CHAPTER 6

Best Practices in Oracle Data Guard with Tips and Techniques

This chapter expands the book's Data Guard coverage and explains how to use many of the new features of Oracle Data Guard so you can start using them and get more use from your Oracle Data Guard databases.

Creating a Standby Database Using DBCA

In 12c R2, it is possible to create Data Guard standby databases with the Database Creation Assistant Utility (DBCA). This can be accomplished with a one-liner CLI call that is parameterized and can be used to greatly enhance and simplify attempts at creating DBaaS offerings that are developed in-house. This feature is an extension of DBCA's capabilities to duplicate databases.

Expected Error Messages per Limitations

While this feature is useful, there are a few drawbacks. For one, it is not possible to make Data Guard standby databases using DBCA when the primary database is a cluster database. The second, and probably most limiting factor, is that container databases (CDBs) are not supported. Therefore, this can be used only on single-instance noncontainer databases.

Y V Ravi Kumar et al., *Oracle High Availability, Disaster Recovery, and Cloud Services*,
https://doi.org/10.1007/978-1-4842-4351-0_6

As stated earlier, if the primary database is a cluster database, DBCA will fail with the following error:

```
[FATAL] [DBT-16056] Specified primary database is not a Single Instance
(SI) database.
   CAUSE: Duplicate database operation is supported only for SI databases.
```

A similar error message is also displayed if the primary database is a CDB.

```
[FATAL] [DBT-16057] Specified primary database is a container database (CDB).
   CAUSE: Duplicate database operation is supported only for non container
databases.
```

Example of a DBCA-Created Standby

Creating a simple Data Guard standby database is fairly straightforward with DBCA. The main flags that need to be passed to DBCA are `-createDuplicateDB` and `-createAsStandby`.

```
[oracle@rac501 ~]$ dbca -silent \
> -createDuplicateDB \
> -primaryDBConnectionString stpldb01-scan:1521/orcl.world \
> -gdbName orcl.world -sid orcldr \
> -createAsStandby -dbUniqueName orcldr \
> -datafileDestination +DATA/
Enter SYS user password:

[WARNING] [DBT-11203] Java pool size specified is too small.
   ACTION: Java pool size of at least (20MB) is recommended.
Listener config step
33% complete
Auxiliary instance creation
66% complete
RMAN duplicate
100% complete
Look at the log file "/u01/app/oracle/cfgtoollogs/dbca/orcldr/orcl.log" for
further details.
```

Alternative Ways to Supply the SYS Password

Whether you're using DBCA to create a Data Guard standby or a normal duplicate database, you will need to specify credentials for the SYS account in some way. There are two main ways this can be done. Credentials can be entered interactively (DBCA requests the SYS password from the standard input, as shown earlier), or credentials can be passed in I/O redirection.

I/O Redirection

Do note that the > characters are the by-product of using \ to escape newline characters and denote a continuation of a single command. In this example, the SYS password `database_123` is being redirected into the command. Special care should be made to ensure that the password is not exposed in the running list of processes viewable by `ps`.

Here is an example of redirecting using a here-document.

```
[oracle@rac501 ~]$ dbca -silent -createDuplicateDB \
> -primaryDBConnectionString  stpldb01-scan:1521/orcl.world \
> -gdbName orcl.world -sid orcldr \
> -createAsStandby -dbUniqueName orcldr \
> -datafileDestination +DATA/ <<EOF
> database_123
> EOF
```

Here is an example of redirecting using a pipe.

```
[oracle@rac501 ~]$ echo "database_123" | dbca -silent -createDuplicateDB \
> -primaryDBConnectionString  stpldb01-scan:1521/orcl.world \
> -gdbName orcl.world -sid orcldr \
> -createAsStandby -dbUniqueName orcldr \
> -datafileDestination +DATA/
```

Diagnostic Pack on Active Data Guard

Starting with 12c R2, Oracle supports the Diagnostic Pack on an Active Data Guard instance database. As more and more people are starting to leverage this Active Data Guard feature to offload read-only workloads to Data Guard standby databases, the need for performance monitoring and tuning features provided by the Diagnostic Pack has been satisfied in 12c R2 with the Remote Management Framework.

Because ADG databases are in read-only mode, they cannot persist AWR information. Therefore, Active Workload Repository (AWR) data is transmitted to the primary database from the standby so that it can be persisted in the Active Workload Repository for later use by AWR reports. The communications between the primary database and the standby databases are mediated by database links that connect as the SYS$UMF user. In the context of the Remote Management Framework, the primary database is considered the *destination database*, and the standby database is considered the *source database*. This is because the standby database is the *source* of the AWR data in question, and the primary database is the *destination* to which the standby database sends its AWR data. The Remote Management Framework is compatible with Oracle's multitenancy feature, and RMF can be configured at the PDB or CDB level. In this chapter, RMF will be configured at the CDB level.

The SYS$UMF User

The SYS$UMF user is the user who is set up by default to have access to all of the Remote Management Framework views and is used for capturing and managing AWR data from standby databases. The SYS$UMF user is locked and expired at database creation and needs to be unlocked.

```
SQL> col username format a10
SQL> col account_status format a20
SQL> set lines 150
SQL> select username, account_status from dba_users where username = 'SYS$UMF';

USERNAME   ACCOUNT_STATUS
---------- --------------------
SYS$UMF    EXPIRED & LOCKED

SQL> show con_name

CON_NAME
------------------------------
CDB$ROOT

SQL> alter user SYS$UMF account unlock identified by "database_123" container=ALL;
```

This changes to this user need to be made in CDB$ROOT with the containers clause. The following is an example of the type of error that will be seen if an attempt is made to modify this user within a primary database (PDB) in a CDB database:

```
SQL> show con_name

CON_NAME
------------------------------
SOMEPDB

SQL> alter user SYS$UMF account unlock identified by "database_123";
alter user SYS$UMF account unlock identified by "database_123"
*
ERROR at line 1:
ORA-65066: The specified changes must apply to all containers

User altered.
```

Creating the DB Links

The DBA can create the database links manually. Two links are needed: one database link for the primary database to connect to the standby database and one database link for the standby database to connect to the primary database.

```
SQL> show con_name

CON_NAME
------------------------------
CDB$ROOT

SQL> create public database link ORCLCDB_TO_ORCLDR connect to SYS$UMF
identified by "database_123" using 'orcldr';

Database link created.

SQL> create public database link ORCLDR_TO_ORCLCDB connect to SYS$UMF
identified by "database_123" using 'orclcdb';

Database link created.
```

Configuring the Remote Management Framework Topology

The primary and standby databases need to be configured with the DBMS_UMF package so that the Remote Management Framework can be created and used. In the following example, all of the calls to the DBMS_UMF package are called from the primary database (known as the *target* database in RMF parlance).

First, a topology will need to be created. All source and target databases belong to a *topology*.

```
SQL> exec DBMS_UMF.create_topology ('RMF_TOPOLOGY');

PL/SQL procedure successfully completed.
```

After the topology has been created, the *source* (standby) node needs to be registered, and the database links need to be specified. One particular argument to note here is the node_name argument. This argument is case sensitive and must match the db_unique_name value of the remote database.

```
SQL> BEGIN
  2         DBMS_UMF.REGISTER_NODE(
  3         topology_name        => 'RMF_TOPOLOGY'
  4         ,node_name            => 'orcldr'
  5         ,dblink_to_node       => 'ORCLCDB_TO_ORCLDR'
  6         ,dblink_from_node     => 'ORCLDR_TO_ORCLCDB'
  7         ,as_source            => 'TRUE'
  8         ,as_candidate_target => 'FALSE'
  9         );
 10  END;
 11  /
```

Once the remote node has been registered with RMF, it need to be registered as a remote database in AWR with DBMS_WORKLOAD_REPOSITORY.

```
SQL> exec DBMS_WORKLOAD_REPOSITORY.register_remote_database(node_
name=>'orcldr');

PL/SQL procedure successfully completed.
```

Viewing the Topology information

The topology information regarding which nodes are registered for remote AWR is exposed via DBA_UMF_* views. DBA_UMF_TOPOLOGY contains information about the various topologies that may exist in a *target* database. DBA_UMF_REGISTRATION contains information about all the nodes that are registered in the Remote Management Framework as well as their respective roles within the framework. DBA_UMF_SERVICE has information that shows which registered nodes have an AWR service running for remote collection (the two views can be joined on NODE_ID). The output shown next will describe the topology that the examples in this chapter created:

```
SQL> col topology_name format a15
SQL> set lines 200

SQL> select * from DBA_UMF_TOPOLOGY;

TOPOLOGY_NAME    TARGET_ID TOPOLOGY_VERSION TOPOLOGY
--------------- ---------- ---------------- --------
RMF_TOPOLOGY     554680322                4 ACTIVE

SQL> select * from DBA_UMF_REGISTRATION;

TOPOLOGY_NAME   NODE_NAME     NODE_ID  NODE_TYPE AS_SO AS_CA STATE
--------------- ---------- ---------- ---------- ----- ----- ----------
RMF_TOPOLOGY    orclcdb     554680322          0 FALSE FALSE OK
RMF_TOPOLOGY    orcldr      301115545          0 TRUE  FALSE OK

SQL> select * from DBA_UMF_SERVICE;

TOPOLOGY_NAME      NODE_ID SERVICE
--------------- ---------- -------
RMF_TOPOLOGY     301115545 AWR

1 row selected.
```

Taking AWR Snapshots on ADG Databases

AWR snapshots can be created with DBMS_WORKLOAD_REPOSITORY by calling the CREATE_REMOTE_SNAPSHOT procedure and referencing the registered node_name for the source (standby database) as an argument.

```
exec DBMS_WORKLOAD_REPOSITORY.CREATE_REMOTE_SNAPSHOT('orcldr');
```

Creating AWR Reports for the Standby Database

It is recommended that you use the awrgrpti.sql script to pull the AWR reports because you can use it to pull a group AWR report of all instances and it allows you to set the DB_ID value. This value will allow the administrator to choose the ADG database's snapshots. While it is true that a standby database actually has the same DB_ID value as the primary database, the Remote Management Framework uses the NODE_ID value, which is found in DBA_UMF_REGISTRATION as a substitute for the source's DB_ID value. The AWR reports can be generated from the primary or standby database; it makes no difference.

```
SQL> @?/rdbms/admin/awrgrpti.sql

 Specify the Report Type
 ~~~~~~~~~~~~~~~~~~~~~~~
 AWR reports can be generated in the following formats.  Please enter the
 name of the format at the prompt. Default value is 'html'.

   'html'          HTML format (default)
   'text'          Text format
   'active-html'   Includes Performance Hub active report

Enter value for report_type: html

Type Specified: html
```

```
Instances in this Workload Repository schema
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

  DB Id      Inst Num   DB Name      Instance     Host
------------ ---------- ---------    ----------   ------
  301115545      1      ORCLCDB      orcldr1      rac501.local
  301115545      2      ORCLCDB      orcldr2      rac502.local
* 2756161440     1      ORCLCDB      orclcdb1     stpldb101.lo
* 2756161440     2      ORCLCDB      orclcdb2     stpldb102.lo

Enter value for dbid: 301115545
Using 301115545 for database Id
Enter value for instance_numbers_or_all: ALL
Using instances ALL (default 'ALL')

Specify the number of days of snapshots to choose from
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

Entering the number of days (n) will result in the most recent
(n) days of snapshots being listed.  Pressing <return> without
specifying a number lists all completed snapshots.

Enter value for num_days: 1


Listing the last day's Completed Snapshots
DB Name      Snap Id       Snap Started    Snap Level
------------ ---------- ------------------ ----------

ORCLCDB              1  25 Aug 2017 18:20    1
                     2  25 Aug 2017 18:21    1

Specify the Begin and End Snapshot Ids
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

Enter value for begin_snap: 1
Begin Snapshot Id specified: 1

Enter value for end_snap: 2
End   Snapshot Id specified: 2
```

```
Specify the Report Name
~~~~~~~~~~~~~~~~~~~~~~~
The default report file name is awrrpt_rac_1_2.html.  To use this name,
press <return> to continue, otherwise enter an alternative.

Enter value for report_name:

Using the report name awrrpt_rac_1_2.html
```

SQL Tuning Advisor in ADG Databases

As of 12c R2, Active Data Guard databases can now make use of the SQL Tuning Advisor. This has been made possible by allowing `DBMS_SQLTUNE.CREATE_TUNING_TASK` to use database links to communicate back to the primary database for data persistence. At tuning task creation time, the argument `database_link_to` can be supplied with the name of a database link that points back to the primary database. In this way, the database link will be used to store data necessary to create and run tuning tasks and for any subsequent SQL plan baseline or SQL profiles that are suggested by the SQL Tuning Advisor.

Creating a Database Link for the SQL Tuning Advisor

To tune queries on an ADG instance, a database link that points to the primary database must exist. If the SQL Tuning Advisor is being called from within a PDB, the database link must connect to the PDB on the primary database. In this example, we will create a `DB_LINK` called `PRIMARY_DB` that connects as `SYS$UMF` to the pluggable database `SOMEPDB` on the primary database `ORCLCDB`. The database link must connect as `SYS$UMF` to the primary database and must be a private database link owned by the SYS user.

```
SQL> alter session set container=SOMEPDB;

Session altered.

SQL> show con_name;

CON_NAME
------------------------------
SOMEPDB
```

```
SQL> create database link PRIMARY_DB connect to SYS$UMF identified by
"database_123" using 'stpldb01-scan:1521/somepdb.world';

Database link created.
```

Tuning a Query

In this section, you'll see an example of a query on HR.EMPLOYEES from the set of sample schemas supplied by Oracle. All the tests will be run from a PDB called SOMEPDB in the CDB ORCLDR, which is the ADG standby database for the primary database called ORCLCDB.

The Example Query

Here is the query in question:

```
SQL> select employee_id, first_name, last_name, phone_number from
hr.employees where first_name = 'Alexis';

EMPLOYEE_ID FIRST_NAME           LAST_NAME                 PHONE_NUMBER
----------- -------------------- ------------------------- ------------
        185 Alexis               Bull                      650.509.2876

SQL> select * from table(dbms_xplan.display_cursor());
SQL_ID  6h7jvaqfuycvj, child number 0
-------------------------------------
select employee_id, first_name, last_name, phone_number from
hr.employees where first_name = 'Alexis'

Plan hash value: 612698390

---------------------------------------------------------------------------
| Id | Operation           | Name        | Rows | Bytes | Cost (%CPU)| Time     |
---------------------------------------------------------------------------
|  0 | SELECT STATEMENT    |             |      |       |    2 (100) |          |
|  1 | TABLE ACCESS BY
       INDEX ROWID BATCHED | EMPLOYEES   | 1    |   34  |    2   (0) | 00:00:01|
|* 2 | INDEX SKIP SCAN     | EMP_NAME_IX | 1    |       |    1   (0) | 00:00:01|
---------------------------------------------------------------------------
```

```
Predicate Information (identified by operation id):
---------------------------------------------------

   2 - access("FIRST_NAME"='Alexis')
       filter("FIRST_NAME"='Alexis')

21 rows selected.
```

Creating a SQL Tuning Task

The following example will show how an SQL tuning task can be created and executed while connected to the ADG instance:

```
SQL> show con_name

CON_NAME
------------------------------
SOMEPDB
SQL> select instance_name from v$instance;

INSTANCE_NAME
----------------
orcldr1

SQL> select open_mode from v$database;


OPEN_MODE
--------------------
READ ONLY WITH APPLY

SQL> variable STMT_TASK varchar2(2000);

SQL> EXEC :stmt_task := DBMS_SQLTUNE.CREATE_TUNING_TASK(sql_id =>
'6h7jvaqfuycvj', database_link_to => 'PRIMARY_DB.WORLD');

PL/SQL procedure successfully completed.

SQL> select :stmt_task from dual;
```

```
:STMT_TASK
----------
TASK_11
SQL> EXEC DBMS_SQLTUNE.EXECUTE_TUNING_TASK(:stmt_task);

PL/SQL procedure successfully completed.
```

You can view the SQL Tuning Task report by using the `DBMS_SQLTUNE.REPORT_TUNING_TASK` function when selecting from dual and referencing the relevant task name. The following is some example output from a SQL Tuning Task Report that was run on the standby database:

```
SQL> SELECT DBMS_SQLTUNE.REPORT_TUNING_TASK(:stmt_task) from dual;

DBMS_SQLTUNE.REPORT_TUNING_TASK(:STMT_TASK)
-------------------------------------------------------------------------
GENERAL INFORMATION SECTION
-------------------------------------------------------------------------
Tuning Task Name   : TASK_11
Tuning Task Owner  : SYS
Workload Type      : Single SQL Statement
Scope              : COMPREHENSIVE
Time Limit(seconds): 1800
Completion Status  : COMPLETED
Started at         : 04/26/2018 12:57:06
Completed at       : 04/26/2018 12:57:08
SQL Tuning at Standby TRUE

-------------------------------------------------------------------------
Schema Name   : SYS
Container Name: SOMEPDB
SQL ID        : 6h7jvaqfuycvj
SQL Text      : select employee_id, first_name, last_name, phone_number from
                hr.employees where first_name = 'Alexis'

-------------------------------------------------------------------------
There are no recommendations to improve the statement.
```

RMAN Support for NONLOGGED BLOCK recovery

In previous versions of Oracle Database, NOLOGGING operations will cause issues with Data Guard standby databases because they mark the affected database blocks as UNRECOVERABLE and any SQL statements that require reading those UNRECOVERABLE blocks on the ADG database will throw errors. For this reason, it is always recommended you keep the primary database in FORCE LOGGING mode. However, in certain cases, this may not be ideal, or the FORCE LOGGING behavior may need to be altered to expedite data loads. If this scenario ever occurs, recovering the blocks affected by NOLOGGING operations is now possible with RMAN without requiring new backups to be taken on the primary database. In this section, you'll see a thorough example of how this feature works, and all of the work will be done in the SOMEPDB PDB in the ORCLCDB CDB with the ORCLDR ADG database.

Creating a Table with the NOLOGGING Clause

A table will be created with the NOLOGGING clause, and the APPEND hint will be used to load data into the table. This will cause an unrecoverable operation because the data required to re-create the changes will not be stored in the redo logs. This will be done in a database where FORCE_LOGGING is disabled at both the CDB and PDB levels. Additionally, the NOLOGGING table will be created in the USERS tablespace, which does not have FORCE_LOGGING enabled.

```
SQL> select db_unique_name from v$database;

DB_UNIQUE_NAME
------------------------------
Orclcdb

SQL> alter database NO FORCE LOGGING;

Database altered.

SQL> select force_logging from v$database;

FORCE_LOGGING
---------------------------------------
NO
```

```
SQL> select force_logging from dba_pdbs where pdb_name='SOMEPDB';

FOR
---
NO

SQL> alter session set container=SOMEPDB;

Session altered.

SQL> show con_name;

CON_NAME
------------------------------
SOMEPDB

SQL> select tablespace_name, logging,force_logging from dba_tablespaces
where tablespace_name = 'USERS';

TABLESPACE_NAME                LOGGING   FOR
------------------------------ --------- ---
USERS                          LOGGING   NO

SQL> create table HR.EMPLOYEES_NOLOG NOLOGGING TABLESPACE USERS as select *
from hr.employees where 'x'='y';

Table created.

SQL> insert /*+ APPEND */ into HR.EMPLOYEES_NOLOG select * from
hr.employees;

107 rows created.

SQL> commit;

Commit complete.
```

The previous example is intended to show a type of NOLOGGING operation. In the next two sections, the effect of the NOLOGGING operation will be shown on the standby database along with the steps necessary to recover from the situation without requiring a new full backup to be taken of the primary's datafiles for the USERS tablespace.

The Effect of NOLOGGING Operations on the Physical Standby

Once the table has been created and loaded with the APPEND hint, select statements against the table on the ADG database will error out with the ORA-26040 error.

```
SQL> select db_unique_name from v$database;

DB_UNIQUE_NAME
------------------------------
orcldr

SQL> show con_name;

CON_NAME
------------------------------
SOMEPDB

SQL> select * from hr.employees_nolog;
select * from hr.employees_nolog
       *
ERROR at line 1:
ORA-01578: ORACLE data block corrupted (file # 14, block # 131)
ORA-01110: data file 14:
'+DATA/ORCLDR/744827E32DAF16D5E053033AA8C089C7/DATAFILE/
users.273.985102721'
ORA-26040: Data block was loaded using the NOLOGGING option
```

Once a select statement has been run on the table, the unrecoverable blocks will be stored in the V$NONLOGGED_BLOCKS view along with some other metadata.

```
SQL> select file#, block#, blocks object# from V$NONLOGGED_BLOCK;

     FILE#     BLOCK#    OBJECT#
---------- ---------- ----------
        14        131          2
```

Because the OBJECT# value of the impacted objects is stored in the V$NONLOGGED_BLOCK view, it can be used to determine which objects cannot be queried because of NOLOGGING operations.

```
SQL> col owner format a5
SQL> col object_name format a30
SQL> col object_type format a10
SQL> set lines 50
SQL> select owner, object_name, object_type from dba_objects where data_
object_id in (select object# from v$nonlogged_block);

OWNER OBJECT_NAME                    OBJECT_TYP
----- ------------------------------ ----------
HR    EMPLOYEES_NOLOG                TABLE
```

Fixing the Problem

As stated, in previous versions, recovering from such a scenario was complicated and cumbersome. However, in 12c R2, this problem can be overcome with one simple RMAN command. To run the RMAN recovery command, you need to stop managed recovery on the standby database.

This can be done via the Broker if it is configured.

```
[oracle@rac501 ~]$ dgmgrl /
DGMGRL for Linux: Release 12.2.0.1.0

Copyright (c) 1982, 2017, Oracle and/or its affiliates.  All rights reserved.

Welcome to DGMGRL, type "help" for information.
Connected to "orcldr"
Connected as SYSDG.
DGMGRL> show database orcldr

Database - orcldr

  Role:               PHYSICAL STANDBY
  Intended State:     APPLY-ON
  Transport Lag:      0 seconds (computed 0 seconds ago)
  Apply Lag:          0 seconds (computed 0 seconds ago)
  Average Apply Rate: 59.00 KByte/s
  Real Time Query:    ON
  Instance(s):
    orcldr1
    orcldr2 (apply instance)

Database Status:
SUCCESS

DGMGRL> edit database orcldr set state='APPLY-OFF';
Succeeded.
DGMGRL>
```

Once you have stopped managed recovery, you can use RMAN to recover the required blocks on the standby database.

```
RMAN> recover database nonlogged block;

Starting recover
using target database control file instead of recovery catalog
allocated channel: ORA_DISK_1
channel ORA_DISK_1: SID=541 instance=orcldr1 device type=DISK

starting recovery of nonlogged blocks
List of Datafiles
=================
File Status Nonlogged Blocks Blocks Examined Blocks Skipped
---- ------ ---------------- --------------- --------------
1    OK     0                0               108799
3    OK     0                0               104959
4    OK     0                0               37759
5    OK     0                0               33279
6    OK     0                0               60159
7    OK     0                0               639
8    OK     0                0               12799
9    OK     0                0               3199
10   OK     0                0               34559
11   OK     0                0               66559
12   OK     0                0               12799
13   OK     0                0               12799
14   OK     0                2               637
```

```
Details of nonlogged blocks can be queried from v$nonlogged_block view

recovery of nonlogged blocks complete, elapsed time: 00:00:02
```

When the `RECOVER DATABASE NONLOGGED BLOCK` command is issued, the standby database will connect to the primary database and fetch the blocks from the primary datafiles. In this way, there is no requirement to take level 0 backups or copy datafiles from the primary site to the standby site as everything is handled at the block level and is automatically taken care of by the standby. In some cases, this operation may fail because the `NONLOGGED` blocks have not been flushed to the physical datafiles on the primary site. In such cases, you can choose to wait for the blocks to be flushed to the datafile or flush the buffer cache on the primary database. The following is an example of the type of alert log entries you may expect to see on a successful recovery of `NONLOGGED` blocks:

```
alter database recover datafile list clear
Completed: alter database recover datafile list clear

Started Nonlogged Block Replacement recovery on file 14 (ospid 2684 rcvid
2503064493696929793)
Data Transfer Cache defaulting to 64MB. Trying to get it from Buffer Cache
for process 2684.
2018-04-26T14:04:30.076444-04:00
Finished Nonlogged Block Replacement recovery on file 14. 0 blocks remain
  Statistics for replacement block source database (service=orclcdb)
  Blocks requested 2, blocks received 2.

  Reason replacement blocks accepted or rejected            Blocks   Last block
  --------------------------------------------------- ---------   ---------
  Accept: SCN in range for classic non-logged block             2         132
```

Data Guard Support for Multiple Observers

As of 12c R2, Oracle Data Guard now supports up to three observers in a Data Guard Broker configuration. In this type of setup, an observer will be chosen at random to be the master observer and to be the one to initiate Fast Start Fail Over (FSFO) operations. If the master observer fails or is shut down, the surviving observer will be elected to become the master. This feature brings high availability to observers and makes FSFO configurations more reliable. In this section of the chapter, the examples will continue to use the primary database ORCLCDB and the standby database ORCLDR. There will be two observers, one on an Oracle Management Server host called oms1.localdomain and another on one of the database servers hosting the standby database rac501.localdomain.

Starting Multiple Observers

When starting observers manually, there is no special syntax that is required to enable multiple observers; this feature works seamlessly with the Oracle Data Guard Broker.

- OMS1.LOCALDOMAIN

```
DGMGRL> connect sys@orclcdb
Password:
Connected to "orclcdb"
Connected as SYSDBA.
DGMGRL> start observer logfile is '/home/oracle/oms1_orclcdb_observer.log';
```

- RAC501.LOCALDOMAIN

```
DGMGRL> connect sys/database_123@orclcdb
Connected to "orclcdb"
Connected as SYSDBA.
DGMGRL> start observer logfile is '/home/oracle/rac501_orclcdb_observer.log'
```

Determining the Master Observer

You can determine which observer is the master observer via DGMGRL.

The master observer will have an asterisk next to the hostname in the Observers list displayed by show fast_start failover.

```
DGMGRL> show fast_start failover;

Fast-Start Failover: ENABLED

  Threshold:          45 seconds
  Target:             orcldr
  Observers:      (*) oms1.localdomain
                      rac501.localdomain
  Lag Limit:          30 seconds
  Shutdown Primary:   TRUE
  Auto-reinstate:     TRUE
  Observer Reconnect: (none)
  Observer Override:  FALSE

Configurable Failover Conditions
  Health Conditions:
    Corrupted Controlfile          YES
    Corrupted Dictionary           YES
    Inaccessible Logfile            NO
    Stuck Archiver                  NO
    Datafile Write Errors          YES

  Oracle Error Conditions:
    (none)
```

Additionally, the command `show observer` will expose the master/backup relationship directly.

```
DGMGRL> show observer

Configuration - orclcdb

  Primary:            orclcdb
  Target:             orcldr

Observer "oms1.localdomain" - Master

  Host Name:                    oms1.localdomain
  Last Ping to Primary:         1 second ago
  Last Ping to Target:          3 seconds ago

Observer "rac501.localdomain" - Backup

  Host Name:                    rac501.localdomain
  Last Ping to Primary:         2 seconds ago
  Last Ping to Target:          2 seconds ago
```

Manually Changing the Master Observer

If the automatically defined master observer is not desired, the master observer role can be assigned to a different observer via DGMGRL.

```
DGMGRL> set masterobserver to rac501.localdomain;
Sent the proposed master observer to the data guard broker configuration.
Please run SHOW OBSERVER to see if master observer switch actually happens.

DGMGRL> show observer;

Configuration - orclcdb

  Primary:            orclcdb
  Target:             orcldr

Observer "rac501.localdomain" - Master

  Host Name:                    rac501.localdomain
  Last Ping to Primary:         2 seconds ago
  Last Ping to Target:          2 seconds ago

Observer "oms1.localdomain" - Backup

  Host Name:                    oms1.localdomain
  Last Ping to Primary:         0 seconds ago
  Last Ping to Target:          0 seconds ago
```

Data Guard Multi-instance Apply

With 12c R2, Oracle Data Guard now supports multi-instance redo apply (MIRA) on physical standby databases. By default, a physical standby database will not apply redo using multiple instances, but this can be changed via DGMGRL by modifying the applyinstances property of the Broker configuration member for the physical standby database.

Broker Example

In this example, all instances of the ORCLDR database will be used to apply redo. There is an alter database command that can be used if a Broker configuration does not exist.

```
DGMGRL> edit database orcldr set state='APPLY-OFF';
Succeeded.

DGMGRL> edit database orcldr set property applyinstances='ALL';
Property "applyinstances" updated

DGMGRL> edit database orcldr set state='APPLY-ON';
Succeeded.
```

SQLPLUS Example

The INSTANCES clause has been added to the ALTER DATABASE RECOVER MANAGED STANDBY DATABASE command to specify which instances should participate in redo apply.

```
SQL> ALTER DATABASE RECOVER MANAGED STANDBY DATABASE DISCONNECT INSTANCES
ALL NODELAY;
```

Selectively Applying Redo for PDBs in Data Guard

It is now possible to selectively enable redo apply for pluggable databases using the ENABLED_PDBS_ON_STANDBY initialization parameter on the physical standby database. In this example, the primary ORCLCDB and the standby ORCLDR will be used to show how this new parameter works. This parameter is valid only on a standby database and is ignored on primary databases. A new PDB TESTPDB has been created to showcase this functionality. The parameter is used to specify which PDBs are eligible to have their redo applied on a standby database. If the parameter is not set, the default of * denoting that all PDBs should have their redo applied is used. The parameter essentially acts a whitelist of PDBs that the managed recovery process will apply redo for in the standby database.

```
SQL> show parameter ENABLED_PDBS_ON_STANDBY

NAME                                 TYPE        VALUE
------------------------------------ ----------- ----------------------
enabled_PDBs_on_standby              string      *

SQL> show spparameter ENABLED_PDBS_ON_STANDBY

SID      NAME                          TYPE        VALUE
-------- ----------------------------- ----------- --------------------
*        enabled_PDBs_on_standby       string
```

When the new database PDB called TESTPDB was created on the primary, it was automatically created on the standby database because the ENABLED_PDBS_ON_STANDBY parameter had a default value of *.

```
SQL> col name format a30
SQL> set lines 200
SQL> select name, recovery_status from v$pdbs;

NAME                           RECOVERY
------------------------------ --------
PDB$SEED                       ENABLED
SOMEPDB                        ENABLED
TESTPDB                        ENABLED
```

Effects of the ENABLED_PDBS_ON_STANDBY Parameter

The parameter can be modified on a running instance, but it must have the same value on all RAC instances. Furthermore, the parameter is evaluated only at PDB creation time; therefore, it cannot be used to disable recovery for an existing PDB, as shown next. The RECOVERY_STATUS column for the TESTPDB does not change despite the PDB not being in the list specified by ENABLED_PDBS_ON_STANDBY.

```
SQL> alter system set ENABLED_PDBS_ON_STANDBY='SOMEPDB' scope=BOTH sid='*';

System altered.

SQL> col name format a30
SQL> set lines 200
SQL> select name, recovery_status from v$pdbs;
```

```
NAME                           RECOVERY
------------------------------ --------
PDB$SEED                       ENABLED
SOMEPDB                        ENABLED
TESTPDB                        ENABLED
```

Creating a NEWPDB Pluggable Database on the Primary

Now observe what happens when a PDB is created on the primary database after this ENABLED_PDBS_ON_STANDBY value has been changed on the standby.

```
SQL> select db_unique_name from v$database;

DB_UNIQUE_NAME
------------------------------
orclcdb

SQL> CREATE PLUGGABLE DATABASE NEWPDB ADMIN USER pdbadmin IDENTIFIED BY
"database_123" ROLES=(CONNECT)  file_name_convert=NONE  STORAGE ( MAXSIZE
UNLIMITED MAX_SHARED_TEMP_SIZE UNLIMITED);

Pluggable database created.

SQL> alter pluggable database NEWPDB open read write instances=ALL;

Pluggable database altered.
```

Alert Log Entries on Standby Database

The PDB is added to the controlfile of the standby database, but its datafiles are not created.

```
Recovery created pluggable database NEWPDB
NEWPDB(5):File #23 added to control file as 'UNNAMED00023'. Originally
created as:
NEWPDB(5):'+DATA/ORCLCDB/745FE299A29B5954E053023AA8C0C85F/DATAFILE/
system.460.985202561'
NEWPDB(5):because the pluggable database was created with nostandby
NEWPDB(5):or the tablespace belonging to the pluggable database is
NEWPDB(5):offline.
NEWPDB(5):File #24 added to control file as 'UNNAMED00024'. Originally
created as:
NEWPDB(5):'+DATA/ORCLCDB/745FE299A29B5954E053023AA8C0C85F/DATAFILE/
sysaux.461.985202561'
NEWPDB(5):because the pluggable database was created with nostandby
NEWPDB(5):or the tablespace belonging to the pluggable database is
NEWPDB(5):offline.
NEWPDB(5):File #25 added to control file as 'UNNAMED00025'. Originally
created as:
NEWPDB(5):'+DATA/ORCLCDB/745FE299A29B5954E053023AA8C0C85F/DATAFILE/
undotbs1.459.985202561'
NEWPDB(5):because the pluggable database was created with nostandby
NEWPDB(5):or the tablespace belonging to the pluggable database is
NEWPDB(5):offline.
(5):File #26 added to control file as 'UNNAMED00026'. Originally created as:
(5):'+DATA/ORCLCDB/745FE299A29B5954E053023AA8C0C85F/DATAFILE/
undo_2.463.985202649'
(5):because the pluggable database was created with nostandby
(5):or the tablespace belonging to the pluggable database is
(5):offline.
```

Recovery Status of the NEWPDB Pluggable Database

The NEWPDB database is not configured to be recovered, and no redo will be applied for this pluggable database.

```
SQL> col name format a30
SQL> set lines 200
SQL> select name, recovery_status from v$pdbs;

NAME                           RECOVERY
------------------------------ --------
PDB$SEED                       ENABLED
SOMEPDB                        ENABLED
TESTPDB                        ENABLED
NEWPDB                         DISABLED
```

Disabling Redo Apply for an Existing PDB

If it is desired to disable redo apply for an already existing PDB, this can be accomplished through an `ALTER PLUGGABLE DATABASE` command. In this example, the PDB `TESTPDB` will be altered. This can be done only after managed recovery has been stopped; otherwise, an ORA-01156 error will be thrown. Additionally, the PDB needs to be closed on all instances, or an ORA-65025 error will be thrown.

```
SQL> alter session set container=TESTPDB;

Session altered.

SQL> alter pluggable database TESTPDB close immediate instances=ALL;

Pluggable database altered.

SQL> alter pluggable database TESTPDB disable recovery;

Pluggable database altered.

SQL> select name, recovery_status from v$pdbs;

NAME                           RECOVERY
------------------------------ --------
TESTPDB                        DISABLED
```

Data Guard Database Compare

In 12c R2, it is now possible to detect lost writes and `NONLOGGED` blocks without using the `DB_LOST_WRITE_PROTECT` parameter. This is possible with the `DBMS_DBCOMP` package. Currently, `DBMS_DBCOMP` has only one procedure, called `DBCOMP`, which can be used to compare datafiles between a primary database and a standby database. It can be called from the standby or the primary database and works only for physical standby databases because it does a block for block comparison between the primary and standby databases.

```
SQL> desc DBMS_DBCOMP
PROCEDURE DBCOMP
 Argument Name                  Type                    In/Out Default?
 ------------------------------ ----------------------- ------ --------
 DATAFILE                       VARCHAR2                IN
 OUTPUTFILE                     VARCHAR2                IN
 BLOCK_DUMP                     BOOLEAN                 IN     DEFAULT
```

The output file can be specified either with an absolute path or with a relative filename. By default, the output is placed in `$ORACLE_HOME/dbs`. The default value for `BLOCK_DUMP` is false, so it is recommended to explicitly set the argument to `TRUE` so that information about blocks that differ between the primary database and the standby database are dumped.

Using DBMS_DBCOMP to Detect Nologging Operations

To borrow an example from a previous section in this chapter, a `NOLOGGING` table will be created and then detected via `DBMS_DBCOMP`.

```
SQL> alter session set container=SOMEPDB;

Session altered.

SQL> create table HR.EMPLOYEES_NOLOG NOLOGGING TABLESPACE USERS as select *
from hr.employees where 'x'='y';

Table created.

SQL> insert /*+ APPEND */ into HR.EMPLOYEES_NOLOG select * from hr.employees;

107 rows created.

SQL> commit;

Commit complete.
```

The comparison details will be dumped into a file called `/home/oracle/dbcomp.txt`.

```
SQL> exec DBMS_DBCOMP.DBCOMP(datafile =>'ALL', outputfile => '/home/oracle/
dbcomp.txt',block_dump =>TRUE);

PL/SQL procedure successfully completed.
```

Contents of the dbcomp.txt File

Because of the NOLOGGING operation that was performed, there are two corrupted blocks. In a healthy database, corrupted database blocks should not be present. The number of corrupted database blocks found during a comparison operation are written into the third CORR column. In this example, there are two corrupted database blocks where ID=6 (transactional data):

```
[oracle@stpldb101 ~]$ cat /home/oracle/dbcomp.txt
Client is connected to database: orclcdb. Role: primary database.
Remote database orcldr.remote db role: physical standby

Slave Id  0
Summary:
****************************************************************************
                       TOTAL: total no. of blocks found
                         |
   +--------+------------+-------+---------+---------+
   |        |                    |         |         |
   |      DIFFV:             LOST_WRITE    |       CORR: corrupted blocks
 SAMEV     diff ver              |       SKIPPED:
   |       block pairs        +--+--+      direct load, empty blocks,
+--+--+--+                    |     |      RMAN optimized blocks,
|  |  |  |                    |     |      flashback optimized blocks
|  |  | SAMEV&C:              |     |
|  |  |  same ver &           |   LWLOC: lost writes at local db
|  |  |  same checksum &     LWRMT: lost writes at remote db
|  |  |  same contents
|  |  |
|  | SAMEV_NO_CHKSUM: same ver & same contents but diff checksum
|  |                 (checksum can be diff but identical contents)
|  |
| DIFFPAIR: same ver but differrent contents (data inconsistency)
|
ENCERR: undecided block pairs due to encryption related issue
        (e.g. when Wallet is not open)
```

```
ID TOTAL   CORR SKIPPED DIFFV   SAMEV   SAMEV&C ENCERR  LWLOC  LWRMT DIFFPAIR
00 0017131 0000 0017131 0000000 0000000 0000000 0000000 000000 000000 0000000
02 0000118 0000 0000000 0000011 0000107 0000107 0000000 000000 000000 0000000
06 0066357 0002 0036590 0000079 0029686 0029686 0000000 000000 000000 0000000
14 0000001 0000 0000000 0000000 0000001 0000001 0000000 000000 000000 0000000
16 0000764 0000 0000000 0000000 0000764 0000764 0000000 000000 000000 0000000
23 0000061 0000 0000000 0000000 0000061 0000061 0000000 000000 000000 0000000
25 0000061 0000 0000000 0000000 0000061 0000061 0000000 000000 000000 0000000
26 0000009 0000 0000000 0000000 0000009 0000009 0000000 000000 000000 0000000
27 0000798 0000 0000000 0000000 0000798 0000798 0000000 000000 000000 0000000
29 0000005 0000 0000000 0000001 0000004 0000004 0000000 000000 000000 0000000
30 0000625 0000 0000000 0000001 0000624 0000624 0000000 000000 000000 0000000
32 0001296 0000 0000000 0000004 0001292 0001292 0000000 000000 000000 0000000
33 0001158 0000 0000000 0000001 0001157 0001157 0000000 000000 000000 0000000
35 0001158 0000 0000000 0000000 0001158 0001158 0000000 000000 000000 0000000
38 0000020 0000 0000000 0000011 0000009 0000009 0000000 000000 000000 0000000
58 0026594 0000 0026591 0000003 0000000 0000000 0000000 000000 000000 0000000
61 0003078 0000 0000000 0000000 0003078 0003078 0000000 000000 000000 0000000
62 0000160 0000 0000000 0000000 0000160 0000160 0000000 000000 000000 0000000
63 0000341 0000 0000000 0000000 0000341 0000341 0000000 000000 000000 0000000
64 0000950 0000 0000000 0000000 0000950 0000950 0000000 000000 000000 0000000
69 0000521 0000 0000000 0000000 0000521 0000521 0000000 000000 000000 0000000
70 0005632 0000 0005627 0000005 0000000 0000000 0000000 000000 000000 0000000
80 0000128 0000 0000000 0000001 0000127 0000127 0000000 000000 000000 0000000
81 0000001 0000 0000000 0000000 0000001 0000001 0000000 000000 000000 0000000
82 0000128 0000 0000000 0000000 0000128 0000128 0000000 000000 000000 0000000
83 0000128 0000 0000000 0000000 0000128 0000128 0000000 000000 000000 0000000
84 0000127 0000 0000000 0000000 0000127 0000127 0000000 000000 000000 0000000

Short description for each block type:
***************************************************************************
02: KTU UNDO BLOCK
06: trans data
14: KTU UNDO HEADER W/UNLIMITED EXTENTS
16: DATA SEGMENT HEADER - UNLIMITED
23: BITMAPPED DATA SEGMENT HEADER
```

```
25: BITMAP INDEX BLOCK
26: BITMAP BLOCK
27: LOB BLOCK
29: KTFB Bitmapped File Space Header
30: KTFB Bitmapped File Space Bitmap
32: FIRST LEVEL BITMAP BLOCK
33: SECOND LEVEL BITMAP BLOCK
35: PAGETABLE SEGMENT HEADER
38: KTU SMU HEADER BLOCK
58: block type 58
61: NGLOB: Hash Bucket
62: NGLOB: Committed Free Space
63: NGLOB: Segment Header
64: NGLOB: Persistent Undo
69: NGLOB: Lob Extent Header
70: block type 70
80: KTFBN File Space Property Map
81: Stats Segment Header
82: Stats Map
83: Stats Map Summary
84: Stats bitmap
```

Password File Change Synchronization

Oracle Data Guard has become easier to manage in 12c R2 because of a change that stores information in the redo logs regarding the Oracle password file changes. This makes things much easier to manage because SYS passwords can be changed on a primary database without having take explicit steps to modify the standby database password files. In previous releases, the Oracle Data Guard Transport service would break down if the primary database's SYS password in the password file was changed to a different value than what the standby database's password file had. This new feature will be shown with a simple check of checksums before and after changing SYS passwords.

Checksums Prior to the SYS Password Change

In the following example, it is shown that both the primary and standby password files have a checksum of 2468345953.

```
[oracle@stpldb101 old_backup]$ srvctl config database -d orclcdb |grep -i
password
Password file: +DATA/orclcdb/pwdorclcdb.ora

[oracle@stpldb101 ~]$ asmcmd cp +DATA/orclcdb/pwdorclcdb.ora /home/oracle/
prim_pw_before_change.ora

[oracle@stpldb101 ~]$ cksum /home/oracle/prim_pw_before_change.ora
2468345953 3584 /home/oracle/prim_pw_before_change.ora

[oracle@rac501 ~]$ srvctl config database -d orcldr |grep -i password
Password file: +DATA/DB_UNKNOWN/PASSWORD/pwddb_unknown.277.985101591

[oracle@rac501 ~]$ asmcmd cp +DATA/DB_UNKNOWN/PASSWORD/pwddb_
unknown.277.985101591 /home/oracle/stby_before_change.ora

[oracle@rac501 ~]$ cksum /home/oracle/stby_before_change.ora
2468345953 3584 /home/oracle/stby_before_change.ora
```

Changing the Password

The password can simply be changed with an ALTER USER command on the primary database.

```
SQL> alter user sys identified by "somenew_password";

User altered.
```

Checksums After to the SYS Password Change

The checksum has changed from 2468345953 to 2291626709 on both the primary and the standby password files without any manual intervention on the standby database. In previous releases of the Oracle Database, this would have required stopping transport services and manually changing the password file values, which can leave databases vulnerable to data loss if an unfortunate catastrophic failure renders the primary database inoperable before it can send archived log files to the standby database.

```
[oracle@stpldb101 ~]$ asmcmd cp +DATA/orclcdb/pwdorclcdb.ora /home/oracle/
prim_pw_after_change.ora

[oracle@stpldb101 ~]$ cksum /home/oracle/prim_pw_after_change.ora
2291626709 3584 /home/oracle/prim_pw_after_change.ora

[oracle@rac501 ~]$ asmcmd cp +DATA/DB_UNKNOWN/PASSWORD/pwddb_
unknown.277.985101591 /home/oracle/stby_after_change.ora

[oracle@rac501 ~]$ cksum /home/oracle/stby_after_change.ora
2291626709 3584 /home/oracle/stby_after_change.ora
```

In-Memory Columnar Store for ADG Instances

Active Data Guard databases can now leverage the powerful in-memory columnar store to improve the performance of reporting queries and other read-only workloads that are being offloaded to Active Data Guard databases.

Enabling this feature is controlled by a new parameter called INMEMORY_ADG_ENABLED that has a default value of TRUE. With this parameter, an INMEMORY_SIZE value can be specified for an ADG instance to enable the use of the in-memory columnar store. However, this feature comes with a few limitations in this version. ADG INMEMORY is not compatible with multi-instance redo apply, and all INMEMORY expressions are evaluated from the primary database. Additionally, INMEMORY join groups and FSFO are not supported as of version 12.2.0.1.

```
SQL> show parameter inmemory_adg_enabled

NAME                                 TYPE        VALUE
------------------------------------ ----------- ----------------------
inmemory_adg_enabled                 boolean     TRUE

SQL> show spparameter inmemory_adg_enabled

SID      NAME                          TYPE       VALUE
-------- ----------------------------- ---------- ---------------------
*        inmemory_adg_enabled          boolean
```

Role Transition Connection Preservation

In 12c R2, it is now possible to preserve connections in a standby database as it undergoes a role change. This can be particularly useful in applications that maintain two connections pools, one to the primary database and another to a designated standby database. In this way, a role transition from standby to primary does not require connections to be recycled and re-established, effectively allowing a zero-downtime transition from an application's perspective.

This new parameter is controlled through the STANDBY_DB_PRESERVE_STATES parameter, which has a default value of NONE and can be set to either NONE, ALL, or SESSION to control which types of sessions are preserved during a role transition. Changing the parameter requires a bounce of the standby database.

```
SQL> show parameter STANDBY_DB_PRESERVE_STATES

NAME                                 TYPE        VALUE
------------------------------------ ----------- ----------------------
standby_db_preserve_states           string      NONE

SQL> show spparameter STANDBY_DB_PRESERVE_STATES

SID      NAME                          TYPE        VALUE
-------- ----------------------------- ----------- --------------------
*        standby_db_preserve_states    string
```

In this example, the parameter will be set to a value of ALL, and a test will be done via SQLPlus. A connection will be made to the PDB SOMEPDB on the ORCLDR CDB ADG database while it is in the database role of PHYSICAL STANDBY. After the connection is made, a role transition will be initiated from a separate session from the Broker.

Establishing the Connection

No special connection-specific details have to be set for this to work; it even works for basic EZCONNECT sessions.

```
[oracle@stpldb101 ~]$ sqlplus hr/test_123@//rac5-scan:1521/somepdb.world

SQL*Plus: Release 12.2.0.1.0 Production
```

```
Copyright (c) 1982, 2016, Oracle.  All rights reserved.

Connected to:
Oracle Database 12c Enterprise Edition Release 12.2.0.1.0 - 64bit Production

SQL> select db_unique_name, database_role from v$database;

DB_UNIQUE_NAME                 DATABASE_ROLE
------------------------------ ----------------
orcldr                         PHYSICAL STANDBY

SQL> show con_name

CON_NAME
------------------------------
SOMEPDB
```

Performing the Role Transition

Here is the command:

```
DGMGRL> switchover to orcldr;
Performing switchover NOW, please wait...
Operation requires a connection to database "orcldr"
Connecting ...
Connected to "orcldr"
Connected as SYSDBA.
New primary database "orcldr" is opening...
Oracle Clusterware is restarting database "orclcdb" ...
Connected to "orclcdb"
Connected to "orclcdb"
Switchover succeeded, new primary is "orcldr"
```

Checking the Existing Session

Finally, here's the check:

```
SQL> select db_unique_name, database_role from v$database;
```

```
DB_UNIQUE_NAME                 DATABASE_ROLE
------------------------------ ----------------
orcldr                         PRIMARY

SQL> show con_name

CON_NAME
------------------------------
SOMEPDB
```

PDB Migrate

It is possible to migrate pluggable databases from one container to another using DGMGRL. This functionality makes it easy to migrate pluggable databases using just one command. A pluggable database can be migrated from its primary container CDB to another read-write CDB, generally on the same primary host, or it can be migrated from its standby CDB to another read-write CDB, generally on the same standby host. The latter scenario is known as a *PDB failover* and can be used to migrate one PDB out of a container in the case that the primary PDB has failed, but performing a failover operation at the CDB level is undesirable because of the downtime requirements of the rest of the tenants in the primary CDB. The chief restriction that exists with PDB migrations is that the target CDB needs to have direct access to the XML manifest file that the source CDB creates during the migration process, either via NFS or through a shared local filesystem if both CDBs are on the same host. Additionally, the PDBs datafiles need to be directly accessible to both the source CDB and the target CDB. Lastly, both the source and the target CDBs of the migration need to be part of their own respective Broker configurations, even if both Broker configurations consist of just primary databases with no standby databases.

In this section, the PDB SOMEPDB will be migrated from its primary CDB, ORCLCDB, to an alternate CDB called ALTCDB, which has been created on the same cluster as ORCLCDB. This type of operation is called a *PDB migration* as opposed to a PDB failover, which is initiated from a standby CDB.

ALTCDB Broker Configuration

The Data Guard command-line interface (DGMGRL) enables you to manage a Data Guard Broker configuration and its databases directly from the command line.

```
DGMGRL> show configuration;

Configuration - altcdb

  Protection Mode: MaxPerformance
  Members:
  altcdb - Primary database

Fast-Start Failover: DISABLED

Configuration Status:
SUCCESS   (status updated 14 seconds ago)

SQL> select db_unique_name from v$database;


DB_UNIQUE_NAME
------------------------------
altcdb

SQL> show pdbs;


    CON_ID CON_NAME                       OPEN MODE  RESTRICTED
---------- ------------------------------ ---------- ----------
         2 PDB$SEED                       READ ONLY  NO
```

Broker Configuration

Check the Broker configuration for the orclcdb database using DGMGRL.

```
DGMGRL> show configuration;

Configuration - orclcdb

  Protection Mode: MaxPerformance
  Members:
  orclcdb - Primary database
```

```
    orcldr  - Physical standby database

Fast-Start Failover: DISABLED

Configuration Status:
SUCCESS   (status updated 50 seconds ago)

SQL> select db_unique_name from v$database;

DB_UNIQUE_NAME
------------------------------
orclcdb

SQL> show pdbs;

    CON_ID CON_NAME                       OPEN MODE  RESTRICTED
---------- ------------------------------ ---------- ----------
         2 PDB$SEED                       READ ONLY  NO
         3 SOMEPDB                        READ WRITE NO
```

The Migration

The pluggable database can be migrated by using the MIGRATE PLUGGABLE DATABASE command while connected to the source CDB through the Broker. The operation is extremely fast and in most cases will complete in about one minute of runtime regardless of database size. This is because the pluggable database is migrated into the target CDB by internally using the NOCOPY clause of the CREATE PLUGGABLE DATABASE command.

```
DGMGRL> connect sys/database_123@orclcdb
Connected to "orclcdb"
Connected as SYSDBA.
DGMGRL> migrate pluggable database somepdb to container altcdb using
'/home/oracle/somepdb.xml' connect as sys/database_123@altcdb
Connected to "altcdb"
Connected as SYSDBA.
```

```
Beginning migration of pluggable database SOMEPDB.
Source multitenant container database is orclcdb.
Destination multitenant container database is altcdb.

Closing pluggable database SOMEPDB on all instances of multitenant
container database orclcdb.
Unplugging pluggable database SOMEPDB from multitenant container database
orclcdb.
Pluggable database description will be written to /home/oracle/somepdb.xml.
Dropping pluggable database SOMEPDB from multitenant container database orclcdb.
Creating pluggable database SOMEPDB on multitenant container database altcdb.
Opening pluggable database SOMEPDB on all instances of multitenant
container database altcdb.
Migration of pluggable database SOMEPDB completed.
Succeeded.
```

Source Database Alert Log Entry

Internally, the pluggable database is unplugged into an XML manifest file, and the datafiles are kept in place.

```
Completed: alter pluggable database SOMEPDB unplug into '/home/oracle/
somepdb.xml'
drop pluggable database SOMEPDB keep datafiles
```

Target Database Alert Log Entry

On the target database, a pluggable database is plugged in with the NOCOPY clause, which is why this operation is performed so quickly.

```
Completed: create pluggable database SOMEPDB using '/home/oracle/somepdb.xml'
nocopy standbys=none tempfile reuse
alter pluggable database SOMEPDB open instances=all
```

State of ALTCDB Post-Migration

At the end of the operation, the pluggable database SOMEPDB will be in the ALTCDB container database with its datafiles unmoved. For this reason, it is recommended that the source and destination container databases of a PDB migrate operation share the same ASM storage or the same faster NFS storage.

```
SQL> select db_unique_name from v$database;

DB_UNIQUE_NAME
------------------------------
Altcdb

SQL> show pdbs;

    CON_ID CON_NAME                       OPEN MODE  RESTRICTED
---------- ------------------------------ ---------- ----------
         2 PDB$SEED                       READ ONLY  NO
         3 SOMEPDB                        READ WRITE NO

SQL> select con_id,name from v$datafile order by con_id;

    CON_ID NAME
---------- ----------------------------------------------------------------
         1 +DATA/ALTCDB/DATAFILE/undotbs2.524.985217387
         1 +DATA/ALTCDB/DATAFILE/users.512.985217191
         1 +DATA/ALTCDB/DATAFILE/undotbs1.511.985217189
         1 +DATA/ALTCDB/DATAFILE/system.509.985217119
         1 +DATA/ALTCDB/DATAFILE/sysaux.510.985217165
         2 +DATA/ALTCDB/4700A987085B3DFAE05387E5E50A8C7B/DATAFILE/
           sysaux.520.985217263
         2 +DATA/ALTCDB/4700A987085B3DFAE05387E5E50A8C7B/DATAFILE/
           system.521.985217263
         2 +DATA/ALTCDB/4700A987085B3DFAE05387E5E50A8C7B/DATAFILE/
           undotbs1.522.985217263
```

```
3 +DATA/ORCLCDB/744827E32DAF16D5E053033AA8C089C7/DATAFILE/
  system.289.985198351
3 +DATA/ORCLCDB/744827E32DAF16D5E053033AA8C089C7/DATAFILE/
  sysaux.392.985198355
3 +DATA/ORCLCDB/744827E32DAF16D5E053033AA8C089C7/DATAFILE/
  undotbs1.393.985198359
3 +DATA/ORCLCDB/744827E32DAF16D5E053033AA8C089C7/DATAFILE/
  undo_2.377.985198359
3 +DATA/ORCLCDB/744827E32DAF16D5E053033AA8C089C7/DATAFILE/
  users.390.985198361
```

Summary

With each new release, Oracle is enhancing Oracle Data Guard and increasingly making it a feature that serves not only as a good disaster recovery solution but also as a feature that can be used to offload read-only workloads. After reading this chapter, you should have a good understanding of many of the new features of Data Guard in 12c R2 that help make maintaining and tuning ADG databases much easier.