the CDD$SYSTEM identifier from a privileged account on both the source and target machines. Use the following commands:

```
$ SET DEFAULT SYS$SYSTEM
$ MCR AUTHORIZE
UAF> SHOW /ID CDD$SYSTEM
```

Caution

Do not change the value for CDD$SYSTEM if there are viable repositories on the target system; otherwise, the repository will have privilege access problems. If this happens, you must change the protection on the repositories you moved after you move them. For information on correcting repository protection, see *Using Oracle CDD/Repository on OpenVMS Systems*.

  If the hex value for CDD$SYSTEM is not the same on both machines, you must change it or modify the protection on the repository after you move it.

  If the target node has no existing repositories that are being preserved, the CDD$SYSTEM identifier can be removed and replaced with the value from the source system. To do this, use the following commands:

```
MCR> REMOVE /ID CDD$SYSTEM
MCR> ADD /ID CDD$SYSTEM /VALUE=IDENTIFIER:%Xhex_value
```

  Add only the last five characters of the hex value. The beginning 800 number is not used.

- The version of Oracle Rdb on the target system must be the same version or a later version than that of the source system.

  All distributed repositories must use compatible versions of Oracle CDD/Repository. See Section 1.3.2.5.

1.3.3 Moving a DMU Dictionary to Another System

Perform the following steps to move a DMU dictionary from one system to another system:

1. Locate any subdictionaries that are part of your logical dictionary structure. On the system from which you are moving the dictionary, use the following DMU LIST/TYPE=SUBDICTIONARY command:

   ```
   $ mcr dmu
   DMU> set default cdd$top
   DMU> list/type=subdictionary cdd$top>
   ```

   The listing includes the OpenVMS file specification of all subdictionaries. For example:

   ```
      .
      .
      .
   PERSONNEL <SUBDICTIONARY>: DB3:[CASADAY.CDD]PERS.DIC
      .
      .
      .
   DMU> exit
   ```

2. Copy the CDD$DICTIONARY:CDD.DIC dictionary from the old system to the new system using the DCL COPY command.

3. Change the definition of the CDD$DICTIONARY logical name in the SYS$STARTUP:CDDSTRTUP.COM procedure to point to the directory on the new system where you placed the dictionary file.

4. Copy all subdictionaries (that were listed when you performed step 1) from the old system to the new system using the DCL COPY command.

   Be sure to place the subdictionary files in the same location within the dictionary structure on the new system as on the old system.

   Otherwise, you must rename each subdictionary you copy to point to the new location, as follows:

   ```
   DMU> RENAME/SUBDICTIONARY=file-specification path-name
   ```

   For example:

   ```
   DMU> RENAME/SUBDICTIONARY=DISK7:[DICT.CDD]PERS.DIC PERSONNEL
   ```

1.4 Deleting a Repository

The CDO DELETE REPOSITORY command allows you to delete all the elements in a repository, and to delete the repository itself. You must have DELETE privilege to the repository.

Caution

When Oracle CDD/Repository deletes a complete repository, all files in the anchor directory are deleted, including the database files. Therefore, you should store only files created by Oracle CDD/Repository in an OpenVMS directory that is dedicated to a repository. If you do store other files in the OpenVMS directory, CDO deletes these files when you delete the repository.

Before you delete a repository, make sure that it does not contain elements used by another element in a different repository (such as records that are based on fields in another repository). Otherwise, an error occurs when you issue the DELETE REPOSITORY command. To see if an element is used by another element, enter the CDO SHOW REPOSITORIES/FULL command.

If necessary, copy the definitions to another repository and change the anchor and path to the definitions, where appropriate. After you copy the definitions that have relationships to other repositories, you must execute the VERIFY /EXTERNAL_REFERENCES/FIX command on the repository to remove all internal relationships with those repositories.

After you execute the VERIFY/EXTERNAL_REFERENCES/FIX command, issue the SHOW REPOSITORIES/FULL command to confirm that there are no longer any relationships. In some cases, you many need to run the CDO VERIFY/EXTERNAL_REFERENCES/FIX command twice to completely remove the relationships.

Once the relationships are removed, you can delete the repository.

2

Enhancing Repository Performance

This chapter provides tuning concepts to enhance repository performance.

2.1 Performance Concepts

It is important to understand the concepts of system performance, because many variables can affect repository performance, such as:

- Different applications running on the system (system load)
- SYSGEN parameter settings
- Disk usage
- Amount of memory available to an application
- Process quotas
- Logical settings

Although performance expectations differ with applications, what really matters is elapsed time. Total elapsed time is affected by the size of the processor used, the number of I/O requests, and the amount of page faults. Therefore, timing will vary, depending on the system used and other processes running simultaneously on the system.

Memory influences the amount of page faults. Usually, as more memory is used, the number of page faults increases and the computer takes longer to process the application. You can see the effect of memory on elapsed time by typing a Ctrl/T key sequence and displaying the output, as in the following example:

```
TLE0:: 11:36:57   (DCL)   CPU=00:00:55.95 PF=20417 IO=9550 MEM=191
TLE0:: 11:37:00   (DCL)   CPU=00:00:55.96 PF=20422 IO=9551 MEM=149
```

You need to be aware of these factors to understand application performance because repository performance is discussed in terms of these factors.

2.2 General System Tuning Tasks

For general system tuning, use the Autogen utility (AUTOGEN). You execute the AUTOGEN command procedure when your system is installed, upgraded, or whenever you make significant changes to your workload. To run AUTOGEN, use the following two steps:

1. Enter the following command:

   ```
   $  @SYS$UPDATE:AUTOGEN SAVPARAMS TESTFILES FEEDBACK
   ```

   Review the AUTOGEN report for changes.

2. If you approve the report, reboot the system by entering the following command:

   ```
   $  @SYS$UPDATE:AUTOGEN GENPARAMS REBOOT
   ```

For more details on how to use AUTOGEN, see the OpenVMS documentation on how to set up an OpenVMS system.

You can improve performance through the following tasks:

- Relocate the page and swap files by placing them on a disk that is separate from the repositories.
- Spread the repositories across several disks.
- Increase the power of the processors.
- Distribute the workload of various applications across multiple processors.
- Decrease the startup time of a CDO command by keeping the database root file (CDD$DATABASE.RDB) of your repository open. To do this, enter the following command:

  ```
  $  RMU/OPEN CDD$DATABASE.RDB
  ```

For additional information on measuring and improving system performance, see the OpenVMS documentation set.

The following Oracle Rdb documentation also provides helpful information:

- *Oracle Rdb Guide to Database Maintenance*
- *Oracle Rdb Guide to Database Performance and Tuning*
- *Oracle RMU Reference Manual*

2.3 Performance Optimizations

This section describes performance optimizations that have been incorporated into Oracle CDD/Repository.

- Users attach only once to the repository.

  Because attaching to the database is expensive in terms of elapsed time, interactive repository sessions are optimized to attach the user to the database once. Therefore, remaining in a single CDO session is more efficient than exiting and entering CDO.

- Type definitions are cached only once in the repository.

  Objects are cached in memory throughout the transaction, or throughout the callable interface transaction. Again, remaining in a single CDO session, rather than exiting and reentering CDO, is recommended.

- Information is stored in memory to save time and database I/O requests.

  Less time is spent waiting for objects to be fetched from memory and more time is spent processing. By using longer transactions you can improve performance; however, longer transactions increase contention. Using a series of CDO commands in a single transaction is described in more detail in Section 2.3.3 through Section 2.5.7.

- Directory files are opened only for create and delete operations.

  The directory hierarchy search is based on OpenVMS directory file names. The depth of hierarchy in the directory path does not affect performance.

- The number of names per directory has been optimized.

  This optimization benefits any operation that reads an element by name or modifies the contents of a directory with a large number of entries. It is recommended that you keep directories uncluttered for optimal performance. The recommended maximum number of names in a repository directory is listed in Table 2–1.

- Logical name translations are saved.

  When you use logicals in element names, Oracle CDD/Repository saves the translations for use when the logical name is used again in the same CDO interactive session.

- Large database integrate operations have been optimized.

  The database integrate operation has been optimized in a number of ways, including one load into memory of all record and field definitions used when comparing the metadata in the database and repository.

For example, in one case, the integration of a large database into a nonexistent path that took 1 hour 15 minutes in Version 5.0 now takes less than 30 minutes.

- CDO uses read-only transactions in VERIFY/NOFIX operations.

  CDO uses read-only transactions for VERIFY operations, except when the /FIX, /REBUILD_DIRECTORY, or /COMPRESS qualifiers are specified. Using read-only transactions reduces the amount of locking in the underlying repository database.

- The CDD$MAX_OBJECTS_IN_MEMORY logical name can be used as a tuning method.

  The CDD$MAX_OBJECTS_IN_MEMORY logical can assist in tuning I/O intensive repository applications. Using the RDM$BIND_BUFFERS logical in conjunction with the CDD$MAX_OBJECTS_IN_MEMORY logical further reduces I/O and improves overall performance of the applications.

  Using these logical names as a tuning method to improve performance is explained in more detail in Section 2.5.3.

2.3.1 Internal Operations During a Session

The following is an example of the operations that occur inside Oracle CDD/Repository during a session. Suppose the following CDO command is executed:

```
CDO> DEFINE FLD1 DATATYPE is TEXT SIZE is 30.
```

Oracle CDD/Repository performs the following internal operations:

- Attaches to the database.
- Starts a read/write transaction on the database.
- Loads the type definitions into memory.
- Reads the directory into memory.
- Creates the field in memory.
- Validates and sets the field property values.
- Creates a relationship and a history entry in memory.
- Determines the request needed to store this field.
- Stores the field as a row in a relation in the database.
- Stores the history and relationship in the database.
- Writes the field name to the directory buffer in memory.

- Stores the modified directory in the database.
- Commits the transaction.

Creating a record would have similar steps. For example:

```
CDO> DEFINE RECORD REC1.
cont> FLD1. END RECORD REC1.
```

The repository is already attached to the database and has already loaded the type definitions. To read the objects, Oracle CDD/Repository performs the following steps:

```
CDO> SHOW RECORD REC1
```

- Starts a read-only transaction.
- Finds the name REC1 in the directory (in memory).
- Uses a key stored with the name to find the record.
- Determines the request needed to read this record.
- Reads the record from the database.
- Builds a key for each type of relationship the record can own.
- Determines the request needed to read this relationship.
- Reads the relationship from the database.
- Uses the key stored in the relationship to find the member.
- Determines the request needed to read this field.
- Reads the field from the database.
- Commits the transaction.

2.3.2 CHANGE and DEFINE FIELD Command Enhancements

When you issue the CDO CHANGE FIELD command on an object with multiple relationships to an object in an external repository, Oracle CDD/Repository reads the information from memory and no longer performs unnecessary (and possibly remote) node lookups.

When you issue the CDO DEFINE FIELD command and the object is defined in a repository by using an object with relationships in another repository, CDO no longer copies these objects into the new repository.

In the following example, two repositories [.DICT1] and [.DICT2], are defined.

Dictionary 1 contains one global field B, field F1 based on B, and 500 records (R1 to R500) also based on B. Dictionary 2 contains an additional record, R501, using the same based on field F1 from Dictionary 1.

In previous versions of Oracle CDD/Repository, the result in [.DICT2] was the following:

- Record R501 was created.
- A local copy of field F1 was created to provide fast access to the field.
- 500 local read copies of the records R1-R500, defined in [.DICT1], were created in the new repository, [.DICT2].

Note

The copy of field F1, which is a clone, cannot be updated in [.DICT2]; it must be updated in [.DICT1], the repository in which it was defined.

Now only record R501 is created, and the local copy of F1 is created for fast access to the field. The 500 local read copies of records R1-R500 are not created.

```
$ REPOSITORY
  Welcome to CDO V6.1
  The CDD/Repository V6.1 User Interface
  Type HELP for help
CDO>  define repository [.dict1].
CDO>  define repository [.dict2].
CDO>  set default [.dict1]
CDO>  define field B datatype text size 25.       ! global field
CDO>  define field f1 based on b.
CDO>  define record  r1.
cont> f1. end.
    .
    .
    .
CDO>  define record  r500.
cont> f1. end.
CDO>  set default [.dict2]
CDO>  define record r501.
cont> [.dict1]f1. end.
CDO>  exit
```

Example 2–1 shows the information that CDO displays for these objects when you issue various SHOW commands.

Example 2–1 CDO SHOW Command Examples

```
 $ REPOSITORY OPERATOR
   .
   .
   .
CDO> SHOW DEFAULT
[.DICT2]
    = DEVICE:[USER.DICT2]

CDO> DIRECTORY
 Directory DEVICE:[USER.DICT2]

CDD$PROTOCOLS                                    DIRECTORY
R501(1)                                          RECORD

CDO> SHOW RECORD R501
Definition of record R501
|   Contains field           F1

CDO> SHOW USED R501
Members of DEVICE:[USER.DICT2]R501(1)
|   DEVICE:[USER.DICT1]F1(1)       (Type : FIELD)
|   |   via CDD$DATA_AGGREGATE_CONTAINS

CDO> SHOW USED/FULL R501
Members of DEVICE:[USER.DICT2]R501(1)
|   DEVICE:[USER.DICT1]F1(1)       (Type : FIELD)
|   |   via CDD$DATA_AGGREGATE_CONTAINS
|   |   DEVICE:[USER.DICT1]B(1)        (Type : FIELD)
|   |   |   via CDD$DATA_ELEMENT_BASED_ON

CDO> SHOW USES R501
   .
   .
   .
CDO> SHOW USED/FULL DEVICE:[USER.DICT1]F1(1)
Members of DEVICE:[USER.DICT1]F1(1)
|   DEVICE:[USER.DICT1]B(1)        (Type : FIELD)
|   |   via CDD$DATA_ELEMENT_BASED_ON
```

(continued on next page)

Example 2–1 (Cont.) CDO SHOW Command Examples

```
CDO> SHOW USES/FULL DEVICE:[USER.DICT1]F1(1)
Owners of DEVICE:[USER.DICT1]F1(1)
|   DEVICE:[USER.DICT1]R1(1)       (Type : RECORD)
|   |   via CDD$DATA_AGGREGATE_CONTAINS
    .
    .
    .
|   DEVICE:[USER.DICT2]R501(1)     (Type : RECORD)
|   |   via CDD$DATA_AGGREGATE_CONTAINS
CDO>
```

2.3.3 CDO Commands Improve Performance

A single repository transaction is defined as an individual CDO command, or a series of CDO commands between the START_TRANSACTION and COMMIT (or ROLLBACK) commands. This provides a substantial savings for large command procedures.

For example, defining 100 fields in a single CDO transaction is 80% faster than defining the same number of fields, each in a separate transaction.

Caution

While a transaction is open, locks are held against the repository database. Using the CDO START_TRANSACTION command can reduce the concurrency of the repository database.

The internal operations that occur after the CDO DEFINE RECORD command is executed are similar to those described in Section 2.3.1.

```
    .
    .
    .
CDO> START_TRANSACTION.
CDO> DEFINE RECORD REC2.
cont> FLD1. END RECORD REC2.
    .
    .
    .
CDO> SHOW RECORD REC2
CDO> COMMIT
```

In this case, the repository is already attached to the database and has already loaded the type definitions. The CDO SHOW command executes only the following steps to read the record:

1. Finds the name REC2 in the directory (in memory).
2. Finds the name REC2, the relationship to FLD1, and FLD1 (loaded in memory when the record was defined).

When you use the CDO START_TRANSACTION and COMMIT commands, the overhead that is associated with these commands is incurred once in the repository and once in the database, rather than once for each CDO command between the START_TRANSACTION and COMMIT commands. Objects are retrieved from memory instead of from disk.

2.4 Tasks for the Repository Administrator

The repository administrator can take an active role in designing the repository and maintaining the optimal performance of the repository. Different applications require different repository design trade-offs to take advantage of the specifics of the application.

Designing a repository involves:

- Designing the repository structure
- Adjusting the working set of the individual user
- Managing the physical repositories

2.4.1 Designing the Repository Structure

The repository contains files for the directory hierarchy, and binary and journal files. This repository structure is now implemented on an Oracle Rdb relational database. Directory contents are stored in the relational database for reliability.

Designing the repository can involve several tasks, such as creating a logical, understandable directory structure, choosing the proper location of physical repositories, implementing a single repository structure on all nodes within the logical framework, or organizing objects within the structure by function.

For optimal performance, when you make design decisions about where the data will reside, it is recommended that you limit the number of names in a repository directory to those listed in Table 2–1.

Table 2–1 Number of Names Defined

Oracle CDD/Repository Version	Recommended Maximum Number
5.0	200
5.1 or higher	500-1000

For example, a group that was performing an ACMS build noticed that a repository directory contained over 2000 definitions. The group moved their critical data to a different directory and cut their repository elapsed time on the build operation by 50%.

2.4.2 Adjust the Working Set of the User

The working set of the individual user affects performance. If the working set is too low for the types of applications that are running, page faulting increases and performance is degraded.

Table 2–2 shows the minimum working set quotas (WSQUO) and paging file limit (PGFLQUOTA) that are recommended.

Table 2–2 Recommended Quotas

Quota	Recommended Minimum Setting
Working Set Quota Minimum	4,096
Working Set Quota, Large Operations	10,240
Memory Page File Limit	40,000

2.4.3 Managing the Physical Repositories

The repository administrator must consider when to distribute and when to replicate data. Users benefit by a design that combines both distributed data (linked repositories that share data) and replicated data (separate repositories).

The advantages and disadvantages of linked repositories and separate repositories must be weighed when seeking improved performance in a repository application. As repository administrator, you should replicate data that does not change often, and distribute other data so that you can easily manage changes in a few places. Ideally, the amount of distributed data should be balanced with the amount of replicated data.

2.4.3.1 Linked Repositories

An advantage of linked repositories is that they allow central modification of shared definitions. CDO provides the BASED ON field property that lets you create local copies of the commonly used definitions. This means that read access to the shared/distributed data is local.

However, linked repositories require access to all the repositories for modifications. Because data is distributed and all repositories must be available, updates are expensive. Performance is significantly degraded if there are too many linked repositories.

In addition, performing backups on linked repositories becomes more complicated, since all repositories must be backed up at the same time to obtain a synchronous snapshot of the environment.

2.4.3.2 Separate Repositories

Separate repositories that contain replicated data offer faster access to definitions because of local availability and reduced contention. In addition, separate repositories let you phase in modifications of shared definitions, and separate repositories are easier to back up because they are not dependent on other repositories. However, too much replicated data can be a management nightmare.

2.5 Maintaining the Repository

You maintain repository application performance by analyzing the repository design with database analysis tools, and by monitoring and tuning available system resources.

Maintaining the repository is an ongoing process. It involves monitoring I/O performance, and identifying and resolving repository database performance problems.

2.5.1 Monitoring Repository I/O Performance and Locking

Use the OpenVMS Monitor utility (MONITOR) to monitor I/O performance by collecting data from a system that is running or from a previously created recording file.

During the first few weeks that Oracle CDD/Repository is installed, check the actual number of locks your system is using with the OpenVMS Monitor utility:

```
$ MONITOR LOCK
```

This displays the maximum number of locks outstanding during the monitor period. You can use this value to fine-tune the LOCKIDTBL_MAX and RESHASHTBL system parameters.

For a complete description of the Monitor utility, the LOCKIDTBL_MAX and RESHASHTBL parameters, and tuning information, see the OpenVMS documentation.

Use Oracle RMU™ to display database information that can help you identify repository performance problems. Oracle RMU displays current information about security audit characteristics, version numbers, active databases, active database users, active recovery units, or database statistics related to database activity, such as locking and contention on your node.

For example:

```
$  RMU/SHOW STATISTICS disk:[anchor_dir]CDD$DATABASE
```

For more information, see the *Oracle RMU Reference Manual*.

2.5.2 Moving Repositories Off the System Disk

If repositories reside on the system disk, often contention on the disk degrades performance. Therefore, you can greatly improve performance by moving repositories off the system disk. To move the repositories, use the following CDO command:

```
CDO> MOVE REPOSITORY [qualifier] disk:[anchor_dir_1]
cont>  TO disk:[anchor_dir_2].
CDO>
```

Using multifile CDD$DATABASE does not improve performance significantly when the size of your repository database is constant. However, you can spread repository files over multiple disks for performance improvements.

If I/O is the bottleneck, try placing the recovery unit journal file (CDD$DATABASE.RUJ) and the snapshot file (CDD$DATABASE.SNP) on separate disks. See Section 2.5.7 for more information.

2.5.3 Trading I/O for Memory

The most effective tuning method for improving performance in repository applications is trading I/O operations for memory. The two logical names, CDD$MAX_OBJECTS_IN_MEMORY and RDM$BIND_BUFFERS, control the size of memory cache in the repository and the database.

These logical names also affect access time for reading objects for SQL INTEGRATE operations and verifications of large databases, and the deletion of an Oracle CODASYL DBMS™ schema.

The following sections describe how using these logical names can improve performance.

2.5.4 Controlling the Number of Objects in Memory

The CDD$MAX_OBJECTS_IN_MEMORY logical name controls the number of objects cached in memory per session, reduces the rereads from memory, and saves up to 50% of the CPU time. An object can be:

- A field
- A record
- A relationship between a record and a field
- A history entry and the relationship between a record and one of its history entries

Increasing the number of objects held in memory reduces the amount of I/O performed by a large application, such as a large database integrate operation.

The number of objects in memory should be increased for SQL INTEGRATE operations of large databases.

In Version 4.3, the number of objects that could be cached was unlimited. In Oracle CDD/Repository Version 5.*n* and higher, you can limit the number of objects that can be cached. The default is 10,000, but you can change this number by defining the CDD$MAX_OBJECTS_IN_MEMORY logical name. For example, the number might be reduced to between 100 and 1000 objects for small operations like CDO commands, or increased to 100,000 for large integrate operations.

Each repository application needs to balance sufficient process memory with system memory. Because process memory is a page file quota (PGFLQUOTA) and system memory is a system parameter (VIRTUALPAGECNT), reducing the number of objects in memory lets you work with a smaller page file quota.

2.5.5 Controlling Pages in Cache Memory

Page faults increase with increased memory usage. The RDM$BIND_ BUFFERS logical name controls the number of pages the database caches in memory and reduces the overall direct I/O by an order of magnitude. Although reducing this number is not necessary or helpful for repository applications (the default is 50), increasing the number can *dramatically* enhance performance for large operations such as SQL integrate operations of large databases.

Using the RDM$BIND_BUFFERS logical name with the CDD$MAX_ OBJECTS_IN_MEMORY logical name also:

- Increases the number of buffers that the underlying database holds in memory
- Further reduces I/O
- Improves overall performance of the applications

For example:

```
$ DEFINE CDD$MAX_OBJECTS_IN_MEMORY 20000
$ DEFINE RDM$BIND_BUFFERS 200
```

Example 2–2 shows how I/O is traded for memory during an SQL integrate operation that involves a database containing 100 tables. The commands show how to change a domain that is used in 70 of the tables in the database. Note that the INTEGRATE DOMAIN feature is new in Version 6.1.

Example 2–2 Trading I/O with Memory During an Integrate Operation

```
$ MCR SQL$
SQL> ATTACH 'FILENAME TRDBRESTORE';
SQL> ALTER DOMAIN CDD$$A_NAME IS CHAR(500);
SQL> COMMIT;
SQL> EXIT
```

After completing the change in the domain, integrate the change into the repository, as follows:

```
$ MCR SQL$
SQL> ATTACH 'PATHNAME TRDBRESTORE';
SQL> INTEGRATE DOMAIN CDD$$A_NAME ALTER DICTIONARY;
SQL> COMMIT;
SQL> EXIT
```

Next, tune the database by increasing the values of CDD$MAX_OBJECTS_IN_ MEMORY and RDM$BIND_BUFFERS logical names.

Performing an SQL integrate operation of the changed domain where both the values of CDD$MAX_OBJECTS_IN_MEMORY and RDM$BIND_BUFFERS logical names were increased reduces CPU time, direct I/O, and overall elapsed time, as shown in Table 2–3.

Table 2–3 SQL INTEGRATE Changed Domain

RDM$BIND_ BUFFERS Value	CDD$MAX_ OBJECTS_IN_ MEMORY Value	Direct I/O	Page Faults	CPU (Minutes)	Elapsed Time (Minutes)
50	50	11,651	57,868	6:08	9:46
200	50	1,142	73,043	5:47	6:37
200	20000	1,111	93,098	3:27	4:17

Notice that page faults increase when both the CDD$MAX_OBJECTS_IN_ MEMORY and RDM$BIND_BUFFERS logical names are increased. You must adjust the working set of the user to reduce page faults. Refer to Section 2.4.2 for information on adjusting the working set.

2.5.6 Using the CDD$WAIT Logical Name

The CDD$WAIT logical name controls how Oracle CDD/Repository handles lock conflicts. If CDD$WAIT is not defined, there are no restrictions on access; read and write requests can access a repository at any time.

PROTECTED locking forces single threading of repository writers; it also allows long batch jobs to complete.

EXCLUSIVE locking single threads all users and is the same as turning off the snapshots. For this reason, exclusive locking is for single-user repositories. Exclusive locking also restricts the repository for maintenance operations. Only one access to the repository is allowed; read and write requests are queued and all operations are serialized.

Caution

Because CDD$WAIT is a process logical name, all repositories you access are affected by how CDD$WAIT is defined.

2.5.7 Improving the I/O Performance of Repository Database

The following operations can be performed to effectively tune the repository database and make the application run under optimal conditions:

- Move the database to multiple disks to reduce contention
- Reduce database extensions to improve I/O performance
- Reduce snapshots of the database to improve I/O performance

- Reduce index node depth

2.5.7.1 Moving Repository Database to Multiple Disks

To improve I/O performance, spread the repository database over multiple disks to reduce disk contention and reduce I/O.

Note

Moving the database will dramatically increase the complexity of performing backup operations.

There are three options you can use:

- Move the database file

  Move the database file when I/O contention is a major problem. Enter the following command to move the database file:

  ```
  $  RMU/MOVE_AREA disk:[anchor_dir]CDD$DATABASE CDD$DATA
  _$  /DIRECTORY=[directory_spec]
  ```

- Move the database journal file

  You can move the recovery unit journal file (.RUJ) for an individual process. If you lose the .RUJ file the repository is irrevocably corrupt, and you must revert to a backup. Enter the following command to move the recovery unit journal file:

  ```
  $  DEFINE RDMS$RUJ DISK:[RDM$RUJ]
  ```

- Move the repository binary files

  You can move the repository binary files only when a repository is created. Enter the following command to move the binary files:

  ```
  $ REPOSITORY OPERATOR DEFINE REPOSITORY disk:[anchor_dir] -
  _$ ALTERNATE_ROOT another_disk:[dir].
  ```

2.5.7.2 Reducing Repository Database Extensions

As a repository database grows, it will be extended. Extensions can cause extra I/O operations, which slow performance considerably.

Use the RMU/DUMP command to assemble an overall picture of the Oracle Rdb relational database. For example, the output from an RMU/DUMP operation reveals clues about the number of times the database has been extended.

In Example 2–3 the current physical page count of the storage area ❶ CDD$DATA is 1246, and the database has been extended ❷ five times.

Example 2–3 RMU/DUMP Output Showing Extensions

```
$ RMU/DUMP SYS$COMMON:[CDDPLUS]CDD$DATABASE
   .
   .
   .
Storage area CDD$DATA
    Area ID number is 2
    Storage area page format is mixed
    Filename is SYS$COMMON:[CDDPLUS]CDD$DATA.RDA;1
    Page size is 8 blocks
    Initial data page count was 501
    Current physical page count is 1246 ❶
    Extends are enabled
      - Area has been extended 5 times ❷
      - Extend area by 20%, minimum of 99 pages, maximum
        of 9999 pages
      - Volume set spreading is enabled
    Area has space management pages
      - Current SPAM page count is 1
      - Interval is 4008 data pages
      - Thresholds are 70%, 85%, and 95%
    Snapshots are allowed and enabled
    Snapshot area ID number is 4
    Area last backed up at 7-JULY-1994 12:18:08.27
    Area has never been incrementally restored
```

Example 2–4 shows how to remove the database extensions to improve I/O performance. In this example, a regular repository backup is performed. The database is then exported and deleted. Finally, the database is imported into an empty directory and the physical page count allocation is reset.

Note

If you use the SQL CREATE STORAGE AREA clause with the SQL IMPORT statement, only the storage area parameters you specify are created. Other unspecified parameters assume the default values; they do not inherit the existing values. If you do not use the CREATE STORAGE AREA clause with the IMPORT statement, the default values are inherited from the export file. For more information on storage area parameters, see the *Oracle Rdb SQL Reference Manual*.

The page allocation is determined by the current physical page count. In Example 2–4, 1250 pages are allocated because the current physical page count of the storage area is 1246.

Example 2–4 Removing the Database Extensions

```
$ BACKUP/VERIFY/EXCLUDE=(.RDA,.RDB,.SNP) -
_$ disk:[anchor_dir...] disk:[different-dir]filename.BCK/SAVE

$ RMU/BACKUP disk:[anchor_dir]CDD$DATABASE -
_$ disk:[different-dir]filename.RBF

$ MCR SQL$
SQL> EXPORT DATABASE FILE [CDDPLUS]CDD$DATABASE.RDB
cont> INTO EXPORTEDFILE.RBR;

SQL> DROP DATABASE FILENAME [CDDPLUS]CDD$DATABASE.RDB;

SQL> IMPORT DATABASE FROM EXPORTEDFILE.RBR
cont> FILENAME [CDDPLUS]CDD$DATABASE
cont> CREATE STORAGE AREA CDD$DATA
cont> ALLOCATION IS 1250
cont> PAGE SIZE IS 8 BLOCKS
cont> PAGE FORMAT IS MIXED
cont> THRESHOLDS ARE (70,85,95)
cont> INTERVAL IS 4008;
SQL> EXIT

$ RMU/DUMP [CDDPLUS]CDD$DATABASE
```

The RMU/DUMP output now shows that the area has never been extended. The reduction in I/O will improve the performance of the repository application.

```
Storage area CDD$DATA
   .
   .
   .
 - Initial data page count was 1245
 - Current physical page count is 1250
Extension...
 - Extends are enabled
 - Extend area by 20%, minimum of 99 pages, maximum of 9999
   pages
 - Volume set spreading is enabled
 - Area has never been extended
   .
   .
   .
```

See the *Oracle RMU Reference Manual* for more information on the RMU/DUMP command.

2.5.7.3 Reducing Snapshots of the Database to Improve I/O

A snapshot is a picture of the database for read-only transactions. Read-only repository transactions are:

- CDO SHOW commands
- Language compilations that use the repository
- DMU access of CDO format definitions

By turning off snapshots for single-user repositories, performance is increased by as much as 20% in the extra I/O saved in the update operations. Disabling the snapshot utility on the database benefits a single, read-only user. This operation reduces the lock conflicts of concurrent accessors.

Periodically, reduce the size of the snapshot file by using the /COMPRESS qualifier with the VERIFY command. For example:

```
CDO> VERIFY/COMPRESS disk:[anchor_dir]
```

The following restrictions apply to using VERIFY/COMPRESS:

- You must be the only user of the repository when you issue the VERIFY /COMPRESS command.
- VERIFY/COMPRESS must be the first command you enter after starting a CDO session.

Three snapshot options are available: ENABLED IMMEDIATE, ENABLED DEFERRED, and DISABLED.

2.5.7.3.1 Enabled Immediate Snapshots By enabling snapshot files, read-only users can access snapshot files and avoid record-locking conflicts with users who are updating the database. This option is the default.

```
SQL> ALTER DATABASE FILENAME dict1:CDD$DATABASE
cont>  SNAPSHOT IS ENABLED IMMEDIATE;
```

2.5.7.3.2 Enabled Deferred Snapshots The deferred option saves overhead by writing to the snapshot file only when a read-only transaction is in progress. Use the deferred option when you often have large updates with no readers in the repository. To defer snapshot files, enter the following SQL command:

```
SQL>  ALTER DATABASE FILENAME dict1:CDD$DATABASE
cont>  SNAPSHOT IS ENABLED DEFERRED;
```

2.5.7.3.3 Disabled Snapshots If snapshot files are disabled, attempts to ready the database in read-only mode start a read/write transaction. In general, use the disabled option only with single-user repositories. To disable snapshot files, enter the following SQL command:

```
SQL>  ALTER DATABASE FILENAME dict1:CDD$DATABASE
cont>  SNAPSHOT IS DISABLED;
```

You can also improve performance by scheduling times when you allow read-only access. For example:

1. Enable snapshot files when update activity is low.
2. Allow read-only access for a certain amount of time.
3. Disable snapshot files again until the next scheduled read-only access time.

For more information on controlling snapshot files, refer to the *Oracle Rdb Guide to Database Maintenance*.

2.5.7.4 Reducing Index Node Depth

Use the RMU/ANALYZE command to see how deep the index structure is for CDD$DATABASE.RDB.

If the index structure is too deep (three or four levels are generally too deep), consider moving some entities to different repositories. If that is not appropriate, use the RMU/ANALYZE/PLACEMENT command to see the index path lengths. For each index affected use SQL DROP INDEX then SQL CREATE INDEX.

3

Upgrading a Dictionary or Repository

This chapter describes how to upgrade existing dictionaries or repositories.

The terms dictionary and repository are basically synonymous, and both are accepted by the product. The term dictionary refers to the product prior to Version 5.0; repository is the term used in Version 5.0 and later.

The recommended method of upgrading existing dictionaries or repositories is to use the CDD$UPGRADE.COM command procedure. The CDD$UPGRADE.COM command procedure automatically invokes the appropriate upgrade utility based on the version number you specify. The procedure is provided with the Oracle CDD/Repository installation.

Note

Oracle CDD/Repository Version 6.1 is a minor point release. The version number was changed only to synchronize with Oracle Rdb. Although the last release of Oracle CDD/Repository was Version 5.3 (and subsequent ECOs), the increase from 5 to 6 does not represent a major change in the product's functionality; an upgrade from a Version 5.*n* repository to a Version 6.1 repository is *not* a major upgrade.

If you are performing an upgrade from a Version 5.*n* repository to a later Version 5.*n* or Version 6.1 repository, CDD$UPGRADE.COM runs the CDO CONVERT utility using the CDO CONVERT/REPOSITORY command. This is referred to as a minor upgrade.

If you are performing an upgrade from a Version 4.*n* dictionary, the CDD$UPGRADE.COM procedure runs CDDX, the repository translation utility, using the REPOSITORY EXPORT and REPOSITORY IMPORT commands. This type of upgrade is referred to as a major upgrade.

Each of these utilities can be run manually. Section 3.3 describes how to use the CDO CONVERT utility; Section 3.4 describes how to use CDDX.

3.1 Preparing for an Upgrade of a Dictionary or Repository

Before you upgrade any dictionary or repository, complete the following steps for each dictionary or repository:

1. Ensure that you have SYSPRV or BYPASS privilege, which is necessary to perform an upgrade.
2. Check that your page file quota (PGFLQUOTA) is sufficient for the upgrade procedure.

   If you are performing a major upgrade (from a Version 4.*n* dictionary) the upgrade procedure invokes CDDX. The REPOSITORY EXPORT portion of the upgrade may require increased page file quota (PGFLQUOTA) while it generates the intermediate .CDDX export file.

   Table 3–1 lists the PGFLQUOTA values required for the upgrade.

Table 3–1 Oracle Rdb Database and PGFLQUOTA Values

CDD$DATABASE.RDB Size	PGFLQUOTA Value
0-20,000 blocks	50,000-200,000 pages
20,000-35,000 blocks	70,000-200,000 pages
35,000-55,000 blocks	100,000-200,000 pages
55,000+ blocks	200,000 pages

   During the upgrade, the procedure will estimate the PGFLQUOTA needed to complete the export. If the PGFLQUOTA values are insufficient, the procedure displays an informational message and asks you to cancel the upgrade and increase the PGFLQUOTA.
3. Verify the repository. See Chapter 1 for instructions on verifying a repository.
4. Make sure you have an adequate backup of the repository before you perform the upgrade. See Section 1.1 for the backup procedure.
5. The CDD$UPGRADE.COM command procedure automatically defines the CDD$SEPARATOR logical name as a slash character (/) before running EXPORT.

   However, if you issue the REPOSITORY EXPORT command directly from DCL rather than use the CDD$UPGRADE.COM procedure, and the repository contains elements with a period (.) in their names, redefine CDD$SEPARATOR as a slash character (/) before you export

the repository. Otherwise, the REPOSITORY IMPORT command cannot correctly interpret the names you exported.

For example:

```
$ DEFINE/USER CDD$SEPARATOR "/"
```

Defining CDD$SEPARATOR as a slash character has no negative effect if there are no names containing periods.

6. If you are installing Oracle CDD/Repository, check for the SYS$SHARE:CDA$ACCESS.EXE image. This image is required by the upgrade images that are installed by Oracle CDD/Repository.
7. Oracle CDD/Repository adds new DCL commands during the installation. If you are installing Oracle CDD/Repository, and the process you are in was created before the installation, then log out and log back in before executing the CDD$UPGRADE.COM command procedure, or run CDD$UPGRADE.COM from the installing account.

3.2 Executing CDD$UPGRADE.COM

Invoke the CDD$UPGRADE.COM command procedure, as follows:

```
$  @SYS$LIBRARY:CDD$UPGRADE
```

When CDD$UPGRADE.COM is executed, it displays the following sequence of questions:

1. The procedure asks you to enter the OpenVMS anchor of the repository to be upgraded:

   ```
   Please enter the OpenVMS anchor of the repository to be upgraded.
   ```

   If an export file named CDD$UPGRADE.CDDX is located in this anchor, the command procedure skips the export operation and performs the import operation. Otherwise, it proceeds to the next step.

   If you are installing Oracle CDD/Repository, it is recommended that the compatibility repository be the first repository you upgrade, and that you upgrade it soon after the installation.
2. Make sure you have an adequate backup of the repository before you continue the upgrade. The CDD$UPGRADE.COM procedure will delete the repository and save only the .CDDX export file. The procedure displays the following prompt:

   ```
   Are you satisfied with the backup of your repository? [ Y | N ] (N):
   ```

If you do not have a backup of your repository, reply No (the default) to this question.

Note

Previously, if you were running the CDD$UPGRADE.COM procedure the default response to this question was Yes. In Version 6.1 the default has been changed to No to be consistent with the default in the other upgrade utilities (CDDX and CDO CONVERT/REPOSITORY). You must explicitly enter Yes to continue the upgrade.

If you answer No, the procedure stops and the DCL level prompt ($) is displayed. Perform a regular backup now so you have a backup if the upgrade procedure fails, then rerun the CDD$UPGRADE.COM command procedure. See Section 1.1 for the backup procedure instructions.

If you reply Yes, Oracle CDD/Repository proceeds with the upgrade.

3. The procedure asks you to specify the version of the repository that you are upgrading:

   ```
   What version is this repository?
   ```

   For example, if you are upgrading from Version 5.1 to Version 6.1, enter V51 at the prompt.

4. Choose whether to just export or to export *and* import your repository.

   ```
   Do you ONLY want to perform an EXPORT of this repository? [Y | N] :
   ```

   If you answer No (the default), the upgrade procedure performs both the export and import of your repository.

Caution

If you have extended your repository by modifying the product-supplied protocols through the Oracle CDD/Repository callable interface or through Oracle CDD/Administrator, and you want to perform an upgrade using REPOSITORY EXPORT and REPOSITORY IMPORT, you must answer Yes to the EXPORT ONLY question. Perform the additional steps in Section 3.2.2, then import the file, as described in Section 3.4.2. Otherwise, you will lose the extensions or any instances of extensions.

If you answer Yes at the prompt, the upgrade procedure only exports your repository, creates an intermediate export file called CDD$UPGRADE.CDDX (the default), and places the export file in the anchor directory of the repository being upgraded.

The following situations require that you perform an EXPORT ONLY upgrade:

- If you are using CDD$UPGRADE to convert your OpenVMS VAX repository to an OpenVMS Alpha system, you must answer Yes to this question and perform an EXPORT ONLY upgrade.

  For an OpenVMS Alpha system installation, you must perform the EXPORT ONLY portion of the upgrade on an OpenVMS VAX system, then perform a separate import operation on an OpenVMS Alpha system. Refer to Section 3.4.2 for instructions on how to perform an import operation.

- If you are performing an upgrade using the REPOSITORY EXPORT and REPOSITORY IMPORT commands and your repository has been extended by modifying the product-supplied protocols through the Oracle CDD/Repository callable interface or through Oracle CDD/Administrator.

  Refer to Section 3.2.2 for instructions on upgrading an extended repository.

If you choose to perform an EXPORT ONLY upgrade, you must run the CDD$UPGRADE.COM procedure again to complete the upgrade. If you are installing Oracle CDD/Repository, run the IVP separately after you have completed the second run of CDD$UPGRADE.COM.

5. The upgrade procedure displays the following prompt, asking if you want to grant READ and WRITE access to the current repository to user [*,*].

```
By default, all users [*,*] will have READ-ONLY access to
this repository.  Do you want to give all users READ/WRITE
access to this repository?  [Y | N] :
```

   Previously, Version 4.*n* granted all users READ and WRITE access to the dictionaries.

   If you answer Yes (the default) and the repository you are upgrading is linked to other repositories, the DEFINE PROTECTION command will fail unless all linked repositories have been upgraded. In this case, you must enter the DEFINE PROTECTION command from CDO after you have upgraded all the linked repositories.

The following is an example of how to manually set the protection and the default protections for a repository. Both commands are required to achieve the protection level specified in the CDD$UPGRADE.COM command procedure.

```
$  REPOSITORY OPERATOR

   Welcome to CDO V6.1
   The CDD/Repository V6.1 User Interface
   Type HELP for help
CDO>  define protection for repository <anchor> -
   position 15 ident [*,*] access change+show+define.
CDO>  define protection for repository <anchor>  -
   position 15 ident [*,*] default_access all.
CDO>
```

6. The upgrade procedure then displays the following information:

```
%CDDX-I-CVT_SUCCESS, repository SYS$COMMON:[CDDPLUS] converted.

Exporting repository SYS$COMMON:[CDDPLUS]...
Importing into repository SYS$COMMON:[CDDPLUS]...

%CDDX-I-DATA, ANSI X3.XXX-199Y EXPORT FILE
%CDDX-I-STMP, FILE EXPORTED ON 21-MAY-1992 11:48:51.70
%CDDX-I-DATA, CDD V4.3-2-0 EXPORT of repository SYS$COMMON:[CDDPLUS],
%CDDX-I-DATA,  repository major: 15,
%CDDX-I-DATA,  repository minor: 18,
%CDDX-I-DATA,  MAX index: 0000000000000CE2
%CDD-I-FIXSNP, dictionary SYS$COMMON:[CDDPLUS] snapshot file size has
               been reset
```

Run the CDD$UPGRADE.COM command procedure for each repository.

During a minor upgrade, the procedure creates a temporary context file in the context subdirectory of the anchor. This file, called CDD$CI_CONTEXT.DIR, is deleted after a successful upgrade, unless the upgrade procedure fails or is aborted. If this file remains in the context subdirectory of the anchor, any subsequent attempts to run the upgrade procedure fail with the following error message:

```
MCS-E-DIREXISTS, directory already exists
```

If this error occurs during an upgrade procedure, exit CDO. Check for the CDD$CI_CONTEXT.DIR file in the subdirectory of the anchor. If the file exists, delete it, then retry the upgrade procedure.

3.2.1 Submitting CDD$UPGRADE.COM in Batch Mode

If you are upgrading a Version 4.*n* dictionary, the upgrade procedure can take several hours for a dictionary that is over 100,000 disk blocks in size. If you run CDD$UPGRADE.COM in batch mode, be sure to include all the required input parameters.

The input parameters are as follows:

- Anchor specification—Enter the OpenVMS anchor of the repository to be upgraded.
- Enter `Yes` or `No` for the question: `Are you satisfied with the backup of your repository?`
- Current repository version—Enter one of the following valid version numbers: V4, V5 (for V5.0), V50, V51, V52, V53, or V61.
- Enter `Yes` or `No` for the question: `Do you ONLY want to perform an EXPORT of the repository?`
- Output device—The upgrade procedure will display at your terminal (SYS$OUTPUT) by default; otherwise, output is not displayed. In batch mode, enter " ".
- Enter `Yes` or `No` for the question: `Give all users [*,*] READ/WRITE access?`

The following is an example of how to submit CDD$UPGRADE.COM in batch mode using these input parameters:

```
$  SUBMIT /NOPRINT/NOTIFY/LOG=CDDUPG.LOG -
_$  /PARAM=("SYS$COMMON:[CDDPLUS]","Y","V4","N"," ","N") CDD$UPGRADE.COM
```

3.2.2 Upgrading an Extended Repository

If you are performing a minor upgrade and you have extended your repository, the CDO CONVERT/REPOSITORY command will automatically perform the steps you need. However, if you are using the REPOSITORY EXPORT and REPOSITORY IMPORT operation and you have extended your repository, perform the following steps before you import the metadata into a new repository. If you do not perform these additional steps, you will lose the extensions and any instances of extensions.

1. Check that you have the .CDDX export file of the anchor directory for the repository to be upgraded. If you do not have an export file, run the CDD$UPGRADE.COM command procedure and specify that you want to perform an EXPORT ONLY, as described in Section 3.2, or use CDDX with the REPOSITORY EXPORT command to create an export file, as described in Section 3.4.1.

2. After you export the repository, delete the original repository (the one you exported) using DCL DELETE. Be sure you do *not* delete the .CDDX file.
3. Define a new repository in the same location.
4. Apply the extensions to the new repository.
5. Import the repository using CDDX with the REPOSITORY IMPORT command, as described in Section 3.4.2, or rerun the CDD$UPGRADE.COM procedure.

3.3 Using the CDO CONVERT/REPOSITORY Command

The CDO CONVERT utility was designed for performing a minor upgrade (from a Version 5.*n* repository to a later Version 5.*n* or Version 6.1 repository). When you perform a minor upgrade, the CDD$UPGRADE.COM procedure invokes the CDO CONVERT utility and uses the CDO CONVERT/REPOSITORY command.

Note

If the CDD$CI_CONTEXT.DIR file exists in the context subdirectory of the anchor, the upgrade procedure will fail with the following error message:

```
MCS-E-DIREXISTS, directory already exists
```

If this error occurs during an upgrade procedure, exit CDO. Check for the CDD$CI_CONTEXT.DIR file in the subdirectory of the anchor. If the file exists, delete it, then retry the upgrade procedure.

The steps in this section describe how to manually run the CDO CONVERT/REPOSITORY utility. You must have SYSPRV or BYPASS privilege to run this utility.

1. Make sure you have prepared for the upgrade by following the steps in Section 3.1.
2. Invoke CDO, then enter the CONVERT/REPOSITORY command, as follows:

```
$ REPOSITORY
CDO> CONVERT/REPOSITORY device:[directory]
```

3. Confirm that you have an adequate backup of the repository. The procedure displays the following prompt:

   ```
   Are you satisfied with the backup of your repository? [Y/N] (N):
   ```

   If you reply `No` (the default) the procedure advises you to back up your repository, and the CDO prompt is displayed. Exit CDO, and perform a regular backup of your repository, as described in Section 1.1, then rerun CDO CONVERT/REPOSITORY.

   If you have an adequate backup, enter `Yes`. CONVERT/REPOSITORY upgrades your repository to the current version by updating the protocols to the ones supplied with the new version of Oracle CDD/Repository. If your repository has been extended, the extensions and instances remain intact.

3.4 Using the CDDX Utility

The CDDX translation utility was designed for performing a major upgrade (from a Version 4.*n* dictionary to a Version 5.*n* or Version 6.1 repository). If you are performing a major upgrade, the CDD$UPGRADE.COM procedure invokes CDDX.

Note

Do not use the CDDX utility for backing up, verifying, or moving a dictionary or repository. Refer to Chapter 1 for information on how to perform those operations.

To perform CDDX REPOSITORY EXPORT and REPOSITORY IMPORT you must have SYSPRV or BYPASS privilege.

For online help on export and import operations, type `HELP CDDX` at the DCL level prompt ($).

3.4.1 Exporting with CDDX

The following section describes how to use the REPOSITORY EXPORT command of CDDX to manually create an export file of your repository metadata. After you export the repository, use the REPOSITORY IMPORT command to import the repository metadata from the export file.

1. Make sure you have prepared for the upgrade by following the steps in Section 3.1.

2. From DCL, issue the REPOSITORY EXPORT command, as follows:

   ```
   $ REPOSITORY EXPORT [qualifiers] repository export-filename
   ```

   If you are upgrading a pre-Version 4.3 dictionary, you must add the /CONVERT qualifier to the EXPORT command.

   The following list describes the qualifiers and parameters for the REPOSITORY EXPORT command.

   - /LOG

     The /LOG qualifier produces a line of text for every object in the repository. Use the /LOG qualifier only when you suspect a problem, because a large repository produces a large amount of output. The default is /NOLOG.

   - /VERSION=*n*

     The /VERSION qualifier specifies the version of the repository you are exporting. Valid version numbers are: V4, V5 or V50 (for Version 5.0), V51, V52, V53, and V61. V52 (Version 5.2) is valid for OpenVMS Alpha systems only.

     If you execute the REPOSITORY EXPORT command directly from DCL, the list of valid version numbers is not displayed.

     The default version is /VERSION=V4.

   - /SCHEMA

     The /SCHEMA qualifier specifies the repository metadata or types to be exported. This qualifier allows user extensions or other product-supplied extensions to the metadata to be exported. The default is /SCHEMA; however, if you are performing a major upgrade you cannot override the default.

     If you specify the /NOSCHEMA qualifier, no repository types (protocols) are exported. This means that no user or other product extensions to the metadata will be exported. This speeds up the export process, especially for small repositories.

   - /CONVERT

     Use the /CONVERT qualifier when you are exporting pre-Version 4.3 dictionaries. When you specify /CONVERT, CDDX first converts the Version 4.0, 4.1, or 4.2 dictionary to Version 4.3 before exporting it. The conversion will be committed, regardless of whether the export succeeds. A repository that cannot be converted will not be exported. The default is /NOCONVERT.

If you specify the /CONVERT qualifier, the following informational message will be displayed when the conversion is completed and before the actual EXPORT starts:

```
%CDDX-I-CVT_SUCCESS  repository device:[directory] converted
```

- *repository*

  The OpenVMS anchor of the repository to be exported.

- *export-filename*

  The name of the export file to be created. The default file extension is .CDDX.

The REPOSITORY EXPORT command creates an intermediate export file of your repository using the file name you supply. It places the file in your current directory by default, or you can specify another device and directory. Be sure you do *not* delete this export file.

Note

If your repository is corrupt and you issue the REPOSITORY EXPORT command, EXPORT issues warning messages. Oracle CDD/Repository attempts to export accessible portions of corrupt elements or skips over elements that are too incomplete to be exported.

3.4.2 Importing a Repository from an Export File

The REPOSITORY IMPORT command of CDDX reloads the exported repository data from a repository export file into a new, empty repository. IMPORT assumes that the location you specify to import the export file into contains either an empty directory, or a directory with a new empty repository at the current version. If you specify an empty directory, IMPORT creates a new repository and imports your metadata into it.

Perform the following steps:

1. Check for a .CDDX export file, which was created during the export. If you do not have the export file, run the CDD$UPGRADE.COM command procedure and specify that you want to perform an EXPORT ONLY of the repository, as described in Section 3.2, or use CDDX with the REPOSITORY EXPORT command, as described in Section 3.4.1.
2. Make sure you have an adequate backup of the repository. See Section 1.1 for the backup procedure.

3. Delete the original repository (the one you exported). Do *not* delete the .CDDX file.
4. If you have extended the repository by modifying the product-supplied types (protocols) using the Oracle CDD/Repository callable interface or through CDD/Adminstrator, define a new repository in the anchor from which the original repository was exported.

   If you do not have an extended repository, ignore this step. Skip to step 6 and issue the REPOSITORY IMPORT command.
5. If you have extended your repository, apply any schema modifications to the new repository.

   If you fail to apply the schema changes, you cannot import instances of extensions. When REPOSITORY IMPORT is executed, you will see error messages for all definitions that could not be imported. However, REPOSITORY IMPORT will continue to import all the definitions it can.
6. From DCL, issue the REPOSITORY IMPORT command. Specify the file name of the intermediate .CDDX export file, and the name of the anchor for the repository you just exported and deleted.

   ```
   $  REPOSITORY IMPORT /LOG export-filename disk:[anchor_dir]
   ```

   The only qualifier for REPOSITORY IMPORT is /LOG, which is optional. The /LOG qualifier produces a line of text for every object in the repository. Use the /LOG qualifier only when you suspect a problem, because a large repository produces a large amount of output. The default is /NOLOG.

   CDDX imports the exported repository to the specified anchor. (If the anchor is empty, CDDX creates a new repository for you.) You *must* import your repository into the same OpenVMS anchor directory from which it was exported.
7. After importing the repository, compress the repository to reduce the size of the snapshot file, CDD$DATABASE.SNP, which may become large during the IMPORT operation. Enter the following command:

   ```
   $  REPOSITORY OPERATOR VERIFY/COMPRESS disk:[anchor_dir]
   ```

3.4.3 Sample Upgrade of Version 4.3 Dictionary

The following is a sample upgrade of a Version 4.3 dictionary:

Note

The following output is a sample; the version numbers, dates, and times that display on your system will be different.

```
$ @SYS$LIBRARY:CDD$UPGRADE
            CDD$UPGRADE.COM
This procedure is used to upgrade repositories that were created
under a previous version of CDD/Plus or CDD/Repository.
Please enter the OpenVMS anchor of the repository to be upgraded.
DEVICE:[DIRECTORY] : CDD$1:[CDD.TEST_DICT]
 Are you satisfied with the backup of your repository? [ Y | N ] (N): y
Do you ONLY want to perform an EXPORT of this repository? [ Y | N ]
[ N ] :  n
By default, all users [*,*] will have READ-ONLY access to this
repository.  Do you want to give all users READ/WRITE access to
this repository?  [ Y | N ]
[ Y ] :  n
There are no more questions.
Exporting repository CDD$1:[CDD.TEST_DICT]...
Importing into repository CDD$1:[CDD.TEST_DICT]...
%CDDX-I-DATA, ANSI X3.xxx-199y EXPORT FILE
%CDDX-I-STMP, File exported on 24-AUG-1994 11:48:51.70
%CDDX-I-DATA, CDD V4.3-2-0 EXPORT of repository CDD$1:[CDD.TEST_DICT]
%CDDX-I-DATA,  repository major: 15,
%CDDX-I-DATA,  repository minor: 18,
%CDDX-I-DATA,  MAX index: 0000000000000C8C
CDD$UPGRADE completed successfully.
```

4

Using Oracle CDD/Repository with Oracle Rdb

This chapter describes how to use Oracle CDD/Repository concurrently with Oracle Rdb. Included are examples of how to create, change, and track repository definitions, which can be shared by multiple Oracle Rdb databases.

To understand this chapter you should be familiar with the repository concepts discussed in Chapter 1 of *Using Oracle CDD/Repository on OpenVMS Systems* and the basic concepts of Oracle Rdb.

4.1 Introduction

Oracle Rdb is a relational database product that supports the features of Oracle CDD/Repository. When you create a database, other databases (and potentially other products) can share the same database metadata to reduce redundancies. Sharing metadata provides consistency across databases and other software products as you develop an application.

There are a variety of CDO commands that are useful for Oracle Rdb users, because they let you perform the following tasks:

- Define metadata
- Track metadata
- Set protection on metadata
- Delete or modify metadata

Oracle CDD/Repository keeps track of all users of a particular repository element, so you can analyze the effects of a change to metadata used throughout an application. Whenever metadata is changed, a message about the change is associated with the repository description of any Oracle Rdb databases using the metadata.

Using Oracle CDD/Repository with Oracle Rdb
4.1 Introduction

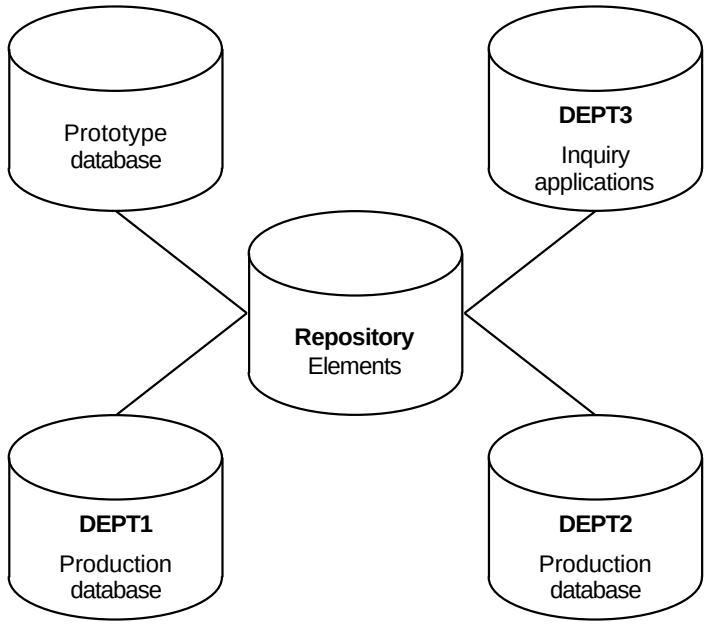
As an Oracle Rdb user, you gain access to repository definitions through CDO to create, modify, delete, or inquire about elements to which you have the appropriate access privileges.

When you use SQL to attach to a database by repository path name, SQL informs you if a message about a repository change is associated with the database. You can read the message from CDO.

Messages warn you that repository definitions may be inconsistent with database metadata. In this way, you can create and maintain standard definitions in Oracle CDD/Repository, which are replicated in any number of Oracle Rdb databases, then share them with other applications.

Figure 4–1 shows a prototype database, two department-wide production databases, and end-user applications that share the same elements through Oracle CDD/Repository.

Figure 4–1 Sharing Repository Definitions Among Database Products



Because the elements reside in the repository and not in any particular database, you can maintain them independently and share them with other databases.

For example, using CDO, you can define a field and base other fields on it, as follows:

```
$ REPOSITORY
Welcome to CDO V6.1
The CDD/Repository V6.1 User Interface
Type HELP for help
CDO> DEFINE FIELD ID_NUMBER
cont> DESCRIPTION IS 'Corporate Standard ABRII'
cont> DATATYPE IS TEXT SIZE IS 5.
CDO> DEFINE FIELD EMPLOYEE_ID
cont> AUDIT IS 'copied from Corporate Standard 5-01-91'
cont>  BASED ON ID_NUMBER.
CDO> DEFINE FIELD STANDARD_DATE
cont> DESCRIPTION IS 'The corporate standard for date'
cont> DATATYPE IS DATE.
```

After defining fields and records in the repository using CDO, you can share them across multiple Oracle Rdb databases by issuing SQL statements to define the fields (which are called domains in SQL) and records (called tables in SQL) from the repository through path names.

Section 4.2 contains more information about this process.

4.2 Creating Shareable Definitions

To create definitions in a repository and make them shareable between the repository and the database or multiple databases, use the following process:

1. Use CDO to define the fields (domains) and records (tables) for a database.
2. Use SQL to define the database.
3. Use SQL to copy the fields (domains) and records (tables) from the repository to the database.
4. Use SQL to create local views, constraints, and indexes. You can mix local and shared database definitions of domains and tables, as needed.
5. Use SQL to include any changed definitions in your repository.
6. Share the definitions, as needed, among other Oracle Rdb databases.

4.2.1 Examples of Creating Shareable Definitions

The following series of examples shows how to use a repository to create definitions for multiple Oracle Rdb databases. These examples assume the following:

- A repository is located at SYS$COMMON:[CDDREP] (the anchor).
- A directory, PERS, where the repository is located, contains the database definitions.

Example 4–1 shows how to use CDO to define a repository field called ID_NUMBER and base another field, EMPLOYEE_ID, on it.

Example 4–1 Defining Shareable Fields in CDO

```
$ REPOSITORY OPERATOR
   .
   .
   .
CDO> SET DEFAULT SYS$COMMON:[CDDREP]PERS
CDO> DEFINE FIELD ID_NUMBER
cont> DESCRIPTION IS 'Corporate Standard ABRII'
cont> DATATYPE IS TEXT SIZE IS 5.
CDO> DEFINE FIELD EMPLOYEE_ID
cont> AUDIT IS 'copied from Corporate Standard 5-01-91'
cont>  BASED ON ID_NUMBER.
```

Define a repository field, STANDARD_DATE, and base other fields on it:

```
   .
   .
   .
CDO> DEFINE FIELD STANDARD_DATE
cont> DESCRIPTION IS 'The corporate standard for date'
cont> DATATYPE IS DATE.
CDO> DEFINE FIELD START_DATE
cont> DESCRIPTION IS 'employee start date'
cont> BASED ON STANDARD_DATE.
```

(continued on next page)

Example 4–1 (Cont.) Defining Shareable Fields in CDO

```
CDO> DEFINE FIELD END_DATE
cont> DESCRIPTION IS 'employee termination date'
cont> BASED ON STANDARD_DATE.

CDO> DEFINE FIELD SALARY_AMOUNT
cont> DATATYPE IS SIGNED QUADWORD SCALE -2.
CDO> DEFINE FIELD BIRTHDAY
cont> DESCRIPTION 'employee date of birth'
cont> BASED ON STANDARD_DATE.
```

Define other repository fields for the database table called EMPLOYEES.

```
CDO> DEFINE FIELD LAST_NAME
cont> DESCRIPTION 'Employee last name'
cont> DATATYPE IS TEXT SIZE IS 14.

CDO> DEFINE FIELD FIRST_NAME
cont> DESCRIPTION 'Employee first name'
cont> DATATYPE IS TEXT SIZE IS 10.

CDO> DEFINE FIELD MIDDLE_INITIAL
cont> DESCRIPTION 'Employee middle initial'
cont> DATATYPE IS TEXT SIZE IS 1.

CDO> DEFINE FIELD ADDRESS_DATA_1
cont> DESCRIPTION IS 'Employee home street address'
cont> DATATYPE IS TEXT SIZE IS 25.

CDO> DEFINE FIELD ADDRESS_DATA_2
cont> DESCRIPTION  IS 'Employee mailing street or PO Box address'
cont> DATATYPE IS TEXT SIZE IS 25.

CDO> DEFINE FIELD CITY
cont> DESCRIPTION 'Employee home city address'
cont> DATATYPE IS TEXT SIZE IS 20.

CDO> DEFINE FIELD STATE
cont> DESCRIPTION 'Employee home state address'
cont> DATATYPE IS TEXT SIZE IS 2.

CDO> DEFINE FIELD POSTAL_CODE
cont> DESCRIPTION 'Employee home 9-digit postal code'
cont> DATATYPE IS TEXT SIZE IS 9.
```

(continued on next page)

Example 4–1 (Cont.) Defining Shareable Fields in CDO

```
CDO> DEFINE FIELD SEX
cont> DESCRIPTION 'M=male or F=female'
cont> DATATYPE IS TEXT SIZE IS 1.

CDO> DEFINE FIELD STATUS_CODE
cont> DESCRIPTION 'F=full-time or P=part-time employee'
cont> DATATYPE IS TEXT SIZE IS 1.
```

Use CDO to verify that the fields have been defined, as shown in Example 4–2.

Example 4–2 Verifying Field Definitions

```
   .
   .
   .
CDO> SET DEFAULT SYS$COMMON:[CDDREP]PERS
CDO> DIRECTORY *

Directory SYS$COMMON:[CDDREP]PERS

ADDRESS_DATA_1(1)                       FIELD
ADDRESS_DATA_2(1)                       FIELD
BIRTHDAY(1)                             FIELD
CITY(1)                                 FIELD
EMPLOYEE_ID(1)                          FIELD
END_DATE(1)                             FIELD
FIRST_NAME(1)                           FIELD
ID_NUMBER(1)                            FIELD
LAST_NAME(1)                            FIELD
MIDDLE_INITIAL(1)                       FIELD
POSTAL_CODE(1)                          FIELD
SALARY_AMOUNT(1)                        FIELD
SEX(1)                                  FIELD
STANDARD_DATE(1)                        FIELD
START_DATE(1)                           FIELD
STATE(1)                                FIELD
STATUS_CODE(1)                          FIELD
   .
   .
   .
```

For more information on an element, issue a CDO SHOW FIELD command.

(continued on next page)

Example 4–2 (Cont.) Verifying Field Definitions

```
   .
   .
   .
CDO> SHOW FIELD /ALL CITY

Definition of field CITY
 |    acl                      (IDENTIFIER=[SYS1,SMITH],ACCESS=READ+WRITE+
MODIFY+ERASE+SHOW+DEFINE+CHANGE+DELETE+CONTROL+OPERATOR+ADMINISTRATOR)
(IDENTIFIER=[*,*],ACCESS=READ+WRITE+MODIFY+ERASE+SHOW+OPERATOR+ADMINISTRATOR)
 |    Created time            4-JAN-1994 11:30:08.07
 |    Description             'Employee home city address'
 |    Modified time           4-JAN-1994 11:30:08.07
 |    Owner                   [SYS1,SMITH]
 |    Datatype                text size is 20 characters
 |     |    History entered by SMITH ([SYS1,SMITH])
 |     |      using CDO V2.3
 |     |      to CREATE definition on 4-MAY-1994 11:30:02.02
   .
   .
   .
```

As you define additional elements, you can identify what other elements use them.

```
   .
   .
   .
CDO> SHOW USES STANDARD_DATE
Owners of SYS$COMMON:[CDDREP]PERS.STANDARD_DATE(1)
 |    SYS$COMMON:[CDDREP]PERS.END_DATE(1)    (Type : FIELD)
 |     |    via CDD$DATA_ELEMENT_BASED_ON
 |    SYS$COMMON:[CDDREP]PERS.START_DATE(1)  (Type : FIELD)
 |     |    via CDD$DATA_ELEMENT_BASED_ON
 |    SYS$COMMON:[CDDREP]PERS.BIRTHDAY(1)    (Type : FIELD)
 |     |    via CDD$DATA_ELEMENT_BASED_ON
   .
   .
   .
```

Example 4–3 shows how to use CDO to create a repository record.

First, define a record in the repository for the database table EMPLOYEES. Then define a record in the repository for the SALARY_HISTORY table in the database.

Example 4–3 Defining a Record

```
   .
   .
   .
CDO> DEFINE RECORD EMPLOYEES
cont>  DESCRIPTION IS /* CORPORATE EMPLOYEE INFORMATION */.
cont> EMPLOYEE_ID.
cont> LAST_NAME.
cont> FIRST_NAME.
cont> MIDDLE_INITIAL.
cont> ADDRESS_DATA_1.
cont> ADDRESS_DATA_2.
cont> CITY.
cont> STATE.
cont> POSTAL_CODE.
cont> BIRTHDAY.
cont> SEX.
cont> STATUS_CODE.
cont> END EMPLOYEES RECORD.

   .
   .
   .
CDO> DEFINE RECORD SALARY_HISTORY
cont> DESCRIPTION IS /* INFO ABOUT EACH JOB HELD */.
cont> EMPLOYEE_ID.
cont> SALARY_AMOUNT.
cont> START_DATE.
cont> END_DATE.
cont> END SALARY_HISTORY RECORD.
   .
   .
   .

CDO> SHOW RECORD EMPLOYEES
Definition of record EMPLOYEES
  |    Description             'CORPORATE EMPLOYEE INFORMATION'
  |    Contains field          EMPLOYEE_ID
  |    Contains field          LAST_NAME
  |    Contains field          FIRST_NAME
  |    Contains field          MIDDLE_INITIAL
  |    Contains field          ADDRESS_DATA_1
  |    Contains field          ADDRESS_DATA_2
  |    Contains field          CITY
  |    Contains field          STATE
  |    Contains field          POSTAL_CODE
  |    Contains field          BIRTHDAY
  |    Contains field          SEX
  |    Contains field          STATUS_CODE
```

(continued on next page)

Example 4–3 (Cont.) Defining a Record

```
CDO> SHOW RECORD SALARY_HISTORY
Definition of record SALARY_HISTORY
 |    Description              'INFO ABOUT EACH JOB HELD'
 |    Contains field           EMPLOYEE_ID
 |    Contains field           SALARY_AMOUNT
 |    Contains field           START_DATE
 |    Contains field           END_DATE
CDO> EXIT
```

The EMPLOYEES and SALARY_HISTORY definitions that you created using CDO can be shared by many Oracle Rdb databases.

Example 4–4 defines two department databases, DEPT1 and DEPT2, that share the EMPLOYEES and SALARY_HISTORY definitions.

Create the DEPT1 database in SQL using a relative path name. The full path name for DEPT1 is SYS$COMMON:[CDDREP]PERS.DEPT1. Create the DEPT2 database using the full path name with the anchor representing the repository origin.

Example 4–4 Using Repository Definitions to Create a Database

```
   .
   .
   .
$ DEFINE CDD$DEFAULT SYS$COMMON:[CDDREP]
$ MCR SQL$
SQL> CREATE DATABASE FILENAME DEPT1
cont>              PATHNAME PERS.DEPT1
cont>              DICTIONARY IS REQUIRED;
SQL> DISCONNECT ALL;

SQL> CREATE DATABASE FILENAME DEPT2
cont>              PATHNAME SYS$COMMON:[CDDREP]PERS.DEPT2;
SQL> DISCONNECT ALL;

SQL> ATTACH 'PATHNAME SYS$COMMON:[CDDREP]PERS.DEPT1';
SQL> SHOW TABLES
User tables in database with path name SYS$COMMON:[CDDREP]PERS.DEPT1(1)
No tables found
   .
   .
   .
```

(continued on next page)

Example 4–4 (Cont.) Using Repository Definitions to Create a Database

Create the EMPLOYEES table and the SALARY_HISTORY table in SQL using the repository definitions:

```
   .
   .
   .
SQL> CREATE TABLE
cont> FROM SYS$COMMON:[CDDREP]PERS.EMPLOYEES;
SQL> CREATE TABLE
cont> FROM SYS$COMMON:[CDDREP]PERS.SALARY_HISTORY;

SQL> SHOW TABLES
User tables in database with pathname SYS$COMMON:[CDDREP]PERS.DEPT1(1)
     EMPLOYEES
     SALARY_HISTORY
SQL> COMMIT;

SQL> SHOW TABLE EMPLOYEES

CDD Pathname:   SYS$COMMON:[CDDREP]PERS.EMPLOYEES(1)

Comment on table EMPLOYEES:
CORPORATE EMPLOYEE INFORMATION

Columns for table EMPLOYEES:
Column Name                    Data Type        Domain
-----------                    ---------        ------
EMPLOYEE_ID                    CHAR(5)          EMPLOYEE_ID
LAST_NAME                      CHAR(14)         LAST_NAME
FIRST_NAME                     CHAR(10)         FIRST_NAME
MIDDLE_INITIAL                 CHAR(1)          MIDDLE_INITIAL
ADDRESS_DATA_1                 CHAR(25)         ADDRESS_DATA_1
ADDRESS_DATA_2                 CHAR(25)         ADDRESS_DATA_2
CITY                           CHAR(20)         CITY
STATE                          CHAR(2)          STATE
POSTAL_CODE                    CHAR(9)          POSTAL_CODE
BIRTHDAY                       DATE             BIRTHDAY
SEX                            CHAR(1)          SEX
STATUS_CODE                    CHAR(1)          STATUS_CODE

   .
   .
   .
```

(continued on next page)

Example 4–4 (Cont.) Using Repository Definitions to Create a Database

```
SQL> SHOW TABLE SALARY_HISTORY

CDD Pathname:   SYS$COMMON:[CDDREP]PERS.SALARY_HISTORY(1)

Comment on table SALARY_HISTORY:
INFO ABOUT EACH JOB HELD

Columns for table SALARY_HISTORY:
Column Name                    Data Type        Domain
-----------                    ---------        ------
EMPLOYEE_ID                    CHAR(5)          EMPLOYEE_ID
SALARY_AMOUNT                  QUADWORD(2)      SALARY_AMOUNT
START_DATE                     DATE             START_DATE
END_DATE                       DATE             END_DATE

SQL> COMMIT;
SQL> DISCONNECT ALL;
```

Use SQL to set the appropriate protection rights on metadata in the database, then commit the transaction.

Use CDO to set protection on the repository definitions. By default, CDO grants all privileges including CONTROL to Owner; World is granted the SHOW privilege only.

See Chapter 6 of *Using Oracle CDD/Repository on OpenVMS Systems* for a discussion of repository protection. After committing the transaction, you can track any modifications to the Oracle Rdb domains (fields) and tables (records) through Oracle CDD/Repository by using the CDO SHOW USES command.

4.2.2 Creating Record Constraints

In Oracle CDD/Repository, you can specify conditions, known as constraints, that affect adding or modifying data in a database table (CDO record). CDO provides syntax for record constraints, including specification of NOT NULL, PRIMARY KEY, FOREIGN KEY, UNIQUE, and CHECK (arbitrary search condition constraint) for fields and records.

Each constraint can be named and supplied with evaluation attributes: DEFERRABLE and NOT DEFERRABLE in SQL Version 4.2 and higher (COMMIT or UPDATE in RDO).

Note

The syntax for attributes not related to the new features has been excluded from the following syntax diagrams. Use these syntax diagrams with the *Oracle CDD/Repository CDO Reference Manual*.

4.2.2.1 CDO DEFINE RECORD Syntax

The updated CDO DEFINE RECORD syntax containing the new constraint features is as follows:

```
DEFINE RECORD record-name
  [ DESCRIPTION IS /*text*/ ]  [ AUDIT IS /*text*/ ]
[ record-property ]
[ constraint      ] ... .
{ field-name [ field-attr ] . }
{ included-name-clause        }
{ structure-clause            } ... . END [ record-name RECORD ] .
{ variant-clause              }

constraint ::=
 CONSTRAINT const-name { primary-constraint   }
                       { foreign-constraint   } [ NOT DEFERRABLE ]
                       { unique-constraint    } [ DEFERRABLE     ]
                       { check-constraint     }

primary-constraint ::= PRIMARY KEY field-name , ...
unique-constraint ::= UNIQUE field-name , ...
foreign-constraint ::=
 FOREIGN KEY fld-name , ... REFERENCES record-name fld-name , ...
check-constraint ::= CHECK ( expression )
field-attr ::= [ description ] [ based-on ] [ aligned-on ] [ field-constr ]
description ::= DESCRIPTION IS quoted-string
based-on ::= BASED ON field-name
aligned-on ::= ALIGNED ON datatype
field-constr ::=
 [ CONSTRAINT constr-name ] NOT NULL [ NOT DEFERRABLE ] ...
                                     [ DEFERRABLE     ]
```

Example 4–5 uses the CDO DEFINE RECORD command syntax to establish constraints on the PARTS record in the ACME Company database.

ACME wants the PART_ID to be the primary key and also the PART_NO to be a unique value across all possible parts. By not specifying whether the constraints are deferrable, ACME accepts the default evaluation time. In CDO, the default evaluation time for constraints is NOT DEFERRABLE. Constraints are evaluated at statement time.

Using CDO, ACME defines the record PARTS with the following attributes:

❶ Primary key PARTS_PMK

❷ Unique constraint PARTS_UNQ

❸ Check constraint PART_CST

❹ Foreign key constraint PART_FRK

The attributes numbered in the list correspond to the callouts in the example.

Note

For the purposes of this example, it is assumed that the field definitions referred to in the record definitions have already been defined in the repository.

Example 4–5 Using the CDO DEFINE RECORD Command to Establish Primary, Unique, Check, and Foreign Key Constraints

```
$ REPOSITORY OPERATOR
   .
   .
   .
CDO> define record PARTS
```

(continued on next page)

Example 4–5 (Cont.) Using the CDO DEFINE RECORD Command to Establish Primary, Unique, Check, and Foreign Key Constraints

```
cont> constraint PARTS_PMK primary key PART_ID ❶
cont> constraint PARTS_UNQ unique PART_NO ❷
cont> constraint PART_CST check ❸
cont> (any p in PARTS with (PART_ID in PARTS = PART_ID_USED_IN in PARTS))
cont> constraint PART_FRK ❹
cont>   foreign key PART_NO references PARTS PART_ID.
cont> PART_NO.
cont> PART_ID.
cont> PART_ID_USED_IN.
cont> PART_QUANT.
cont> end.

CDO> show record PARTS/full
Definition of record PARTS
|   Contains field          PART_NO
|   |   Datatype                signed word
|   Contains field          PART_ID
|   |   Datatype                signed longword
|   Contains field          PART_ID_USED_IN
|   |   Based on                ID_DOM
|   |   |   Datatype                signed longword
|   Contains field          PART_QUANT
|   |   Datatype                signed word
|   Constraint PARTS_PMK  primary key PART_ID NOT DEFERRABLE
|   Constraint PARTS_UNQ  unique PART_NO NOT DEFERRABLE
|   Constraint PART_CST (ANY (P IN PARTS WITH (PART_ID IN PARTS EQ
PART_ID_USED_IN IN PARTS))) NOT DEFERRABLE
|   Constraint PART_FRK  foreign key PART_NO references PARTS PART_ID
NOT DEFERRABLE
```

A database at Smitti's Grocery Store provides the PRODUCE record used in Example 4–6 to display combinations of NOT NULL and DEFERRABLE field attributes in a record definition.

Each item sold at the store contains several attributes that can be defined in a record: UPC number, weight, price, and quantity.

Because NOT NULL is specified on each field, when the store later inserts values into the PRODUCE record, a value must be supplied for *each* field in the PRODUCE record. The field PRICE ❷ has the DEFERRABLE attribute, allowing its value to be evaluated at commit time.

Note

If you do not specify a name for the NOT NULL constraint, as displayed for the field UPC_CODE ❶, Oracle CDD/Repository generates its own name ❸; in this instance the name is CDD$NOT_NULL_ 0096C4CF2C3448EA.

If an error pertaining to this constraint occurs, the output of the error will contain this generated name and can cause confusion. It is recommended that you always specify a constraint name when defining constraints.

Example 4–6 Using the CDO DEFINE RECORD Command with NOT NULL and DEFERRABLE Field Attributes

```
   .
   .
   .
CDO> define record PRODUCE.
cont> UPC_CODE ❶ not null DEFERRABLE.
cont> WEIGHT constraint WNOTNULL not null.
cont> PRICE constraint PNOTNULL not null DEFERRABLE. ❷
cont> QUANTITY constraint QNOTNULL not null not DEFERRABLE.
cont> end.

CDO> show record PRODUCE/full
Definition of record PRODUCE
|   Contains field          UPC_CODE
|   |   Datatype                signed longword
|   Contains field          WEIGHT
|   |   Datatype                signed word
|   Contains field          PRICE
|   |   Datatype                signed longword
|   Contains field          QUANTITY
|   |   Datatype                signed word
|   Constraint ❸ CDD$NOT_NULL_0096C4CF2C3448EA  not null UPC_CODE DEFERRABLE
|   Constraint WNOTNULL  not null WEIGHT NOT DEFERRABLE
|   Constraint PNOTNULL  not null PRICE DEFERRABLE
|   Constraint QNOTNULL  not null QUANTITY NOT DEFERRABLE
```

The ADDRESS record for an EMPLOYEES database in Example 4–7 shows the use of local field definitions using BASED ON attributes from the CDO DEFINE RECORD syntax diagram.

In this example, the fields FIRST_NAME ❶ and LAST_NAME ❷ are both based on the 30-character text string, NAME_STRING, and they contain descriptions. The field ZIP_CODE must be specified when you insert values into the record. This requirement is established by the NOT NULL ❸ attribute.

Example 4–7 Using CDO DEFINE RECORD Field Attributes to Specify BASED ON Attributes

```
CDO> define record ADDRESS.
cont>  FIRST_NAME ❶
cont>           description /* employee first name */
cont>           based on NAME_STRING.
cont>  LAST_NAME ❷
cont>           description /* employee last name */
cont>           based on NAME_STRING.
cont>  STREET.
cont>  CITY.
cont>  STATE.
cont>  ZIP_CODE not null. ❸
cont>end.

CDO> show record address/full
Definition of record ADDRESS
|   Contains field          FIRST_NAME
|   |   Description             /* employee first name  */
|   |   Based on                NAME_STRING
|   |   |   Datatype                text size is 30 characters
|   Contains field          LAST_NAME
|   |   Description             /* employee last name  */
|   |   Based on                NAME_STRING
|   |   |   Datatype                text size is 30 characters
|   Contains field          STREET
|   |   Datatype                text size is 40 characters
|   Contains field          CITY
|   |   Datatype                text size is 20 characters
|   Contains field          STATE
|   |   Datatype                text size is 2 characters
|   Contains field          ZIP_CODE
|   |   Datatype                text size is 5 characters
|   Constraint CDD$NOT_NULL_0096C4D0443E0F10  not null ZIP_CODE NOT DEFERRABLE
```

The CAR record in Example 4–8 is from a database at Herb's Auto Dealer. The record has several fields to define a car's make, year, color, cost, and the date sold.

The local field MAKE ❶ is based on the CAR_MAKES field, and the local field CAR_YEAR ❷, which defined the year of the car, is based on the YEAR field. The COST field ❸ contains the attributes aligned on quad and not null. The local field DATE_SOLD ❹, is based on the DATE field.

Example 4–8 Using CDO DEFINE RECORD Field Attributes to Specify BASED ON and ALIGNED ON Attributes

```
CDO> define record CAR.
cont> MAKE ❶
cont>       description /* make of car */
cont>       based on CAR_MAKES.

cont> CAR_YEAR ❷
cont>       description /* year of car */
cont>       based on YEAR.

cont> COLOR description /* color of car */.
cont> COST  description /* price of car */ ❸
cont>       based on PRICE aligned on quad not null.
cont> DATE_SOLD ❹
cont>       description /* date of sale */
cont>       based on DATE.
cont> end.

CDO> show record car/full
Definition of record CAR
|   Contains field          MAKE
|   |   Description             /* make of car  */
|   |   Based on                CAR_MAKES
|   |   |   Datatype                text size is 10 characters
|   Contains field          CAR_YEAR
|   |   Description             /* year of car  */
|   |   Based on                YEAR
|   |   |   Datatype                signed word
```

(continued on next page)

Example 4–8 (Cont.) Using CDO DEFINE RECORD Field Attributes to Specify BASED ON and ALIGNED ON Attributes

```
|   Contains field          COLOR
|   |   Description             /* color of car  */
|   |   Based on                COLOR
|   |   |   Datatype                text size is 12 characters
|   Contains field          COST
|   |   Aligned on quadword boundary
|   |   Description             /* price of car  */
|   |   Based on                PRICE
|   |   |   Datatype                signed longword
|   Contains field          DATE_SOLD
|   |   Description             /* date of sale  */
|   |   Based on                DATE
|   |   |   Datatype                date
|   Constraint CDD$NOT_NULL_0096C4D3A9B805E6  not null COST NOT DEFERRABLE
```

Example 4–9 displays how to use fields from different directories in record definitions when using the BASED ON attribute.

The repository in this example has two directories: CUST ❶ containing customer information, and EMPLOY ❷ containing employee information.

In the top level default directory, two fields are created: FIRST_NAME ❸ and LAST_NAME ❹. These fields are used in the record definition for CUSTOMER ❺ and EMPLOYEE ❻ in their respective directories. Defining fields using BASED ON within record definitions can eliminate some CDD-E-DUPGLOBAL errors.

Example 4–9 Defining Fields Using BASED ON Attributes within Record Definitions in Different Directories

```
CDO> define directory cdd$default.CUST. ❶
CDO> define directory cdd$default.EMPLOY. ❷
CDO> define field cdd$default.FIRST_NAME ❸
cont> datatype is text size is 10.
CDO> define field cdd$default.LAST_NAME ❹
cont> datatype is text size is 15.
CDO> set default cdd$default.CUST
CDO> define record CUSTOMER. ❺
cont> CUST_FNAME based on cdd$default.FIRST_NAME.
cont> CUST_LNAME based on cdd$default.LAST_NAME.
cont> end.
```

(continued on next page)

Example 4–9 (Cont.) Defining Fields Using BASED ON Attributes within Record Definitions in Different Directories

```
CDO> show record/full CUSTOMER
Definition of record CUSTOMER
|   Contains field           CUST_FNAME
|   |   Based on                FIRST_NAME
|   |   |   Datatype              text size is 10 characters
|   Contains field           CUST_LNAME
|   |   Based on                LAST_NAME
|   |   |   Datatype              text size is 15 characters
CDO> set default cdd$default.employ
CDO> define record EMPLOYEE. ❻
cont> EMP_FNAME based on cdd$default.FIRST_NAME.
cont> EMP_LNAME based on cdd$default.LAST_NAME.
cont> end.

CDO> show record/full EMPLOYEE
Definition of record EMPLOYEE
|   Contains field           EMP_FNAME
|   |   Based on                FIRST_NAME
|   |   |   Datatype              text size is 10 characters
|   Contains field           EMP_LNAME
|   |   Based on                LAST_NAME
|   |   |   Datatype              text size is 15 characters
```

Section 4.2.2.2 shows the updated CDO CHANGE RECORD syntax.

Use the record-change-clause of the CDO CHANGE RECORD command to add a new field or constraint. Use the DELETE clause to delete an existing field or constraint.

4.2.2.2 CDO CHANGE RECORD Syntax

The updated CDO CHANGE RECORD syntax containing the new constraint features is as follows:

```
CHANGE RECORD record-name
[ DESCRIPTION IS /*text*/ | NODESCRIPTION ]  [ AUDIT IS /*text*/ ]

[ constraint | DELETE CONSTRAINT constr-name ] ... .

[ record-change-clause | DELETE name . | field-name [ field-attr ] ] ...  END [ record-name RECORD ] .
```

record-change-clause ::=

```
DEFINE { included-name-clause | structure-clause | variant-clause } END [ name DEFINE ] .
```

included-name-clause ::=

```
field-name { field-constr | ALIGNED ON datatype BOUNDARY | NOALIGNED } .
```

If you specify the field name (no special keyword) and one of the attributes (field-constr, ALIGNED ON datatype, or NOALIGNED), the field's attributes will be changed.

If you specify the constraint name and one of the attributes shown in the CDO DEFINE RECORD syntax diagram, the constraint attributes will be changed.

Example 4–10 uses the CHANGE RECORD command syntax to add the telephone number to the PERSON record and to ensure that a value is supplied for all people.

Example 4–10 Using the CHANGE RECORD Command to Add a New Field

```
   .
   .
   .
CDO> change record PERSON.
cont> define TELEPHONE_NUMBER
cont>    description is 'Home Telephone Number'
cont>    constraint NO_PHONE_NUMBER
cont>    NOT NULL NOT DEFERRABLE.
cont> end.
cont> end.
CDO> show record PERSON/full
Definition of record PERSON
|   Contains field           NAME
|   |   Datatype                text size is 20 characters
|   Contains field           AGE
|   |   Datatype                signed word
|   Contains field           TELEPHONE_NUMBER
|   |   Description             /* Home Telephone Number */
|   |   Based on                TELEPHONE_NUMBER
|   |   |   Datatype                signed longword
|   Constraint NO_PHONE_NUMBER  not null TELEPHONE_NUMBER NOT DEFERRABLE
```

Example 4–11 returns to the PARTS record at the ACME Company, which was described in Example 4–5. Now the ACME Company decides the foreign key on the PARTS record is no longer useful. The PARTS record can be altered by using the CHANGE RECORD command with the DELETE clause ❶ specified with the constraint name.

The ACME Company also wants a new not null field, SUPPLIER, to identify the supplier of each part. To accomplish this, ACME uses the DEFINE clause ❷, as shown in Example 4–11.

Example 4–11 Using the CHANGE RECORD Command to Delete a Constraint and Add a NOT NULL Field

```
   .
   .
   .
CDO> change record PARTS
cont>  delete constraint PART_FRK. ❶
cont>  define SUPPLIER    ❷
cont>       constraint NO_SUPPLIER not null not deferrable.
cont>   end.
cont>  end.

CDO> show record PARTS/full
Definition of record PARTS
|   Contains field           PART_NO
|   |   Datatype                signed word
|   Contains field           PART_ID
|   |   Datatype                signed longword
|   Contains field           PART_ID_USED_IN
|   |   Based on                ID_DOM
|   |   |   Datatype                signed longword
|   Contains field           PART_QUANT
|   |   Datatype                signed word
|   Contains field           SUPPLIER
|   |   Datatype                text size is 20 characters
|   Constraint PARTS_PMK  primary key PART_ID NOT DEFERRABLE
|   Constraint PARTS_UNQ  unique PART_NO NOT DEFERRABLE
|   Constraint PART_CST (ANY (P IN PARTS WITH (PART_ID IN PARTS EQ
PART_ID_USED_IN IN P))) NOT DEFERRABLE
|   Constraint NO_SUPPLIER  not null SUPPLIER NOT DEFERRABLE
```

In Example 4–12, Smitti's Grocery Store decides the UPC_CODE, which was defined in Example 4–6, should be a primary key constraint for the PRODUCE record. To accomplish this, Smitti uses the CHANGE RECORD command and specify the PRIMARY KEY on the local field UPC_CODE ❶.

Smitti also wants to add the STATUS_CODE field to help determine if the item is currently in stock ❷.

Example 4–12 Using the CDO CHANGE RECORD Command to Specify a Primary Key Constraint

```
   .
   .
   .
CDO> change record PRODUCE
cont> constraint U_PKEY primary key UPC_CODE. ❶
cont> define STATUS_CODE
cont>     description /* in stock status */ ❷
cont>     based on STATUS.
cont> end.
cont>end.

CDO> show record PRODUCE/full
Definition of record PRODUCE
|   Contains field          UPC_CODE
|   |   Datatype                signed longword
|   Contains field          WEIGHT
|   |   Datatype                signed word
|   Contains field          PRICE
|   |   Datatype                signed longword
|   Contains field          QUANTITY
|   |   Datatype                signed word
|   Contains field          STATUS_CODE
|   |   Description             /* if in stock  */
|   |   Based on                STATUS
|   |   |   Datatype                text size is 1 characters
|   Constraint CDD$NOT_NULL_0096C56941B9497D  not null UPC_CODE DEFERRABLE
|   Constraint WNOTNULL  not null WEIGHT NOT DEFERRABLE
|   Constraint PNOTNULL  not null PRICE DEFERRABLE
|   Constraint QNOTNULL  not null QUANTITY NOT DEFERRABLE
|   Constraint U_PKEY  primary key UPC_CODE NOT DEFERRABLE
```

In Example 4–13, the managers in charge of the EMPLOYEES database decide the ADDRESS record must be changed by deleting the FIRST_NAME field and redefining the CITY field to be based on the NAME_STRING.

Example 4–13 shows how they accomplish this by first deleting the FIRST_ NAME ❶ field and deleting the CITY ❷ field. Then the CITY field is redefined by specifying the field with new attributes ❸ and supplying an end ❹.

Example 4–13 Using the CHANGE RECORD Command to Delete a Field, Then Redefine It

```
   .
   .
   .
CDO> change record ADDRESS.
cont>   delete FIRST_NAME. ❶
cont>   delete CITY.    ❷
cont>   define CITY based on NAME_STRING. ❸
cont>   end. ❹
cont>end.

CDO> show record ADDRESS/full
Definition of record ADDRESS
|   Contains field          LAST_NAME
|   |   Based on                NAME_STRING
|   |   |   Description             /* standard name  */
|   |   |   Datatype                text size is 30 characters
|   Contains field          STREET
|   |   Datatype                text size is 40 characters
|   Contains field          STATE
|   |   Datatype                text size is 2 characters
|   Contains field          ZIP_CODE
|   |   Datatype                text size is 5 characters
|   Constraint CDD$NOT_NULL_0096C4D0443E0F10  not null ZIP_CODE NOT
DEFERRABLE
|   Contains field          CITY
|   |   Based on                NAME_STRING
|   |   |   Description             /* standard name  */
|   |   |   Datatype                text size is 30 characters
```

4.2.3 Differences Between CDO IN Clause and SQL IN Operator

This section explains the differences between using the CDO IN keyword, which is part of a relation-clause in a record selection expression (RSE), and the SQL IN operator.

The CDO IN keyword in a relation-clause declares context variables for a record stream or loop. For example, the context variable PART_ID specifies a temporary name that identifies the record stream to the product evaluating the clause:

```
PART_ID in PARTS
```

You then use the context variable to refer to fields from that relation. The relation name PARTS specifies the relation from which CDO takes the records in the record stream.

For more information, refer to the description of record selection expressions in the *Oracle CDD/Repository CDO Reference Manual*.

The SQL IN operator compares a value with another value or a collection of values. For example:

```
STATE in ('RI','MA','NY')
```

In SQL, value-expr IN (value-expr1,value-expr2,value-expr3) is the same as the complex predicate, as follows:

```
value-expr = value-expr1
OR
value-expr = value-expr2
OR
value-expr = value-expr3
```

Unlike SQL, CDO does not have an IN operator; therefore, to perform the same operation within a CDO expression, you must specify the complex predicate form. For example:

```
CDO> define record rec1
cont> constraint cons check ((STATE IN rec1 EQ "RI") OR
                             (STATE IN rec1 EQ "MA") OR
                             (STATE IN rec1 EQ "NY")).
cont>   test.
cont> end.
```

4.3 Modifying Repository Definitions and Database Metadata

When you create an Oracle Rdb database, you can store metadata in the database, or in the database and the repository. There may be times when the database metadata and the definitions in the repository no longer match. When they no longer match, you can update one source using the other.

The following sections describe situations that might cause discrepancies between the repository definitions and the database metadata, and give examples of how to modify them so that they match.

4.3.1 Modifying Repository Definitions Using CDO

You can modify repository definitions by using the CDO DEFINE command to create a new version of an element, or use the CDO CHANGE command to modify an element.

Oracle CDD/Repository lets you store several versions of the same element in much the same way as the OpenVMS operating system stores multiple versions of files. When you store several versions of the same definition, you can specify and use any of these versions. If you specify a definition without a version number, CDO assigns the highest version number to the definition by default.

You can issue a CDO SHOW USES command to see if an older version is part of the database. For example:

```
CDO> SHOW USES STANDARD_DATE
 Owners of SYS$COMMON:[CDDREP]PERS.STANDARD_DATE(1)
 |    SYS$COMMON:[CDDREP]PERS.START_DATE(1)  (Type : FIELD)
 |     |    via CDD$DATA_ELEMENT_BASED_ON
 |    SYS$COMMON:[CDDREP]PERS.END_DATE(1)    (Type : FIELD)
 |     |    via CDD$DATA_ELEMENT_BASED_ON
 |    SYS$COMMON:[CDDREP]PERS.BIRTHDAY(2)    (Type : FIELD)
 |     |    via CDD$DATA_ELEMENT_BASED_ON
 |    SYS$COMMON:[CDDREP]PERS.BIRTHDAY(1)    (Type : FIELD)
 |     |    via CDD$DATA_ELEMENT_BASED_ON
```

Use the CDO CHANGE commands to modify an element created by CDO. If the element is controlled, you can modify only the highest visible version, and you must reserve the element before you can change it.

You control an element by opening a context and sending a control message to it. This places the element on the base partition of that context. You cannot modify the controlled versions without using the CDO RESERVE/REPLACE model.

Use the CDO CHANGE command to modify an uncontrolled element created by CDO. The CHANGE command changes the original definition without creating a new version of the definition. See the *Oracle CDD/Repository CDO Reference Manual* for more information on the CDO CHANGE, RESERVE, and REPLACE commands.

```
   .
   .
   .
CDO> SHOW FIELD ID_NUMBER
Definition of field ID_NUMBER
 | Description             'Corporate Standard ABRII'
 | Datatype                text size is 5 characters
CDO> CHANGE FIELD ID_NUMBER DATATYPE TEXT SIZE IS 6.
CDO> SHOW FIELD ID_NUMBER

 Definition of field ID_NUMBER
 | Description             'Corporate Standard ABRII'
 | Datatype                text size is 6 characters
```

Because the ID_NUMBER field is changed in the repository, the database and the repository no longer match. Use the SQL INTEGRATE DATABASE statement to alter the database definitions inclusively to match those of the repository.

```
$ MCR SQL$
SQL> INTEGRATE DATABASE PATHNAME SYS$COMMON:[CDDREP]PERS.DEPT1
cont> ALTER FILES;
SQL> COMMIT;
```

Section 4.3.2 describes the SQL INTEGRATE statement in more detail.

If you want to update an individual repository field, use the SQL INTEGRATE DOMAIN statement. To update an individual record, use the SQL INTEGRATE TABLE statement. This functionality is supported in Oracle CDD/Repository Version 6.1. See the Oracle CDD/Repository Version 6.1 release notes for details.

4.3.2 Using SQL INTEGRATE to Synchronize the Repository and the Database

This section shows how you can use the SQL INTEGRATE statement to do the following:

- Create repository definitions for the first time by using the database metadata as the source.
- Update repository definitions using the database metadata as the source.
- Update the database metadata using repository definitions as the source.

Four possible combinations of the SQL CREATE DATABASE statement and the SQL ATTACH statement affect how a repository is updated. The following list explains these combinations and the actions you must take in each situation to update the repository:

- Create a database using the DICTIONARY IS REQUIRED option and the PATHNAME option of the SQL CREATE DATABASE statement. You can

subsequently attach to the database by using the PATHNAME option of the SQL ATTACH statement.

SQL updates the repository and the database with any changes you make during that attachment to the database. You do not need to use the SQL INTEGRATE statement.

- Create a database using the DICTIONARY IS REQUIRED option and the PATHNAME option of the SQL CREATE DATABASE statement. You can subsequently attach to the database using the FILENAME option of the SQL ATTACH statement.

  SQL produces an error message in response to subsequent data definition statements because the database definition specifies the DICTIONARY IS REQUIRED option. In this case, you should attach to the database using the PATHNAME option because the FILENAME option specifies that only the database is updated.

  When you specify the PATHNAME option on the SQL ATTACH statement, SQL updates both the repository and the database.

- Create a database using the DICTIONARY IS NOT REQUIRED (the default) option and the PATHNAME option of the SQL CREATE DATABASE statement. You subsequently attach to the database using the FILENAME option of the SQL ATTACH statement.

  SQL stores the initial database definitions in the repository when you specify the PATHNAME option. Because the repository is not required, and because you attach to the database using the FILENAME option, any changes you make to the database metadata during that attachment to the database are entered only into the database, not into the repository.

  To update repository definitions inclusively, use the SQL INTEGRATE DATABASE . . . PATHNAME statement with the ALTER DICTIONARY clause. This statement alters the repository definitions so that they are the same as those in the database. However, by altering definitions in the repository, you may affect other repository entities that refer to these definitions.

  Example 4–14 is a typical situation where you would use the INTEGRATE DATABASE statement with the ALTER DICTIONARY clause.

  If you want to update an individual repository field, use the SQL INTEGRATE DOMAIN statement. To update an individual record, use the SQL INTEGRATE TABLE statement. See the Oracle CDD/Repository Version 6.1 release notes for details.

Note

You cannot use the SQL INTEGRATE DOMAIN or the SQL INTEGRATE TABLE statements to create an object in the database or repository; you can only modify an existing definition.

- Create a database using the DICTIONARY IS NOT REQUIRED option (the default) of the SQL CREATE DATABASE statement, but *do not* specify the PATHNAME option. You can subsequently attach to the database using the FILENAME option of the SQL ATTACH statement.

  Because you do not specify the PATHNAME option, SQL does not store the original database metadata in the repository. When you specify the FILENAME option, SQL enters any database metadata changes you make during that attachment to only the database, not to the repository.

  To store existing database metadata in the repository for the first time, use the SQL INTEGRATE DATABASE . . . FILENAME statement with the CREATE PATHNAME clause. This statement builds repository definitions using the database as the source.

  Example 4–15 creates a database and stores the metadata in only the database. It then uses the SQL INTEGRATE statement with the CREATE PATHNAME clause to integrate the metadata from the database to the repository.

4.3.2.1 Modifying Repository Definitions Using SQL

If you create a database using the DICTIONARY IS NOT REQUIRED option and the PATHNAME option, SQL stores the initial definitions in the repository. If you subsequently make changes to the database definitions without using the PATHNAME option, SQL stores those changes only in the database, not in the repository.

If you specify the DICTIONARY IS REQUIRED clause when you create the database, then try to modify metadata using the FILENAME option, you will see an error message indicating that updates are not allowed.

Example 4–14 shows how to integrate the database with the repository. At the beginning of this example, the repository and the database match. The database contains a table called ORDER, and the repository contains a record also called ORDER.

The ORDER table has four fields (FIRST, SECOND, THIRD, and FOURTH) that are based on four domains (FIRST_DOM, SECOND_DOM, THIRD_DOM, and FOURTH_DOM).

Example 4–14 Modifying Repository Definitions Using the INTEGRATE Statement with ALTER DICTIONARY Clause

```
   .
   .
   .
SQL> CREATE DATABASE FILENAME TEST1
cont> PATHNAME SYS$COMMON:[CDDREP]TEST1;
SQL> CREATE DOMAIN FIRST_DOM CHAR(4);
SQL> CREATE DOMAIN SECOND_DOM CHAR(4);
SQL> CREATE DOMAIN THIRD_DOM CHAR(4);
SQL> CREATE DOMAIN FOURTH_DOM CHAR(4);

SQL> CREATE TABLE ORDER
cont> (FIRST FIRST_DOM,
cont> SECOND SECOND_DOM,
cont> THIRD THIRD_DOM,
cont> FOURTH FOURTH_DOM);
SQL> COMMIT;
SQL> DISCONNECT ALL;
```

Attach to the database with the FILENAME clause so the repository is not updated, then use the SQL SHOW TABLE statement to see what fields and domains are part of the table ORDER.

```
   .
   .
   .
SQL> ATTACH 'FILENAME TEST1';
SQL> SHOW TABLE ORDER
Columns for table ORDER:
Column Name                    Data Type        Domain
-----------                    ---------        ------
FIRST                          CHAR(4)          FIRST_DOM
SECOND                         CHAR(4)          SECOND_DOM
THIRD                          CHAR(4)          THIRD_DOM
FOURTH                         CHAR(4)          FOURTH_DOM
   .
   .
   .
```

(continued on next page)

Example 4–14 (Cont.) Modifying Repository Definitions Using the INTEGRATE Statement with ALTER DICTIONARY Clause

Create a new domain called FIFTH_DOM. Add a new column to the ORDER table. Name the column FIFTH and base it on the domain FIFTH_DOM:

```
   .
   .
   .
SQL> CREATE DOMAIN FIFTH_DOM CHAR(4);
SQL> ALTER TABLE ORDER ADD FIFTH FIFTH_DOM;

   .
   .
   .
SQL> SHOW TABLE ORDER

Columns for table ORDER:
Column Name                    Data Type        Domain
-----------                    ---------        ------
FIRST                          CHAR(4)          FIRST_DOM
SECOND                         CHAR(4)          SECOND_DOM
THIRD                          CHAR(4)          THIRD_DOM
FOURTH                         CHAR(4)          FOURTH_DOM
FIFTH                          CHAR(4)          FIFTH_DOM

   .
   .
   .
SQL> COMMIT;
SQL> EXIT
```

Invoke CDO, and check the record ORDER. The field FIFTH is not part of the record ORDER. Now the database and the repository no longer match.

```
$ REPOSITORY
   .
   .
   .
CDO> SHOW RECORD ORDER FROM DATABASE TEST1
Definition of the record ORDER
|   Contains field             FIRST
|   Contains field             SECOND
|   Contains field             THIRD
|   Contains field             FOURTH

CDO> EXIT
```

(continued on next page)

Example 4–14 (Cont.) Modifying Repository Definitions Using the INTEGRATE Statement with ALTER DICTIONARY Clause

Enter SQL again. Use the SQL INTEGRATE DATABASE statement with the ALTER DICTIONARY clause to resolve this situation. Alter the repository to add the field FIFTH and the domain FIFTH_DOM to the repository using the database as the source.

The SQL INTEGRATE statement implicitly attaches to the database, using the default authorization identifier.

```
$ MCR SQL$
SQL> INTEGRATE DATABASE PATHNAME TEST1 ALTER DICTIONARY;
SQL> COMMIT;
SQL> EXIT
```

Note

You must integrate the entire database; you cannot use the SQL INTEGRATE DOMAIN statement to create FIFTH_DOM in the repository. Creating individual objects using INTEGRATE is not supported.

Enter CDO again. Use the CDO SHOW RECORD command to see that the field FIFTH is now part of the record ORDER. The repository and the database match again.

```
$ REPOSITORY
   .
   .
   .
CDO> SHOW RECORD ORDER FROM DATABASE TEST1
Definition of the record ORDER
|   Contains field              FIRST
|   Contains field              SECOND
|   Contains field              THIRD
|   Contains field              FOURTH
|   Contains field              FIFTH

CDO> EXIT
```

4.3.2.2 Creating Repository Definitions Using SQL

If you create a database using the DICTIONARY IS NOT REQUIRED option and you do not specify the PATHNAME option, SQL does not store the original definitions in the repository.

Example 4–15 shows how to store existing database file definitions in the repository for the first time.

This example creates a database called DOGS. It then creates a table for the breed of dog, POODLES. The columns in the table are types of poodles. The example shows how to use the SQL INTEGRATE statement with the CREATE PATHNAME clause to integrate the metadata from the database into the repository.

Example 4–15 Storing Existing Definitions in the Repository

```
$ MCR SQL$
SQL> CREATE DATABASE ALIAS DOGS;
SQL> CREATE TABLE DOGS.POODLES (STANDARD CHAR(10),
cont> MINIATURE CHAR(10), TOY CHAR(10));

   .
   .
   .
SQL> SHOW TABLE DOGS.POODLES

Columns for table DOGS.POODLES:
Column Name                    Data Type        Domain
-----------                    ---------        ------
STANDARD                       CHAR(10)
MINIATURE                      CHAR(10)
TOY                            CHAR(10)

   .
   .
   .

SQL> COMMIT;
SQL> EXIT
```

The CDO DIRECTORY command shows that the database DOGS is not part of the repository:

(continued on next page)

Example 4–15 (Cont.) Storing Existing Definitions in the Repository

```
$ REPOSITORY
   .
   .
   .
CDO> DIR

Directory SYS$COMMON:[CDDREP]DEPT32.FIELDMAN
FIRST(1)                                        FIELD
FOURTH(1)                                       FIELD
ORDER(1)                                        RECORD
PERSONNEL(1)                                    CDD$DATABASE
SECOND(1)                                       FIELD
TEST1(1)                                        CDD$DATABASE
THIRD(1)                                        FIELD
CDO> EXIT
```

Enter SQL again. Use the SQL INTEGRATE DATABASE FILENAME statement to integrate the database DOGS into the repository:

```
$ MCR SQL$
SQL> INTEGRATE DATABASE FILENAME DISK01:[FIELDMAN.KENNELS]DOGS
cont> CREATE PATHNAME SYS$COMMON:[CDDREP]DEPT32.FIELDMAN.DOGS;
SQL> COMMIT;
SQL> EXIT
```

Enter CDO again, and check to see that the database DOGS has been integrated into the repository:

```
$ REPOSITORY
   .
   .
   .
CDO> DIR
DOGS(1)                                         CDD$DATABASE
FIRST(1)                                        FIELD
FOURTH(1)                                       FIELD
ORDER(1)                                        RECORD
SECOND(1)                                       FIELD
TEST1(1)                                        CDD$DATABASE
THIRD(1)                                        FIELD
PERSONNEL(1)                                    CDD$DATABASE
   .
   .
   .
```

(continued on next page)

Example 4–15 (Cont.) Storing Existing Definitions in the Repository

You can also use the CDO SHOW USED_BY command to see that the record (table) POODLES and the fields (columns) STANDARD, MINIATURE, and TOY are part of the database DOGS.

```
   .
   .
   .
CDO> SHOW USED_BY/FULL DOGS

Members of SYS$COMMON:[CDDREP]DEPT32.FIELDMAN.DOGS(1)
|   SYS$COMMON:[CDDREP]DEPT32.FIELDMAN]DOGS    (Type : CDD$FILE)
|   |   via CDD$DATABASE_FILE
|   DOGS                            (Type : CDD$RDB_DATABASE)
|   |   via CDD$DATABASE_SCHEMA

   .
   .
   .

|   SYS$COMMON:[CDDREP]CDD$RDB_SYSTEM_METADATA.RDB$CDD_NAME(1)(Type : FIELD)
|   |   |   via CDD$DATA_AGGREGATE_CONTAINS
|   |   POODLES                           (Type : RECORD)
|   |   |   via CDD$RDB_DATA_AGGREGATE
|   |   |   STANDARD                        (Type : FIELD)
|   |   |   |   via CDD$DATA_AGGREGATE_CONTAINS
|   |   |   |   SQL$10CHR                       (Type : FIELD)
|   |   |   |   |   via CDD$DATA_ELEMENT_BASED_ON
|   |   |   MINIATURE                       (Type : FIELD)
|   |   |   |   via CDD$DATA_AGGREGATE_CONTAINS
|   |   |   |   SQL$10CHR                       (Type : FIELD)
|   |   |   |   |   via CDD$DATA_ELEMENT_BASED_ON
|   |   |   TOY                             (Type : FIELD)
|   |   |   |   via CDD$DATA_AGGREGATE_CONTAINS
|   |   |   |   SQL$10CHR                       (Type : FIELD)
|   |   |   |   |   via CDD$DATA_ELEMENT_BASED_ON
CDO> EXIT
```

4.3.3 Making a Database Table Shareable in the Repository

To make a database table shareable in a repository, use the CDO ENTER command. For example:

```
$ REPOSITORY
   .
   .
   .
CDO> ENTER RECORD POODLES FROM DATABASE DOGS
CDO>
```

The CDO ENTER command lets you assign a directory name to an element that exists in the repository, so that you can display and share the definition in the database. You must be sure that your current default directory does not contain a directory name that is the same as the processing name of the element to be shared. If it does, the CDO ENTER command will fail.

4.3.4 Changing a Field's Datatype Using CDO

If you change the datatype of a field using CDO and the field is used in an Oracle Rdb hashed index, you must delete the index in order to change the field, then re-create the index.

4.3.5 Updating the Database File Using the Repository Definitions

The previous sections described how the SQL INTEGRATE statement uses the database metadata to create and update repository definitions. Sometimes, you need to perform the operation in the opposite direction; that is, use repository definitions to update the database.

If you have created a table or domain with the SQL CREATE TABLE . . . FROM statement or the SQL CREATE DOMAIN . . . FROM statement, and then changed the definitions in the repository, you can use the SQL INTEGRATE DATABASE . . . ALTER FILES statement to update the database metadata to match the repository definitions.

The SQL INTEGRATE . . . ALTER FILES statement has no effect on metadata that was not created using the SQL CREATE TABLE . . . FROM or the SQL CREATE DOMAIN . . . FROM statements.

To update the database metadata so that it matches the repository definitions, use the SQL INTEGRATE DATABASE . . . PATHNAME . . . ALTER FILES statement.

Example 4–16 shows a typical situation where you would use the SQL INTEGRATE statement with the ALTER FILES clause. In the example, fields and records are defined in the repository. Then a table is created in SQL based on the repository definitions. The repository definitions are subsequently changed, and the database metadata and the repository definitions no longer match.

The SQL INTEGRATE statement resolves this situation by altering the database using the repository definitions as the source.

Example 4–16 Updating the Database to Match Repository Definitions

```
   .
   .
   .
CDO> SET DEFAULT SYS$COMMON:[CDDREP]DEPT32.FIELDMAN
CDO> DEFINE FIELD FIRST DATATYPE IS TEXT 4.
CDO> DEFINE FIELD SECOND DATATYPE IS TEXT 4.

CDO> DEFINE RECORD ORDER.
CDO> FIRST.
CDO> SECOND.
CDO> END ORDER RECORD.
CDO> EXIT
```

Enter SQL, and create the database TEST1. Then create a table, ORDER, in the TEST1 database. Use the table just created in the repository, as follows:

```
$ MCR SQL$
SQL> CREATE DATABASE FILENAME TEST1 PATHNAME TEST1;
SQL> CREATE TABLE FROM SYS$COMMON:[CDDREP]DEPT32.FIELDMAN.ORDER;
SQL>
```

Use the SQL SHOW TABLE statement to see information about the table:

```
   .
   .
   .
SQL> SHOW TABLE ORDER

Columns for table TEST1.ORDER:
Column Name                    Data Type        Domain
-----------                    ---------        ------
FIRST                          CHAR(4)          FIRST
SECOND                         CHAR(4)          SECOND

   .
   .
   .
SQL> COMMIT;
SQL> EXIT
```

(continued on next page)

Example 4–16 (Cont.) Updating the Database to Match Repository Definitions

Enter CDO again. Verify which fields are in the record ORDER:

```
$ REPOSITORY
   .
   .
   .
CDO> SHOW RECORD ORDER
Definition of record ORDER
|   Contains field          FIRST
|   Contains field          SECOND
```

Define the fields THIRD and FOURTH and add them to the record ORDER. Now the repository definitions and the database metadata no longer match:

```
   .
   .
   .
CDO> SET DEFAULT SYS$COMMON:[CDDREP]DEPT32.FIELDMAN
CDO> DEFINE FIELD THIRD DATATYPE IS TEXT 4.
CDO> DEFINE FIELD FOURTH DATATYPE IS TEXT 4.
CDO> CHANGE RECORD ORDER.
CDO> DEFINE THIRD.
CDO> END.
CDO> DEFINE FOURTH.
CDO> END.
CDO> END ORDER RECORD.


CDO> SHOW RECORD ORDER
Definition of record ORDER
|   Contains field          FIRST
|   Contains field          SECOND
|   Contains field          THIRD
|   Contains field          FOURTH

CDO> EXIT
```

Enter SQL again. Use the SQL INTEGRATE statement to alter the database, making it the same as the repository. The SQL INTEGRATE statement implicitly attaches to the database:

```
$ MCR SQL$
SQL> INTEGRATE DATABASE PATHNAME SYS$COMMON:[CDDREP]DEPT32.FIELDMAN.TEST1
cont> ALTER FILES;
   .
   .
   .
```

(continued on next page)

Example 4–16 (Cont.) Updating the Database to Match Repository Definitions

The SQL SHOW TABLE statement indicates that the table ORDER has changed. SQL has integrated the THIRD and FOURTH domains into the database:

```
   .
   .
   .
SQL> SHOW TABLE ORDER

Columns for table ORDER:
Column Name                    Data Type        Domain
-----------                    ---------        ------
FIRST                          CHAR(4)          FIRST
SECOND                         CHAR(4)          SECOND
THIRD                          CHAR(4)          THIRD
FOURTH                         CHAR(4)          FOURTH

SQL> COMMIT;
SQL> EXIT
```

4.3 Modifying Repository Definitions and Database Metadata

Table 4–1 summarizes the SQL statements and steps you use to modify repository definitions and database metadata using CREATE DATABASE and INTEGRATE DATABASE statements.

Table 4–1 Modify Repository Definitions and Database Metadata

To:	Use SQL Statements:
Update repository from database source	1. CREATE DATABASE . . . PATHNAME . . . DICTIONARY IS REQUIRED 2. ATTACH . . . PATHNAME
Update database only[1]	1. CREATE DATABASE . . . PATHNAME . . . DICTIONARY IS NOT REQUIRED 2. ATTACH . . . FILENAME
Update existing database from repository source	INTEGRATE DATABASE . . . PATHNAME . . . ALTER FILES
Create a new repository definition from existing database source	INTEGRATE DATABASE . . . FILENAME . . . CREATE PATHNAME
Update repository from existing database source	INTEGRATE DATABASE . . . PATHNAME . . . ALTER DICTIONARY

[1] If you issue the SQL CREATE DATABASE . . . PATHNAME statement with the DICTIONARY IS REQUIRED clause and later use the ATTACH . . . FILENAME statement, you will update the database *only*; an error will occur when you attempt to update the repository because DICTIONARY IS REQUIRED was specified in the CREATE statement, and the attach is by FILENAME.

(continued on next page)

Table 4–1 (Cont.) Modify Repository Definitions and Database Metadata

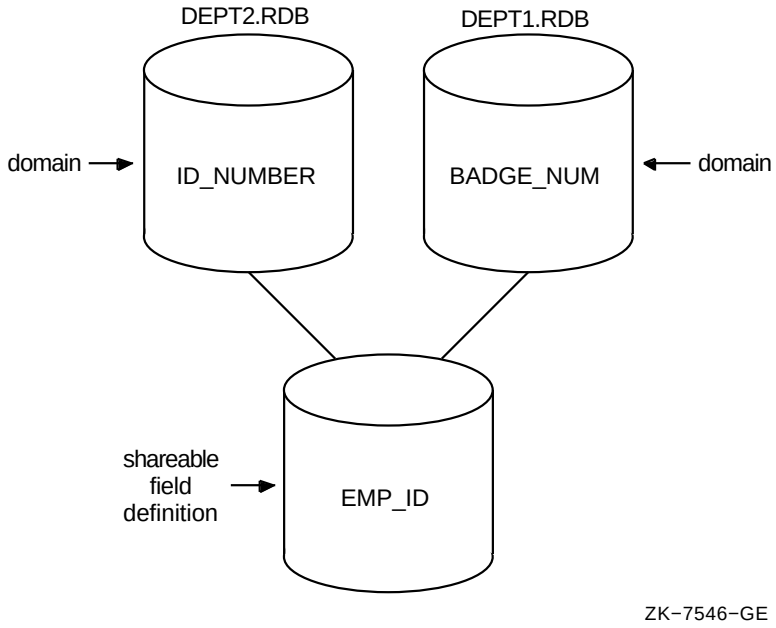
To:	Use SQL Statements:
Update a previously defined, single-named global field in the repository from the database	1. CREATE DATABASE . . . DICTIONARY IS REQUIRED 2. INTEGRATE DOMAIN . . . ALTER DICTIONARY
Update a previously defined, single-named relation in the repository from the database	1. CREATE DATABASE . . . DICTIONARY IS REQUIRED 2. INTEGRATE TABLE . . . ALTER DICTIONARY
Update a previously defined, single-named global field in the database from the repository	1. CREATE DATABASE . . . DICTIONARY IS REQUIRED 2. INTEGRATE DOMAIN . . . ALTER FILES
Update a previously defined, single-named relation in the database from the repository	1. CREATE DATABASE . . . DICTIONARY IS REQUIRED 2. INTEGRATE TABLE . . . ALTER FILES

For more information on using the SQL INTEGRATE statement, see the *Oracle Rdb SQL Reference Manual.*

4.4 Deleting Definitions Using SQL and CDO

You cannot use SQL to delete shared fields (domains) that reside in repositories. For example, Figure 4–2 shows two fields in two separate Oracle Rdb databases that depend on one Oracle CDD/Repository shareable field definition.

Figure 4–2 Shareable Fields in Oracle CDD/Repository



Because the EMP_ID field is used by both ID_NUMBER and BADGE_NUM, do not use SQL to delete EMP_ID. However, when you attach to a database by path name and issue a DROP DOMAIN statement, you delete that database's connection between that field and the record in the repository. The field still exists in the repository for other databases that share the field. For example:

```
$ MCR SQL$
SQL> ATTACH 'PATHNAME DEPT2';
SQL> DROP DOMAIN ID_NUMBER;
SQL> COMMIT;
SQL> ATTACH 'PATHNAME DEPT1';
SQL> DROP DOMAIN BADGE_NUM;
SQL> COMMIT;
SQL>
```

You can delete a field if it is not used by another element. Use the CDO SHOW USES command to see the uses of the field EMP_ID, as in Example 4–17.

Example 4–17 Using the CDO SHOW USES Command to Track the Uses of a Field

```
$ REPOSITORY
   .
   .
   .
CDO> SHOW USES EMP_ID

Owners of SYS$COMMON:[CDDREP]PERS.EMP_ID(1)
|    SYS$COMMON:[CDDREP]PERS.BADGE_NUM(1)  (Type : FIELD)
|     |    via CDD$DATA_ELEMENT_BASED_ON
|    SYS$COMMON:[CDDREP]PERS.ID_NUMBER(1)    (Type : FIELD)
|     |    via CDD$DATA_ELEMENT_BASED_ON
```

Because the field is used by other entities, you cannot delete it.

```
CDO> DELETE FIELD EMP_ID.
%CDO-E-NOTDELETED, entity EMP_ID not deleted
-CDD-E-INUSE, entity is the member of a relationship; it cannot be deleted
```

Example 4–18 also shows how a shared definition cannot be deleted from the repository using SQL. In this example, the repository record EMPLOYEES contains the record ADDRESS_DATA_2, and the database table EMPLOYEES contains the domain ADDRESS_DATA_2.

Example 4–18 Repository Definition Not Removed After SQL DROP COLUMN

```
$ REPOSITORY
   .
   .
   .
CDO> SET DEFAULT SYS$COMMON:[CDDREP]PERS
CDO> SHOW RECORD EMPLOYEES
Definition of record EMPLOYEES
 |    Description              'CORPORATE EMPLOYEE INFORMATION'
 |    Contains field           EMPLOYEE_ID
 |    Contains field           LAST_NAME
 |    Contains field           FIRST_NAME
 |    Contains field           MIDDLE_INITIAL
 |    Contains field           ADDRESS_DATA_1
 |    Contains field           ADDRESS_DATA_2
 |    Contains field           CITY
 |    Contains field           STATE
 |    Contains field           POSTAL_CODE
 |    Contains field           BIRTHDAY
 |    Contains field           SEX
 |    Contains field           STATUS_CODE

$ MCR SQL$
SQL> ATTACH 'FILENAME DEPT1';
SQL> SHOW TABLE EMPLOYEES

CDD Pathname:   SYS$COMMON:[CDDREP]PERS.EMPLOYEES(1)

Comment on table EMPLOYEES:
CORPORATE EMPLOYEE INFORMATION
```

(continued on next page)

Example 4–18 (Cont.) Repository Definition Not Removed After SQL DROP COLUMN

```
Columns for table EMPLOYEES:
Column Name                    Data Type        Domain
-----------                    ---------        ------
EMPLOYEE_ID                    CHAR(5)          EMPLOYEE_ID
LAST_NAME                      CHAR(14)         LAST_NAME
FIRST_NAME                     CHAR(10)         FIRST_NAME
MIDDLE_INITIAL                 CHAR(1)          MIDDLE_INITIAL
ADDRESS_DATA_1                 CHAR(25)         ADDRESS_DATA_1
ADDRESS_DATA_2                 CHAR(25)         ADDRESS_DATA_2
CITY                           CHAR(20)         CITY
STATE                          CHAR(2)          STATE
POSTAL_CODE                    CHAR(9)          POSTAL_CODE
BIRTHDAY                       DATE             BIRTHDAY
SEX                            CHAR(1)          SEX
STATUS_CODE                    CHAR(1)          STATUS_CODE
```

If you enter the SQL ALTER TABLE statement and use the DROP COLUMN clause, the definition of the field ADDRESS_DATA_2 is not removed from the repository.

```
   .
   .
   .
SQL> ALTER TABLE EMPLOYEES
cont>      DROP COLUMN ADDRESS_DATA_2;
SQL> COMMIT;
SQL> EXIT;
```

(continued on next page)

Example 4–18 (Cont.) Repository Definition Not Removed After SQL DROP COLUMN

```
$ REPOSITORY
   .
   .
   .
CDO> SET DEFAULT SYS$COMMON:[CDDREP]PERS
CDO> SHOW RECORD EMPLOYEES
Definition of record EMPLOYEES
 |    Description             'CORPORATE EMPLOYEE INFORMATION'
 |    Contains field          EMPLOYEE_ID
 |    Contains field          LAST_NAME
 |    Contains field          FIRST_NAME
 |    Contains field          MIDDLE_INITIAL
 |    Contains field          ADDRESS_DATA_1
 |    Contains field          ADDRESS_DATA_2
 |    Contains field          CITY
 |    Contains field          STATE
 |    Contains field          POSTAL_CODE
 |    Contains field          BIRTHDAY
 |    Contains field          SEX
 |    Contains field          STATUS_CODE
```

However, the link between the database and the field is deleted.

```
   .
   .
   .
CDO> EXIT
$ MCR SQL$
SQL> SHOW TABLE EMPLOYEES

CDD Pathname:   SYS$COMMON:[CDDREP]PERS.EMPLOYEES(1)

Comment on table EMPLOYEES:
CORPORATE EMPLOYEE INFORMATION
```

(continued on next page)

Example 4–18 (Cont.) Repository Definition Not Removed After SQL DROP COLUMN

```
Columns for table EMPLOYEES:
Column Name                    Data Type        Domain
-----------                    ---------        ------
EMPLOYEE_ID                    CHAR(5)          EMPLOYEE_ID
LAST_NAME                      CHAR(14)         LAST_NAME
FIRST_NAME                     CHAR(10)         FIRST_NAME
MIDDLE_INITIAL                 CHAR(1)          MIDDLE_INITIAL
ADDRESS_DATA_1                 CHAR(25)         ADDRESS_DATA_1
CITY                           CHAR(20)         CITY
STATE                          CHAR(2)          STATE
POSTAL_CODE                    CHAR(9)          POSTAL_CODE
BIRTHDAY                       DATE             BIRTHDAY
SEX                            CHAR(1)          SEX
STATUS_CODE                    CHAR(1)          STATUS_CODE
```

To delete repository definitions, use the SQL DROP PATHNAME statement. The SQL DROP PATHNAME statement does not delete the database metadata; it deletes only the repository definitions. Shared definitions are not deleted.

4.5 Deciding Whether to Use Oracle CDD/Repository

If your application consists of many Oracle Rdb databases, you should decide whether or not to use Oracle CDD/Repository. Consider the following tradeoffs when making the decision:

- Field and record definitions in the repository are stored in a generic format that can be interpreted appropriately by different products.
- You cannot rename a repository definition as you include it into a database.
- Shareable definitions created in CDO can be changed from either CDO or SQL.
- Whenever database metadata is changed without the corresponding repository change, inconsistencies can exist between copies of the definitions in the repository and database. If you use SQL and invoke a database using the PATHNAME clause, you will receive a message that the repository definitions have changed. For example:

```
      .
      .
      .
   SQL> ATTACH 'PATHNAME DEPT1';

   %SQL-I-DIC_DB_CHG1, A dictionary definition used by schema
     SYS$COMMON:[CDDREP]PERS.DEPT1(1) has changed
   -SQL-I-DIC_DB_CHG2, Use the INTEGRATE statement to resolve any
     differences between the dictionary and the database
   %CDD-I-MESS, entity has messages
      .
      .
      .
```

If your database design requires that all metadata updates be maintained in the repository, use the SQL DICTIONARY IS REQUIRED clause when you issue the SQL CREATE DATABASE statement.

Then, if you use SQL and attach to a database using the FILENAME clause and you attempt to manipulate database definitions, you will receive the following messages:

```
   .
   .
   .
SQL> CREATE DOMAIN EMPLOYEE_NAME IS CHAR (10);
%RDB-E-NO_META_UPDATE, metadata update failed
-RDMS-F-CDDISREQD, CDD required for metadata updates is not
   being maintained
SQL> ROLLBACK;
SQL> DISCONNECT ALL;
```

Storing database definitions in the repository provides a central source of shareable field and record definitions. To avoid data definition inconsistencies, attach to the database using the Oracle CDD/Repository path name, with the SQL ATTACH . . . PATHNAME statement. By doing this, the database definitions are available to other products that use the repository.

In Oracle CDD/Repository and Oracle Rdb environments, you can use either CDO or SQL to define fields (domains) and records (tables). In some cases, however, using CDO has certain advantages because records and fields can be defined, shared, tracked, and controlled independently of a particular database. You must define fields and records in CDO or use the CDO ENTER command for them to be shareable, as described in Section 4.3.3.

You should consider implementing database definitions through Oracle CDD/Repository if you need to track and share definitions.

Not all database designs benefit equally from a centrally placed common repository. You need not create definitions in the repository if the definitions are local to one application and are likely to remain local, or if requirements for control of definitions are met by existing SQL statements.

4.6 Restoring a Database That Uses Shareable Repository Definitions

You can copy an Oracle Rdb database using either of the following two methods:

- RMU/BACKUP and RMU/RESTORE commands—for regular maintenance backups or disaster recovery
- SQL IMPORT and EXPORT statements—for unloading and reloading databases, restructuring physical database files, or migrating a database such as Oracle Rdb to another database.

4.6.1 Backing Up and Restoring Databases

Do not use the OpenVMS BACKUP procedure to back up your database. Use the RMU/BACKUP command.

The RMU/BACKUP command creates a backup copy of an Oracle Rdb database and places it in an .RBF file. You can back up the entire database or you can request an incremental backup that backs up only the pages that have changed since the last full backup. In the event of subsequent damage to the database, you can specify backup files in an RMU/RESTORE command to restore the database to the condition it was in when you backed it up.

The RMU/RESTORE command re-creates all the relationships between the database structure and shared definitions individually defined in CDO.

You can rename or move the files that comprise a database by using the RMU/BACKUP and RMU/RESTORE command combination. To move a multifile Oracle Rdb database, you *must* use the RMU/BACKUP and RMU/RESTORE commands or the SQL EXPORT and IMPORT statements.

Do not use the DCL COPY command with a multifile database; otherwise, the resulting database will be corrupt and unusable.

For more information about how to perform an RMU backup operation, see the Oracle Rdb documentation.

4.6.2 Restructuring and Reloading Databases

You can use the SQL EXPORT and IMPORT statements to perform the following tasks:

- Migrate a database from one system to another
- Change parameters, such as a storage area definition or page size
- Reload a database (leaving the storage area definitions the same, but changing the device specifications)

For more details about using SQL IMPORT and EXPORT statements, see the *Oracle Rdb SQL Reference Manual*.

4.7 Deleting Databases

You can delete Oracle Rdb databases with the SQL DROP DATABASE statement, which has the following two clauses:

- PATHNAME—Deletes the database and the repository definition for the database
- FILENAME—Deletes only the database

Caution

Use the SQL DROP DATABASE statement with care. You cannot use the ROLLBACK statement to cancel a DROP DATABASE statement.

The SQL DROP DATABASE statement deletes the database root file (.RDB), its snapshot files (.SNP), and any storage area files (.RDA), which include all data and all definitions. The PATHNAME clause deletes the repository database definitions from the repository in addition to these files.

Issue the SQL DROP DATABASE statement before attaching to the database or after issuing the DISCONNECT ALL statement, because you cannot delete a database when there are active users for that database.

When you use the PATHNAME clause, you can specify either one of the following:

- A full repository path name, such as CDD$USER:[JONES.PERS]DEPT1
- A relative repository path name, such as DEPT1

If you use a relative path name, be sure that the current repository default directory is defined to be all of the path segments preceding the relative path name.

Example 4–19 shows how to delete database DEPT1 using the SQL DROP DATABASE . . . PATHNAME statement. In this example, the database, DEPT1, is in the current Oracle CDD/Repository directory, CDD$USER:[JONES.PERS]:

Example 4–19 Deleting a Database Using the SQL DROP DATABASE PATHNAME Statement

```
$ REPOSITORY OPERATOR
Welcome to CDO V6.1
The CDD/Repository V6.1 User Interface
Type HELP for help
CDO> show database

Definition of database  DEPT1
|   database uses RDB database DEPT1
|   database in file DEPT1
|   |   fully qualified file CDD$USER:[JONES.PERS]DEPT1.RDB;
Definition of database  FOO1
|   database uses RDB database FOO1
|   database in file FOO1
|   |   fully qualified file CDD$USER:[JONES.PERS]FOO1.RDB;
Definition of database  TEST1
|   database uses RDB database TEST1
|   database in file TEST1
|   |   fully qualified file CDD$USER:[JONES.PERS]TEST1.RDB;
CDO> exit

$ mcr sql$
SQL> attach 'pathname dept1';
SQL> show databases
Default alias:
    data dictionary pathname is CDD$USER:[JONES.PERS]DEPT1
    DEC Rdb database in file dept1
SQL> drop database pathname dept1;
SQL> attach 'pathname dept1';
%CDD-E-NOT_A_DB, dept1, is not the name of a CDD/Plus dictionary database
```

Because you have successfully deleted DEPT1, SQL shows no databases in the current Oracle CDD/Repository directory when you issue the SQL ATTACH statement. Using CDO, you can also see that the database has been deleted from the repository.

```
   .
   .
   .
SQL> $ mcr cdo
Welcome to CDO V6.1
The CDD/Repository V6.1 User Interface
Type HELP for help
CDO> directory

 Directory CDD$USER:[JONES.PERS]
   .
   .
   .
FOO1(1)                                          CDD$DATABASE
   .
   .
   .
TEST1(1)                                         CDD$DATABASE
   .
   .
   .
CDO> exit
```

When you delete a database using the SQL DROP DATABASE . . . FILENAME statement, the database is deleted, but the database definition remains in the repository.

You can use either a full or partial file specification when deleting the database using the DROP DATABASE . . . FILENAME statement.

Example 4–20 shows how to delete the database FOO1, which was created with the DICTIONARY IS NOT REQUIRED clause.

Example 4–20 Deleting a Database Using the SQL DROP DATABASE FILENAME Statement

```
   .
   .
   .
SQL> drop database filename 'foo1';
SQL> show databases
%SQL-F-ERRATTDEF, Could not use database file specified by SQL$DATABASE
-RDB-E-BAD_DB_FORMAT, SQL$DATABASE does not reference a database known to Rdb
-RMS-E-FNF, file not found
SQL> attach 'filename foo1';
%SQL-F-ERRATTDEC, Error attaching to database foo1
-RDB-E-BAD_DB_FORMAT, foo1 does not reference a database known to Rdb
-RMS-E-FNF, file not found
```

(continued on next page)

Example 4–20 (Cont.) Deleting a Database Using the SQL DROP DATABASE FILENAME Statement

Although the database has been deleted, the definition remains in the repository:

```
SQL> $ mcr cdo
Welcome to CDO V6.1
The CDD/Repository V6.1 User Interface
Type HELP for help
CDO> directory

 Directory CDD$USER:[JONES.PERS]
    .
    .
    .
 FOO1(1)                                          CDD$DATABASE
    .
    .
    .
TEST1(1)                                         CDD$DATABASE
    .
    .
    .
```

You can delete the CDD$DATABASE definition for FOO1 by using the CDO DELETE GENERIC command, as follows:

```
   .
   .
   .
CDO> delete generic cdd$database foo1.
CDO> exit
SQL> $ directory *.rdb

Directory CDD$USER:[JONES.PERS]

TEST1.RDB;1           1097
SQL> exit
```

Index

D

E

F

H

I

L

M

N

O

P

Q

R

S