ORACLE UNIVERSITY

Hardware and Software
Engineered to Work Together

Oracle Database 12*c*: RAC and Grid Infra Deployment Workshop

Student Guide
D87007GC10
Edition 1.0 | November 2014 | D89032

Learn more from Oracle University at **oracle.com/education/**

ORACLE

Authors

Peter Fusek
Jim Womack

Technical Contributors and Reviewers

Allan Graves
Andrey Gusev
Anil Nair
Branislav Valny
Dominique Jeunot
Donna Keesling
Douglas Williams
Harald van Breederode
Harendra Mishra
Harish Nandyala
Janet Stern
Jean-Francois Verrier
Jerry Lee
Jim Williams
Joel Goodman
John McHugh
Jonathan Creighton
Larry Carpenter
Mark Scardina
Markus Michalewicz
Prasad Bagal
Raj Kammend
Rick Wessman
Sean Kim
Soma Prasad
Subhransu Basu
Srinagesh Battula

Editors

Aju Kumar
Anwesha Ray
Malavika Jinka

Graphic Designer

Rajiv Chandrabhanu

Publishers

Veena Narasimhan
Jayanthy Keshavamurthy
Syed Ali

Contents

1

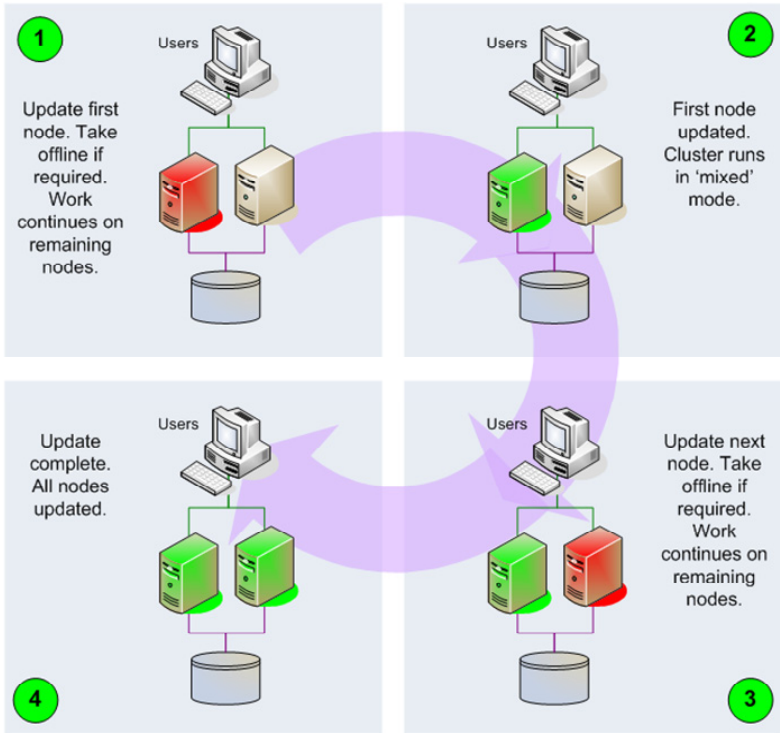
Rolling Database Upgrade Using Transient Logical Standby

Objectives

After completing this lesson, you should be able to:

- Describe the procedure for database rolling upgrade using transient logical standby
- Describe the key differences between database rolling upgrade using transient logical standby and alternative upgrade methods
- Perform a database rolling upgrade using transient logical standby

Rolling Upgrade: Introduction



The diagram in the slide illustrates the general concept of rolling upgrade. The diagram shows an example using a two-node (server) cluster; however, the concept also applies to larger clusters.

Rolling upgrade is not an Oracle Database–specific concept. Rather, the idea of rolling upgrades has been associated with high availability and multi-server computing architectures for many years.

The basic idea is that servers are upgraded one at a time, and while each node is being upgraded, the rest of the cluster remains available to process the workload.

Ideally, a rolling upgrade eliminates system down time. However, sometimes a rolling upgrade requires a global restart of cluster software or hardware components. In such cases, the purpose of rolling upgrade is to minimize system down time.

Although rolling upgrades can eliminate (or at least minimize) system down time, it is important to note that they may not be the perfect solution for every situation. The procedure for rolling upgrades can be more complex and time-consuming than a nonrolling upgrade. Also, while a rolling upgrade is occurring, the overall system capacity is diminished.

Rolling Upgrade and Oracle RAC

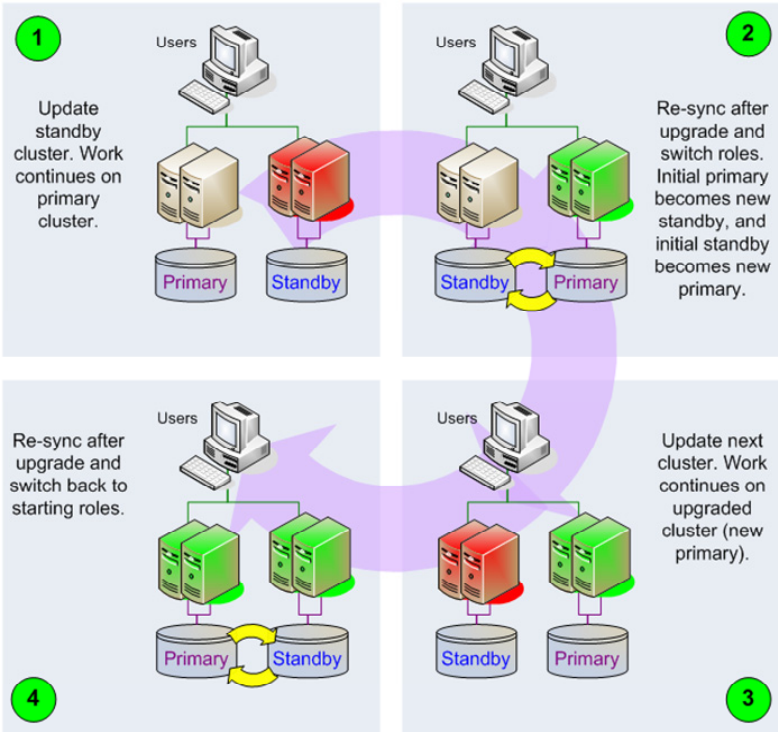
- Oracle supports rolling upgrade for Grid Infrastructure:
  - Includes Clusterware and ASM
  - Is integrated into Oracle Universal Installer
  - Is automated, and is quick and easy to perform
- Rolling upgrade is not supported for Oracle RAC in a single cluster.
  - Most patches can be applied in a rolling manner.
  - New release upgrades cannot be rolling upgrades.
- Rolling upgrade can be achieved by using two separate clusters:
  - One cluster processes work while the other is being upgraded.

Oracle supports rolling upgrades for Oracle Grid Infrastructure, which includes Oracle Clusterware and Oracle Automatic Storage Management (ASM). The procedure for rolling upgrades of Oracle Grid Infrastructure is built into Oracle Universal Installer, which makes it both quick and easy to perform. Regardless of how you perform your database upgrade, it is generally recommended to upgrade Oracle Grid Infrastructure by using the rolling upgrade procedure.

Currently, you can apply most database patches (one-off patches), and all database patch set updates (PSUs) and critical patch updates (CPUs) by using a rolling patch installation procedure in an Oracle Real Application Clusters (RAC) environment. However, you cannot upgrade to a new major release, maintenance release, or patch release of Oracle Database by using a rolling upgrade process within a single cluster. In other words, most changes to an existing Oracle Database installation can be rolling in nature, whereas updates that create a new Oracle Home directory cannot be rolling in nature.

Although rolling database upgrade is currently not possible within a single cluster, it is possible to perform a rolling upgrade by using two clusters. Using this strategy, one cluster maintains system availability while the other cluster is upgraded.

Rolling Database Upgrade with Logical Standby Database



The diagram in the slide illustrates the concept of rolling upgrade using an Oracle Data Guard logical standby database with SQL Apply. The diagram shows an example with two 2-node clusters; however, the concept applies to single-instance databases and Oracle RAC databases with any number of instances.

With this configuration, the rolling database upgrade procedure is as follows:

1. Work continues on the primary database while the standby database is upgraded using the normal (nonrolling) upgrade procedure.
2. After the standby database is upgraded, it is started and synchronized with the primary database. Then the primary and standby databases switch roles and the workload moves over to the upgraded database, which is now the new primary database.
3. Next, the new standby (original primary) database is upgraded using the normal upgrade procedure.
4. Finally, the newly updated standby database is started and re-synchronized with the primary database. After synchronization, the databases can optionally be switched back to their original roles.

Throughout this process, the primary database is available at all times except for the brief periods when a role switch is happening.

Optionally, you can use an archive log repository or a bystander database so that redo is still saved away during each database upgrade. This provides additional protection in cases where the primary database suffers a failure while the standby database is being upgraded.

Rolling Database Upgrade with Transient Logical Standby Database

- Starting point:
  - Oracle Data Guard physical standby configuration
  - Flashback Database enabled
  - Data Guard Broker disabled
- Process:
  1. Create a guaranteed restore point.
  2. Convert the physical standby to a transient logical standby.
  3. Upgrade the transient logical standby database.
  4. Resynchronize the upgraded standby database.
  5. Switch over the database roles.
  6. Flash back the logical standby (original primary) to the guaranteed restore point created at step 1.
  7. Remount the logical standby using upgraded Oracle binaries.
  8. Convert the logical standby back to a physical standby.
  9. Resynchronize the standby database, which automatically completes the upgrade process.
  10. Switch back to the original database configuration.

Users with existing Oracle Data Guard physical standby database configurations can perform rolling database upgrade by using a transient logical standby database. Apart from Oracle Data Guard physical standby, the procedure also uses the Flashback Database feature. An outline of the procedure is listed in the slide.

Performing a rolling upgrade by using a transient logical standby is similar to a rolling upgrade by using a logical standby with SQL Apply (outlined previously), but with the following differences:

- A guaranteed restore point is created on the primary database for the purpose of flashing it back to become a physical standby after the switchover.
- The conversion of the physical standby to a logical standby uses the `KEEP IDENTITY` clause to retain the same `DB_NAME` and `DBID` as the primary database. This is the characteristic that differentiates a transient logical standby from a normal logical standby.
- The `ALTER DATABASE CONVERT TO PHYSICAL STANDBY` statement is used to convert the original primary from a logical standby to a physical standby.
- The original primary is actually upgraded by means of Redo Apply after it is converted from a logical standby to a physical standby.

`physru.sh` Script

- Usage:

```
$ physru.sh <username> <primary_tns> <standby_tns> <primary_name>
   <standby_name> <upgrade_version>
```

- The `physru.sh` script:
  - Simplifies and automates rolling database upgrade with transient logical standby database
  - Contains many automatic checks to minimize errors
  - Provides guidance for additional manual tasks
  - Is available through My Oracle Support bulletin 949322.1

ORACLE

The process for rolling database upgrade with transient logical standby database requires many operations and role changes to be performed in a precise order. To simplify the process and reduce the likelihood of administrator error, the `physru.sh` script is available through the My Oracle Support bulletin 949322.1. This script automates much of the process and provides guidance regarding the required manual steps. The script also performs many checks along the way to maximize the probability of success.

Rolling Database Upgrade with Transient Logical Standby Database: Benefits and Challenges

- Benefits:
  - Minimal business disruption
    - The required down time ranges from seconds to a few minutes depending on the environment.
    - The upgrade process can be paused to enable down time in a scheduled period.
    - Apart from the required down time, the full processing power of the primary system is always available.
  - In-place physical database upgrade
    - There is no need to upload and reload data.
    - Existing security controls are maintained.
  - Opportunity to test the upgraded standby before upgrading the primary
- Challenges:
  - Process requires an Oracle Data Guard standby database.
  - Process is long and complex.
    - Mitigated by using `physru.sh`
  - Unsupported data types
    - Various options exist.

ORACLE

The slide lists the various benefits and challenges associated with rolling database upgrade by using transient logical standby database.

Benefits

- **Minimal business disruption:** Rolling database upgrade with transient logical standby database requires much less system down time compared with the normal (nonrolling) database upgrade procedure. Depending on the environment, down time can range from seconds to a few minutes. Because each database is upgraded while it is in the standby role, the full processing power of the primary system remains available for business processing. Also, the process can be executed over an extended time period, which might allow you to fit the required down time into existing maintenance periods.
- **In-place physical database upgrade:** Because the upgrade is in-place, there is no need to unload and reload your data, which can be particularly useful for large databases. Also, because the data remains inside the database, your existing security controls are maintained, which is particularly useful for sensitive databases.
- **Opportunity to test the upgraded standby before upgrading the primary:** After the standby database has been upgraded, it can be used to evaluate performance and stability of the new release before there is any impact to production.

Challenges

- **Process requires an Oracle Data Guard standby database:** This might not be an issue for existing customers of Oracle Data Guard. However, customers that do not already use Oracle Data Guard might not be able to justify the investment in a standby environment solely for the purpose of performing an upgrade.
- **Process is quite long and complex:** The process requires numerous tasks to be performed correctly and in a precise order. This opens up the possibility for administrator error, which could lead to unforeseen costs and delays. These risks can be mitigated by thorough planning and testing, and also by using `physru.sh` to simplify and automate the process.
- **Unsupported data types:** When setting up a logical standby database, you must ensure that the logical standby database can maintain the data types and tables in your primary database. Limits and restrictions associated with Oracle Data Guard logical standby database may result in some data not being replicated on the standby database. You should review Appendix C titled “Data Type and DDL Support on a Logical Standby Database” in the *Oracle Data Guard Concepts and Administration* guide for complete details.

  Even if unsupported data types are identified, there are cases when a transient logical standby procedure can still be used. The determination has to be made if there is a satisfactory way to handle the unsupported data types. Options for using rolling upgrade when unsupported data types exist include the following:
  - Temporarily suspend or prohibit changes to the unsupported tables for the period of time it takes to perform the upgrade procedure.
  - If you cannot prevent changes to unsupported tables during the upgrade, any unsupported transactions that occur are recorded in the `DBA_LOGSTDBY_EVENTS` table on the logical standby database. After the upgrade is completed, you could use Oracle Data Pump or the Export/Import utility to copy the changed tables.
  - Use Extended Datatype Support (EDS), which enables SQL Apply to replicate changes to tables that contain some data types that are not natively supported. See the chapter titled “Managing a Logical Standby Database” in the *Oracle Data Guard Concepts and Administration* guide for complete details.

Other Upgrade Options

	Benefits	Challenges
Nonrolling upgrade using DBUA or scripts	• Commonly used • Universally applicable	• Requires substantial down time
Data migration using Data Pump Export and Import	• Commonly used • Universally applicable	• May require substantial down time
Data migration using transportable tablespaces	• Conceptually simple • No need to unpack/repack data	• May need to use other methods to copy associated metadata and PL/SQL objects
Data migration using replication technologies	• Little impact on the original system while the replica is built • Opportunity to test the stability and performance of the replica before switchover	• May be costly and complex to implement • May not be practical for large databases
Data migration using pluggable databases	• Quick and easy to unplug/replug a database	• Source database must already be a PDB (> version 12.1)
Rolling upgrade using `DBMS_ROLLING`	• Highly automated process • Minimal down time	• Requires Data Guard standby • Source database must be > version 12.1.0.2

ORACLE

The table in the slide outlines other upgrade options and lists some of their benefits and challenges. Note that the table does not discuss all the possible alternatives, nor does it provide comprehensive details regarding each alternative. Rather, the aim is to provide an overview of a range of alternatives along with enough information to allow for comparisons between them and with the rolling upgrade options introduced earlier in the lesson.

- Database Upgrade Assistant (DBUA) interactively steps you through the upgrade process and automatically configures the database for the new Oracle Database software release. When automatic configuration is not possible, DBUA makes appropriate recommendations for manual configuration tasks.
  A manual upgrade consists of running SQL scripts and utilities from a command line to upgrade a database. Although a manual upgrade gives you finer control over the upgrade process, it is more susceptible to error if any of the upgrade or preupgrade steps are either not followed or are performed out of order.
  Both of these processes are commonly used and are designed to be universally applicable. However, the main drawback with both approaches is that a substantial amount of down time is required. Depending on the environment, the required down time may range from many minutes to hours.

- Unlike DBUA or a manual command-line upgrade, the Oracle Data Pump Export and Import utilities migrate a copy of data from your current database to a new database in the new release. The new database must exist before the contents of the export dump files can be loaded.
  The export and import data migration method does not change the current database, which enables the database to remain available throughout the upgrade process. However, to maintain data consistency, the source database must be protected from changes until the upgraded database is ready for use. Often, that means that the source database is unavailable throughout the export and import operations.
  Depending on the environment and the amount of data being moved, the export and import processes could take a substantial amount of time.
- Using the transportable tablespace feature reduces database upgrade time by moving all user tablespaces from a database running an earlier software release to an empty destination database running the upgraded software release. With transportable tablespace, tablespace data files are plugged in to the database by copying the data files to the destination database, and then importing the object metadata into the destination database.
  This method is particularly well suited for databases that have simple schemas and where the data files do not need to be transferred as part of the transport process (such as when the data files will be used in place). Note that you will need to use other methods (such as Data Pump) to copy PL/SQL program objects and other definitions such as users, grants, and synonyms.
- You can use data replication technologies, such as Oracle GoldenGate, to migrate data from an existing database into an upgraded database. After the replica is up-to-date, you can move your workload to the upgraded database with very little down time. The main drawback to this approach is that you must maintain two copies of the database during the upgrade process, and you must configure and manage the replication process.
- The multitenant architecture enables an Oracle database to function as a multitenant container database (CDB) that includes zero, one, or many customer-created pluggable databases (PDBs). A PDB is a portable collection of schemas, schema objects, and nonschema objects that appears to an Oracle Net client as a fully self-contained database.
  Using the multitenant architecture, you can quickly and easily unplug a PDB and plug it into an upgraded CDB. However, to enable this capability, you must first migrate to the Oracle Multitenant option, and this process may require you to first upgrade to Oracle Database 12*c* by using some other means.
- Oracle Database 12*c* introduces a streamlined method for performing rolling upgrades. It is implemented using the new `DBMS_ROLLING` PL/SQL package, which allows you to upgrade the database software in an Oracle Data Guard configuration in a rolling fashion. `DBMS_ROLLING` simplifies and automates the rolling upgrade process. It also removes many of the restrictions associated with rolling upgrades using logical standby databases.

- You will be able to use this feature to perform database version upgrades starting with the first patch set of Oracle Database 12*c* (version 12.1.0.2); that is, you cannot use it to upgrade from any earlier version. This means that rolling database upgrade with transient logical standby database must still be used when upgrading from Oracle Database 11*g* to Oracle Database12*c*, or when upgrading from the initial Oracle Database 12*c* release to later releases.

For further information about any of these upgrade options, refer to the Oracle Database documentation library along with the white papers that are available at otn.oracle.com.

Quiz

What is the main benefit of performing a rolling upgrade using an Oracle Data Guard logical standby database?

a. Simplicity
b. Speed
c. Minimal down time
d. Automation

Answer: c

Quiz

Apart from a rolling upgrade, which of the following options can be used to upgrade from Oracle Database 11*g* release 2 to Oracle Database 12*g*?

a. Nonrolling upgrade using DBUA or scripts
b. Data migration using Data Pump Export and Import
c. Data migration using transportable tablespaces
d. Data migration using replication technologies
e. Data migration using pluggable databases
f. Rolling upgrade using `DBMS_ROLLING`

Answer: a, b, c, d

Data migration using pluggable databases requires the source database to be pluggable database, which implies that the source database must be version 12.1 or later.

Rolling upgrade using `DBMS_ROLLING` requires the source database to be version 12.1.0.2 or later.

Summary

In this lesson, you should have learned how to:

- Describe the procedure for database rolling upgrade using transient logical standby
- Describe the key differences between database rolling upgrade using transient logical standby and alternative upgrade methods
- Perform a database rolling upgrade using transient logical standby

Practice 1 Overview: Database Rolling Upgrade Using Transient Logical Standby

In this practice, you will perform a Database Rolling Upgrade Using Transient Logical Standby. The initial Oracle Database software version on both clusters is 11.2.0.4, and the target software version for the upgraded environment is 12.1.0.1.

2

ASM Filter Driver

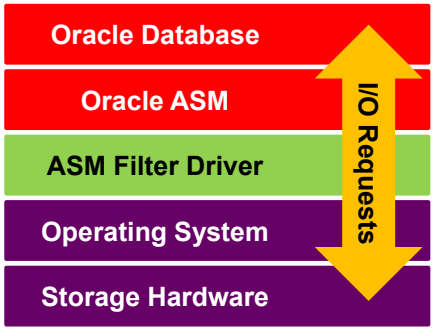
Objectives

After completing this lesson, you should be able to:

- Describe the purpose of ASM Filter Driver
- Describe the architecture of ASM Filter Driver
- Describe how to configure ASM Filter Driver
- Upgrade an existing database to use ASM Filter Driver and see its effect

ASM Filter Driver: Introduction

- First available with Oracle Database release 12.1.0.2
  - Initially on Linux only
  - Implemented as a kernel module situated in the ASM I/O path
- Key capabilities:
  - Reject non-Oracle write I/O:
    - Stops OS utilities from overwriting ASM disks
    - Protects database files
  - Enable device name persistence:
    - Uses labels to identify each disk
  - Reduce OS resource usage:
    - Fewer open file descriptors
  - Enable faster node recovery:
    - Restarts Oracle Custerware instead of restarting the node



ORACLE

Oracle ASM Filter Driver (ASMFD) is a kernel module that resides in the I/O path of the Oracle ASM disks. Logically, ASMFD provides an interface between Oracle binaries and the underlying operating environment, which includes the storage hardware interfaces.

ASMFD is available in Linux versions of Oracle Database starting with version 12.1.0.2. Key capabilities of ASMFD include:

- **Reject non-Oracle write I/O:** As a manager of Oracle storage, ASM is exposed to the OS capabilities when it comes to dealing with storage devices. In particular, non-Oracle commands have the ability to overwrite the contents of ASM disks, which may lead to unrecoverable data loss. ASMFD allows writes only by using an Oracle-specific interface and prevents non-Oracle applications from writing to ASM disks. This protects ASM from accidental corruption.
- **Enable device name persistence:** ASMFD uses a label to persistently identify each disk. Therefore, systems using ASMFD do not require additional configuration to ensure device name persistence by using `udev` rules or third-party storage drivers.

- **Reduce OS resource usage:** An ASM instance contains numerous processes (or threads on Windows). Without ASMFD, each process that is I/O capable needs to have its own dedicated open file descriptor for each disk. When thousands of processes access hundreds of disks, the ensuing explosion of file descriptors leads to considerable OS resource consumption. ASMFD exposes a portal device that can be used for all I/O on a particular host. The same portal device can be shared by all the processes associated with multiple database instances. Therefore, you can drastically reduce the required number of open file descriptors by using ASMFD.
- **Enable faster node recovery:** Node fencing, also known as node eviction, protects the integrity of the cluster by removing failed or misbehaving nodes. Historically, Oracle Clusterware used `init.d` scripts to reboot a fenced node. Although effective, this solution is costly because of the time required to restart the node and restart all required processes. ASMFD allows Oracle Clusterware to perform node-level fencing without a restart. Therefore, with ASMFD, it is possible to achieve the same result by restarting the Oracle software stack instead of restarting the entire node. This process is just as effective, but far quicker.

Configuring ASM Filter Driver

- ASMFD is installed during ASM installation.
- To configure ASMFD:
  1. Modify the ASM disk string:

```
$ asmcmd dsset '/dev/xvd*','AFD:*'
```

  2. Configure ASMFD: Load the kernel module, create device proxies, and update resource dependencies.

```
# asmcmd afd_configure
```

     - Must be performed by the `root` OS user on every node
     - Must stop CRS
     - Can be performed as a rolling operation
  3. Set the ASMFD disk string:

```
$ asmcmd afd_dsset '/dev/xvd*'
```

     - Must be performed on every node

ORACLE

ASMFD software is installed as part of a fresh installation of, or an upgrade to, Oracle Database version 12.1.0.2 or later. With version 12.1.0.2, ASMFD is not configured by default. To use ASMFD requires a one-time configuration process on each node. Following this configuration, each disk must be labeled to enable management by ASMFD.

The slide shows the steps required to configure ASMFD. The disk labeling procedure is introduced later in the lesson. Note the following:

1. You must modify the ASM disk string to enable the discovery of disks labeled by ASMFD. Initially, you should append `'AFD:*'` to the existing ASM disk string for your environment so that ASM can discover the current devices along with any ASMFD labeled devices. Later, after all the disks are labeled, you can optionally remove the non-ASMFD portion of the ASM disk string to ensure that ASM uses only the ASMFD labeled disks. Note that the ASM disk string can be set once and that the setting is automatically visible to all the ASM instances in the cluster.
2. You must configure ASMFD by executing `asmcmd afd_configure` as `root` on every node in the cluster. Oracle Clusterware must be shut down on the node to perform the configuration. However, the configuration can be performed in a rolling manner.
3. You must set a new ASMFD disk string to enable ASMFD to discover the correct disk devices. Unlike the ASM disk string, the ASMFD disk string must be set on each node.

Note that the procedure is slightly different if you are migrating from using ASMlib. Migration from ASMlib to ASMFD is discussed later in this lesson.

Labeling Disks for ASM Filter Driver

- Disks must be labeled to enable management by ASMFD.
- To label disks:
  1. Prepare for disk labeling:
     - You must shut down Oracle Clusterware on all cluster nodes to label disks in a disk group containing the OCR or voting files.
     - You must dismount the associated disk group to label disks in a disk group NOT containing the OCR or voting files.
  2. Label disks:

```
$ asmcmd afd_label <Label Name> <Device> [--migrate]
```

     - Label disks only once on any cluster node.
  3. Discover the labeled disks:

```
$ asmcmd afd_scan
```

     - Labeled disks must be discovered on every cluster node.

ORACLE

To enable ASMFD management, each disk device must be labeled by ASMFD.

1. If the disk currently belongs to a disk group, before labeling, the disk group must be dismounted across the entire cluster. If the disk group contains the Oracle Cluster Registry (OCR) or voting files, you must shut down Oracle Clusterware on all the cluster nodes before labeling the disk.
2. You can use the `asmcmd afd_label` command or the `ALTER SYSTEM LABEL SET` SQL statement to label each disk device. If the disk currently belongs to a disk group, you must use the `migrate` option. Disk labeling is a one-time operation that can be performed on any cluster node.
3. Labeled disks must be discovered on every cluster node. You can use the `asmcmd afd_scan` command or the `ALTER SYSTEM LABEL SCAN` SQL statement to perform a scan on each cluster node. You do not need to perform an ASMFD disk scan on the node where you labeled your disks. For example, if you labeled all your disks using one cluster node, you would need to perform a scan on all the other cluster nodes. However, if you labeled different disks using different cluster nodes, you would need to perform a scan on all the nodes to ensure that all the disks were discovered on all the nodes.

Migrating from ASMLib to ASMFD

1. Reset the ASM disk string to an empty string:

```
$ asmcmd dsset ''
```

2. Stop Oracle Clusterware.
3. Configure ASMFD:

```
# asmcmd afd_configure
```

   – This must be performed by the `root` OS user on every node.
   – ASMlib labels are automatically converted to ASMFD.

4. Discover the labeled disks:

```
$ asmcmd afd_scan '/dev/xvd*'
```

5. Restart Oracle Clusterware.
6. Set the ASMFD disk string:

```
$ asmcmd afd_dsset '/dev/xvd*'
```

ORACLE

To migrate ASMlib disks to ASMFD, users must first reset the ASM disk string to an empty string. An empty ASM disk string causes ASM to scan for disks in a number of default locations, including the default ASMlib path `ORCL:*`, so your ASMlib disks will remain visible to ASM. However, this setting is required by the ASMFD configuration routines, which will otherwise raise an error if ASMlib disks are found in conjunction with a nonempty ASM disk string.

After the ASM disk string is reset, the remaining tasks must be performed on each cluster node. Note that the ASMFD configuration process (step 3) automatically disables ASMlib and converts all ASMlib disk labels to ASMFD labels. Following this step, ensure that the newly relabeled disks are visible to ASMFD by executing a disk scan (step 4). At this point, the ASMFD disk string is not yet set; therefore, you must provide the appropriate device disk string to the `asmcmd afd_scan` command. After you restart Oracle Clusterware (step 5), you should set the ASMFD disk string (step 6) to ensure that the disk devices are automatically discovered by ASMFD from then on. Note that the ASMFD disk string must be set on each node.

Unlabeling Disks and Deconfiguring ASMFD

- Unlabeling a disk:

```
# asmcmd afd_unlabel <Label Name> [-f]
```

  - Can happen at any time if the disk is not in a disk group
- Deconfiguring ASMFD:

```
# asmcmd afd_deconfigure
```

  - Must be performed by the `root` OS user
  - Must be performed separately on each node
  - Must stop CRS

The ASMCMD utility provides commands for unlabeling disks and deconfiguring ASMFD.

To unlabel a disk, use the `asmcmd afd_unlabel` command and supply the label name for the disk being unlabeled. You can unlabel a disk at any time if it does not belong to an ASM disk group. The most likely use case occurs when you return a disk device to a pool of non-Oracle storage. If you want to unlabel a disk that belongs to an ASM disk group, you must at least unmount the disk group. If the disk group in question contains the OCR or voting disks, you must also stop Oracle Clusterware on all the cluster nodes before you can unlabel the disk.

If you want to completely deconfigure ASMFD for any reason, you can use the `asmcmd afd_deconfigure` command. This command must be performed separately on each cluster node by the `root` OS user and Oracle Clusterware must be shut down during this process.

Quiz

Identify the correct statements regarding ASM Filter Driver.

a. ASMFD protects database files by rejecting non-Oracle write I/Os.

b. ASMFD drastically reduces the number of open file descriptors that are required to support large numbers of Oracle Database processes.

c. ASMFD enables Oracle Clusterware to perform node-level fencing without a restart.

Answer: a, b, c

Summary

In this lesson, you should have learned how to:

- Describe the purpose of ASM Filter Driver
- Describe the architecture of ASM Filter Driver
- Describe how to configure ASM Filter Driver
- Upgrade an existing database to use ASM Filter Driver and see its effect

Practice 2 Overview: Configuring and Using ASM Filter Driver

In this practice, you will configure the Oracle ASM Filter Driver on a new cluster running Oracle Grid Infrastructure version 12.1.0.2, and you will perform tests to demonstrate how ASM Filter Driver protects data integrity.

3 Flex ASM

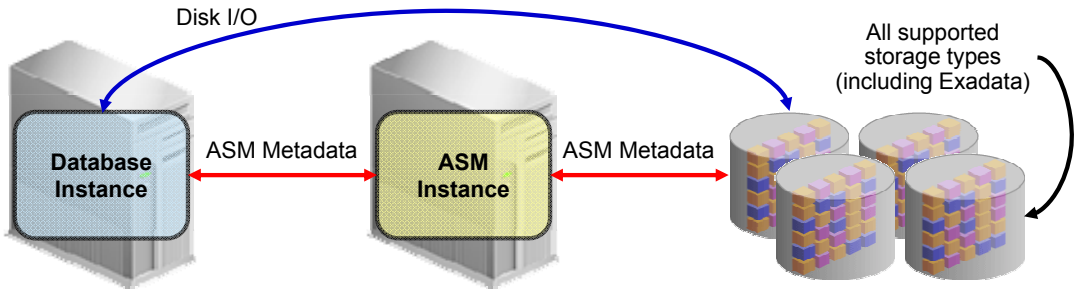
Objectives

After completing this lesson, you should be able to:

- Describe the architecture and components of Flex ASM
- Install and configure Flex ASM
- Administer Flex ASM

Flex ASM: Overview

- In previous versions, ASM clients can access ASM only by using an ASM instance on the same host.
  - Resources are consumed by ASM on every database server.
  - If an ASM instance fails, its clients must fail.
- With Flex ASM, ASM clients can use a network connection to access ASM.
  - Resources are saved because ASM is not required on every database server.
  - If an ASM instance fails, its clients can connect to another instance.



Before Oracle Database 12*c*, an ASM client (database instance or ASM Cluster File System [ACFS]) could connect only to an ASM instance running on the same host. This required every database server to dedicate system resources to ASM, which increased the overall system resource requirement to run Oracle Database in conjunction with ASM. This tightly coupled model also had availability concerns because if an ASM instance failed, all ASM clients on that host also failed.

Oracle Database 12*c* introduces Flex ASM. Flex ASM enables ASM clients to connect to ASM over a network. By relaxing the hard dependency between ASM and its clients, the previous architectural limitations are overcome. With Flex ASM, a smaller pool of ASM instances can be used to serve a large pool of database servers. If an ASM instance fails, its clients can reconnect to another ASM instance.

Note that ASM continues to support the same architecture as previous releases where clients are coupled with ASM instances on the same host. This mode of deployment is called standard ASM.

Flex ASM Instance Changes

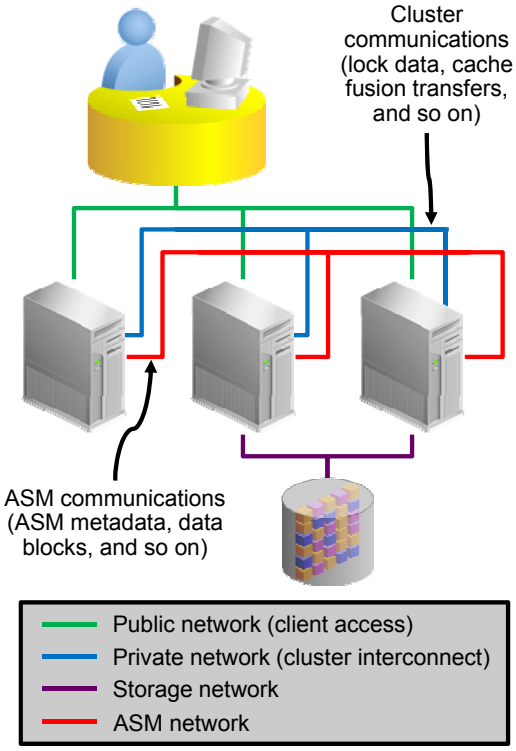
- ASM instances are no longer required to run on every node in a cluster.
- Administrators specify the cardinality for ASM.
  - Cardinality sets the number of instances across the cluster.
  - The default is 3.
- All disk groups are mounted by all ASM instances.

Flex ASM relaxes the requirement for an ASM instance to exist on every node in the cluster. Instead, the administrator specifies the cardinality for ASM. This number specifies the number of ASM instances that should be made available in the ASM cluster. The default cardinality setting for ASM instances is three.

Because an ASM instance can now service any database instance across the cluster, all disk groups are typically mounted on all the ASM instances. As is the case in previous versions, the ASM instance running on a node has its ORACLE_SID set to `+ASM<node number>`.

ASM Network

- In previous versions, Oracle Clusterware requires:
  - A public network for client application access
  - One or more private networks for inter-node communication within the cluster
- Flex ASM adds the ASM network, which is used for communication between ASM and ASM clients.



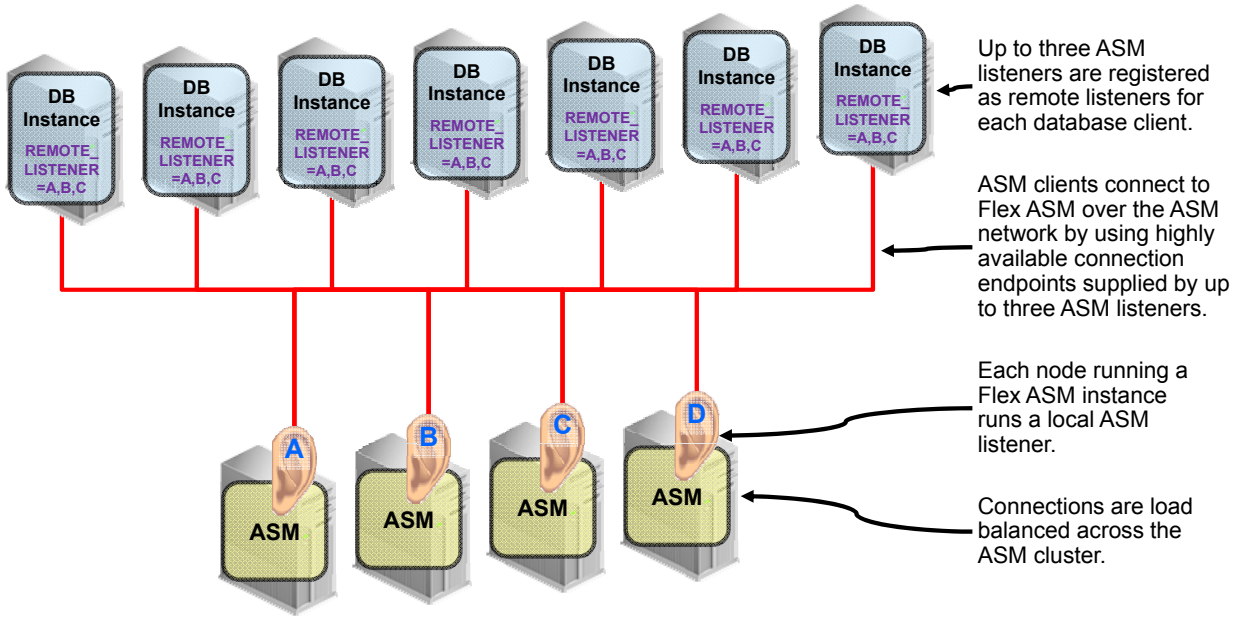
In previous versions, Oracle Clusterware requires access to a public network and one or more private networks. Clients outside the cluster use the public network to connect to servers in the cluster. The private networks are predominantly used for inter-node communication within the cluster. Sometimes, the private network also serves as the storage network. This is the case with Oracle Exadata Database Machine.

Flex ASM introduces a new type of network called an *ASM network*. The ASM network is used for all communication between ASM and its clients. There can be more than one ASM network in a customer environment. ASM provides its services on all the ASM networks, and this requires all ASM networks to be accessible on all the nodes hosting ASM instances.

All ASM clients running within the ASM cluster can use any of the ASM networks to communicate with ASM.

It is possible to configure a network both as a private and an ASM network. That is, a single network can perform both functions.

ASM Listeners



To support Flex ASM connectivity, a set of ASM listeners is configured for every ASM network. The ASM listeners are in addition to other listeners such as the SCAN listeners and the local database listeners. The diagram in the slide illustrates the arrangement of ASM listeners.

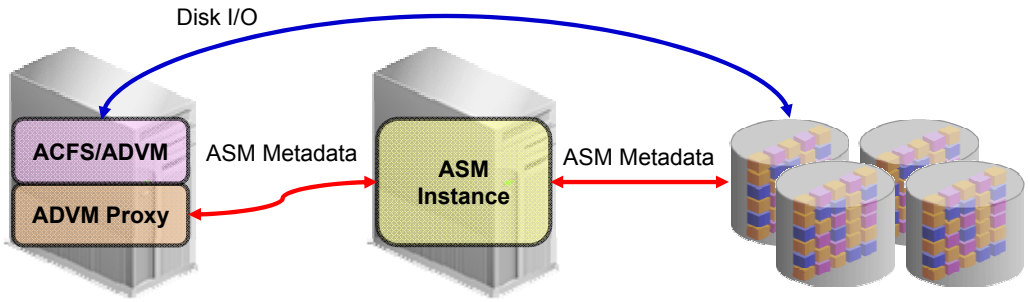
Each node hosting an ASM instance hosts one local ASM listener for each ASM network. Each ASM listener can service client connections over the corresponding ASM network. Up to three ASM listener addresses are registered as remote listeners in each client database instance. Using this arrangement, clients have a highly available connection endpoint to facilitate connection to ASM instances.

While connection is initiated by using one of the registered remote listeners, all client connections are load balanced across the entire set of available ASM instances. The load-balancing mechanism is connect-time load balancing.

ADVM Proxy

The ADVM Proxy is a special Oracle instance.

- It enables ADVM to connect to Flex ASM.
- It is required to run on the same node as ADVM and ACFS.
- By default, it is configured on every node in a standard cluster or every Hub Node in a Flex Cluster.
- It can be shut down when ACFS is not running.

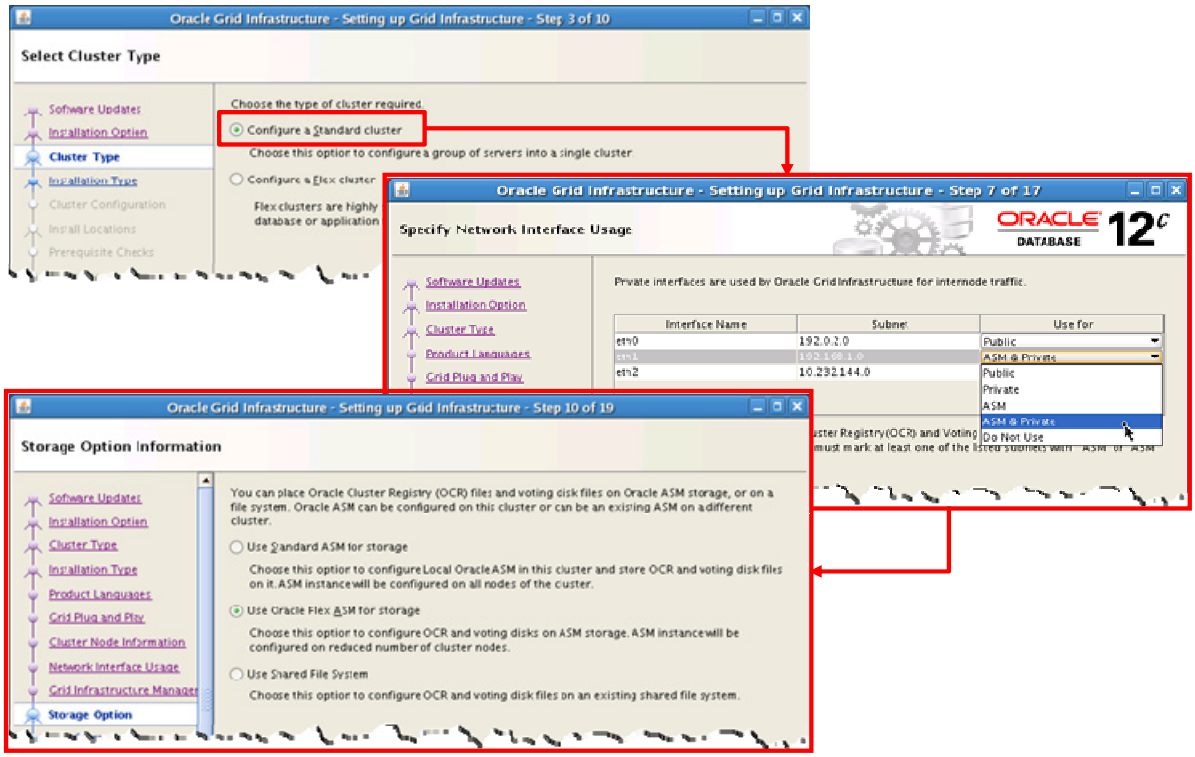


The ADVM Proxy is a special Oracle instance. Its sole purpose is to enable ASM Dynamic Volume Manager (ADVM), and through it ASM Cluster File System (ACFS), to connect to Flex ASM.

In release 12.1, ACFS, ADVM, and the ADVM proxy must reside on the same node. Therefore, by default, the ADVM proxy is configured to run on every node in a standard cluster or every Hub Node in a Flex Cluster. Administrators can shut down the ADVM proxy if ACFS is not running on the node.

The ADVM proxy instance has its ORACLE_SID set to `+APX<node number>`.

Configuring Flex ASM on a Standard Cluster

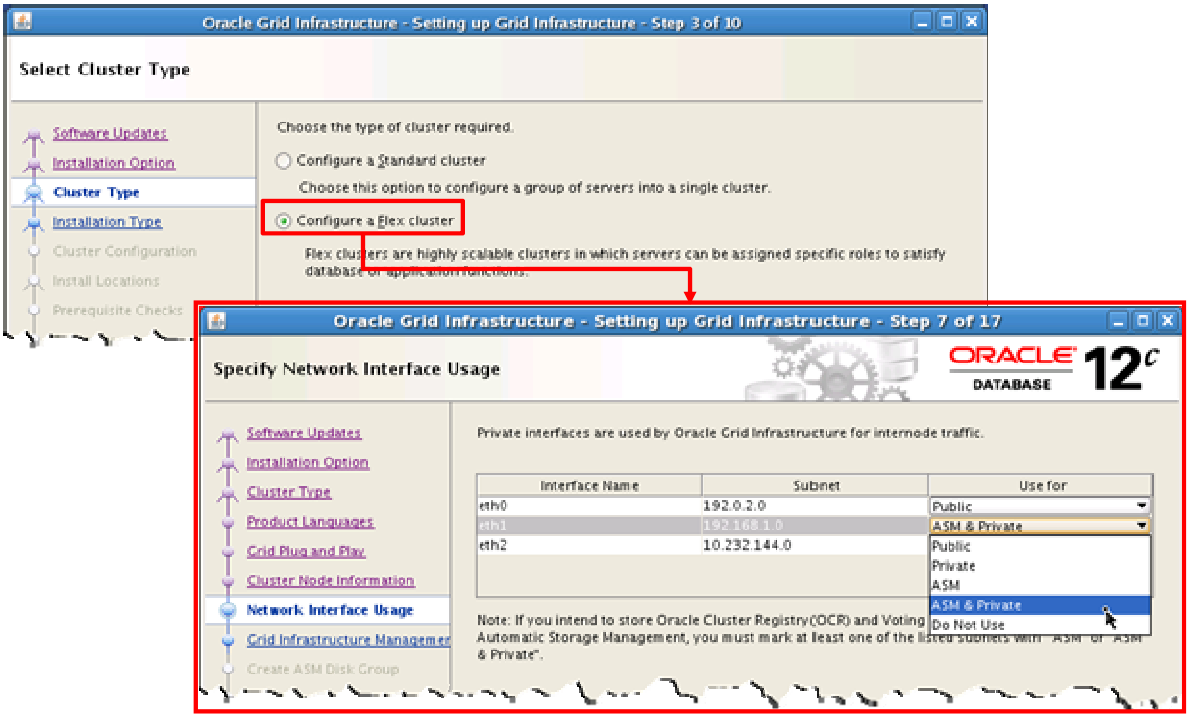


Oracle Universal Installer (OUI) facilitates configuration of Flex ASM. The procedure for configuring Flex ASM using OUI differs slightly, depending on whether or not the cluster is also being configured as a Flex Cluster.

To configure Flex ASM on a standard cluster, the following steps are required:

1. Select “Configure a Standard cluster” on the Select Cluster Type screen.
2. Specify an ASM network on the Specify Network Interface Usage screen.
3. Select “Use Oracle Flex ASM for storage” on the Storage Option Information screen.

Configuring Flex ASM on a Flex Cluster



ORACLE

If the option to configure a Flex Cluster is selected on the Select Cluster Type screen, Flex ASM is implicitly configured and you must specify an ASM network on the Specify Network Interface Usage screen.

Managing Flex ASM Instances

Flex ASM is designed to require minimal monitoring and ongoing management.

- The primary concern is that instances are up and running.

```
$ srvctl status asm -detail
ASM is running on c00n03,c00n02,c00n01
ASM is enabled.
$ srvctl status asm -proxy -detail
ADVM Proxy is running on c00n04,c00n03,c00n02,c00n01
ADVM Proxy is enabled.
```

- No Flex ASM–specific instance parameters are required.
- Default settings will effectively support most situations.
- ASM and ADVM Proxy instances use automatic memory management.
  - Minimum default setting: `MEMORY_TARGET=1076M`

Flex ASM is designed to require minimal monitoring and ongoing management after initial configuration. The primary concern for administrators is that the ASM instances are up and running. This can be verified by using the `srvctl status` commands shown in the slide.

In release 12.1, no new instance parameters are specific to Flex ASM. In addition, the default parameter settings have been adjusted to suit the Flex ASM architecture, making them sufficient to effectively support most situations.

Automatic memory management is used for ASM instances. In release 12.1, the default setting for `MEMORY_TARGET` is based on various attributes of the node hosting the instance, such as the physical memory size and the number of processor cores.

Note that the minimum default `MEMORY_TARGET` setting (`1076M`) is significantly larger than the default `MEMORY_TARGET` setting used by ASM instances in previous versions.

Stopping, Starting, and Relocating Flex ASM Instances

- ASM instances:

```
$ srvctl status asm -detail
ASM is running on c00n03,c00n02,c00n01
ASM is enabled.
$ srvctl stop asm -node c00n03 -f
$ srvctl start asm -node c00n04
$ srvctl status asm -detail
ASM is running on c00n04,c00n02,c00n01
ASM is enabled.
$ srvctl relocate asm -currentnode c00n04 -targetnode c00n03
$ srvctl status asm -detail
ASM is running on c00n03,c00n02,c00n01
ASM is enabled.
```

- ADVM Proxy instances:

```
$ srvctl stop asm -proxy -node c00n03
$ srvctl start asm -proxy -node c00n04
```


At times it may be useful for administrators to control an individual ASM instance or ADVM Proxy instance. The slide shows examples of the `srvctl` commands to stop, start, and relocate individual Flex ASM instances.

Setting the Number of Flex ASM Instances

```
$ srvctl config asm
ASM home: <CRS home>
Password file: +DATA/orapwASM
ASM listener: LISTENER
ASM instance count: 3
Cluster ASM listener: ASMNET1LSNR_ASM
$ srvctl modify asm -count 4
$ srvctl config asm
ASM home: <CRS home>
Password file: +DATA/orapwASM
ASM listener: LISTENER
ASM instance count: 4
Cluster ASM listener: ASMNET1LSNR_ASM
```


The slide shows examples of the commands required to manage the number of Flex ASM instances. To view the current setting, use the `crsctl config asm` command as shown in the slide. To modify the setting, use the `srvctl modify` command shown in the slide.

Note that an error is generated if you attempt to set the number of instances to a value smaller than the current number of running instances. In this case, you must first stop the extra instances, and then modify the number of instances.

Monitoring Flex ASM Connections

```
SQL> select distinct i.instance_name asm_instance_name,
  2  c.instance_name client_instance_name, c.db_name, c.status
  3  from gv$instance i, gv$asm_client c
  4  where i.inst_id=c.inst_id;

ASM_INSTANCE_NAME    CLIENT_INSTANCE_NAME DB_NAME  STATUS
-------------------- -------------------- -------- ------------
+ASM1                +APX1                +APX     CONNECTED
+ASM1                +ASM1                +ASM     CONNECTED
+ASM1                orcl_2               orcl     CONNECTED
+ASM1                orcl_5               orcl     CONNECTED
+ASM1                orcl_7               orcl     CONNECTED
+ASM2                +APX2                +APX     CONNECTED
+ASM2                +ASM2                +ASM     CONNECTED
+ASM2                orcl_1               orcl     CONNECTED
+ASM2                orcl_4               orcl     CONNECTED
+ASM3                +APX3                +APX     CONNECTED
+ASM3                +ASM3                +ASM     CONNECTED
+ASM3                orcl_3               orcl     CONNECTED
+ASM3                orcl_6               orcl     CONNECTED
+ASM3                orcl_8               orcl     CONNECTED
```


At times it may be useful for administrators to know which clients are connected to each ASM instance. This knowledge may be especially useful when considering the impact of shutting down a node for scheduled maintenance or if a change in the cardinality setting for ASM instances is being considered.

To determine the database instances that are connected to a specific ASM instance, ASM administrators can connect to an ASM instance and query the `GV$ASM_CLIENT` table. The example in the slide shows the distribution of eight database instances (`orcl_1` to `orcl_8`) across three Flex ASM instances (`+ASM1`, `+ASM2`, `+ASM3`).

Relocating an ASM Client

- Clients are automatically relocated to another instance if an ASM instance fails.
  - Clients reconnect and the connection is load balanced to an available instance.
- Clients can be manually relocated using the `ALTER SYSTEM RELOCATE CLIENT` command.
  - The command syntax is:

```
SQL> ALTER SYSTEM RELOCATE CLIENT '<instance_name>:<db_name>';
```

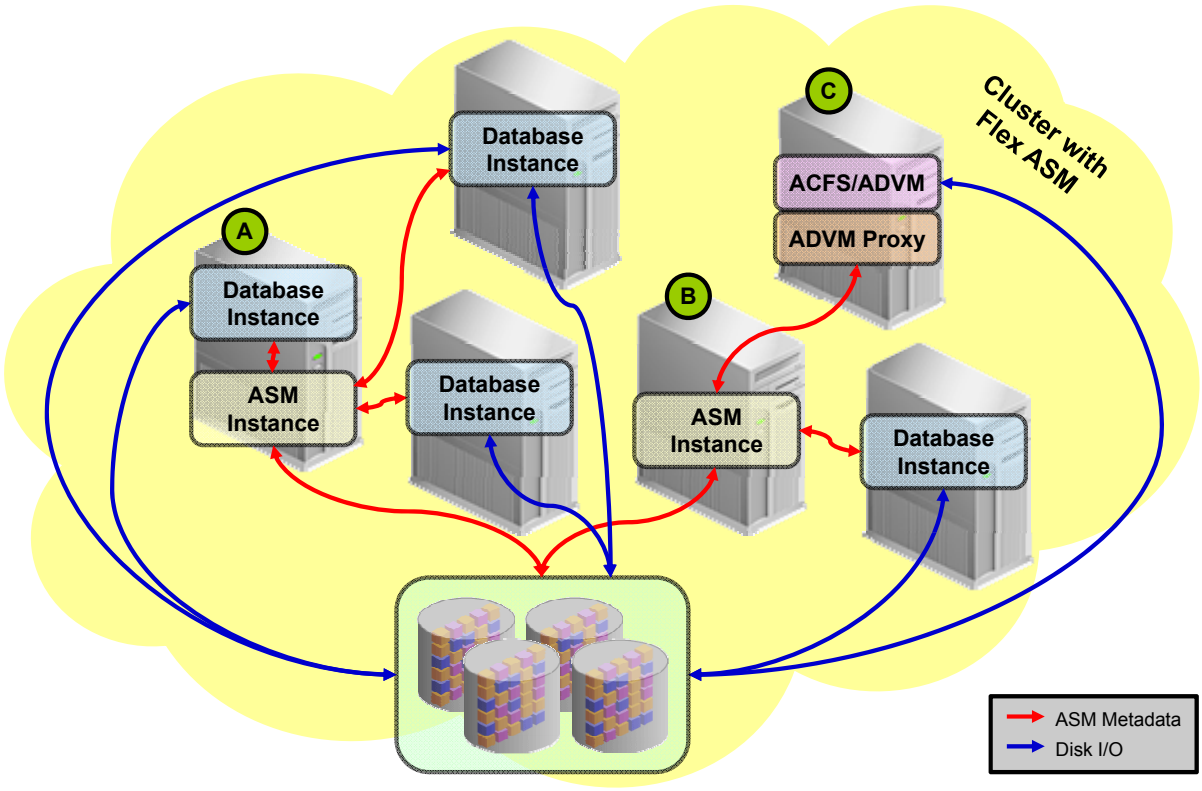
    - Query `GV$ASM_CLIENT` to determine `instance_name` and `db_name`.
  - This is useful for manually adjusting the workload balance between instances.

With Flex ASM, if an ASM instance fails, clients are automatically relocated to another instance. When the failure is detected by the client, it reconnects to an available instance. Like any other connection request, reconnection requests are subject to connection load balancing. Relocation due to failure is automatic and transparent to end users.

In addition to automatic relocation, the `ALTER SYSTEM RELOCATE CLIENT` command can be used to perform manual relocation. This command results in the server terminating the connection to the client, which forces the client to reconnect to the least-loaded ASM instance. Manual relocation is useful for manually adjusting the workload balance between ASM instances when a significant imbalance is detected.

Note that if an ASM client is already connected to the least-loaded ASM instance, the `ALTER SYSTEM RELOCATE CLIENT` command will cause the client to disconnect; however, it will reconnect to the same ASM instance. In this case, to force an ASM client to relocate off the ASM instance, you would need to shut down the ASM instance.

Flex ASM Deployment: Example



ORACLE

The diagram in the slide shows a Flex ASM deployment example. The following list summarizes some of the key points relating to Flex ASM:

- The diagram illustrates a standard cluster running Flex ASM. The diagram also illustrates the Hub Nodes of a Flex Cluster.
- ASM clients can run on any node in a standard cluster, or on any Hub Node in a Flex Cluster. In release 12.1, Flex ASM does not support clients on Leaf Nodes.
- Flex ASM enables a smaller number of ASM instances (two in this example) to service a larger number of clients (four database instances and one ACFS in this example).
- Flex ASM enhances the availability of Oracle Database and ACFS by helping to protect against various ASM failures. If, for example, the ASM instance at node A failed, the three database instances it supports would transparently connect to the ASM instance at node B.
- The ASM cardinality setting specifies the number of ASM instances that should be made available in the cluster. In this example, the ASM cardinality is two. The default cardinality setting for ASM instances is three.

- Depending on the distribution of clients and ASM instances, an ASM client may reside on the same node as an ASM instance (as shown on node A in the diagram), or the ASM instance may reside on a node separate from the ASM clients (as shown on node B in the diagram).
- By default, the ADVM proxy runs on every node in a standard cluster or every Hub Node in a Flex Cluster. For the sake of simplicity, the ADVM proxy is shown only on node C in the diagram, which in this example is the only node running an ASM Cluster File System.

Flex ASM and Flex Clusters

- Flex Clusters require Flex ASM.
  - Standard ASM is not supported on a Flex Cluster.
- Flex ASM does not require a Flex Cluster.
  - Flex ASM can run on a standard cluster servicing clients across the cluster.
  - Flex ASM can run on the Hub Nodes of a Flex Cluster servicing clients across the Hub Nodes of the Flex Cluster.
- The benefits of Flex ASM apply regardless of cluster type:
  - Smaller ASM resource footprint
  - Protection from ASM failure

ORACLE

Standard ASM is not supported on a Flex Cluster. Therefore, Flex Clusters require Flex ASM. However, Flex ASM does not require a Flex Cluster. Flex ASM can be configured on either a standard cluster or a Flex Cluster.

When Flex ASM runs on a standard cluster, ASM services can run on a subset of cluster nodes servicing clients across the cluster. When Flex ASM runs on a Flex Cluster, ASM services can run on a subset of Hub Nodes servicing clients across all the Hub Nodes in the Flex Cluster.

The fundamental benefits of Flex ASM apply regardless of the type of cluster being used. That is:

- The overall resource footprint is smaller because a smaller pool of ASM instances can be used to serve a larger pool of database servers
- Higher availability can be achieved because if an ASM instance fails, its clients can reconnect to another ASM instance

Quiz

Identify the correct statements about Flex ASM.

a. Flex ASM requires an ASM instance on each cluster node running an Oracle Database instance.
b. Flex ASM allows ASM clients to remotely connect to ASM over a network.
c. With Flex ASM, a small pool of ASM instances can be used to serve a larger pool of database servers.
d. If an ASM instance fails, the database clients and ASM cluster file systems can reconnect to another ASM instance.

ORACLE

Answer: b, c, d

Flex ASM relaxes the hard dependency between ASM and database clients. Therefore, Flex ASM does not require an ASM instance on each cluster node running an Oracle Database instance.

Quiz

If OUI is used to install and configure a four-node standard cluster with Flex ASM, which statement describes the resulting configuration?

a. ASM instances run on two cluster nodes for high availability, and more ASM instances are started as the number of ASM clients increases.
b. ASM instances run on the first three cluster nodes.
c. Three ASM instances are spread across the cluster.
d. Each of the four nodes runs an ASM instance.

Answer: c

The default cardinality for ASM instances is three. Regardless of the cluster size, Clusterware attempts to start three ASM instances when a new cluster is configured with Flex ASM. Fewer than three instances may start only if an error prevents an ASM instance from starting, or if there are fewer than three nodes in a standard cluster, or fewer than three Hub Nodes in a Flex Cluster. The ASM instances may start on the first three nodes, as suggested in answer b; however, this will not always be the case.

Quiz

Which statement best describes the relationship between Flex Clusters and Flex ASM?

a. There is no relationship, except that both have “Flex” in their names.
b. A Flex Cluster requires Flex ASM, but Flex ASM does not require a Flex Cluster.
c. Flex ASM requires a Flex Cluster, but a Flex Cluster does not require Flex ASM.
d. Flex Clusters and Flex ASM always require each other.

Answer: b

A Flex Cluster requires Flex ASM. However, Flex ASM can also run on a standard cluster providing I/O services on a subset of the cluster nodes.

Summary

In this lesson, you should have learned how to:

- Describe the architecture and components of Flex ASM
- Install and configure Flex ASM
- Administer Flex ASM

Practice 3 Overview: Converting to Flex ASM and using Flex ASM

In these practices, you will:

- Convert an existing two-node cluster from standard ASM to Flex ASM
- Set the number of ASM instances
- Monitor Flex ASM to determine how database clients are using the available ASM instances
- See how Flex ASM insulates database clients from ASM instance failures, thus increasing database availability

ORACLE

4

Policy-Based Cluster Management, Policy-Managed Database, and Oracle Multitenant Architecture

Objectives

After completing this lesson, you should be able to:

- Describe the purpose of policy-based cluster management
- Describe the architecture and components of policy-based cluster management
- Implement a policy set and enforce a policy
- Describe the purpose and architecture of policy-managed database
- Describe how policy-based cluster management interacts with policy-managed database
- Convert an existing administrator-managed database into a policy-managed database
- Create a new policy-managed database
- Describe how the Oracle Multitenant architecture works with policy-based cluster management and policy-managed databases
- Expose PDBs as services and integrate them with policy-based management

ORACLE

Policy-Based Cluster Management Enhancements: Overview

- In previous releases:
  - A cluster can be logically divided into server pools
    - The server pools collectively define a policy.
    - All nodes are assumed to be equal.
  - Quality of Service Management uses a different policy
    - Potential for overlap, confusion, and inconsistency
- With Oracle Clusterware 12*c*, policy-based cluster management is enhanced to provide:
  - Extended server attributes to govern node placement
  - A library of policy definitions with an easy way of switching between policies
  - Unification of policies for Oracle Clusterware and Quality of Service Management

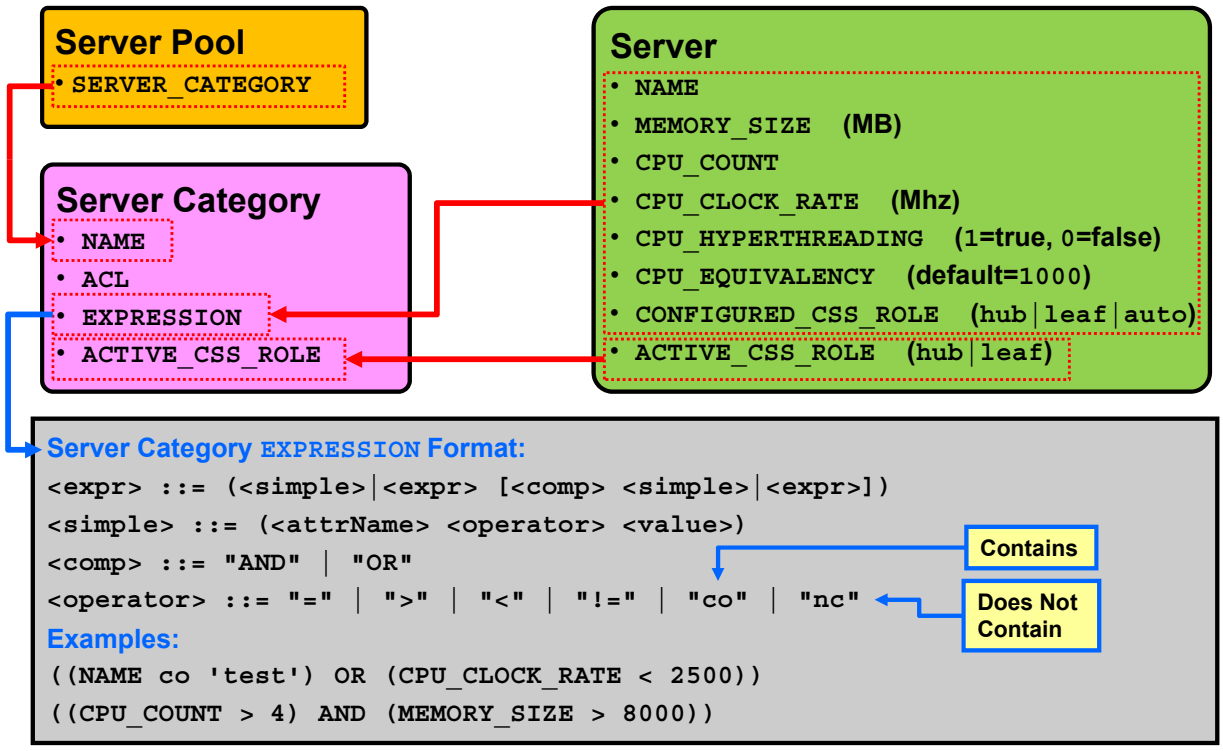
With the policy-based cluster management capability introduced in Oracle Clusterware 11*g*, release 2, a cluster can be logically divided into groups of servers known as server pools. The placement of nodes in each server pool is governed by the relative importance assigned to each server pool, along with other attributes such as the minimum and maximum number of nodes assigned to each server pool.

The server pool definitions effectively define a policy, which enables dynamic capacity assignment and fast resource failover when the number of nodes in the cluster changes. With release 11.2, all nodes are assumed to be equal; that is, there is no way to specify server attributes that favor server placement in a particular server pool.

In release 11.2, the Quality of Service (QoS) Management feature defines a separate policy for cluster resource management, which can be confusing for administrators unless they are familiar with all the policies. In addition, there exists a potential to create policies that are contrary to each other.

In Oracle Clusterware 12*c*, policy-based cluster management is enhanced in three important ways. Firstly, numerous server attributes allow for greater flexibility and control over node assignments to different server pools. Secondly, an extended policy framework allows administrators to maintain a library of policies and easily switch between them as required. Finally, policy-based cluster management has been unified with QoS Management.

Server Categorization



In previous versions, the assignment of servers to server pools was based on the relative importance of each server pool and a few basic attributes, such as the minimum and maximum number of servers associated with the server pool. Because there was no way to differentiate between servers, all servers were assumed to be homogeneous with respect to CPU, memory, and other resources.

With Oracle Clusterware 12*c*, the notion of server categorization is introduced. Categorization allows servers to be differentiated and provides a mechanism for automatically controlling the composition of each sever pool. Server categorization works as follows:

- Every server contains a set of new attributes. Most of the attributes specify key physical characteristics of the server, such as `MEMORY_SIZE` and `CPU_COUNT`, or they contain configuration settings relating to Oracle Clusterware, such as `CONFIGURED_CSS_ROLE`. For a complete list of server attribute definitions, refer to the *Oracle Clusterware Administration and Deployment Guide 12c Release 1 (12.1).*
- A new Clusterware object defines server categories. The main attribute of each server category is an expression that is evaluated against the attributes of each server to determine whether the server belongs to the category.
- A new attribute, `SERVER_CATEGORY`, is added to each server pool definition. This attribute allows a server category to be associated with each server pool, thereby governing which servers can be in the pool.

Administering Server Categorization: Server Attributes

- Most server attributes are automatically discovered by Oracle Clusterware.
- Viewing attribute settings example:

```
$ crsctl status server c00n01 -f
NAME=c00n01
MEMORY_SIZE=4006
CPU_COUNT=1
CPU_CLOCK_RATE=2857
CPU_HYPERTHREADING=0
CPU_EQUIVALENCY=1000
DEPLOYMENT=other
CONFIGURED_CSS_ROLE=hub
RESOURCE_USE_ENABLED=1
SERVER_LABEL=UNAVAILABLE
PHYSICAL_HOSTNAME=UNAVAILABLE
STATE=ONLINE
ACTIVE_POOLS=Free
STATE_DETAILS=
ACTIVE_CSS_ROLE=hub
```


Most server attribute settings are automatically discovered by Oracle Clusterware. An exception is the `SERVER_LABEL` attribute, which administrators can set by using the `crsctl modify server` command.

Administrators can view the server attributes by using the `crsctl status server` command. An example is displayed in the slide. Note the use of the `-f` option to display a full listing of all server attributes.

Administering Server Categorization: Server Categories

- Creating a new server category:

```
$ crsctl add category <catName> -attr "<attrName>=<value>[,...]"
```

```
$ crsctl add category small -attr "EXPRESSION='(CPU_COUNT = 1)'"
```

- Modifying an existing server category:

```
$ crsctl modify category <catName> -attr "<attrName>=<value>[,...]"
```

```
$ crsctl modify category small -attr "ACTIVE_CSS_ROLE='hub'"
```

- Viewing a category:

```
$ crsctl status category <catName>
```

```
$ crsctl status category small
NAME=small
ACL=owner:grid:rwx,pgrp:oinstall:rwx,other::r--
ACTIVE_CSS_ROLE=hub
EXPRESSION=(CPU_COUNT = 1)
```

- Deleting a category:

```
$ crsctl delete category <catName>
```

ORACLE

The slide shows examples of the commands that can be used to create, modify, view, and delete a server category.

When creating or modifying a server category, note that the key attribute is `EXPRESSION`, which defines the servers that can belong to the category. With the `ACTIVE_CSS_ROLE` attribute, administrators can specifically define different server categories for Hub Nodes and for Leaf Nodes. The `ACTIVE_CSS_ROLE` attribute should not be referenced in the `EXPRESSION` string.

Administering Server Categorization: Server Categories

- Listing servers in a category:

```
$ crsctl status server -category <catName>
```

```
$ crsctl status server -category small
NAME=c00n01
STATE=ONLINE
...
```

- Listing categories for a server:

```
$ crsctl status category -server <serverName>
```

```
$ crsctl status category -server c00n01
NAME=ora.hub.category
ACL=owner:root:rwx,pgrp:root:r-x,other::r--
ACTIVE_CSS_ROLE=hub
EXPRESSION=

NAME=small
ACL=owner:grid:rwx,pgrp:oinstall:rwx,other::r--
ACTIVE_CSS_ROLE=hub
EXPRESSION=(CPU_COUNT = 1)
```

ORACLE

After a category is defined, it may be useful to understand which servers are members of that category. This can be achieved by using the `crsctl status server` command with the `-category` option, as shown in the slide.

It is also possible to list all the categories that apply to a specific server by using the `crsctl status category` command with the `-server` option. Note that a server can belong to multiple categories at the same time, as shown in the example in the slide. The `ora.hub.category` category is an internal category that is used to categorize Hub Nodes.

Administering Server Categorization: Server Pools

- Specifying the SERVER_CATEGORY attribute:

```
$ crsctl add serverpool hr -attr "SERVER_CATEGORY='medium'" ...
```

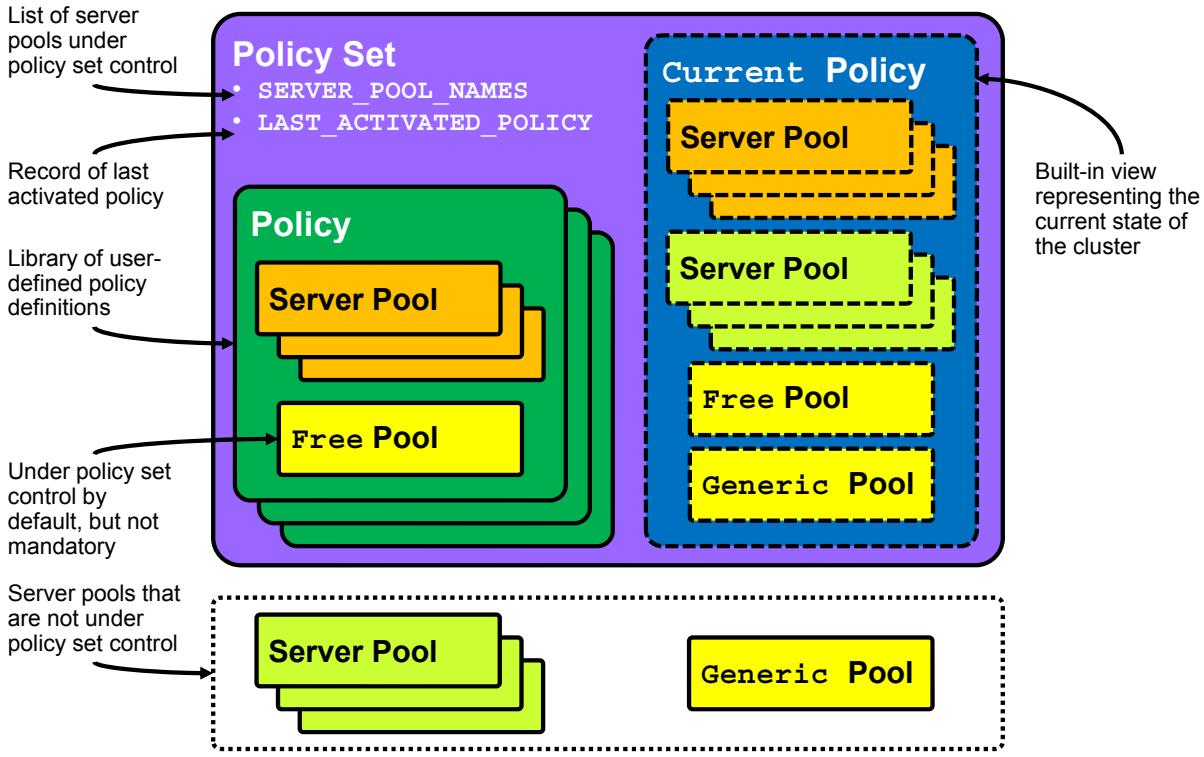
```
$ crsctl modify serverpool dev -attr "SERVER_CATEGORY='small'"
```

- Viewing the SERVER_CATEGORY attribute:

```
$ crsctl status serverpool dev -f
NAME=dev
IMPORTANCE=0
MIN_SIZE=0
MAX_SIZE=-1
SERVER_NAMES=
PARENT_POOLS=
EXCLUSIVE_POOLS=
ACL=owner:grid:rwx,pgrp:oinstall:rwx,other::r--
SERVER_CATEGORY=small
ACTIVE_SERVERS=c00n01 c00n02
```

Server categories are applied to server pools by using the new SERVER_CATEGORY attribute. This attribute can be specified for new and existing server pools, as shown in the examples in the slide. To view the setting of the SERVER_CATEGORY attribute, use the `crsctl status serverpool` command with the `-f` option.

Policy Set: Overview



The diagram in the slide illustrates the policy set contained in Oracle Clusterware 12*c*.

There is always exactly one policy set defined and used by the cluster. The policy set contains two attributes. The `SERVER_POOL_NAMES` attribute defines the names of all server pools controlled by the policy set. The policy set also contains an attribute that records the last activated policy.

The policy set contains zero or more user-defined policies. Each policy contains exactly one definition for each server pool controlled by the policy set. Typically, administrators create policies to address different priorities at different times.

Server pools may also exist outside the control of the policy set. For example, a server pool can be created with the `srvctl add serverpool` command and not be listed in the `SERVER_POOL_NAMES` policy set attribute. Server pools that are outside the control of the policy set are not directly affected by policy changes; however, such server pools can be indirectly affected by policy changes.

For example, a server pool that is outside the control of the policy set must yield a server when a policy change increases the number of servers allocated to a server pool with a higher priority, and there are no free servers or lower priority server pools elsewhere. Likewise, a server pool that is outside the control of the policy set may grow in size because a policy change frees up a server.

The policy set always contains a special built-in policy, named `Current`, representing the configuration that is currently in effect. The `Current` policy includes all server pools that are not under policy set control. When a user-defined policy is activated, its attributes are reflected in the `Current` policy. Over time, the `Current` policy may cease to reflect the last activated policy because of changes made outside the policy set; for example, when a server pool associated with a release 11.2 policy-managed database is added to the system.

In previous versions, two built-in server pools existed: `Generic` and `Free`. With Oracle Clusterware 12*c*, these built-in server pools remain; however, they are handled differently. By default, the `Free` server pool is implicitly controlled by the policy set. However, administrators can choose to remove the `Free` server pool from the `SERVER_POOL_NAMES` list if they want to place the `Free` pool outside direct policy set control. The `Generic` server pool is never under direct policy set control. It is listed as a server pool in the `Current` policy view.

Policy-Based Cluster Management: Configuration Methods

Two methods for configuring and running policy-based cluster management:

- User-defined policy management
  - Clusterware administrators manually configure the policy set.
  - Clusterware administrators activate different policies as required.
    - Administrators can use a job scheduling system to automatically activate specific policies at different times.
- Quality of Service (QoS) Management
  - QoS Management interfaces are used to configure the policy set.
    - Administrators cannot directly modify the policy set.
  - QoS Management automatically adjusts resource allocations in response to workload demands.

There are two distinct modes of operation that apply to policy-based cluster management.

- **User-defined policy management:** With user-defined policy management, Clusterware administrators manually configure the policy set, policies, server pools, and their attributes. After configuration, it is the responsibility of the Clusterware administrator to activate the required policy. Policies can also be activated automatically by using a job scheduling system or another program that uses the supplied commands and application programming interfaces (APIs).
- **QoS Management:** During the implementation of QoS Management, the QoS Management interfaces are used to configure a QoS Management policy set, which also contains a Clusterware policy set definition. When QoS Management is activated, the associated Clusterware policy set definition is activated and locked so that Clusterware administrators cannot manually modify the policy set. This allows QoS Management to automatically adjust policy settings to fulfill the prescribed performance objectives.

  In essence, QoS Management extends the functionality that is available with user-based policy management. Direct policy manipulation outside the QoS Management interfaces is not possible while QoS Management is enabled.

The remainder of this lesson focuses on user-defined policy management as a new feature of Oracle Clusterware 12*c*. QoS Management is outside the scope of this course.

Viewing the Policy Set

```
$ crsctl status policyset
ACL=owner:grid:rwx,pgrp:oinstall:rwx,other::r-x
LAST_ACTIVATED_POLICY=
SERVER_POOL_NAMES=Free
POLICY
  NAME=Current
  DESCRIPTION=This policy is built-in and managed automatically to
reflect current configuration
  SERVERPOOL
    NAME=Free
    ACTIVE_SERVERS=c00n01 c00n02
    EXCLUSIVE_POOLS=
    IMPORTANCE=0
    MAX_SIZE=-1
    MIN_SIZE=0
    PARENT_POOLS=
    SERVER_CATEGORY=
    SERVER_NAMES=
...
```


A policy set is implicitly created in every cluster. Because there is exactly one policy set per cluster, policy sets cannot be created or deleted and no name is required to identify the policy set.

The `crsctl status policyset` command can be used to view the current policy set attributes, including all policies and server pools that are defined on the cluster. The output also includes information about the current state of the cluster, which is listed under the special built-in policy named `Current`.

Configuring a User-Defined Policy Set: Method 1

1. Set the SERVER_POOL_NAMES policy set attribute:

```
$ crsctl modify policyset -attr "SERVER_POOL_NAMES='dev test'"
```

2. Add the policies:

```
$ crsctl add policy day -attr "DESCRIPTION='The day policy'"

$ crsctl add policy night -attr "DESCRIPTION='The night policy'"
```

3. Set the server pool attributes for each policy:

```
$ crsctl modify serverpool dev -attr
"IMPORTANCE=10,MAX_SIZE=2,MIN_SIZE=1,SERVER_CATEGORY=small" -policy day

$ crsctl modify serverpool test -attr
"IMPORTANCE=5,MAX_SIZE=2,MIN_SIZE=1" -policy day

$ crsctl modify serverpool dev -attr
"IMPORTANCE=5,MAX_SIZE=2,MIN_SIZE=0,SERVER_CATEGORY=small" -policy night

$ crsctl modify serverpool test -attr
"IMPORTANCE=10,MAX_SIZE=2,MIN_SIZE=2" -policy night
```

ORACLE

The policy set can be configured by using the `crsctl` command-line utility. The slide outlines one method and shows a series of example commands. Perform the following steps:

1. Set the SERVER_POOL_NAMES policy set attribute. This attribute formally defines the scope of the server pools that are controlled by the policy set. In addition, any server pool named in the SERVER_POOL_NAMES policy set attribute is implicitly created if it did not previously exist.
2. Add the policies. Each policy that is created in this phase is automatically created with a default set of attributes describing each of the server pools named in the previous step.
3. Set the attributes for the server pools in each policy.

Configuring a User-Defined Policy Set: Method 2

1. Create a policy set definition file:

```
$ cat policyset.txt
SERVER_POOL_NAMES=dev test
POLICY
  NAME=day
  DESCRIPTION=The day policy
  SERVERPOOL
    NAME=dev
    IMPORTANCE=10
    MAX_SIZE=2
    MIN_SIZE=1
    SERVER_CATEGORY=small
  SERVERPOOL
    NAME=test
    IMPORTANCE=5
    MAX_SIZE=2
    MIN_SIZE=1
```

```
POLICY
  NAME=night
  DESCRIPTION=The night policy
  SERVERPOOL
    NAME=dev
    IMPORTANCE=5
    MAX_SIZE=2
    MIN_SIZE=0
    SERVER_CATEGORY=small
  SERVERPOOL
    NAME=test
    IMPORTANCE=10
    MAX_SIZE=2
    MIN_SIZE=2
```

2. Modify the policy set:

```
$ crsctl modify policyset -file policyset.txt
```

ORACLE

The example in this slide shows another way of configuring the policy set. It yields the same result as the commands for method 1.

This method of policy set configuration uses a text file that contains the policy set attribute definitions that are to be implemented. You can use the text file shown in this example as a template for configuring your own policy sets. In this example, the policy set contains two policies and two server pools; however, any number of policies and server pools can be specified this way. After the policy set definition file is created, the policy set can be modified by using the `crsctl modify policyset` command shown in the slide.

Administrators can also use the `crsctl status policyset` command with the `-file` option to dump the current policy set definition into a text file. The resulting text file can then be edited and loaded back into the system by using the `crsctl modify policyset` command shown in the slide. This method is an effective way to configure the policy set when administrators start with existing server pool definitions that were created with previous Clusterware releases.

Modifying a User-Defined Policy Set

- Method 1
  - Modify the policy set directly by using `crsctl` commands.
  - Examples:

```
$ crsctl add policy day -attr "DESCRIPTION='The day policy'"
```

```
$ crsctl modify serverpool dev
  -attr "IMPORTANCE=10,MAX_SIZE=2,MIN_SIZE=1,SERVER_CATEGORY=small"
  -policy day
```

- Method 2
  1. Create a policy set definition file:

```
$ crsctl status policyset -file policyset.txt
```

  2. Edit the policy set definition file.
  3. Modify the policy set:

```
$ crsctl modify policyset -file policyset.txt
```


There are essentially two ways to modify a user-defined policy set:

- Administrators can directly modify the policy set by using specific `crsctl` commands. Examples of direct manipulation are listed in the slide.

  By default, attempts to modify policies and server pools fail if the modification causes a cluster-managed resource, such as a database instance, to shut down. Alternatively, the `-f` command-line option can be added to force the change. In addition, what-if command evaluation can be used to test the effect of a change prior to implementation.
- Administrators can use the `crsctl status policyset` command with the `-file` option to dump the current policy set definition into a text file. The resulting text file can then be modified to define an updated policy set. Finally, the updated policy set can be loaded into the system by using the `crsctl modify policyset` command with the `-file` option.

  This method also provides an effective way to configure the policy set when administrators start with existing server pool definitions that were created with Oracle Clusterware, release 11.2.

Activating a User-Defined Policy

- Set the LAST_ACTIVATED_POLICY policy set attribute:

```
$ crsctl modify policyset -attr "LAST_ACTIVATED_POLICY='day'"
```

- Verify the policy settings:

```
$ crsctl status policyset
...
LAST_ACTIVATED_POLICY=day
SERVER_POOL_NAMES=dev test Free
POLICY
  NAME=Current
...
  SERVERPOOL
    NAME=dev
    ACTIVE_SERVERS=c00n01
...
  SERVERPOOL
    NAME=test
    ACTIVE_SERVERS=c00n02

...
```

```
$ crsctl status policy -active
POLICY
  NAME=Current
...
  SERVERPOOL
    NAME=dev
    ACTIVE_SERVERS=c00n01
...
  SERVERPOOL
    NAME=test
    ACTIVE_SERVERS=c00n02
...
```

ORACLE

After the policy set is initially configured, none of the defined policies are active. To activate a policy, the LAST_ACTIVATED_POLICY policy set attribute must be set. An example is shown at the top of the slide. The active policy can be changed at will by using this command. Alternatively, system administrators can use a job scheduling system or other management tools to automatically activate different policies based on different times of day or other circumstances.

When a new policy is activated, nodes are automatically reassigned to server pools, and relevant resources are automatically started or stopped in line with the new active policy.

You can examine the active policy by using the crsctl status commands shown in the slide. Examine the LAST_ACTIVATED_POLICY policy set attribute, and also check the server assignments in each server pool along with other server pool attributes to verify the policy settings.

Policy-Managed Databases Versus Administrator-Managed Databases

- Administrator-managed databases:
  - Administrator specifies database placement.
  - Administrator specifies service placement.
- Policy-managed databases:
  - Administrator associates database services with server pools.
  - Oracle automatically manages database instances and services based on the available resources.
    - No direct association between servers and databases

Oracle provides flexible workload management capabilities that allow an Oracle RAC database to assume different personalities based on different workload requirements and node availability. There are two fundamentally different management styles available.

For administrator-managed RAC databases, the administrator specifies a list of cluster nodes where RAC instances will run. Database services may also be associated with preferred and alternative nodes. With administrator-managed database, there is an explicit administrator-defined association between database services, instances, and cluster nodes.

Oracle Database 11*g* Release 2 introduced the policy-managed database feature, which breaks the explicit association between database services, instances, and cluster nodes.

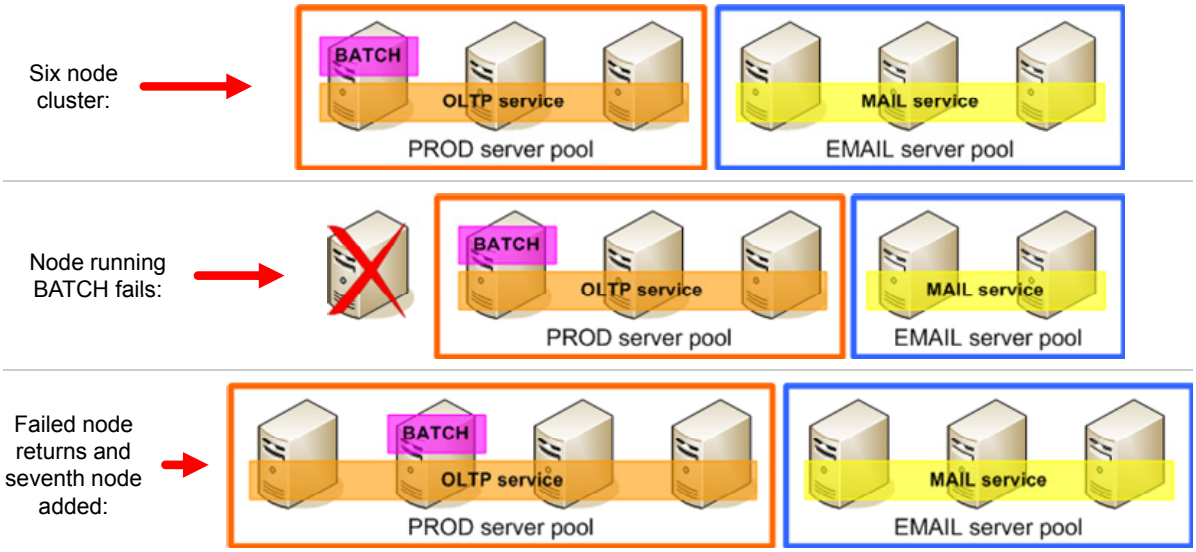
For policy-managed databases, the administrator specifies associations between database services and server pools. Then, Oracle automatically starts and stops RAC instances and database services in line with the available cluster resources.

A policy-managed database consistently delivers Oracle RAC database services based on the total available resources in the cluster, rather than which nodes happen to be available. By using policy-managed databases, you can also ensure that the most important database services are always prioritized over less important services; that is, less important services will automatically shut down to ensure that the most important services get the resources that they require.

Policy-Managed Database: Example

Scenario: Policy-managed RAC databases sharing a cluster

- Production database
  - Services: OLTP (uniform), BATCH (singleton)
  - Associated with PROD server pool
    - Pool attributes: min=3, max=4, importance=10
- Email database
  - Service: MAIL (uniform)
  - Associated with EMAIL server pool
    - Pool attributes: min=3, max=5, importance=5



ORACLE

Consider the policy-based example in this slide. For the sake of simplicity, assume that policy-based cluster management is not being used, so the server pool specifications are not subject to change. Also, assume that the servers all have the same hardware specification, making them interchangeable in the cluster.

By applying the specifications listed in the slide to a six-node cluster, three nodes would be allocated to the PROD server pool and three nodes would be allocated to the EMAIL server pool. The OLTP and MAIL services would both span all the servers in their respective server pools, while the BATCH service would reside on only one server on the PROD pool.

If the server running the BATCH service failed, one of the servers from the EMAIL server pool would be dynamically reallocated to the PROD server pool because the failure would cause PROD to fall below its minimum and it has a higher importance than EMAIL. In that case, the OLTP service would be extended to the reallocated node and the BATCH service would be restarted on one of the surviving PROD nodes.

If the failed node later returned to the cluster, it would be allocated to the EMAIL pool to satisfy the minimum number of servers for that server pool.

If the cluster grew to seven nodes, the additional node would be allocated to the PROD pool because of its higher importance. The OLTP service would also be extended to the new node.

If the cluster grew to eight nodes, the eighth node would be added to the EMAIL pool because the PROD pool would already contain its specified maximum of four nodes.

Policy-Managed Databases and Policy-Based Cluster Management

- Policy-based cluster management:
  - It is primarily concerned with how servers map to server pools.
  - The policy set governs how servers are allocated to server pools.
  - A policy change may trigger an automatic server reallocation.
- Policy-managed databases:
  - It is primarily concerned with how databases map to server pools.
  - Database services are associated with server pools.
  - Automatic placement of instances and services is based on the current server pool allocation.
- How they work:
  - Together: Both technologies provide a foundation for consolidating multiple databases on a cluster.
    - Policies govern the composition of the cluster, which determines the amount of server resources allocated to each database service.
    - Policy changes automatically start and stop policy-managed database instances and services as required.
  - Separate: Both technologies *can* be used in isolation.

ORACLE

In simple terms, policy-based cluster management controls how server resources are grouped into different logical groups known as server pools. Policy-managed databases use server pools to control the placement of database instances and services.

Policy-based cluster management and policy-managed databases are complementary technologies that provide a foundation for consolidating multiple databases on a cluster. You can define polices and policy-managed databases that satisfy your business priorities and cater for different workload profiles. Also, you can quickly and easily change policies to meet the demands of different workload cycles. Then, when a policy change causes a server pool change, the associated policy-managed database instances and services are automatically stopped and started as required.

However, policy-based cluster management and policy-managed databases are separate technologies that do not absolutely rely on each other. For example, you could run multiple policy-managed databases on a cluster without ever defining your own policy set. In such cases, you could define and adjust your server pools manually by using the appropriate `srvctl` commands. Also, you can use policy-based cluster management to manage application resources without ever using a policy-managed database.

Converting to a Policy-Managed Database

- Basic procedure:
  1. Create the required server pools.
  2. Associate the database with the server pools.
  3. (Optional) Associate the database services with a server pool.
  4. Restart the database.
- Single server pool example using only the default service:

```
$ srvctl add srvpool -serverpool poolA
$ srvctl modify database -d orclB -serverpool poolA
$ srvctl stop database -d orclB
$ srvctl start database -d orclB
```

- Multiple server pool example with nondefault database services:

```
$ srvctl add srvpool -serverpool poolX
$ srvctl add srvpool -serverpool poolY
$ srvctl modify database -d orclZ -serverpool "poolX,poolY"
$ srvctl modify service -db orclZ -service srvV -serverpool poolX
$ srvctl modify service -db orclZ -service srvW -serverpool poolY
$ srvctl stop database -d orclZ
$ srvctl start database -d orclZ
```

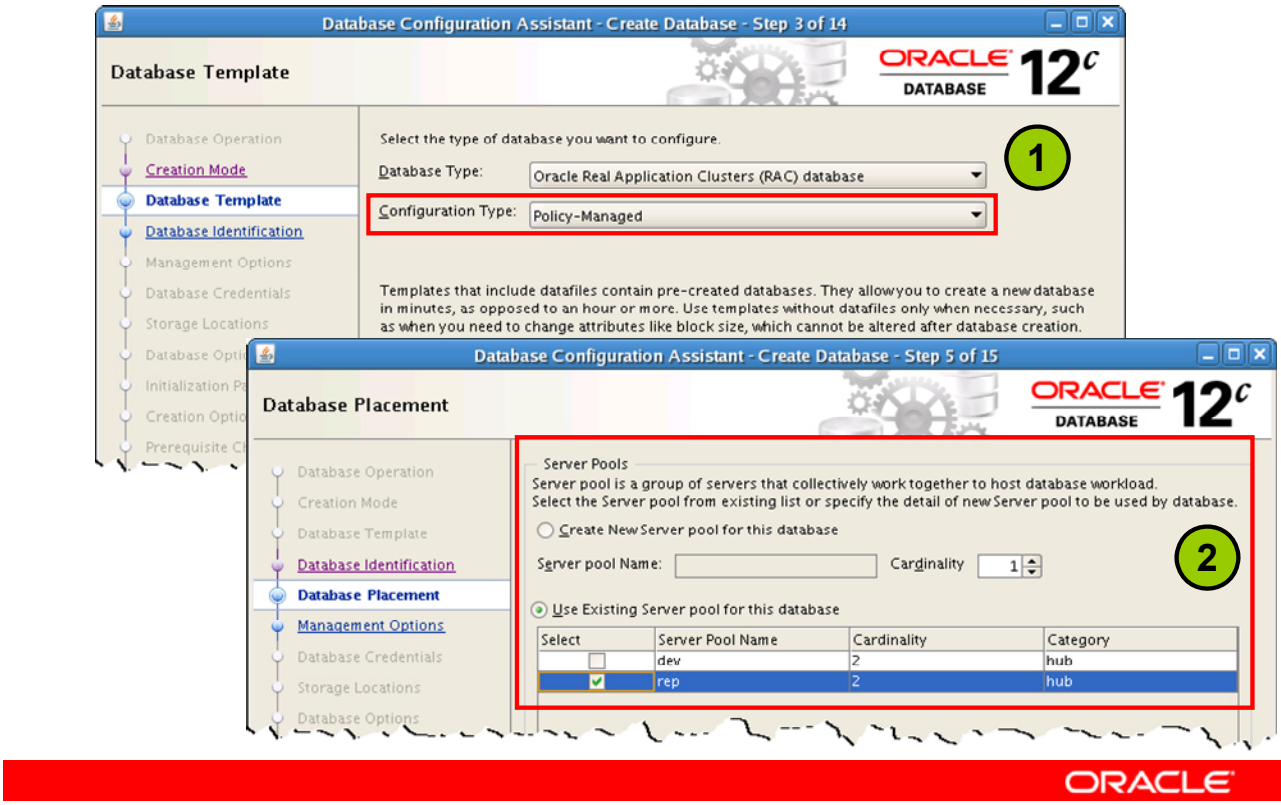
ORACLE

The slide outlines the basic procedure for converting an existing administrator-managed database to a policy-managed database. Two examples are also provided.

In the first example, an existing Oracle RAC database (`orclB`) is associated with a newly created server pool (`poolA`). After the database is restarted, each server in `poolA` will host an instance of `orclB` along with the default service for `orclB`.

The second example assumes the existence of an Oracle RAC database (`orclZ`) that supports two database services (`srvV` and `srvW`). The example adds two new server pools (`poolX` and `poolY`). At the end of the configuration, each service is associated with a separate server pool, and the database is associated with both of the server pools. After the database is restarted, all the servers in `poolX` and `poolY` will host instances of `orclZ`. However, `srvV` will be offered only on the servers in `poolX` and `srvW` will be offered only on the servers in `poolY`.

Creating a New Policy-Managed Database

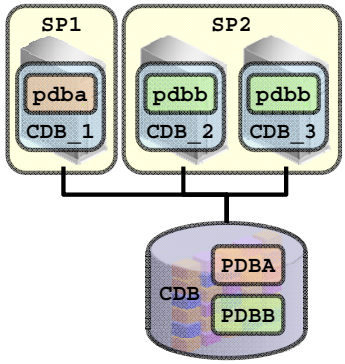


You can easily create a new policy-managed database by using the Database Configuration Assistant (DBCA). To do so, you must first select Policy-Managed from the Configuration Type drop-down list on the Database Template Page. Then, later on the Database Placement page, you must associate your new database with at least one server pool.

On the Database Placement page, you can specify a new server pool, or you can select from a list of existing server pools. It is generally recommended that you define the server pools before using DBCA because you can precisely set the server pool attributes to meet your requirements.

Policy-Managed Databases and Policy-Based Cluster Management with Oracle Multitenant

- Oracle Multitenant enables multiple PDBs to share the resources of a single CDB:
  - Enables more efficient consolidation
  - Maintains isolation between PDBs
- Using Oracle Multitenant with policy-managed databases and policy-based cluster management:
  - Each PDB can be exposed by using database services
  - Each PDB service can be associated with a server pool
  - The placement of PDB services can be controlled using policy-based cluster management



A multitenant container database (CDB) is an Oracle database that includes one or more pluggable databases (PDBs). A PDB is a portable collection of schemas, schema objects, and nonschema objects that appear to an application client as a self-contained database. The name of this capability is Oracle Multitenant.

First introduced in Oracle Database 12*c*, Oracle Multitenant enables multiple PDBs to share the resources of a single CDB instance while maintaining database-level isolation. This enables more efficient consolidation of databases while maintaining separate storage (data files) and security definitions (users, schemas, privileges, and so on).

In a policy-managed multitenant database, each PDB can be exposed using database services. Furthermore, the PDB services can be associated with server pools that are managed in association with policy-based cluster management.

The diagram in the slide shows an example where a CDB with two PDBs (`PDBA` and `PDBB`) is hosted on a three-node cluster that is organized into two server pools (`SP1` and `SP2`). In the example, `PDBA` is exposed on `SP1` and `PDBB` is exposed on `SP2`. However, you have a great deal of freedom to organize your server pools and services. You can even create multiple services for one PDB, which enables a PDB to span across multiple server pools.

For more information about Oracle Multitenant, refer to the Oracle Database 12*c* documentation library.

Quiz

Identify the correct statements regarding server categorization.

a. Server categorization provides a mechanism to control which servers can be in a server pool.
b. A server category must contain one or more servers.
c. A server can belong to only one server category at a time.
d. Servers can be categorized with user-defined expressions that combine various server attributes.

Answer: a, d

A server category can contain no servers if none meets the criteria for entry into the category. Servers can belong to multiple categories simultaneously; however, they can belong to only one server pool at a time.

Quiz

Identify the correct statements regarding policy-based cluster management.

a. Administrators can create multiple policy sets to cater for different workloads and priorities.
b. Administrators can create multiple policies to cater for different workloads and priorities.
c. The policy set automatically manages all server pools defined in the cluster.
d. The policy set automatically manages the server pools identified in the SERVER_POOL_NAMES attribute, and policy changes have no effect on other server pools.
e. The policy set automatically manages the server pools identified in the SERVER_POOL_NAMES attribute, and policy changes may indirectly affect other server pools.

ORACLE

Answer: b, e

There is always exactly one policy set defined in the cluster, and the policy set can contain zero or more policy definitions. The policy set automatically manages the server pools identified in the `SERVER_POOL_NAMES` attribute, and policy changes may indirectly affect other server pools.

Quiz

Fill in the blank: The `Current` policy _____ matches the policy definition specified in the `LAST_ACTIVATED_POLICY` policy set attribute.

a. sometimes
b. always
c. never

Answer: c

The `Current` policy never matches the policy definition specified in the `LAST_ACTIVATED_POLICY` policy set attribute because it includes the `Generic` built-in server pool, which is never under policy set control. In addition, the `Current` policy contains the current state of server pools not under policy set control. It is fair to say that the attributes of the `LAST_ACTIVATED_POLICY` may be reflected in the `Current` policy; however, those attributes may also vary over time because of changes made outside the policy set.

Quiz

When you create a policy-based RAC database, you specify which nodes the RAC database instances will run on.

a. True
b. False

Answer: b

Policy-based management introduces the concept of server pools, which are logical divisions of a cluster that are dynamically allocated based on relative importance. Database services are associated with server pools and RAC instances are automatically started to satisfy the service-to-server pool associations. You do not specify which nodes run the instances for a policy-managed RAC database.

Summary

In this lesson, you should have learned how to:

- Describe the purpose of policy-based cluster management
- Describe the architecture and components of policy-based cluster management
- Implement a policy set and enforce a policy
- Describe the purpose and architecture of policy-managed database
- Describe how policy-based cluster management interacts with policy-managed database
- Convert an existing administrator-managed database into a policy-managed database
- Create a new policy-managed database
- Describe how the Oracle Multitenant architecture works with policy-based cluster management and policy-managed databases
- Expose PDBs as services and integrate them with policy-based management

Practice 4 Overview: Using Policy-Based Cluster Management with Oracle RAC

In this practice, you will use policy-based cluster management capabilities to manage cluster resources in conjunction with Oracle RAC. You will:

- Convert an existing administrator-managed RAC database to a policy-managed database
- Define a cluster policy set and examine how cluster policies can be used to share cluster resources between multiple Oracle RAC databases
- Create a container database and examine how policy-based management can be used to control access to pluggable database services

5

Flex Clusters

Objectives

After completing this lesson, you should be able to:

- Describe the purpose of Flex Clusters
- Describe the architecture and components of Flex Clusters
- Install and configure a Flex Cluster
- Describe the effect of node failure in a Flex Cluster
- Convert an existing standard cluster into a Flex Cluster
- Add Leaf Nodes to a Flex Cluster
- Configure and manage highly available application resources on Flex Cluster Leaf Nodes

ORACLE

Flex Clusters: Overview

- In previous releases, most large clusters contain between 32 and 64 nodes.
- With Oracle Clusterware 12*c*, Flex Clusters are designed to scale up to 2000 nodes. Use cases include:
  - Large pools of highly available application resources
  - Multiple databases and applications running in one cluster

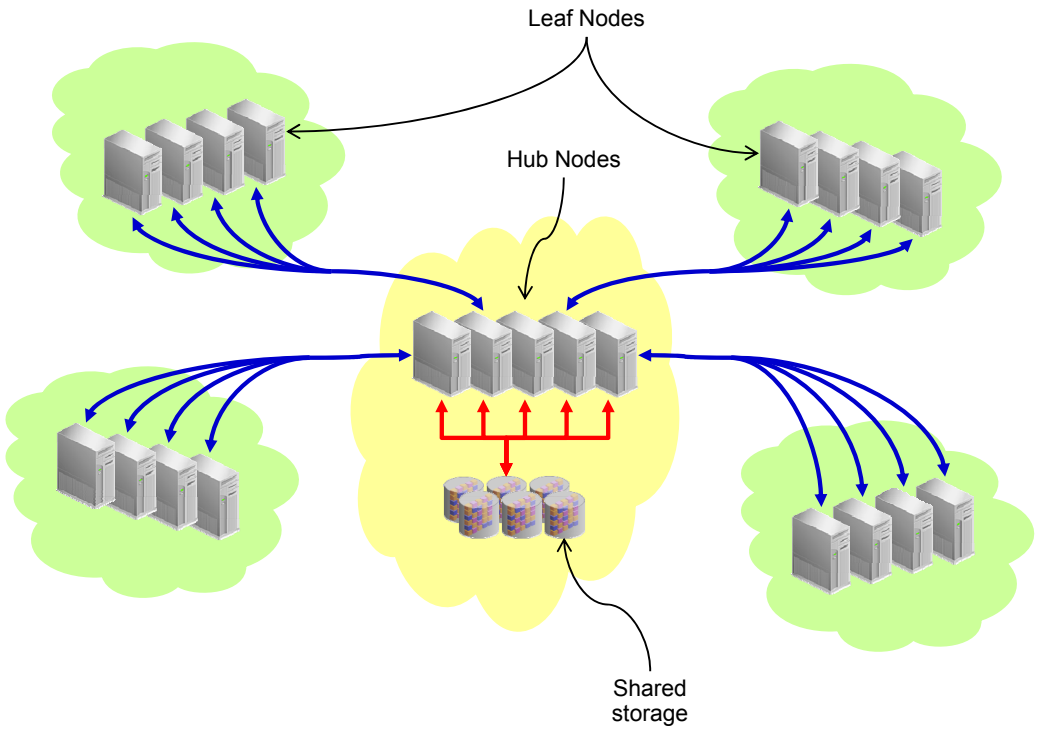
Previous releases of Oracle Clusterware have been used to build large production clusters containing between 32 and 64 nodes. A few clusters larger than 100 nodes have been successfully deployed.

With Oracle Clusterware 12*c*, a new set of features enables Flex Clusters. In this release, Flex Clusters are designed to scale up to 2000 nodes.

In release 12.1, you can use Flex Clusters to:

- Manage large pools of application resources with high-availability and failover protection
- Efficiently support multiple highly available databases and applications running in a single cluster

Flex Cluster Architecture



Flex Clusters use a hub-and-spoke topology, as illustrated in the slide.

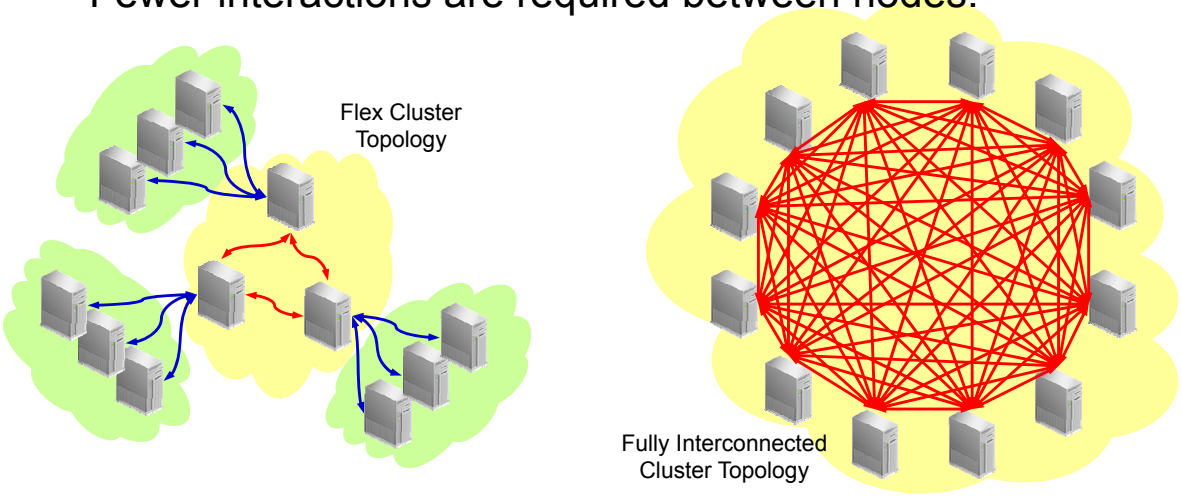
The core of a Flex Cluster is a group of Hub Nodes. The group is essentially the same as a release 11.2 cluster, and can scale up to the size of an existing release 11.2 cluster. There must be only one group of Hub Nodes in a Flex Cluster deployment, and like a release 11.2 cluster, each Hub Node must be connected to storage that is shared across the group of Hub Nodes.

Zero or more Leaf Nodes may be connected to a Flex Cluster. Each Leaf Node is connected to the cluster through a Hub Node. Leaf Nodes do not require direct access to the shared storage connected to the Hub Nodes.

Flex Cluster Scalability

The Flex Cluster hub-and-spoke topology segments the cluster into more manageable groups of nodes.

- Only the Hub Nodes require direct access to the OCR and voting disks.
- Fewer interactions are required between nodes.



The two-layered hub-and-spoke topology is the key architectural feature that allows a Flex Cluster to scale well beyond previous limits. In essence, the hub-and-spoke topology segments the cluster into groups of nodes, and each group contains a manageable number of nodes. This segmentation has two fundamental impacts:

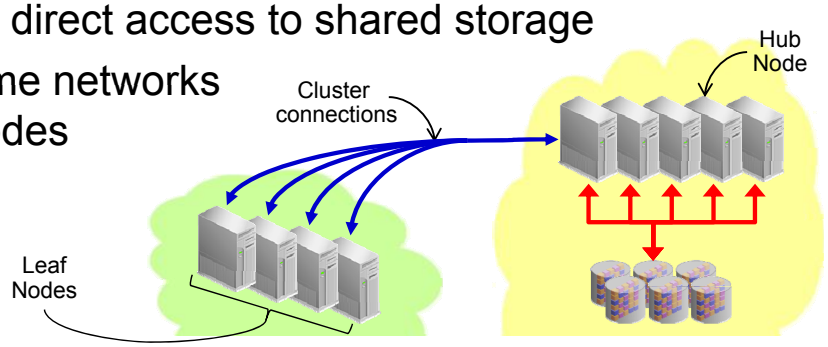
First, by limiting the size of the hub, contention for key Clusterware resources, such as the Oracle Cluster Registry (OCR) and voting disks, does not increase significantly due to the addition of the Leaf Nodes. This is important because contention for the voting disks can lead to nodes being evicted from a cluster.

Second, fewer network interactions are required between nodes in the cluster. Consequently, there is less administrative network traffic, such as heartbeats, exchanged between the nodes. This is illustrated in the diagram in the slide. On the left side, the 12-node Flex Cluster contains 12 interaction paths. On the right side, the fully interconnected 12-node cluster contains 66 possible interaction paths. For a 1000-node cluster, the difference would be far more noticeable. Assuming 40 Hub Nodes, with 24 Leaf Nodes per Hub Node, a Flex Cluster contains 1740 possible interaction paths. In comparison, a 1000-node fully interconnected cluster contains 499500 interaction paths.

Leaf Node Characteristics

Leaf Nodes:

- Are more loosely coupled to the cluster than Hub Nodes
- Automatically discover the Hub Nodes at startup
- Connect to the cluster through a Hub Node
  - Failure of the Hub Node or network failure results in eviction of associated Leaf Nodes.
  - Functioning Leaf Nodes can be brought back into the cluster.
- Do not require direct access to shared storage
- Are on the same networks as the Hub Nodes



In comparison to Hub Nodes, Leaf Nodes are loosely coupled to the cluster. When Oracle Clusterware is started on a Leaf Node, the Leaf Node automatically discovers the Hub Nodes and is associated with a single Hub Node. For cluster membership purposes, Hub Nodes periodically exchange heartbeat messages with their associated Leaf Nodes. Similar mechanisms are used for other services.

If a Hub Node fails, or if the network link between a Hub Node and a Leaf Node fails, the associated Leaf Nodes may be removed from the cluster. In any case, if there is no fault with the Leaf Node, it can be brought back into the cluster by restarting Oracle Clusterware on the Leaf Node.

A Leaf Node does not require direct access to shared storage. This means that Leaf Nodes can participate in the cluster without storage-related hardware and network connections, such as fiber-channel network connections and host bus adapters.

In release 12.1, all Leaf Nodes are on the same public and private networks as the Hub Nodes.

Grid Naming Service and Flex Clusters

Clients on Leaf Nodes use GNS to locate Hub Node services.

- The GNS server location is stored in the cluster profile.
- Leaf Node services issue DNS queries to GNS.
  - Particularly during Leaf Node startup
- A fixed GNS VIP is required on one of the Hub Nodes.
  - So Leaf Node clients have a reliable, well-known location to contact.
- DNS forwarding is not required for Flex Clusters.
  - But it can be implemented to better integrate GNS with DNS.

Flex Clusters require the Grid Naming Service (GNS) to be configured with a fixed virtual IP (VIP) on one of the Hub Nodes.

GNS is used to dynamically register and resolve names within the cluster. In particular, GNS is referenced as the Leaf Node is associated with a Hub Node for cluster membership purposes during the Clusterware startup process on a Leaf Node. This requires access to GNS through a fixed VIP running on one of the Hub Nodes, so that Leaf Nodes have a reliable, well-known naming service within the cluster.

Domain name server (DNS) forwarding is not required to facilitate discovery of Clusterware services by Leaf Nodes; however, it can still be configured to integrate GNS with a wider network-naming service.

Cluster Mode: Overview

- Oracle Clusterware 12*c* introduces a new cluster mode setting to enable Flex Cluster functionality.
  - Users must explicitly enable Flex Cluster functionality.
- The default cluster mode setting disables Flex Cluster functionality.
  - Users who do not implement Flex Clusters are not exposed to the new code.
  - Performance and stability of standard clusters is not impacted by Flex Cluster functionality.

ORACLE

You can choose to enable or disable the Flex Cluster functionality by using the new cluster mode setting. By default, the Flex Cluster functionality is disabled.

Configuring the Cluster Mode

- Showing the current cluster mode:

```
$ crsctl get cluster mode status
```

- Converting from a standard cluster to a Flex Cluster:
  1. Ensure that GNS is configured with a fixed VIP:

```
# srvctl add gns -i <Fixed GNS VIP address> -d <cluster domain>
```

  2. Enable Flex ASM in the cluster using ASMCA
  3. Set the cluster mode:

```
# crsctl set cluster mode flex
```

  4. Stop Oracle Clusterware on each node:

```
# crsctl stop crs
```

  5. Start Oracle Clusterware on each node:

```
# crsctl start crs
```

ORACLE

At any time, a cluster administrator or system administrator can check the current cluster mode by using the `crsctl get cluster mode status` command.

To convert from a standard cluster to a Flex Cluster, an administrator should first ensure that GNS is configured with a fixed VIP. If GNS is not configured with a fixed VIP, the remainder of the procedure will fail. Also, Flex ASM must be enabled on the cluster prior to setting the cluster mode. Next, the system administrator (`root`) can set the cluster mode by using the `crsctl set cluster mode flex` command. Finally, the system administrator must restart the cluster by using the `crsctl stop crs` command on each node in the cluster followed by the `crsctl start crs` command on each node in the cluster. Note that you cannot avoid cluster down time when changing the cluster mode.

To convert from a Flex Cluster to a standard cluster, the system administrator must set the cluster mode by using the `crsctl set cluster mode standard` command and Clusterware must be stopped and restarted across the cluster. There is no requirement to reconfigure GNS or Flex ASM, because the configuration required for Flex Cluster mode is also compatible with standard cluster mode.

Note that any node which is unable to join the reconfigured cluster is left out of the cluster and eventually dropped from it. This can occur when, for example, a Leaf Node having no access to shared storage cannot join a cluster converted to standard mode.

Configuring the Node Role

- Showing the current node role:

```
$ crsctl get node role status -node <hostname>
```

```
$ crsctl get node role status -node c00n02
Node 'c00n02' active role is 'hub'
```

- Setting the node role:

```
# crsctl set node role { hub | leaf | auto } -node <hostname>
```

```
# crsctl set node role leaf -node c00n02
# crsctl get node role config -node c00n02
Node 'c00n02' configured role is 'leaf'
# crsctl get node role status -node c00n02
Node 'c00n02' active role is 'hub'
```

ORACLE

One of the key configuration tasks for a Flex Cluster is specifying which nodes are Hub Nodes and which are Leaf Nodes.

At any time, a cluster administrator or system administrator can check the current role for a node by using the following command:

```
crsctl get node role status -node <hostname>
```

Configuring the node role can be achieved in two ways:

- A system administrator can explicitly specify the node role as `hub` for a Hub Node, or `leaf` for a Leaf Node, by using the `crsctl set node role` command. Explicitly setting the node role ensures the node type. The example in the slide shows the node `c00n02` being configured as a Leaf Node. Note that node role changes do not take effect until the next time that Oracle Clusterware is started on the node. This is evident in the example in the slide, where the configured node role is `leaf` but the active node role is still `hub` immediately after the node role change.
- A system administrator can also set the node role to `auto` by using the `crsctl set node role` command. This setting allows the cluster to decide which role a node will perform based on the composition of the cluster. The cluster administrator must ensure that the node can fulfill either role, `hub` or `leaf`, in order to use the `auto` setting.

Configuring the Hub Size

- Showing the current hub size:

```
$ crsctl get cluster hubsize
CRS-4950: Current hubsize parameter value is 32
```

- Setting the hub size:

```
# crsctl set cluster hubsize <number>
```

```
# crsctl set cluster hubsize 16
# crsctl get cluster hubsize
CRS-4950: Current hubsize parameter value is 16
```


The `auto` node role setting works hand in hand with the cluster `hubsize` setting. When a node with the `auto` node role setting attempts to join the cluster, Oracle Clusterware examines the `hubsize` setting. If the number of Hub Nodes is smaller than the `hubsize` setting, the node joins the cluster as a Hub Node. Otherwise, the node joins the cluster as a Leaf Node.

The examples in the slide show how administrators can examine and set the cluster hub size. Note that setting the cluster hub size requires system administrator privileges. Also note that the hub size setting is effective immediately, but it does not impact the node role of any nodes already in the cluster.

Configuring Miss Count for Leaf Nodes

- Viewing and setting `leafmisscount`:

```
# crsctl get css leafmisscount
CRS-4678: Successful get leafmisscount 30 for Cluster Synchronization
Services.
# crsctl set css leafmisscount 45
CRS-4684: Successful set of parameter leafmisscount to 45 for Cluster
Synchronization Services.
# crsctl get css leafmisscount
CRS-4678: Successful get leafmisscount 45 for Cluster Synchronization
Services.
```

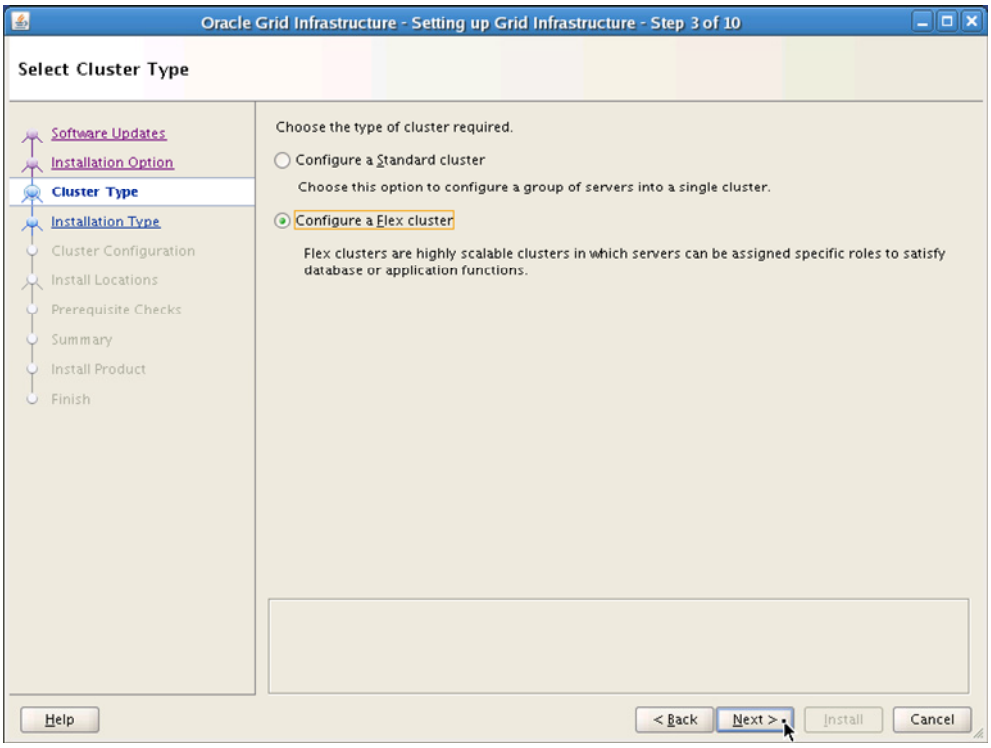

The `leafmisscount` attribute determines how Cluster Synchronization Services (CSS) handles network connectivity issues between a Leaf Node and the Hub Node that connects it to the cluster.

The `leafmisscount` setting defines the threshold duration (in seconds) for tolerable communication failures. If the communication between a Hub Node and associated Leaf Node is interrupted and restored before the amount of time specified by `leafmisscount`, then the cluster continues to operate normally. If communication is lost for a period exceeding the `leafmisscount` setting, then the interruption is assumed to be significant and the Leaf Node is evicted from the cluster. The default `leafmisscount` setting is 30 seconds.

The examples in the slide show how to query and set the `leafmisscount` attribute.

Note that the `leafmisscount` attribute is separate from the `misscount` attribute, which existed in previous releases and continues to exist in release 12.1.

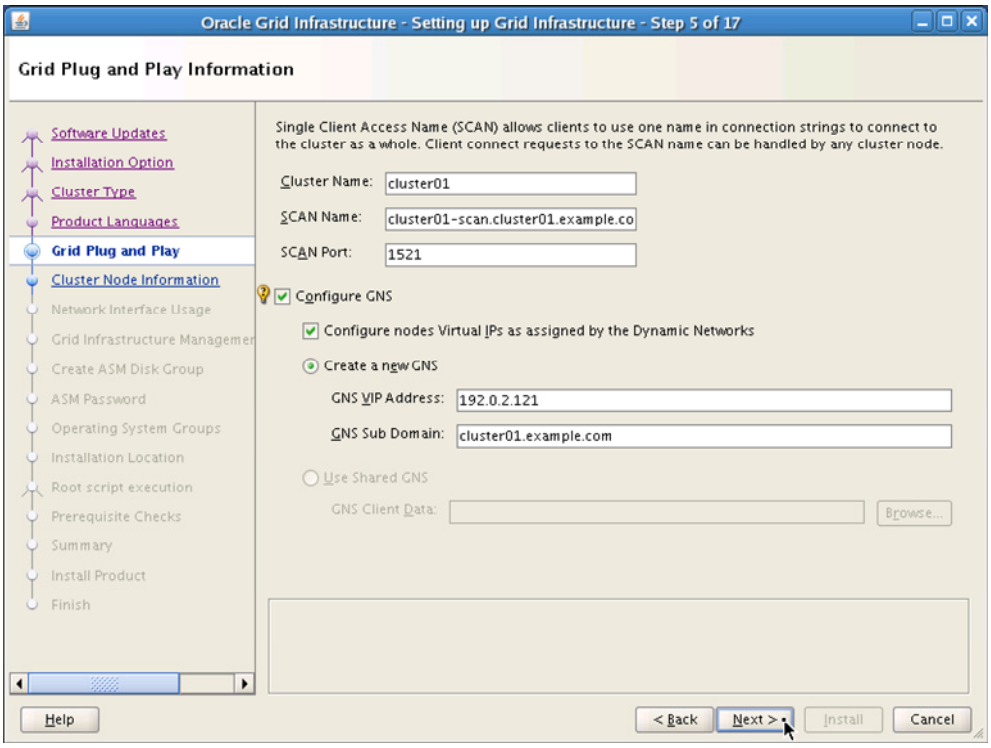
Configuring a Flex Cluster with OUI: Selecting the Cluster Type



The previous four pages introduced the commands required to configure a Flex Cluster and convert a standard cluster to a Flex Cluster. For new clusters, Oracle Universal Installer (OUI) has been updated to facilitate the configuration of Flex Clusters.

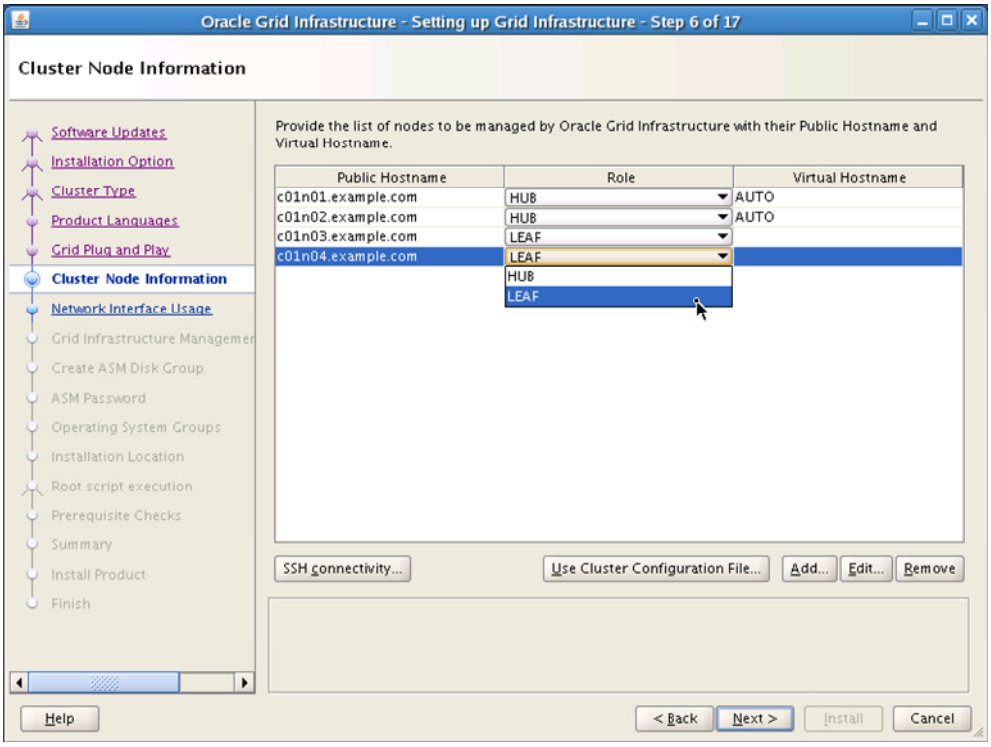
The screenshot in the slide shows the OUI interface for the step where administrators select the cluster type. To configure a Flex Cluster, administrators must select the “Configure a Flex cluster” option.

Configuring a Flex Cluster with OUI: Configuring GNS



Flex Clusters require the Grid Naming Service (GNS) to be configured with a fixed virtual IP (VIP) on one of the Hub Nodes. If you selected the option to configure a Flex Cluster, OUI will not progress before GNS is configured on the Grid Plug and Play Information screen. The screenshot in the slide shows an example of the required configuration.

Configuring a Flex Cluster with OUI: Selecting the Node Type



When you select the option to configure a Flex Cluster, the Cluster Node Information page looks like the example shown in this slide. You can use this interface to specify the cluster nodes and you can set the node type to `HUB` or `LEAF` for each cluster node.

To configure the cluster by using the `auto` node role setting in conjunction with the cluster `hubsize` setting, administrators must wait until after the cluster is initially configured with OUI. Then, they can use the commands introduced earlier in this lesson to adjust the node role and cluster `hubsize` settings.

Flex Clusters and Node Failure

- Nodes that are evicted from the cluster do not require a restart; only a cluster software restart.
- If a Hub Node fails:
  - The node is evicted from the cluster.
    - Services are relocated to other Hub Nodes if possible.
  - Corresponding Leaf Nodes are also evicted from the cluster.
    - Services are relocated to other Leaf Nodes if possible.
- If a Leaf Node fails:
  - The node is evicted from the cluster.
    - Services are relocated to another Leaf Node where possible.
  - The impact of the failure is contained where possible.

ORACLE

In previous releases of Oracle Clusterware, any node that was evicted from the cluster would panic the operating system kernel and cause an immediate shutdown of the node. This measure is very effective at protecting the cluster from the effects of a rogue node. However, restarting the node results in the cluster working with diminished capacity for many minutes. With Oracle Clusterware 12*c*, node eviction does not require a node restart. Rather, where possible, only the cluster software is restarted, which significantly reduces the amount of time that the cluster is affected.

If a Hub Node fails, the node is evicted from the cluster in essentially the same way as any node in a standard cluster. As part of dealing with the node failure, the cluster attempts to start as many services as possible on surviving cluster nodes. Leaf Nodes that were associated with the failed Hub Node are also evicted from the cluster.

If a Leaf Node fails, the node is evicted from the cluster and the cluster attempts to relocate services running on the Leaf Node to other Leaf Nodes connected to the same Hub Node. This means that the effect of a Leaf Node failure is usually contained within the group of Leaf Nodes connected to the same Hub Node. Thus, the performance and availability of the rest of the cluster is not affected by the failure of a Leaf Node.

Quiz

Identify the use cases supported by Flex Clusters:

a. Large-scale decision support databases where parallel queries operations can be spread across database instances running on the Leaf Nodes
b. Large-scale online transaction processing (OLTP) databases where many thousands of user connections can be spread across database instances running on the Leaf Nodes
c. Mixed environments where databases run on the Hub Nodes and highly available application resources run on the Leaf Nodes

Answer: c

In release 12.1, Oracle Database instances are supported only on Hub Nodes. Leaf Nodes can be used to host application services, which can leverage the high-availability framework of Oracle Clusterware.

Quiz

Flex Clusters achieve greater scalability than standard clusters because:

a. Fewer physical network connections are required between the cluster nodes
b. Leaf Nodes do not require direct access to shared storage
c. By limiting the size of the hub, contention for key Clusterware resources is controlled
d. Fewer network interactions are required between the cluster nodes to maintain the cluster
e. The cluster hub size can be set to a larger value than in previous versions

ORACLE

Answer: c, d

The two-layered hub-and-spoke topology of a Flex Cluster achieves greater scalability because of two fundamental reasons:

1. By limiting the size of the hub, contention for key clusterware resources, such as the OCR and voting disks, does not increase significantly because of the addition of the Leaf Nodes.
2. Fewer network interactions are required between nodes in the cluster, and consequently, there is less administrative network traffic, such as heartbeats, exchanged between the nodes.

Answer A is not correct because the number of physical network connections is the same for both cluster types.

Answers B is a correct statement however this fact does not improve cluster scalability by itself.

Answer E is not correct because the cluster hub size setting does not improve cluster scalability by itself.

Summary

In this lesson, you should have learned how to:

- Describe the architecture, components, and purpose of Flex Cluster
- Install and configure a Flex Cluster
- Describe the effect of node failure in a Flex Cluster
- Convert an existing standard cluster into a Flex Cluster
- Add Leaf Nodes to a Flex Cluster
- Configure and manage highly-available application resources on Flex Cluster Leaf Nodes

ORACLE

Practice 5 Overview: Configuring and Using a Flex Cluster

In this practice, you will:

- Convert an existing 2-node standard cluster to a Flex Cluster
- Add two leaf nodes to your Flex Cluster
- Create a series of highly-available application resources running on one of the Flex Cluster Leaf Nodes

6

Oracle Database In-Memory

Objectives

After completing this lesson, you should be able to:

- Describe the benefits and architecture of Oracle Database In-Memory
- Enable Oracle Database In-Memory in a database
- Configure database objects to use Oracle Database In-Memory
- Describe how Oracle Database In-Memory performs optimized query processing
- Examine meta-data and statistics relating to Oracle Database In-Memory
- Describe how Oracle Database In-Memory works in conjunction with Oracle RAC

ORACLE

Introducing Oracle Database In-Memory

- Oracle Database In-Memory is a query-optimized in-memory data store:
  - Particularly useful for analytical applications
  - Does not replace existing Oracle Database capabilities
- Key benefits:
  - Instant query responses without indexes using scans, joins, filters, and aggregates on any column
  - Full application transparency: ORACLE E-BUSINESS SUITE, ORACLE FUSION APPLICATIONS, ORACLE SIEBEL, ORACLE JD EDWARDS, ORACLE PEOPLESOFT
  - Space saving and faster DML because of fewer indexes
  - Easy setup and management

ORACLE

Oracle Database In-Memory enables tables, partitions, and column data to be stored in-memory in a columnar format, which is optimized for query operations. This enables scans, joins, and aggregates to perform much faster than the traditional on-disk format, resulting in fast reporting and DML performance for both OLTP and DW environments. This is particularly useful for analytic applications, which operate on a few columns but return many rows, as opposed to OLTP operations, which process many columns but return a few rows.

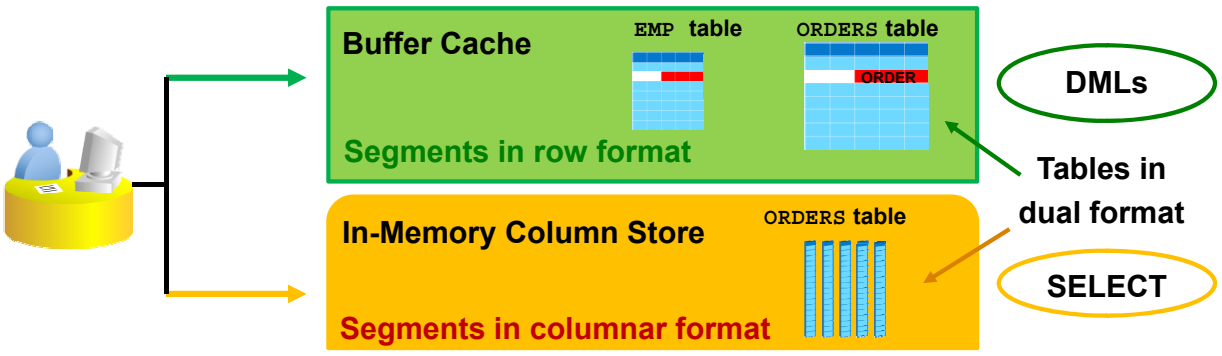
The in-memory columnar format does not replace the on-disk or buffer cache format. Rather, it is a consistent copy of the data that is independent of the disk format and only available in memory. To accommodate data in the new columnar format, a new memory pool can be configured. The new pool is known as the In-Memory Column Store (IMCS).

Oracle Database In-Memory offers four main benefits:

- Queries can run more than 100 times faster thanks to the compressed columnar format.
- Because the IMCS does not replace existing Oracle Database functionality, applications are able to transparently use Oracle Database In-Memory without any changes.
- Oracle Database In-Memory removes the need for analytical indexes, which results in space saving and faster DML processing.
- Oracle Database In-Memory is easy to configure and manage.

In-Memory Column Store

- The In-Memory Column Store (IMCS) is a new SGA pool.
- It contains a copy of selected data segments.
- Data in the IMCS is in compressed columnar format.
- IMCS data is consistent with the buffer cache.
- There are no additional disk writes.
- There is no additional segment storage.



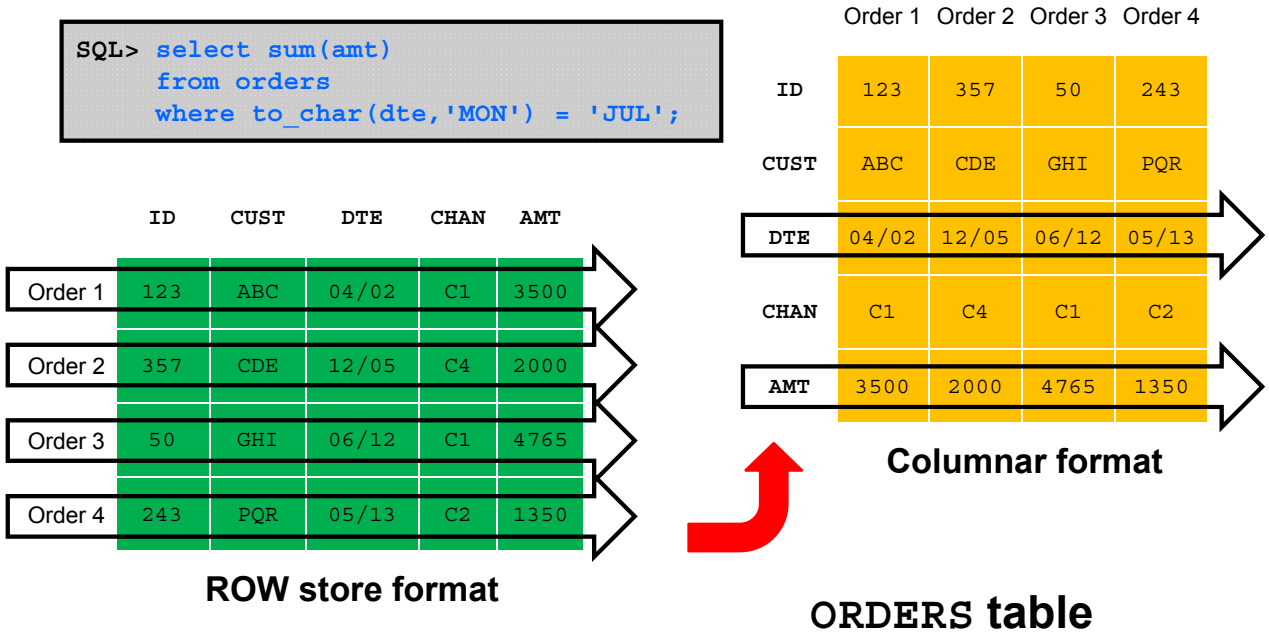
The in-memory columnar format does not replace the on-disk or buffer cache format. This means that when a segment, such as a table or a partition, is populated into the In-Memory Column Store (IMCS), the data is automatically converted into a columnar format and optionally compressed. The columnar format is a pure in-memory format. It never causes additional writes to disk, and therefore, does not require any logging or undo space.

Note that the IMCS contains a copy of the data from selected segments and that all data is still persisted to disk. Because of this, there is no special crash recovery for the IMCS; it is simply repopulated after the instance restarts.

Moreover, data in the IMCS is a transactionally-consistent copy of the data on disk or in the buffer cache. Transaction consistency between the two data copies is maintained automatically.

Because queries against the IMCS perform much faster, the improved performance allows more ad-hoc queries to be executed directly on the real-time transaction data without impacting the existing workload.

Row Store Versus Columnar Format

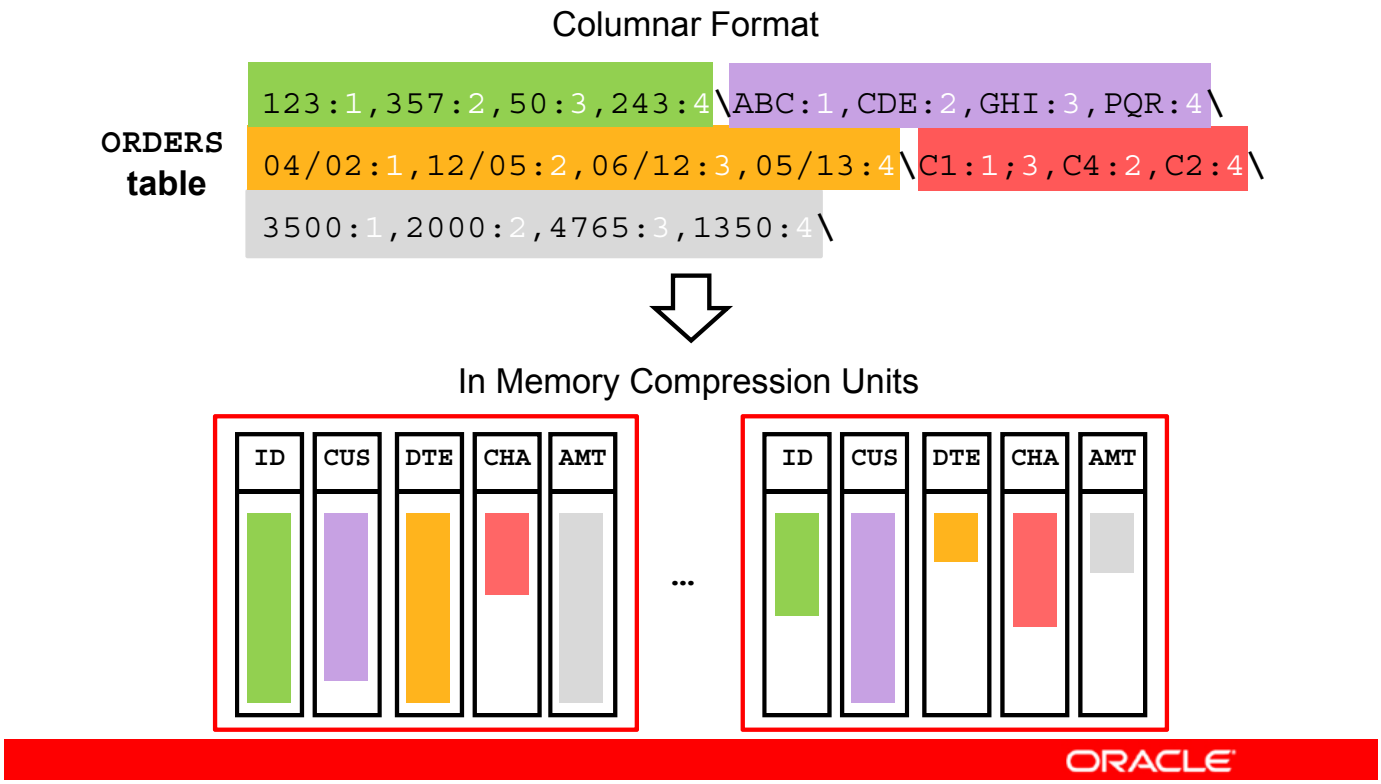


Relational databases organize data into tables. Each table consists of a series of rows and each row contains attribute data in a series of columns. For example, the left side of the slide shows a table that contains sales order data (ORDERS). Each row represents a single order and contains an order identifier (ID), customer identifier (CUST), order date (DTE), channel identifier (CHAN), and order value (AMT).

Traditionally, table data is stored in a row format where the columns of each row are stored together. This representation is well suited to manipulating individual rows. However, it is not really convenient for searches on a small number of attributes. For example, to calculate the value of all orders in July, the query must read every row to find the right date information and order value. Note that each time the query reads a row, it reads all of the columns even though only two columns are required to satisfy the query. Indexes can help the situation; however, they incur significant processing and space costs.

Columnar format groups together the data in each column. An example, based on the ORDERS table, is shown on the right side of the slide. This representation is not well suited to manipulating individual rows, but it is very well suited to analytical query operations. For example, to calculate the value of all orders in July, the query needs to scan only the DTE column to locate the July orders, and the AMT column to locate the corresponding order values. Because the query scans only two columns, it requires much less IO. Reading only the required columns and ignoring the rest is known as *columnar projection*.

In-Memory Compression Unit

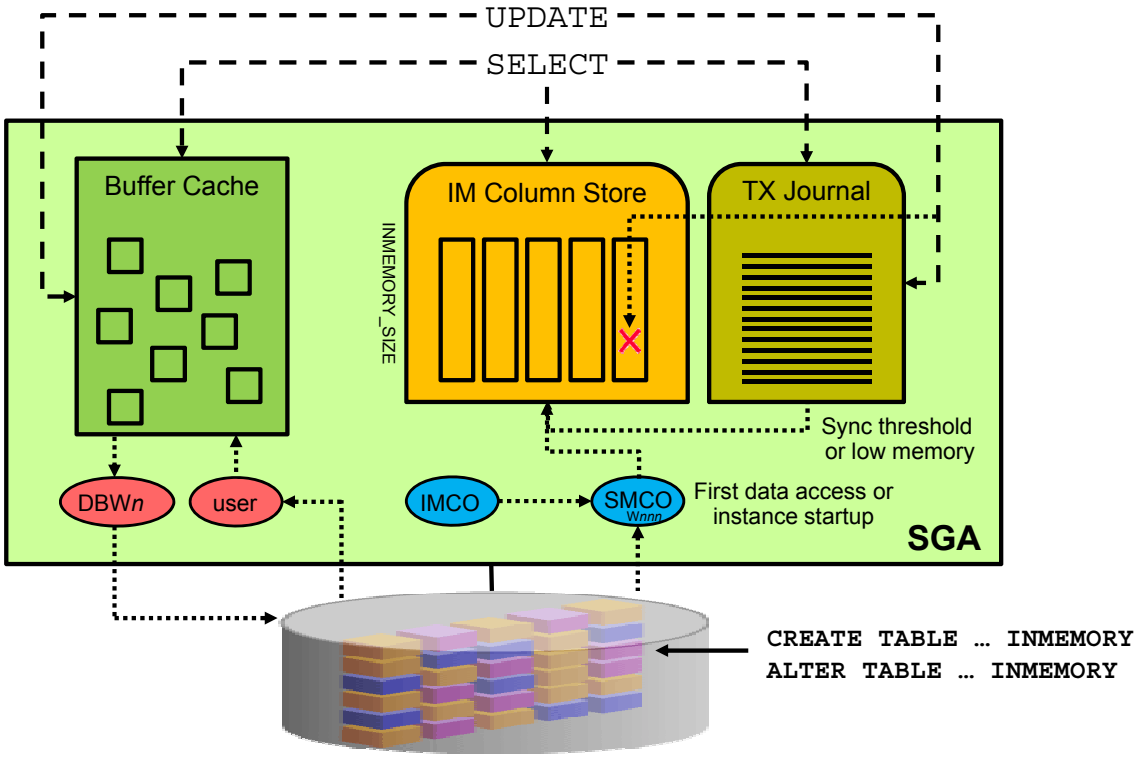


When data is populated into the In-Memory Column Store, it is divided into chunks known as In-Memory Compression Units (IMCUs). Unlike conventional database blocks, which are typically 8 KB in size, IMCUs are 1 MB in size.

Within the IMCU, each column is stored separately in a contiguous region of memory. Data compression is also applied at the IMCU level; that is, every IMCU is self-contained and there is no need to reference another IMCU for compression and decompression.

Each IMCU contains data from a complete number of rows; that is, the data for a row is contained within only one IMCU. The exact number of rows that are populated in each IMCU is determined at runtime and is based on numerous factors including table size, table structure, compression level, and memory constraints.

IMCS Architecture: Overview



There are new background processes that create and refresh IMCUs to populate and repopulate the In-Memory Column Store. `IMCO` is the background coordinator process. It wakes up approximately every two minutes to schedule data population. `SMCO` and `Wnnn` are the background worker processes that actually perform population operations.

When rows in the table are updated, the corresponding entries in the IMCUs are marked as stale. The updated row version is recorded in a transaction journal. IMCU synchronization is performed by the in-memory background processes by using the updated rows populated in the transaction journal. IMCU synchronization is automatically triggered by various events including:

- The number of invalidations for an IMCU reaching an internal threshold
- The transaction journal reaching a low memory threshold

Enabling Oracle Database In-Memory

- Enable the In-Memory Column Store:
  - Set `INMEMORY_SIZE` as desired.
  - Minimum allowed is 100 MB.
- Other considerations:
  - Verify the database compatibility value.
    - Set `COMPATIBLE = 12.1.0.0.0` or higher.
  - Enable IMCS population.
    - Ensure `INMEMORY_MAX_POPULATE_SERVERS > 0`
    - Default is based on system characteristics.
  - Enable in-memory parallel execution.
    - Set `PARALLEL_DEGREE_POLICY = AUTO`
    - It is important in Oracle RAC environments.

Oracle Database In-Memory is shipped in the installation package for Oracle Database 12*c* version 12.1.0.2 and no special installation procedure is required. By default, Oracle Database In-Memory is not enabled. To enable Oracle Database In-Memory, you must configure the database instance parameters, including the following:

- `INMEMORY_SIZE` sets the size of the IMCS on a database instance. A nonzero value is required to enable Oracle Database In-Memory. The database must be restarted after setting this parameter to enable the IMCS. The minimum size allowed for this parameter is 100 MB. This pool, like the log buffer, KEEP, RECYCLE, and other block size pools is a static pool that is not affected by Automatic Shared Memory Management (ASMM). The memory allocated to this pool is fixed and deducted from the total available memory for `SGA_TARGET` when ASMM is enabled. This parameter can also be set for each pluggable database (PDB) to limit the maximum memory used by each PDB. Note that the sum of the per-PDB values does not necessarily have to be to equal the container database (CDB) value, and it may even be greater.
- `COMPATIBILITY` must be set to 12.1.0 or higher.

- `INMEMORY_MAX_POPULATE_SERVERS` specifies the maximum number of background populate servers to use for IMCS population. A nonzero value is required to enable IMCS population. Care should be taken to ensure that this parameter is not set so high that population tasks monopolize the system CPU resources.
- `PARALLEL_DEGREE_POLICY` must be set to `AUTO` to enable in-memory parallel execution. This is particularly important in Oracle RAC environments in order to perform a single query that can reference the IMCS on multiple database instances.

Configuring IMCS Candidate Objects

- Object configuration examples:
  - Deferred population:

```
SQL> CREATE TABLE t1 (...) INMEMORY;
```

```
SQL> ALTER TABLE t2 INMEMORY PRIORITY NONE;
```

```
SQL> CREATE TABLE t3 ... PARTITION ...
     (PARTITION p1 ... INMEMORY, PARTITION p2 ... NO INMEMORY);
```

  - Automatic population:

```
SQL> CREATE TABLE t4 (...) INMEMORY PRIORITY CRITICAL;
```

  - Disable an object:

```
SQL> CREATE TABLE t5 (...) NO INMEMORY;
```

```
SQL> ALTER TABLE t1 NO INMEMORY;
```


You must specify the `INMEMORY` clause to configure an object as a candidate for population in the IMCS. The clause can be specified during object creation or later as an alteration. By default, all columns in the candidate object are populated in the IMCS.

If you do not specify the `PRIORITY` subclause or if you specify `PRIORITY NONE`, the object will be subject to deferred population. With deferred population, the object is populated in the IMCS after it is first accessed.

By using different priority levels, an object can be automatically populated into the IMCS. The different `PRIORITY` values are covered later in the lesson.

Note that for a partitioned table, it is possible to configure a subset of partitions as in-memory candidates while specifically excluding others.

IMCS Supported and Unsupported Data

Supported data objects: Tables, partitions, subpartitions, inline LOBs, materialized views, materialized join views, and materialized view logs

- Unsupported data objects:
  - Clustered tables, index-organized tables
  - Tables owned by `SYS` and in the `SYSTEM` or `SYSAUX` tablespaces
- Unsupported data types:
  - LONG or LONG RAW columns
  - out-of-line columns (LOBs, varrays, nested table columns)
  - Extended data type columns

ORACLE

The objects that are compatible with the `INMEMORY` clause are: tables, partitions, subpartitions, inline LOBs, materialized views, materialized join views, and materialized view logs.

Clustered tables and index-organized tables (IOTs) are not supported with the `INMEMORY` clause. Also, you cannot specify the `INMEMORY` clause for tables that are owned by the `SYS` schema and reside in the `SYSTEM` or `SYSAUX` tablespace.

Finally, the IMCS does not support `LONG` or `LONG RAW` columns, out-of-line columns (LOBs, varrays, nested table columns), or extended data type columns. If you specify the `INMEMORY` clause for a table with unsupported columns, then only the supported columns will be populated and the unsupported columns will not be populated in the IMCS.

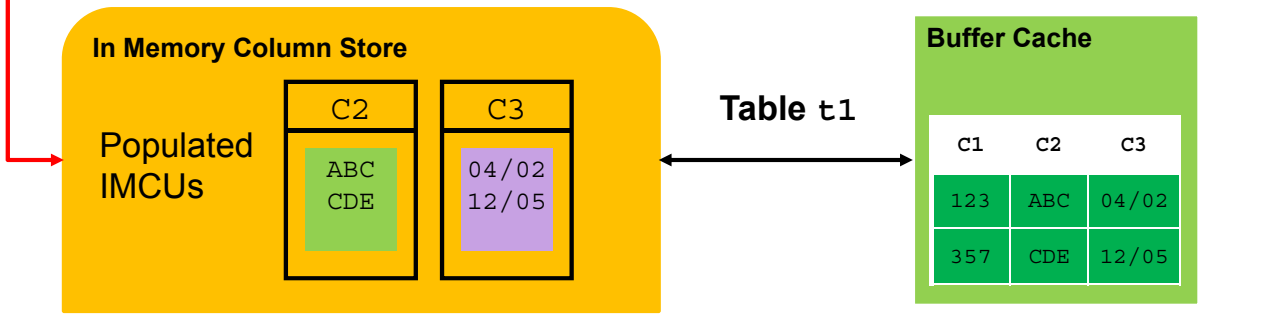
Configuring IMCS Candidate Objects: Column Subsets

```
SQL> CREATE TABLE t1(c1 NUMBER, c2 CHAR(2), c3 DATE)
     INMEMORY NO INMEMORY (c1);

SQL> ALTER TABLE t2 NO INMEMORY (c8, c9);

SQL> CREATE TABLE t3 (c1 NUMBER, c2 CHAR(2)) NO INMEMORY INMEMORY (c2);

ORA-64361: column INMEMORY clause may only be specified for an inmemory
           table
```



You can configure objects so that only a subset of columns are populated in the IMCS. Some examples are shown in the slide. In the first example, table `t1` is specified as an in-memory candidate; however, column `c1` is specifically excluded. The result is that only columns `c2` and `c3` may be populated in the IMCS. This is also illustrated in the diagram at the bottom of the slide.

The second example assumes that table `t2` is already in memory. In this example, columns `c8` and `c9` are explicitly removed as IMCS candidates.

Note that you cannot specify a table as `NO INMEMORY` and also populate a subset of columns into the IM column store. This is shown in the third example.

Defining the Population Priority

Use **PRIORITY** to define the population priority level:

```
SQL> CREATE TABLE ... INMEMORY
        PRIORITY [ NONE | LOW | MEDIUM | HIGH | CRITICAL ];
```

- PRIORITY NONE:
  - Is the default setting
  - Disables automatic population
- Other priorities:
  - Enable automatic population
  - Apply to the population queue
  - Influence how the remaining free space is populated
  - Do NOT affect population speed
  - Do NOT affect objects already in the IMCS

The population of in-memory objects is controlled by the PRIORITY value. The default is PRIORITY NONE, which results in deferred population. Otherwise, there are four priority levels: LOW, MEDIUM, HIGH, and CRITICAL. If any of these levels is specified, the object is subject to automatic population.

With automatic population, objects are populated in priority order (CRITICAL, HIGH, MEDIUM, and LOW) at instance startup or when IMCO next wakes up, which is approximately every 2 minutes.

Note that the priority setting applies to the population queue, and only influences how the remaining free space is populated in the IMCS.

The priority setting does not affect the population rate; that is, no additional processing power or algorithm enhancements are applied to higher priority objects.

Also, if a new critical priority object cannot fit into the IMCS, space is not automatically created by removing lower priority objects. However, the next time the database restarts, the critical priority object is populated before any lower priority objects.

Defining the Compression Level

Use **MEMCOMPRESS** to define the compression level:

```
SQL> CREATE TABLE ... INMEMORY [ NO MEMCOMPRESS |
     MEMCOMPRESS FOR { DML | QUERY [LOW|HIGH] | CAPACITY [LOW|HIGH] }];
```

- Use different settings to change the balance between capacity and performance.
- Compare stored bytes on disk with columnar format compressed bytes by using V$IM_SEGMENTS.

- Compression level change does not automatically repopulate existing IMCUs.

ORACLE

The MEMCOMPRESS subclause distinguishes in-memory compression from on-disk compression. In ascending order (least compression to most compression), the available compression level options are:

- NO MEMCOMPRESS does not compress the data.
- MEMCOMPRESS FOR DML optimizes the data for DML operations.
- MEMCOMPRESS FOR QUERY LOW is optimized for the best query performance. This is the default when the INMEMORY clause is specified without a compression method or when MEMCOMPRESS FOR QUERY is specified without including either LOW or HIGH.
- MEMCOMPRESS FOR QUERY HIGH provides excellent query performance with greater compression than the previous options.
- MEMCOMPRESS FOR CAPACITY LOW provides good query performance with even greater compression than the previous options. This is the default when MEMCOMPRESS FOR CAPACITY is specified without including either LOW or HIGH.
- MEMCOMPRESS FOR CAPACITY HIGH is optimized for the greatest compression.

Use V$IM_SEGMENTS to compare the actual data size on disk (BYTES column) with the data size in the IMCS (INMEMORY_SIZE column).

Note that when the compression level is changed, the data dictionary is immediately updated and the new compression level is applied to newly populated data. Existing data in the IMCS is not automatically repopulated.

Controlling Data Distribution

Use **DISTRIBUTE** to control data distribution:

```
SQL> CREATE TABLE ... INMEMORY DISTRIBUTE [ AUTO | BY ROWID RANGE |
     BY PARTITION | BY SUBPARTITION ];
```

- Applicable only for Oracle RAC
- Available options:
  - DISTRIBUTE AUTO:
    - It is the default setting.
    - It allows Oracle to control data distribution.
  - DISTRIBUTE BY PARTITION or SUBPARTITION:
    - Distribution is according to the partitioning strategy.
    - GV$IM_SEGMENTS shows the partition-to-instance mapping.
  - DISTRIBUTE BY ROWID RANGE:
    - It ensures distribution regardless of the table size.
- In all cases, Oracle decides the data-to-instance mapping.

ORACLE

The DISTRIBUTE subclause is applicable only if you are using Oracle RAC. It controls how table data in the IMCS is distributed across Oracle RAC instances. The available options are:

- DISTRIBUTE AUTO allows Oracle Database to control how data is distributed across Oracle RAC instances. Large tables are distributed across Oracle RAC instances depending on their access patterns. Small tables are not distributed, and may be populated in only one instance. If you specify only the DISTRIBUTE keyword or omit the DISTRIBUTE clause entirely, then AUTO is the default.
- DISTRIBUTE BY PARTITION specifies that whole primary partitions are distributed to different Oracle RAC instances. If composite partitioning is in used, then all subpartitions are collocated in the same instance.
- DISTRIBUTE BY SUBPARTITION specifies that subpartitions are distributed to different Oracle RAC instances.
- DISTRIBUTE BY ROWID RANGE explicitly instructs Oracle Database to distribute the data across the available Oracle RAC instances. You can use this setting to ensure that an unpartitioned table is distributed regardless of its size.

Note that in all cases, Oracle Database decides how the data is distributed to the available instances. There are no controls that allow a user to specify the instance that a data item (row, table, partition, and so on) is populated in.

Controlling Data Duplication

Use **DUPLICATE** to control data duplication:

```
SQL> CREATE TABLE ... INMEMORY [ NO DUPLICATE |
     DUPLICATE | DUPLICATE ALL ];
```

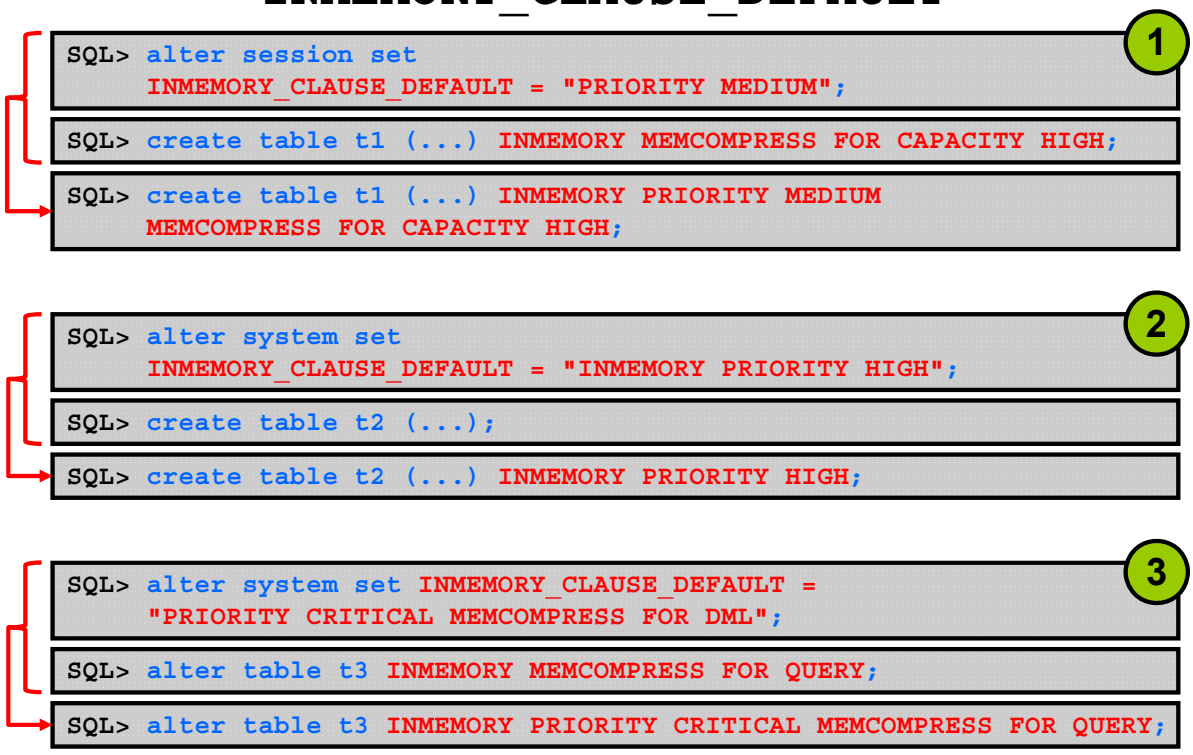
- Only applicable with Oracle RAC
- Available options:
  - NO DUPLICATE:
    - Is the default setting
    - Only one copy of the data resides across all of the instances.
  - DUPLICATE:
    - Two data copies reside on separate instances.
    - Only available on Engineered Systems
  - DUPLICATE ALL:
    - Data is duplicated on every available instance.
    - Implies data distribution using DISTRIBUTE ALL
    - Only available on Engineered Systems

The DUPLICATE subclause is applicable only if you are using Oracle RAC, and some options are available only on Oracle Engineered Systems, such as Oracle Exadata Database Machine. It controls how table data in the IMCS is duplicated across Oracle RAC instances. The available options are:

- NO DUPLICATE specifies that data is not duplicated across Oracle RAC instances. Or in other words, each data item resides in the IMCS on only one instance. This is the default setting.
- DUPLICATE specifies that data is primarily populated in one instance and duplicated on one other Oracle RAC instance. In other words, each data item resides in the IMCS on two Oracle RAC instances.
- DUPLICATE ALL specifies that data is duplicated across all Oracle RAC instances. If you specify DUPLICATE ALL, then the database automatically uses the DISTRIBUTE AUTO setting, regardless of the DISTRIBUTE clause setting.

Note that the DUPLICATE and DUPLICATE ALL options are available only on Oracle Engineered Systems. On all other platforms, these settings are ignored and effectively set to NO DUPLICATE.

Setting INMEMORY Clause Defaults: INMEMORY_CLAUSE_DEFAULT



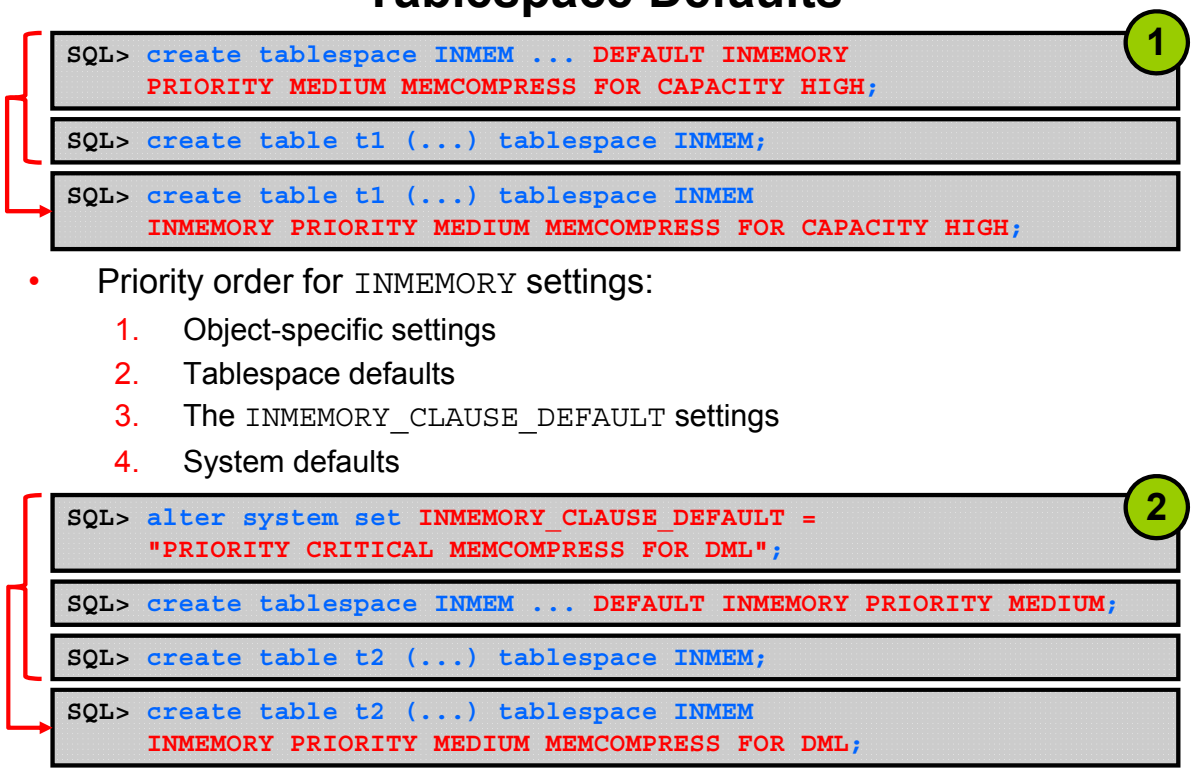
Instead of specifying the full INMEMORY clause each time the user creates or alters an object, users can specify customized defaults by setting the INMEMORY_CLAUSE_DEFAULT parameter. By default, the parameter is set to an empty string, which means that the system defaults apply. The parameter can be set at the system level or at the session level.

If the INMEMORY_CLAUSE_DEFAULT parameter is set, then its settings override the system defaults. However, settings specified in CREATE and ALTER statements still override the INMEMORY_CLAUSE_DEFAULT parameter settings.

If the INMEMORY_CLAUSE_DEFAULT parameter contains the INMEMORY keyword, then any newly created tables or materialized views are automatically enabled as candidate objects for the IMCS, unless NO INMEMORY is explicitly specified in the SQL CREATE statement.

The slide contains three examples, which illustrate how the INMEMORY_CLAUSE_DEFAULT parameter works. In each example, the bottom statement shows the cumulative effect of the top two commands.

Setting INMEMORY Clause Defaults: Tablespace Defaults



- Priority order for INMEMORY settings:
  1. Object-specific settings
  2. Tablespace defaults
  3. The INMEMORY_CLAUSE_DEFAULT settings
  4. System defaults

In addition to setting the INMEMORY_CLAUSE_DEFAULT parameter, a DEFAULT INMEMORY clause can be associated with a tablespace. If present, the settings in this clause override the system defaults and the settings in the INMEMORY_CLAUSE_DEFAULT parameter. Object-specific settings specified in CREATE or ALTER SQL commands still override the tablespace defaults.

The slide contains two examples that illustrate how the tablespace DEFAULT INMEMORY settings work. In each example, the bottom statement shows the cumulative effect of the previous commands.

Examining Candidate Objects

View the in-memory attributes by using data dictionary views:

```
SQL> SELECT TABLE_NAME, INMEMORY, INMEMORY_PRIORITY "IM_PRIORITY",
     INMEMORY_DISTRIBUTE "DIST", INMEMORY_COMPRESSION "COMP",
     INMEMORY_DUPLICATE "DUPL" FROM USER_TABLES;


TABLE_NAME INMEMORY PRIORITY DIST   COMP              DUPL
---------- -------- -------- ------ --------------- -------------
T1         DISABLED
T2         ENABLED  MEDIUM   AUTO   FOR QUERY LOW   NO DUPLICATE
```


Various data dictionary views, such as `USER_TABLES` and `DBA_TABLES`, now include columns that contain the in-memory settings for candidate objects. The slide shows an example query and the corresponding output.

If an object is not a candidate for population in the IMCS, the `INMEMORY` column contains `DISABLED` and the in-memory attribute columns are all `NULL`. The is the case for table `T1` in the slide example.

If an object is a candidate for population in the IMCS, the `INMEMORY` column contains `ENABLED` and the in-memory attribute columns contain the current attribute setting. This is the case for table `T2` in the slide example.

Examining the IMCS: Segment Information

V$IM_SEGMENTS contains segment-level metadata for objects in the IMCS:

```
SQL> SELECT SEGMENT_NAME, INMEMORY_SIZE, BYTES, BYTES_NOT_POPULATED,
     POPULATE_STATUS "POP_STAT", INMEMORY_PRIORITY "PRI",
     INMEMORY_DISTRIBUTE "DIST", INMEMORY_DUPLICATE "DUPL",
     INMEMORY_COMPRESSION "COMP" FROM V$IM_SEGMENTS;

SEGMENT_NAME INMEMORY_SIZE      BYTES BYTES_NOT_POPULATED POP_STAT
------------ ------------- ---------- -------------------- -----------
PRI          DIST          DUPL               COMP
---------- ---------- ---------------- ----------------
T2                  34799616   83886080                    0 COMPLETED
NONE         AUTO          DUPLICATE ALL    FOR QUERY LOW
```

Segment size in memory

Segment size on disk

Amount of the on-disk size not populated

Population status: STARTED, COMPLETED or FAILED

```
   BYTES / INMEMORY_SIZE = Compression Ratio
83886080 /      34799616 =                2.41
```


V$IM_SEGMENTS contains segment-level metadata for objects in the IMCS. You can use this view to examine the currently populated objects, including the settings (governing priority, distribution, duplication, and compression) that were used to populate the object. Also, the view contains the following useful information:

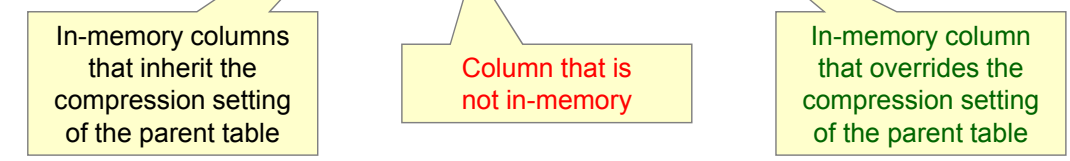
- INMEMORY_SIZE displays the size of the segment in memory
- BYTES displays the total on-disk segment size
- POPULATION_STATUS displays the status of the population operation; either STARTED, COMPLETED or FAILED.
- BYTES_NOT_POPULATED displays the amount of the on-disk segment size that is not populated in the IMCS. If zero is displayed, then the whole segment is populated in the IMCS. A non-zero value may mean one of the following:
  - Population is still in progress. In this case, expect to see POPULATION_STATUS=STARTED.
  - Population failed because of an error. In this case, expect to see POPULATION_STATUS=FAILED.
  - The object is distributed across multiple instances. This is likely if POPULATION_STATUS=COMPLETED. To confirm this possibility, examine the IMCS on every instance. You can use GV$IM_SEGMENTS to do this.
  - Incomplete population occurred because the IMCS ran out of space. This is also a possibility if POPULATION_STATUS=COMPLETED. To confirm this possibility, examine the IMCS memory status by using V$INMEMORY_STATUS.

Examining the IMCS: Column Information

V$IM_COLUMN_LEVEL contains column-level metadata for objects in the IMCS:

```
SQL> SELECT TABLE_NAME, COLUMN_NAME, INMEMORY_COMPRESSION
     FROM V$IM_COLUMN_LEVEL;

TABLE_NAME COLUMN_NAME INMEMORY_COMPRESSION
---------- ----------- --------------------
T2         COL1        DEFAULT
T2         COL2        DEFAULT
T2         COL3        MEMCOMPRESS FOR DML
T2         COL4        NO INMEMORY
```

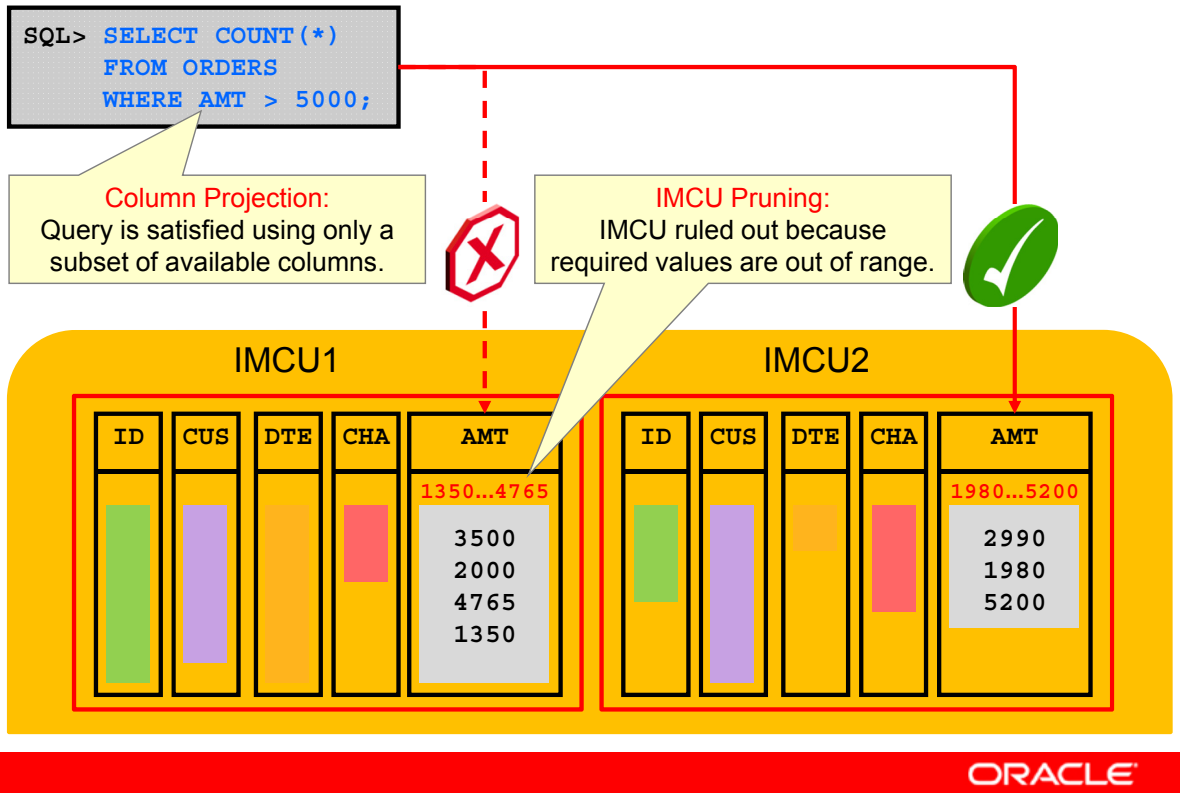


V$IM_COLUMN_LEVEL contains column-level metadata for objects in the IMCS.

You can use the V$IM_COLUMN_LEVEL view to find:

- Columns that are not populated in the IMCS
- Columns that override the compression settings of the parent table

Column Projection and IMCU Pruning



The example in the slide illustrates two of the optimizations that occur when queries access the IMCS.

When a query references only a subset of the available columns, the column-oriented nature of the IMCS allows for the unused columns to simply be skipped. This is called column projection. The example in the slide shows column projection because the query is completely satisfied by using only the `AMT` column.

In addition to the data values in each IMCU, each column has a metadata area that includes the minimum and maximum column values for the IMCU. This information is used in query processing to determine whether or not the IMCU contains relevant data. If it does not, the IMCS is not scanned any further. This optimization is know as IMCU pruning, and it is conceptually similar to the storage index concept in Exadata Storage Server. IMCU pruning applies to queries with predicate filters on a single column that contain operator such as: `=`, `<>`, `<`, `<=`, `>`, `>=`, `IS NULL`, `IS NOT NULL`, `LIKE`, `SUBSTRING`.

The example in the slide shows IMCU pruning because the predicate `AMT>5000` fails when tested against IMCU1. Therefore, the query is satisfied by scanning the `AMT` column in IMCU2.

You can view the IMCU minimum and maximum column values by querying `V$IM_COL_CU`.

IMCU Pruning Statistics

```
SQL> SELECT display_name, value
     FROM   v$mystat m, v$statname n
     WHERE  m.statistic# = n.statistic#
     AND    display_name IN (
                  'IM scan segments minmax eligible',
                  'IM scan CUs pruned',
                  'IM scan CUs optimized read',
                  'IM scan CUs predicates optimized');

DISPLAY_NAME                                                     VALUE
---------------------------------- --------------------
IM scan segments minmax eligible                                    2
IM scan CUs pruned                                                  1
IM scan CUs optimized read                                          0
IM scan CUs predicates optimized                                    1
```


The example in the slides shows IMCU pruning statistics. Following is a description for each statistic:

- `IM scan segments minmax eligible` displays the number of IMCUs eligible for pruning.
- `IM scan CUs pruned` displays the number of IMCUs where no rows satisfied the `WHERE` clause predicate, and the IMCU was no scanned.
- `IM scan CUs optimized read` displays the number of IMCUs where all rows satisfied the `WHERE` clause predicate. Or in other words, every rows in the IMCU was required to satisfy the query.
- `IM scan CUs predicates optimized` displays the number of IMCUs where either no rows satisfied the `WHERE` clause predicate, or all rows satisfied the `WHERE` clause predicate. Or in other words, `IM scan CUs predicates optimized` is the sum of `IM scan CUs pruned` and `IM scan CUs optimized read`.

The example in the slide shows the values you would expect to see for the example query on the previous page. In this case, two IMCUs are eligible for pruning. One is pruned, and the other is scanned.

In-Memory Query Statistics

```
SQL> SELECT display_name, value
     FROM   v$mystat m, v$statname n
     WHERE  m.statistic# = n.statistic#
     AND    name like '%IM%';

NAME                                                  VALUE
---------------------------------------------------- ----------
...
session logical reads - IM                             10097
IM scan bytes in-memory                             33490629
IM scan bytes uncompressed                          65769131
IM scan CUs columns accessed                               34
IM scan rows                                           700000
IM scan rows projected                                 700000
...
IM scan segments disk                                      0
table scan disk non-IMC rows gotten                        0
...
```

High counts indicate in-memory scanning

High counts indicate that in-memory scanning is disabled or the data is not populated

In addition to the IMCU pruning statistics, there are over 200 statistics related to all aspects of Oracle Database In-Memory. The slide contains an example query and partial output that includes a selection of the most used statistics for analyzing in-memory query performance. Following is a description for each statistic:

- `session logical reads - IM` displays the number of blocks scanned in-memory.
- `IM scan bytes in-memory` displays the total compressed size of scanned IMCUs.
- `IM scan bytes uncompressed` displays the total decompressed size of scanned IMCUs.
- `IM scan CUs columns accessed` displays the number of columns accessed.
- `IM scan rows` displays the total number of rows scanned in the IMCS.
- `IM scan rows projected` displays the total number of rows projected out of in-memory scans after the application of filters.
- `IM scan segments disk` displays the number of times a segment was retrieved from the traditional IO path because a column was not in memory.
- `table scan disk non-IMC rows gotten` displays the number of rows fetched during table scans, but not from the IMCS.

Simple Query Execution Plans

```
SQL> SELECT COUNT(*) FROM ORDERS WHERE AMT > 5000;
```

- Query plan without in-memory table access:

```
-------------------------------------------------------------------------
| Id  | Operation            | Name    | Rows  | Bytes | Cost (%CPU)|
-------------------------------------------------------------------------
|   0 | SELECT STATEMENT     |         |     1 |     6 |  2746   (1)|
|   1 |  SORT AGGREGATE      |         |     1 |     6 |            |
|*  2 |   TABLE ACCESS FULL  | ORDERS  | 24841 |  145K |  2746   (1)|
-------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------
2 - filter("AMT">5000)
```

1

- Query plan with in-memory table access:

```
...
|*  2 |   TABLE ACCESS INMEMORY FULL | ORDERS | 24841 |   145K|
--------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------
   2 - inmemory("AMT">5000)
       filter("AMT">5000)
```

2

The top of the slide shows a simple query with a single column comparison predicate. Assuming that there are no indexes, the optimizer chooses a table scan for such queries.

Without Oracle Database In-Memory, the query performs a full table scan (`TABLE ACCESS FULL`), as shown in query plan number one. The full table scan performs disk I/Os to read the table data or it uses the contents of the buffer cache if it can.

With Oracle Database In-Memory, the scan may be performed by using data in the IMCS. In such cases, the query plan includes `TABLE ACCESS INMEMORY FULL`. The `INMEMORY` keyword indicates that the full table scan operation is eligible to have some or all of the data returned from the IMCS. An example is shown in the slide in query plan number 2.

Notice also that the predicate information section of query plan number 2 includes `inmemory("AMT">5000)`. This indicates that in-memory predicate processing can be applied, including IMCU pruning.

If a table is only partially populated in the IMCS, queries are able to use the part that is in-memory and use other access methods to read the rest.

In-Memory Joins

```
SQL> SELECT SUM(lo_extendedprice * lo_discount) revenue
     FROM   orderline l, date_dim d
     WHERE  l.lo_orderdate = d.d_datekey AND l.lo_discount BETWEEN 2 AND 3
     AND    l.lo_quantity < 24 AND d.d_date='December 24, 1996';

-------------------------------------------------------------------------
| Id  | Operation                     | Name      | Rows  | Bytes |
-------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |           |     1 |    43 |
|   1 |  SORT AGGREGATE               |           |     1 |    43 |
|*  2 |   HASH JOIN                   |           |    38 |  1634 |
|   3 |    JOIN FILTER CREATE         | :BF0000   |     1 |    25 |
|*  4 |     TABLE ACCESS INMEMORY FULL| DATE_DIM  |     1 |    25 |
|   5 |    JOIN FILTER USE            | :BF0000   | 92597 |  1627K|
|*  6 |     TABLE ACCESS INMEMORY FULL| ORDERLINE | 92597 |  1627K|
-------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------
   2 - access("L"."LO_ORDERDATE"="D"."D_DATEKEY")
   4 - inmemory("D"."D_DATE"='December 24, 1996')
       filter("D"."D_DATE"='December 24, 1996')
   6 - inmemory("L"."LO_DISCOUNT"<=3 AND "L"."LO_QUANTITY"<24 AND
              "L"."LO_DISCOUNT">=2 AND
     SYS_OP_BLOOM_FILTER(:BF0000,"L"."LO_ORDERDATE"))
       filter("L"."LO_DISCOUNT"<=3 AND "L"."LO_QUANTITY"<24 AND
              "L"."LO_DISCOUNT">=2 AND
     SYS_OP_BLOOM_FILTER(:BF0000,"L"."LO_ORDERDATE"))
```

ORACLE

SQL statements that join multiple tables can be processed very efficiently in the IMCS by using bloom filters. A bloom filter transforms a join into a filter that can be applied as part of the scan of the larger table. Bloom filters were originally introduced in Oracle Database 10*g* to enhance hash join performance and are not specific to Oracle Database In-Memory. However, they are very efficiently applied to column format data.

When two tables are joined via a hash join, the first table (typically the smaller table) is scanned and the rows that satisfy the `WHERE` clause predicates are used to create a hash table in PGA memory. During the hash table creation, a bloom filter (or bit vector) is also created based on the join column. The bloom filter is then sent as an additional predicate to the scan of the second table. After the `WHERE` clause predicates are applied to the second table scan, the resulting rows have their join column hashed and compared to values in the bloom filter. If a match is found, the row is sent to the hash join. If no match is found, the row is discarded.

The slide shows an in-memory join example using a bloom filter. It is executed as follows:

1. A full in-memory table scan is performed on the `DATE_DIM` table (operation 4). The scan returns the rows that satisfy the predicate `D_DATE='December 24, 1996'`.
2. Using the result of the `DATE_DIM` scan, a bloom filter (`:BF0000`) based on the `D_DATEKEY` join column is also created (operation 3).

3. The bloom filter is sent as an additional predicate to the `ORDERLINE` table scan (`SYS_OP_BLOOM_FILTER(:BF0000,"L"."LO_ORDERDATE"`).
4. The `ORDERLINE` table is scanned to find the rows that satisfy the predicates `LO_DISCOUNT<=3 AND LO_QUANTITY<24 AND LO_DISCOUNT>=2` (operation 6).
5. For the resulting rows, the join column `LO_ORDERDATE` is hashed compared to the `D_DATEKEY` values in the bloom filter (`:BF0000`) (operation 5). If a match is found, the row is sent to the hash join. If no match is found, the row is discarded.
6. The results from both table scans are joined by using a hash join (operation 2). The bloom filter optimizes the hash join by reducing the number of rows that must be processed by the hash join operation.

Joining In-Memory and Non-In-Memory Tables

```
SQL> SELECT SUM(lo_extendedprice * lo_discount) revenue
     FROM   orderline l, date_dim_not_in_mem d
     WHERE  l.lo_orderdate = d.d_datekey AND l.lo_discount BETWEEN 2 AND 3
     AND    l.lo_quantity < 24 AND d.d_date='December 24, 1996';

-------------------------------------------------------------------------------
| Id  | Operation                     | Name                | Rows  | Bytes |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |                     |     1 |    43 |
|   1 |  SORT AGGREGATE               |                     |     1 |    43 |
|*  2 |   HASH JOIN                   |                     |    38 |  1634 |
|   3 |    JOIN FILTER CREATE         | :BF0000             |     1 |    25 |
|*  4 |     TABLE ACCESS FULL         | DATE_DIM_NOT_IN_MEM |     1 |    25 |
|   5 |    JOIN FILTER USE            | :BF0000             | 92597 |  1627K|
|*  6 |     TABLE ACCESS INMEMORY FULL| ORDERLINE           | 92597 |  1627K|
-------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------
   2 - access("L"."LO_ORDERDATE"="D"."D_DATEKEY")
   4 - filter("D"."D_DATE"='December 24, 1996')
   6 - inmemory("L"."LO_DISCOUNT"<=3 AND "L"."LO_QUANTITY"<24 AND
              "L"."LO_DISCOUNT">=2 AND
       SYS_OP_BLOOM_FILTER(:BF0000,"L"."LO_ORDERDATE"))
       filter("L"."LO_DISCOUNT"<=3 AND "L"."LO_QUANTITY"<24 AND
              "L"."LO_DISCOUNT">=2 AND
       SYS_OP_BLOOM_FILTER(:BF0000,"L"."LO_ORDERDATE"))
```

ORACLE

In situations where in-memory tables are joined with non-in-memory tables, Oracle still masks optimal use of the available in-memory resources. For example, the slide shows an example with the same query as on the previous page; however, this time the date dimension table (DATE_DIM_NOT_IN_MEM) is not in-memory.

Notice that the execution plan for this query is almost identical to the previous example, when both tables were in-memory. The only difference is that the scan on DATE_DIM_NOT_IN_MEM is not an in-memory scan (operation 4). However, notice that the query still uses a bloom filter to optimize the hash join, and that the bloom filter is still applied using in-memory predicate processing (operations 5 and 6).

DML Processing with Oracle Database In-Memory

- Oracle Database In-Memory manages DML for OLTP
  - Changes are written to an in-memory transaction journal
  - Statistics include number of transactions and journaled rows:

```
SQL> SELECT name, value FROM   v$mystat m, v$statname n
     WHERE  m.statistic# = n.statistic#
     AND    name like 'IM transactions%';

NAME                                                  VALUE
---------------------------------------------------- ----------
IM transactions                                            1
IM transactions rows journaled                         10224
...
```

- Oracle Database In-Memory manages bulk data loads
  - The IMCS is populated in the background during direct-path loads
  - Partial population occurs if there is insufficient memory available

ORACLE

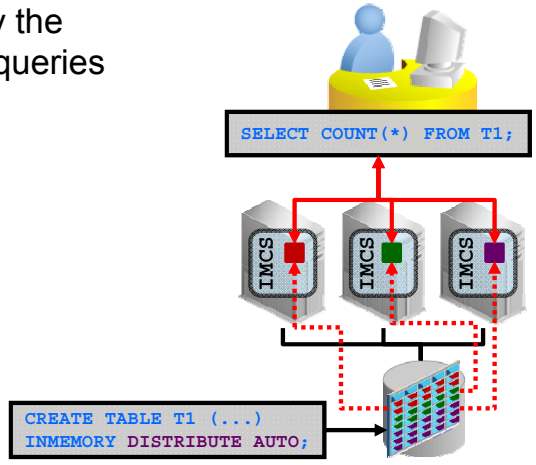
Oracle Database In-Memory manages OLTP DML operations to ensure that the IMCS remains transactionally consistent with the data on disk or in the buffer cache. When data in the IMCS changes, the populated information is invalidated and a change record is written to the in-memory transaction journal. Periodically the journal is used to re-populate the IMCS with a clean data copy, which includes all of the changes in the journal.

Oracle Database In-Memory includes statistics that provide information about the number of transactions performed against data in the IMCS, and the number of rows written into the journal. The slide contains an example that shows these statistics.

In addition to catering for OLTP operations, Oracle Database In-Memory is also able to perform direct-path loads by populating the IMCS along with loading the data to disk. In this case, a background population is performed and the bulk load operation does not commit until the background memory population completes. If the population fails due to a shortage of available memory, the bulk load is committed, leaving the application to use a combination of the on-disk data and the partially populated in-memory version. For bulk data loads, the transaction journal is not used, and therefore, the `IM transactions` statistics are not incremented.

Oracle Database In-Memory and Oracle RAC

- Each Oracle RAC instance contains a separate IMCS
  - By default, large tables are populated across multiple column stores, but small tables reside in one column store only
  - Use the **DISTRIBUTE** and **DUPLICATE** clauses to adjust and control object population for Oracle RAC
  - Data duplication is only supported on Engineered Systems
- Use in-memory parallel query to apply the collective power of RAC to individual queries
  - PARALLEL_DEGREE_POLICY must be set to AUTO to enable in-memory parallel execution
  - Otherwise, traditional data access paths may be used
  - Use statistics to determine exactly what happened
- Cache fusion is used to maintain IMCS transactional consistency



ORACLE

When Oracle Database In-Memory is used in conjunction with Oracle RAC, each instance contains an IMCS. By default, Oracle Database automatically populates large table across multiple instances, while small tables are only populated in a single instance. You can control this behaviour by using the DISTRIBUTE and DUPLICATE clauses as discussed earlier in the lesson. Note that data duplication is not supported on platforms other than Oracle Engineered Systems, such as Exadata Database Machine.

By using in-memory parallel query, you can apply the collective power of multiple instances to individual queries. This enables extremely fast queries across huge data sets. Note that PARALLEL_DEGREE_POLICY must be set to AUTO to enable in-memory parallel execution. If in-memory parallel query is not enabled, or if the optimizer decides against using parallel query, then a query can only leverage the contents of the IMCS on the current instance. If the IMCS in the current instance does not contain the required data, then Oracle Database automatically falls back to using the traditional buffer-cache and disk I/O paths, even if the query plan indicates in-memory scan operations. You can use system-level and session-level statistics from across all of the database instances to determine precisely how Oracle Database In-Memory works in conjunction with Oracle RAC to execute queries.

Cache fusion has been extended to ensure that transactional consistency is maintained across the database instances. For RAC row invalidations that occur during DML operations, the row invalidation occurs both in the buffer cache and in the IMCS.

Quiz

What is the primary purpose of the IMCS?

a. To store any type of data
b. To maintain an in-memory copy of data, that is optimized for query performance
c. To keep a copy of data in memory when it is aged out of the cache
d. To compress data in memory in the same way as on disk

Answer: b

Quiz

What happens when IMCS runs out of space during a table population operation?

a. The population is rolled back and space remains available for population of smaller tables.
b. A warning or alert is issued by ADDM.
c. The table remains partially populated in the IMCS and any future queries on that table may use a combination of the populated data and traditional access methods.

Answer: c

Answer B is not true because the Automatic Database Diagnostic Monitor (ADDM) is not aware of the In-Memory Column Store.

Summary

In this lesson, you should have learned how to:

- Describe the benefits and architecture of Oracle Database In-Memory
- Enable Oracle Database In-Memory in a database
- Configure database objects to use Oracle Database In-Memory
- Describe how Oracle Database In-Memory performs optimized query processing
- Examine meta-data and statistics relating to Oracle Database In-Memory
- Describe how Oracle Database In-Memory works in conjunction with Oracle RAC

Practice 6 Overview: Using Oracle Database In-Memory in conjunction with Oracle RAC

In this practice, you will use the new Oracle Database In-Memory features on a 2-node Oracle RAC database. You will see how data is populated in the In-Memory Column Store (IMCS) across multiple instances and how queries can use data that is distributed across multiple instances.

7

Application Continuity

Objectives

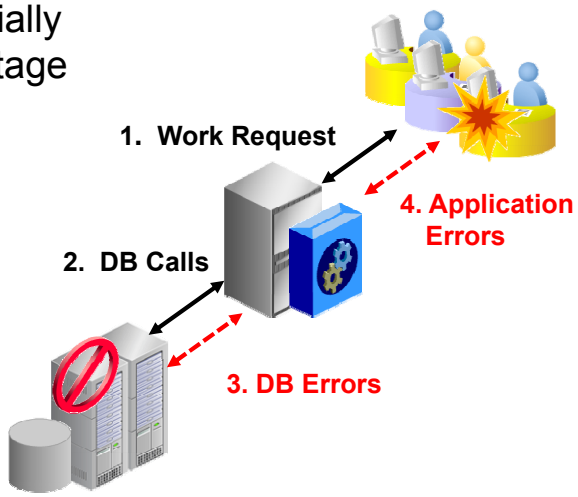
After completing this lesson, you should be able to:

- Describe the purpose of Transaction Guard and Application Continuity
- Describe the key concepts of Application Continuity
- Describe the side effects and restrictions of Application Continuity
- Describe the requirements for developing applications that leverage Application Continuity
- Configure Application Continuity

The Situation Prior to Application Continuity

- Database outages can cause:
  - In-flight work to be lost, leaving users in doubt
  - Users to restart applications and reenter data, leading to duplicate submission of requests
  - Additional failures, potentially prolonging the system outage
- Solving the problem inside the application is difficult and expensive.

Did the last transaction commit?



ORACLE

For end users, database session outages can lead to undesirable results:

- Confusion: Users do not know what happened to their application's transactions, such as fund transfers, orders, payments, and bookings.
- Duplication: Doubt over a failed transaction may lead users to reenter it. This may lead to undesirable duplication resulting in overpayments, overordering or overbooking, and so on.
- Loss of productivity: Even if duplication is not an issue, user productivity is adversely affected by the need to restart applications and reenter data.
- Prolonged system outages: A database failure may cause related applications or other infrastructure, like sensors and communication equipment, to stall or fail. This may require applications and other components to be rebooted or reinitialized, resulting in a prolonged system outage.

Developing a solution for these problems inside the application has traditionally been difficult and expensive for the following reasons:

- Every possible exception must be considered and handled.
- If a failure occurs during a commit, it is difficult to determine whether the commit occurred.
- To legitimately replay the work associated with a failed session, the application must ensure that the database is in the correct state; otherwise, the replay may be invalid.

Introducing Transaction Guard and Application Continuity

- Transaction Guard is a new reliable protocol and API that returns the outcome of the last transaction after a recoverable error has occurred.
- Application Continuity is a feature that attempts to mask database session outages by recovering the in-flight work for requests submitted to the database.

Oracle Database 12*c* introduces two fundamental capabilities for ensuring continuity of applications after database outages:

1. A foolproof way for applications to know the outcome of transactions
2. The ability to mask outages by reconnecting to the database and replaying the workload

These capabilities are provided by two new features: Transaction Guard and Application Continuity.

Transaction Guard is an API that applications use in error handling. It is a new and reliable way to return the outcome of the last transaction after a recoverable error has occurred. By itself, Transaction Guard can dramatically improve the end-user experience by erasing the doubt over whether the last transaction was committed or not.

Application Continuity is a feature that masks recoverable outages from end users and applications. Application Continuity attempts to replay the transactional and nontransactional work that constitute a database request. When replay is successful, the outage appears to the end user as if the execution was slightly delayed. With Application Continuity, the end user experience is improved because users may never sense that an outage has occurred. Furthermore, Application Continuity can simplify application development by removing the burden of dealing with recoverable outages.

Key Concepts of Application Continuity

Database Request: Unit of work submitted from the application, typically SQL, PL/SQL, RPC calls

Recoverable Error: Error that arises due to an external system failure, independent of the application

Commit Outcome: Outcome of last transaction; made durable by Transaction Guard

Mutable Functions: Functions that change their results each time they are executed

Session State Consistency: Describes how the application changes the nontransactional state during a database request

The slide introduces some terms and concepts that are associated with Transaction Guard and Application Continuity. The following notes elaborate further:

Database Request

A database request is a unit of work submitted from the application. A database request typically consists of SQL statements, PL/SQL blocks, local procedure calls, and database Remote Procedure Calls (RPCs), in a single web request on a single database connection. Database requests are generally demarcated by the calls made to check out and check in a database connection from a connection pool. For recoverable errors, Application Continuity reestablishes the database session and repeats the database request safely.

Recoverable Error

A recoverable error is an error that arises due to an external system failure, independent of the application session logic that is executing. Recoverable errors occur following planned and unplanned outages of foregrounds, networks, nodes, storage, and databases. The application receives an error code (such as `ORA-03113: end-of-file on communication channel`); however, the status of the last submitted operation may be unclear. Recoverable errors have been enhanced in Oracle Database 12*c* to include more errors, and now include a public API for OCI.

Application Continuity reestablishes database sessions and resubmits the pending work for recoverable errors. Application Continuity does not resubmit work following call failures due to nonrecoverable errors. An example of a nonrecoverable error is submission of an invalid data value, such as an invalid date or invalid numeric value.

Commit Outcome

In Oracle Database, a transaction is committed by updating its entry in the internal transaction table. Oracle Database generates a redo log record corresponding to this update. After the redo log record is written out to the redo log on disk, the transaction is deemed committed at the database. From a client perspective, the transaction is committed when a commit outcome message, generated after that redo is written, is received by the client. However, the commit outcome message is not durable. For example, the commit outcome can be lost if the client is disconnected from the database after the commit outcome is generated by the database but before it is received by the client. Transaction Guard provides a reliable commit outcome because it can be used to reliably obtain the commit outcome when it has been lost following a recoverable error that results in the failure of a database session.

Mutable Functions

Mutable functions are functions that can change their results each time that they are called. Mutable functions can cause replay to be rejected because the results visible to the application change at replay. Consider `sequence.NEXTVAL` that is often used in key values. If a primary key is built with a sequence value and it is later used in foreign keys or other binds, at replay, the same function result must be returned if the application is using it.

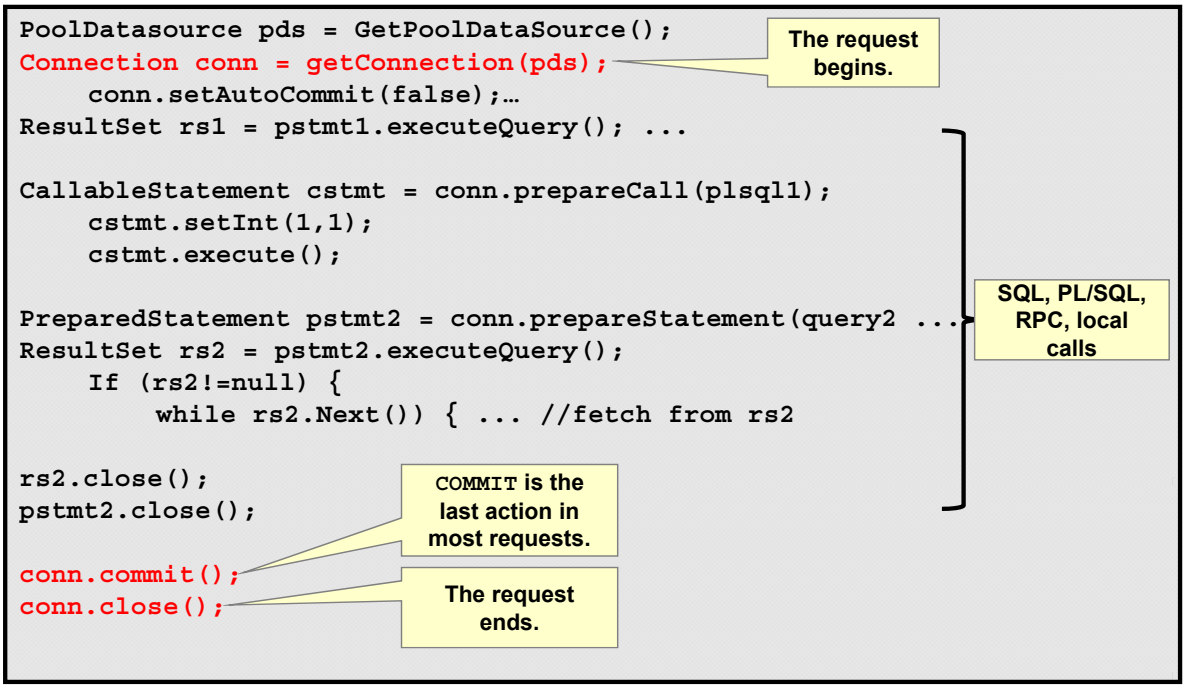
Application Continuity provides mutable object value replacement at replay for granted Oracle function calls to provide opaque bind-variable consistency. If the call uses database functions that are mutable, including `sequence.NEXTVAL`, `SYSDATE`, `SYSTIMESTAMP`, and `SYSGUID`, the original values returned from the function execution are saved and are reapplied at replay. If an application decides not to grant mutables, replay for these requests may be rejected.

Session State Consistency

Nontransactional state includes national language support (NLS) settings, cursors, events, and global PL/SQL package states. After a `COMMIT` statement has been executed, if the nontransactional state was changed in that transaction, it is not possible to replay the transaction to reestablish that state if the session is lost. When configuring Application Continuity, the applications are categorized depending on whether the session state after the initial setup is static or dynamic, and thus whether it is correct to continue past a `COMMIT` operation within a request.

While configuring Application Continuity, almost all applications should use the default `DYNAMIC` mode. In the `DYNAMIC` mode, replay is disabled from `COMMIT` until the end of the request. This is not a problem for most applications, because almost all requests have zero or one commit, and commit is most often the last statement in a database request.

Workflow of a Database Request



ORACLE

A database request is a logical unit of work.

For example:

- Every action that the client driver receives from an Object Relational Manager (ORM)
- Every action that you submit at your automatic teller machine
- Every action that you submit at your browser while browsing, shopping, making bill payments, and so on

Typically, all database requests that use JDBC follow a standard pattern.

The slide contains a code segment that shows how typical JDBC applications are coded:

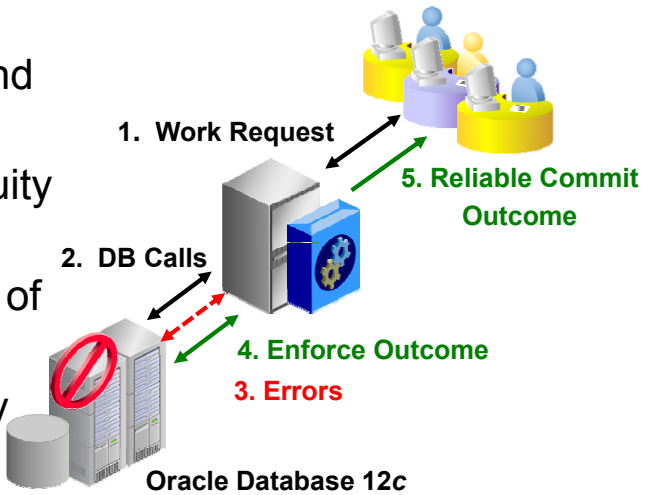
1. A database request begins with the call to `getConnection` to the `PoolDatasource`.
2. The application's logic is executed. This execution could include SQL, PL/SQL, RPC, or local procedure calls.
3. The transaction is committed.
4. The database request ends when the connection is returned to the connection pool.

What Is Transaction Guard?

Transaction Guard is:

- A tool that provides a reliable commit outcome for the last transaction after errors
- An API available for JDBC Thin, C/C++ (OCI/OCCI), and ODP.NET
- Used by Application Continuity for at-most-once execution
- Can be used independently of Application Continuity

Without Transaction Guard, retry can cause logical corruption.

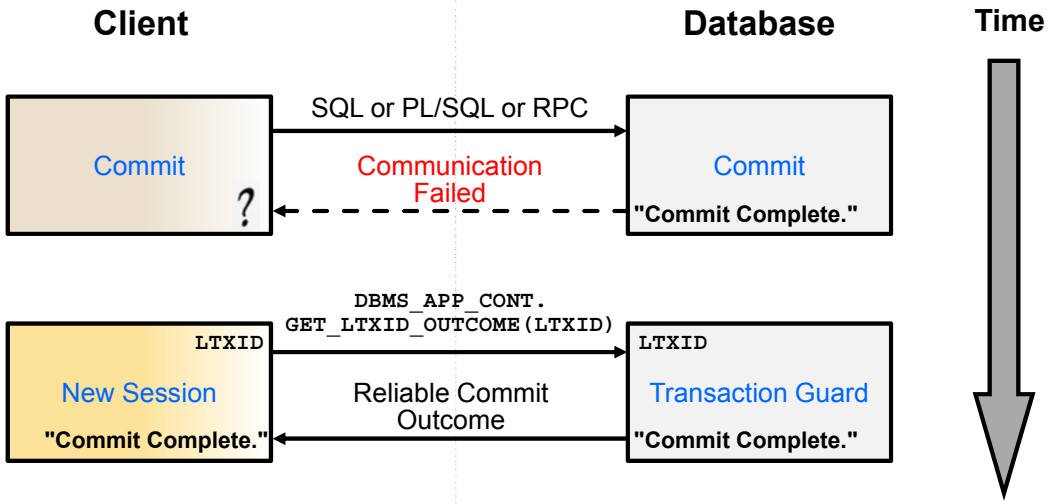


Transaction Guard is a reliable protocol and API that applications use to provide a reliable commit outcome. The API is typically embedded in error handling and should be called following recoverable errors. The outcome indicates whether the last transaction was committed and completed. After the commit outcome is returned to the application, the outcome persists. Therefore, if Transaction Guard returns committed or uncommitted, the status stays this way. This enables the application or user to proceed with confidence.

Transaction Guard is used by Application Continuity and is automatically enabled by it, but it can also be enabled independently. Transaction Guard prevents the transaction being replayed by Application Continuity from being applied more than once. If the application has implemented application-level replay, integration with Transaction Guard can be used to ensure that committed transactions are not replayed.

How Transaction Guard Works

Transaction Guard is a reliable protocol and API that enables applications to know the outcome of the last transaction.



In the standard commit case, the database commits a transaction and returns a success message to the client. In the illustration shown in the slide, the client submits a commit statement and receives a message stating that communication failed. This type of failure can occur due to several reasons, including a database instance failure or network outage. In this scenario, without Transaction Guard, the client does not know the outcome of the transaction.

Oracle Database solves the problem by using a globally unique identifier called a logical transaction ID (LTXID). When the application is running, both the database and client hold the logical transaction ID. The database supplies the client with a logical transaction ID at authentication and at each round trip from the client driver that executes one or more commit operations.

When a recoverable outage occurs, the logical transaction ID uniquely identifies the last database transaction submitted on the session that failed. A new PL/SQL interface (`DBMS_APP_CONT.GET_LTXID_OUTCOME`) interface returns the reliable commit outcome. Further detail on using Transaction Guard APIs is provided later in the lesson.

Using Transaction Guard

- Supported transaction types:
  - Local
  - Auto-commit and Commit on Success
  - Commit embedded in PL/SQL
  - DDL, DCL, and Parallel DDL
  - Remote, Distributed
- Not supported in release 12.1:
  - XA
  - Read-write database links from Active Data Guard
- Server configuration:
  - Set the `COMMIT_OUTCOME=TRUE` service attribute
  - Optionally, set the `RETENTION_TIMEOUT` service attribute
- Supported clients:
  - JDBC Thin, OCI, OCCI, and ODP.NET

Transaction Guard supports all the transaction types listed in the slide. The primary exclusions in Oracle Database12*c* release 12.1 are:

- Transaction Guard is not supported for applications developed by using Oracle XA.
- Transaction Guard is not supported if you are using Active Data Guard with read/write database links to another database.

To enable Transaction Guard, set the service attribute `COMMIT_OUTCOME=TRUE`. Optionally, change the `RETENTION_TIMEOUT` service attribute to specify the amount of time that the commit outcome is retained. The retention timeout value is specified in seconds; the default is 86400 (24 hours), and the maximum is 2592000 (30 days).

Benefits of Transaction Guard

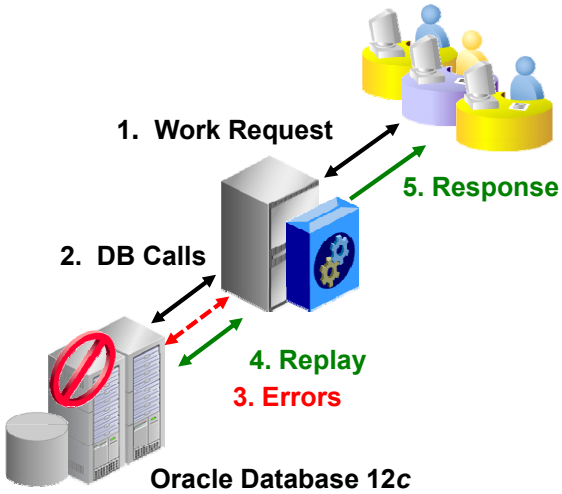
- After outages, users know what happened to their in-flight transactions, such as fund transfers, flight bookings, and bill payments.
- Transaction Guard provides better performance and reliability than home-built code to determine transaction outcome following a failure.

Transaction Guard is a powerful feature. Some of the benefits are:

- For applications that integrate Transaction Guard, users can know what happened to their last submission and proceed with confidence and certainty. Without Transaction Guard, doubt following a failure can lead to resubmitting a database request, which can cause logical corruption.
- Because it is integrated into the database kernel, Transaction Guard provides better performance and reliability than home-built code that attempts to determine the transaction outcome following a failure.

What Is Application Continuity?

- Replays in-flight work on recoverable errors
- Masks many hardware, software, network, storage errors, and outages, when successful
- Improves the end-user experience



Without Application Continuity, network outages, instance failures, hardware failures, repairs, configuration changes, patches, and so on can result in the failure of a user session followed by an error message of some sort.

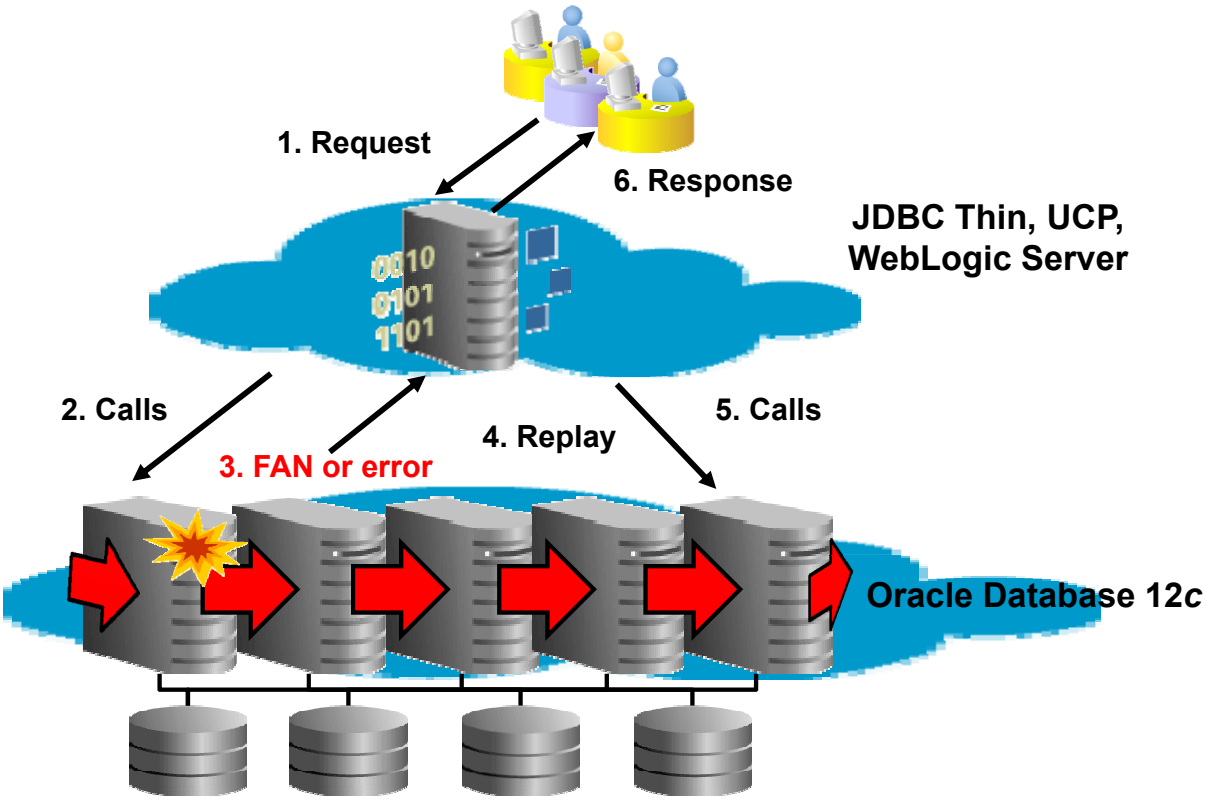
Application Continuity masks many recoverable database outages from applications and users. It achieves the masking by restoring the database session, the full session (including session state, cursors, and variables), and the last in-flight transaction (if there is one).

If the database session becomes unavailable due to a recoverable error, Application Continuity attempts to rebuild the session and any open transactions to the correct states. If the last transaction was successful and does not need to be reexecuted, the successful status is returned to the application. Otherwise, Application Continuity replays the last in-flight transaction.

To be successful, the replay must return to the client exactly the same data that the client received previously in the original request. This ensures that any decisions based on previously queried data are honored during the replay. If the replay is successful, the application continues safely without duplication.

If the replay is not successful, the database rejects the replay and the application receives the original error. This ensures that the replay does not proceed if circumstances change between the original request and the replay.

How Does Application Continuity Work?



The graphic in the slide illustrates how Application Continuity works. Following is a description of a typical workflow involving Application Continuity:

1. The client sends a work request to the application.
2. The application sends the calls that make up the request to the database using the JDBC replay driver.
3. The JDBC replay driver receives a Fast Application Notification (FAN) notification or a recoverable error.
4. The replay driver performs the following actions:
   - It checks that the request has replay enabled and that the replay initiation timeout has not expired. If all is in order, the driver obtains a new database session. If a callback is registered, the callback is executed to initialize the session.
   - It checks with the database to determine whether the last transaction completed.
   - If replay is required, the JDBC replay driver resubmits calls, receiving directions for each call from the database. Each call must result in the same client-visible state.
5. When the last call is replayed, the replay driver ends the replay and returns to normal runtime mode.
6. If the replay succeeds, the application responds normally to the user.

Using Application Continuity

- Supported database operations:
  - SQL, PL/SQL, and JDBC RPC:
    - `SELECT`, `ALTER SESSION`, DML, DDL, `COMMIT`, `ROLLBACK`, `SAVEPOINT`, and JDBC RPCs
  - Transaction models:
    - Local, Parallel, Remote, Distributed, and Embedded PL/SQL
  - Mutable functions
  - Transaction Guard
- Works in conjunction with:
  - Oracle RAC and RAC One Node
  - Oracle Active Data Guard
- Hardware acceleration on current Intel and SPARC chips
- Supported clients:
  - JDBC Thin driver, Universal Connection Pool, and WebLogic Server

ORACLE

The slide lists key points relating to the use of Application Continuity. The following notes elaborate further:

- Application Continuity recovers the database request, including any in-flight transaction and the database session states. The requests may include most SQL and PL/SQL, RPCs, and local JDBC calls. Note that for remote and distributed transactions, all databases involved must be release 12.1 or later.
- Application Continuity offers the ability to keep the original values for some Oracle functions, such as `SEQUENCE.NEXTVAL`, that change their values each time that they are called. This improves the likelihood that the replay will succeed.
- Application Continuity uses Transaction Guard. Transaction Guard tags each database session with a logical transaction ID (LTXID), so that the database recognizes whether a request committed the transaction before the outage.
- Application Continuity works in conjunction with Oracle Real Application Clusters (Oracle RAC), RAC One Node, and Oracle Active Data Guard.
- On the database server, the validation performed by Application Continuity is accelerated using processor extensions built into current SPARC and Intel chips.
- Application Continuity provides client support for thin JDBC, Universal Connection Pool, and WebLogic Server.

Application Continuity Processing Phases

Normal Run Time	Reconnect	Replay
• Demarcates the database request • Builds proxy objects • Holds original calls with validation • Manages queues	• Ensures that the request has replay enabled • Handles timeouts • Creates a new connection • Validates the target database • Uses Transaction Guard to enforce last commit	• Replays held calls • Continues replay, if user-visible results match, based on validations • Continues the request

ORACLE

The operation of Application Continuity can be divided into three distinct phases:

Normal run time: During normal run time, each new database request is tagged with a request beginning, either by checking out of the Universal Connection Pool or WebLogic Server Connection Pool, or by adding begin and end request markers to your own application or to your own Java connection pool at checkout and checkin. The JDBC replay driver and Oracle Database 12*c* collaborate so that calls in the database request are held in queues, together with validation received from the database. The JDBC Replay driver holds the calls until the end of the database request or until the replay is disabled. The JDBC replay driver is responsible for managing the queues and building proxy objects to maintain a record of nontransactional state, allowing objects to be replaced if replay is needed.

Reconnect: The reconnect phase of Application Continuity is triggered when a recoverable outage occurs. In this phase, the request is checked to see whether replay is still enabled, and the replay initiation timeout is checked to ensure that is has not expired. If both checks are in order, a new connection to the database is obtained. Because the reconnection to the database can take some time, you may need to set `FAILOVER_DELAY` and `FAILOVER_TIMEOUT` to allow the service to be reestablished. After the driver has established a connection to the database, it checks whether the database is a valid target and whether the last transaction was committed successfully.

Replay will not occur if the connection is to a logically different database, or to the same database but transactions have been lost (for example, the database has been restored to a prior point in time). Also, Application Continuity uses Transaction Guard to ensure that committed transactions are not replayed.

Replay: The replay phase starts when a new connection to the database is obtained. All calls held in the queues are replayed, and the request continues from the point where it failed. Replay is disabled if there are any user-visible changes in the results for any request that is replayed.

Restrictions

Global	Request	Target Database
• Do not use the default database service. • Replay is not supported for an Oracle XA data source. • Deprecated Java concrete classes are not supported.	• For Java streams, replay is on "best effort" basis. • Restricted calls: – `ALTER SYSTEM` – `ALTER DATABASE` • Replay is not supported for Active Data Guard with read/write database links	• Does not support: – Logical Standby – GoldenGate

The slide lists some restrictions and other considerations that apply to Application Continuity. There are basically three levels of restrictions for Application Continuity:

Global: The following restrictions prevent Application Continuity from being enabled or used on any request.

- Do not use the default service corresponding to the `DB_NAME` or `DB_UNIQUE_NAME`. The use of the database service is not recommended for high availability, because this service cannot be enabled or disabled, and cannot be relocated on Oracle RAC or switched over to Oracle Data Guard.
- Replay is not supported for applications developed by using Oracle XA.
- For applications using JDBC, there is no support for oracle.sql deprecated concrete classes like `BLOB`, `CLOB`, `BFILE`, `OPAQUE`, `ARRAY`, `STRUCT`, or `ORADATA`.

Request: The following restrictions disable Application Continuity for part of a request.

- For JDBC stream arguments, replay is on a "best effort" basis. For example, if the application is using physical addresses, the address is no longer valid with the outage and cannot be repositioned.
- Replay is disabled if the transaction executes the `ALTER SYSTEM` or `ALTER DATABASE` statement.
- Replay is not supported if you are using Active Data Guard with read/write database links to another database.

Target: Application Continuity is not supported for logically different databases (Oracle Logical Standby and Oracle GoldenGate).

Potential Side Effects

- Some applications may need to use the `disableReplay` API if there are any requests that should not be replayed.
- Examples of calls that create side effects are:
  - Autonomous transactions
  - `DBMS_ALERT` calls (email or other notifications)
  - `DBMS_FILE_TRANSFER` calls (copying files)
  - `DBMS_PIPE` and `RPC` calls (to external sources)
  - `UTL_FILE` calls (writing text files)
  - `UTL_HTTP` calls (making HTTP callouts)
  - `UTL_MAIL` calls (sending email)
  - `UTL_SMTP` calls (sending SMTP messages)
  - `UTL_TCP` calls (sending TCP messages)
  - `UTL_URL` calls (accessing URLs)

ORACLE

Applications that use external actions should be reviewed to decide whether requests with side effects should be replayed or not.

Application Continuity replays SQL, PL/SQL, and RPCs. Application Continuity serves to rebuild the session as if the user submission were delayed. When a session is rebuilt, all states are rebuilt, including reexecuting statements that leave side effects. These side effects may be exactly what is required, such as writing a report, completing some auditing, or obtaining custom primary key ranges. However, the calls that are replayed might include some calls that should not be replayed. The application may want to take action to accommodate or mitigate the effects of the replay. Developers can elect to use the `disableReplay` API for requests that contain calls that they do not want to replay.

The slide shows some of the actions that create side effects, such as Autonomous transactions, email, or other notifications by using `DBMS_ALERT` calls, copying files by using `DBMS_PIPE` and RPC calls, writing text files by using `UTL_FILE` calls, making HTTP callouts by using `UTL_HTTP` calls, sending email by using `UTL_MAIL` calls, sending SMTP messages by using `UTL_SMTP` calls, sending TCP messages by using `UTL_TCP` calls, and accessing URLs by using `UTL_URL` calls.

Actions That Disable Application Continuity

Normal Run Time	Reconnect	Replay
Any calls in same request after: • Successful commit in dynamic mode (the default) • A restricted call • `disableReplay` API call	• Error is not recoverable • Reconnection failure – Replay initiation timeout – Max connection tries – Max retries per incident • Target database is not valid for replay • Last call committed, in dynamic mode	• Validation detects different results

If it is disabled, Application Continuity is ineffective until the next request. The following events disable replay for the request:

- **Normal run time:** Capture happens on all original calls until a successful `COMMIT` (in dynamic mode) or unless it is disabled for the request.
- **Reconnect:** Replay does not occur if the error is not recoverable or if:
  - Timeouts have been exceeded
  - The target database is not the same or an ancestor
  - The request has committed
- **Replay:** The calls must return the same user-visible results that the application has previously seen and potentially used to make a decision. If the validation for the replayed call does not pass, replay is aborted and the original error is returned.

When Is Application Continuity Transparent?

- Application Continuity is transparent for J2EE applications that:
  - Use standard JDBC and Oracle connection pools
  - Do not have external actions, or have external actions and correctness is preserved at replay
- For situations where Application Continuity is not transparent, the following changes are typically required:
  - Request boundaries are specified using the Application Continuity APIs.
  - The `disableReplay` API is used to selectively stop replay for calls that the application never wants to replay.

Application Continuity is transparent (performed automatically) for J2EE applications that use standard JDBC and that use Oracle connection pools (UCP or WLS Active GridLink). For applications with external actions (for example, autonomous transactions or using UTL_HTTP to issue a SOA call), Application Continuity is still transparent only if the application's correctness is preserved when these external actions are replayed after a failure.

For other scenarios in which Application Continuity is not transparent, the following infrastructure changes may be needed:

- If the connection pool or container does not use an Oracle connection pool, the application must use Application Continuity APIs to mark request boundaries. Request boundaries are needed to reclaim the memory used for holding calls, and to establish a point at which to resume recording following nonreplayable operations.
- If the application has requests that the application does not want repeated, the application can explicitly call an API to disable replay for those requests. Such calls are likely to be isolated to one or a few specialized pieces of application code.

Benefits of Application Continuity

- Uninterrupted user service, when replay is successful
- Possible to relocate database sessions to remaining servers for planned outages
- Improved developer productivity by masking outages that can be masked
- Few or no application changes
- Simple configuration

Here are some benefits of Application Continuity:

- User service is uninterrupted when the request replay is successful.
- Application Continuity can be used to migrate the database sessions to the remaining servers without the users perceiving an outage.
- Masking outages that can be masked improves developer productivity. Error handling code is invoked less often and can potentially be simplified.
- Replay often requires few or no application changes.
- Application Continuity is simple to configure.

Application Assessment for Using Application Continuity

Decide	What You Should Do
Request Boundaries	Mark request boundaries if you are not using Oracle Pools.
JDBC Concrete Classes	Replace deprecated concrete classes with Java interfaces.
Side Effects	Use the `disable` API if a database request has an external call that should not be replayed.
Callbacks	Ensure that a callback is registered if the state changes outside a request (WLS/UCP labeling included by default).
Mutable Functions	Grant keeping mutable values if they are compatible with your application.

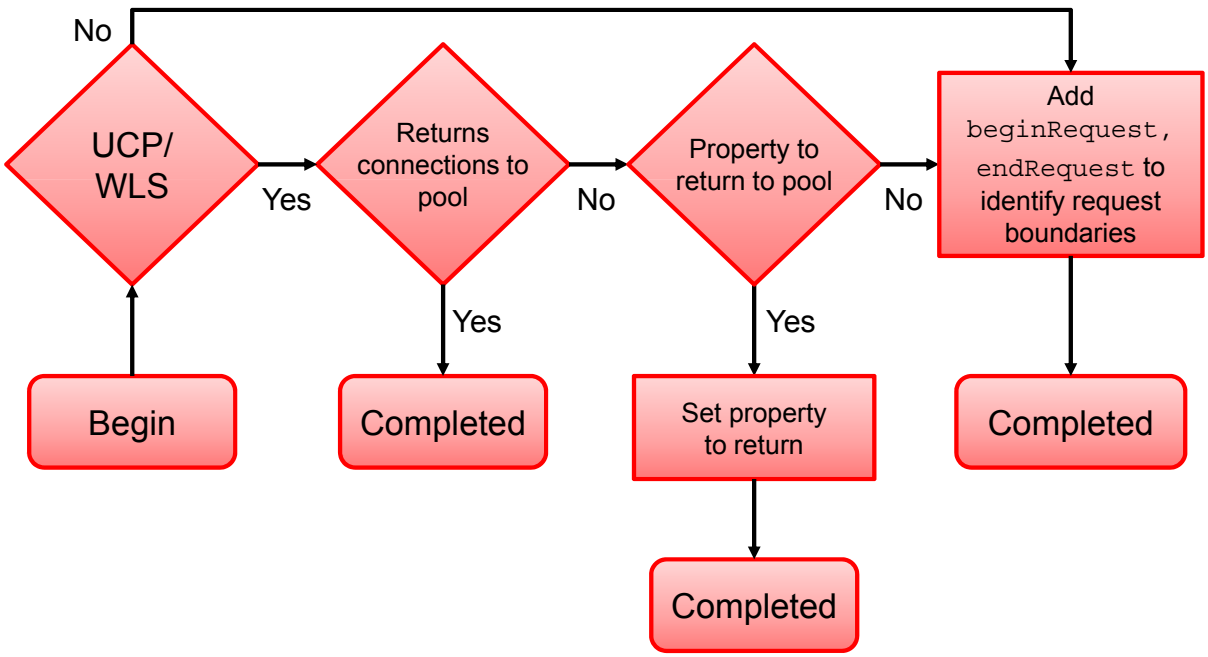
ORACLE

Before using Application Continuity in conjunction with an application, the application owner must check the following:

1. Mark request boundaries if the application is not using Oracle Pools or is not releasing connections to Oracle Pools. Request boundaries tag the beginning and end of each request. Determine whether the application borrows and returns connections from the WebLogic Server Pool or Universal Connection Pool for each request, or whether `beginRequest` and `endRequest` APIs should be added to the application's own connection pool to identify request boundaries. If you are using Oracle Pools, but not releasing connections, consider your options to do so.
2. Determine whether the application uses Oracle JDBC concrete classes. To use Application Continuity, the deprecated concrete classes must be replaced. For information about the deprecation of concrete classes, including actions to take if an application uses them, see My Oracle Support Note 1364193.1.
3. Assess the impact of restrictions and side effects on your application. Use the `disableReplay` API for any request that should not be replayed. For example, if a request makes external calls by using one of the external PL/SQL messaging actions, decide whether the external call should be replayed or not.

4. Assess whether your application sets state outside the database request. If a state is set when a user starts a request and it is outside that request, replay needs to know about it in order to reexecute the calls.
   When using Oracle WebLogic Server or the Universal Connection Pool, connection labeling is recommended. The labeling is used for both run time and replay.
   If you use an application's own callback, register it at the WebLogic Admin Console, or at Universal Connection Pool, or JDBC replay driver levels.
5. Decide whether the application can keep mutables at replay. When a request is replayed, this function obtains a new value every time it is called. Keep mutable values when they are compatible with the application. Keep original values for `seq.NEXTVAL`, `SYSDATE`, `SYSTIMESTAMP`, and `SYS_GUID` during failover.

Handling Request Boundaries



Request boundaries are used to tag the beginning and end of each request. They are required to resume recording following a nonreplayable operation, and also to reclaim the memory used for holding calls.

The flow chart shows how to decide whether request boundaries should be added:

1. Determine whether the application borrows and returns connections from the WebLogic Server Pool or Universal Connection Pool for each request, or whether `beginRequest` and `endRequest` APIs should be added to the application's own connection pool to identify request boundaries.
2. If you are using Oracle Pools, but you are not releasing connections to the pool, there is often a property to be set to release connections. Releasing connections scales much better and automatically embeds the request boundaries. This allows planned and unplanned failover support and better load balancing.

Handling Request Boundaries: Example

```
doSQL(con, "CREATE SEQUENCE seq01 keep");

((oracle.jdbc.replay.ReplayableConnection)con).beginRequest();
System.out.println("\ncon.beginRequest()");

con.setAutoCommit(false);
System.out.println("\ncon.getAutoCommit()="+con.getAutoCommit());

String q = "select c1, seq01.NEXTVAL from test_tab";
System.out.println("\nExecute - "+q);

Statement s = con.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
  ResultSet.CONCUR_READ_ONLY);
ResultSet r= s.executeQuery(q);

while (r.next())
{
  System.out.println("c1="+r.getInt(1)+", seq="+r.getString(2));
}

r.close();
s.close();

((oracle.jdbc.replay.ReplayableConnection)con).endRequest();
System.out.println("\ncon.endRequest()");
```

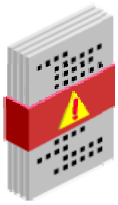
ORACLE

The slide shows an example code fragment. In the example, the request boundary API calls are highlighted in red. If your application manages its own connection pool, these boundaries are best added at connection checkout (get) and connection release (close).

An application should use the Application Continuity APIs to mark request boundaries only if the connection pool or container does not use an Oracle connection pool or cannot release connections to the pool.

Disabling Replay by Using the `disableReplay` API

- Use the `disableReplay` API for requests that should not be replayed.
- Make a conscious decision to replay external actions.
  - Autonomous transactions, `UTL_HTTP`, `UTL_URL`, `UTL_FILE`, `UTL_FILE_TRANSFER`, `UTL_SMTP`, `UTL_TCP`, `UTL_MAIL`, `DBMS_PIPE`, and `DBMS_ALERT`
- In addition, consider disabling replay when the application:
  - Synchronizes independent sessions
  - Uses time at the middle-tier in a logic execution
  - Assumes that `ROWID` values do not change
  - Assumes that location values do not change



By default, the JDBC replay driver replays following a recoverable error. If the application has requests that the application does not want to be repeated, the application can explicitly call the `disableReplay` API to disable replay for those requests.

Application developers should make a conscious decision to allow replay for external actions, and especially for procedures that use the UTL_HTTP package. If they should not be replayed, use the `disableReplay` API.

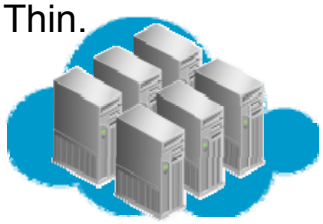
Additionally, it is recommended that you consider disabling replay when the application:

- Synchronizes independent sessions
- Uses time at the middle-tier in a logic execution
- Assumes that `ROWID` values do not change
- Assumes that location values do not change

This is because the assumptions contained in these scenarios may not be valid during replay, potentially yielding incorrect results. For further discussion on these scenarios, refer to the chapter entitled Ensuring Application Continuity in the *Oracle Database Development Guide 12c Release 1 (12.1)*.

Connection Initialization Options

- No callback
  - The application builds up its own state during each request.
- Connection labeling
  - Performance is improved by avoiding some initialization.
  - It is available with UCP or WebLogic Server.
  - The replay driver uses Connection Labeling when present.
- Connection initialization callback
  - The replay driver uses callback to set the initial state of the session at run time and replay.
  - Register with WebLogic, UCP, and JDBC Thin.



Nontransactional session state (NTSS) is state of a database session that exists outside transactions and is not protected by recovery. For applications that use stateful requests, the nontransactional state is reestablished as the session is rebuilt by Application Continuity.

For applications that set state only at the beginning of a request, or for stateful applications that gain performance benefits from using connections with a preset state, choose one of the following callback options:

- **No callback:** In this scenario, the application builds up its own state during each request.
- **Connection labeling:** Connection Labeling is a generic pool feature that is recommended for its excellent performance. When Connection Labeling is present, Application Continuity uses it. Connection Labeling is a feature of Universal Connection Pool (UCP) and Oracle WebLogic server. The application can be modified to take advantage of the preset state on connections. Connection Labeling APIs determine how well a connection matches, and use the labeling callback to populate the gap when a connection is borrowed. Not all applications can use Connection Labeling, because it requires some application recoding.

- **Connection initialization callback**: In this scenario, the replay driver uses an application callback to set the initial state of the session during run time and replay. The JDBC replay driver provides an optional connection initialization callback interface and methods to register and unregister connection initialization callbacks in the `oracle.jdbc.replay.OracleDataSource` interface.

When registered, the initialization callback is executed at each successful checkout at run time and each reconnection following a recoverable error. If the callback invocation fails, replay is disabled on that connection. Use the connection initialization callback only when the application has not implemented Connection Labeling, and needs state information that is not established at the request.

Callback registration is performed by using the WebLogic Admin Console, in UCP, or in the JDBC Thin Replay driver.

Mutable Objects and Application Continuity

- Requests using mutable functions can fail to replay if the function result changes.
- When a mutable privilege is granted, replay applies the original function result.

Mutable Function	Application A	Application B	Application C
SYSDATE, SYSTIMESTAMP	Original	Current	Not Applicable
SEQUENCE.NEXTVAL, SEQUENCE.CURRVAL	Original	Original	Current
SYS_GUID	Original	Not Applicable	Not Applicable

A mutable object is a function that obtains a new value every time it is called. An example of a mutable is a call to the `SYSTIMESTAMP` function. When a request is replayed, the default and desired treatment of mutable objects can vary. Applications using Application Continuity can determine whether to keep the original values from mutable functions if the request is replayed.

Support for keeping mutable object values is currently provided for `SYSDATE`, `SYSTIMESTAMP`, `SYS_GUID`, and `SEQUENCE.NEXTVAL`. If the original values are not kept and if different values for these mutable objects are returned to the client, replay is rejected because the client sees different results.

The table in the slide shows examples of the different approaches that different applications might use for mutable objects. No approach is recommended over another. They are presented only to illustrate some of the possibilities that might reasonably exist.

Keeping Mutable Objects for Replay

- SQL commands:

```
SQL> GRANT [KEEP DATE TIME | KEEP SYSGUID] TO <user>;
SQL> REVOKE [KEEP DATE TIME | KEEP SYSGUID] FROM <user>;
SQL> GRANT KEEP SEQUENCE ON <sequence_name> TO <user>;
SQL> REVOKE KEEP SEQUENCE ON <sequence_name> FROM <user>;
SQL> CREATE SEQUENCE <sequence_name> [KEEP|NOKEEP];
SQL> ALTER SEQUENCE <sequence_name> [KEEP|NOKEEP];
```

- Examples:

```
SQL> CREATE SEQUENCE new_seq KEEP;
SQL> ALTER SEQUENCE existing_seq KEEP;
SQL> GRANT KEEP SEQUENCE on sales.seq1 TO user2;
SQL> GRANT KEEP DATE TIME TO user2;
SQL> GRANT KEEP SYSGUID TO user2;
```


The slide shows the SQL commands that you can use to configure mutable objects for replay. The following notes elaborate further:

- The database user running the application may have the `KEEP DATE TIME` and `KEEP SYSGUID` privileges granted, and the `KEEP SEQUENCE` object privilege on each sequence whose value is to be kept. Note that these privileges cannot be granted using `GRANT ALL ON <object>`. In addition, it is not recommended to grant DBA privileges to database users running applications for which you want replay to be enabled. Grant only privileges that are necessary for such users.
- Sequences may be defined with the `KEEP` attribute, which keeps the original values of `SEQUENCE.NEXTVAL` for the sequence owner. Note that specifying `KEEP` or `NOKEEP` with `CREATE SEQUENCE` or `ALTER SEQUENCE` applies to the owner of the sequence.
- If granted privileges are revoked between run time and failover, the mutable values that are collected are not applied for replay.
- If new privileges are granted between run time and failover, mutable values are not collected and, therefore, cannot be applied for replay.

Configuring the JDBC Replay Data Source

Use the **`oracle.jdbc.replay.OracleDataSourceImpl`** data source to obtain JDBC connections.

- Configure the data source in the property file for UCP or WebLogic Server.
- Reference the data source in JDBC Thin applications.

To use Application Continuity for Java, you must use the `oracle.jdbc.replay.OracleDataSourceImpl` data source to obtain JDBC connections. This data source supports all the properties and configuration parameters of all the Oracle JDBC data sources (for example, `oracle.jdbc.pool.OracleDataSource`).

You can configure the replay data source by changing the data source property for Universal Connection Pool or by setting it using the WebLogic Console. You can also set the replay data source for JDBC Thin applications in the property file or directly in the application.

Configuring Database Services for Application Continuity

Example:

```
$ srvctl add service -db orcl -service acservice
      -serverpool ora.orcldb
      -cardinality singleton
      -failovertype TRANSACTION
      -commit_outcome TRUE
      -clbgoal SHORT
      -rlbgoal SERVICE_TIME
      -failoverretry 50
      -failoverdelay 5
      -retention 86400
      -replay_init_time 1800
      -notification TRUE
```

Mandatory Settings for Application Continuity

Recommended Settings for Application Continuity with Oracle RAC

Optional Settings for Application Continuity

To use Application Continuity, set the following mandatory service attributes:

- `-failovertype TRANSACTION` enables Application Continuity
- `-commit_outcome TRUE` enables Transaction Guard

You can also set the following attributes:

- `-clbgoal` specifies the connection load balancing mode. `SHORT` is the recommended setting for Application Continuity in conjunction with Oracle RAC.
- `-rlbgoal` specifies runtime load balancing goal. `SERVICE_TIME` is the recommended setting for Application Continuity in conjunction with Oracle RAC.
- `-failoverretry` specifies the number of connection retries for each replay attempt. If replay does not start within the specified number of retries, it is abandoned.
- `-failoverdelay` specifies the delay in seconds between connection retries.
- `-retention` specifies the number of seconds that the commit outcome is retained. The default is 86400 (24 hours), and the maximum is 2592000 (30 days).
- `-replay_init_time` specifies the number of seconds within which replay must start. If replay does not start within the specified time then it is abandoned.
- `-notification TRUE` enables Fast Application Notification (FAN).

Resource Requirements for Application Continuity

- Application Continuity
  - Java client
    - Increase memory to maintain replay queues.
    - Additional CPU is consumed for garbage collection and to build proxies.
  - Database server
    - Additional CPU is required for validation.
    - CPU overhead is minimal on current Intel and SPARC CPUs where validation is hardware-assisted.
- Transaction Guard
  - Built into the Database kernel
  - Minimal overhead
  - Excellent scalability

Application Continuity has the following resource requirements:

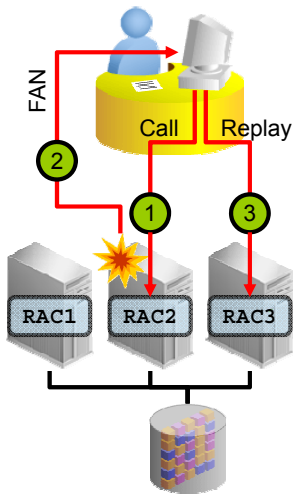
- **CPU:** Application Continuity operates on both the database server and application client and needs additional CPU resources to operate. Application Continuity consumes CPU resources on the database server to perform validation during replay. Note that CPU overhead is reduced on platforms with current Intel and SPARC CPUs where validation is assisted in hardware (using Cyclic Redundancy Checks [CRCs]). On the client side, Application Continuity uses some CPU resources above the base driver. Additional CPU costs are incurred for building proxy objects and garbage collection.
- **Memory:** The JDBC replay driver requires more memory than the base driver because the calls are retained until the end of a request. At the end of the request, the calls are released to the garbage collector. This action differs from the base driver, which releases closed calls.

The memory consumption of the replay driver depends on the number of calls per request. If this number is small, the additional memory consumption of the replay driver is less and is comparable to the base driver.

To obtain the best performance, set the same value for both the `-Xmx` and `-Xms` parameters on the client. This sets the initial heap size and maximum heap size to the same size ensuring that JVM performance is not impacted by attempts to use lower settings.

Application Continuity and Oracle RAC

- With Oracle RAC, Application Continuity can replay requests on another RAC instance
- Using Application Continuity with RAC provides:
  - Protection against a wider variety of failure scenarios
  - Faster reconnect and replay



ORACLE

When Application Continuity is used in conjunction with Oracle RAC, failed sessions can be quickly restarted and replayed on another RAC database instance. Using Application Continuity in conjunction with Oracle RAC provides protection against a wider variety of possible failures compared with using Application Continuity against a single instance database. In addition, using Application Continuity in conjunction with Oracle RAC enables quicker replay because the client can immediately reconnect to another instance rather than waiting for the database to be restarted or waiting for a Data Guard failover operation.

Quiz

Application Continuity attempts to mask recoverable database outages from applications and users by restoring database sessions and replaying database calls.

a. True
b. False

Answer: a

Quiz

Which of these is not one of the phases in Application Continuity?

a. Run time
b. Replay
c. Reconnect
d. Restore

Answer: d

Quiz

Select the statements that apply to Application Continuity:

a. Application Continuity is supported with thin JDBC clients.
b. The replay target database must have the same database ID, ancestors, and descendants as the source database.
c. Replay is not supported if the transaction executes the `ALTER SYSTEM` or `ALTER DATABASE` statement.
d. Replay is supported for applications developed by using Oracle XA.
e. Replay is supported if you are using Active Data Guard with read/write database links to another database.

ORACLE

Answer: a, b, c

Summary

In this lesson, you should have learned how to:

- Describe the purpose of Transaction Guard and Application Continuity
- Describe the key concepts relating to Application Continuity
- Describe the side effects and restrictions relating to Application Continuity
- Describe the requirements for developing applications that leverage Application Continuity
- Configure Application Continuity

Practice 7 Overview: Using Application Continuity

In this practice, you will use Application Continuity against a RAC database to demonstrate how Application Continuity helps an application to seamlessly recover after an the failure of a RAC instance.

8

Oracle Global Data Services

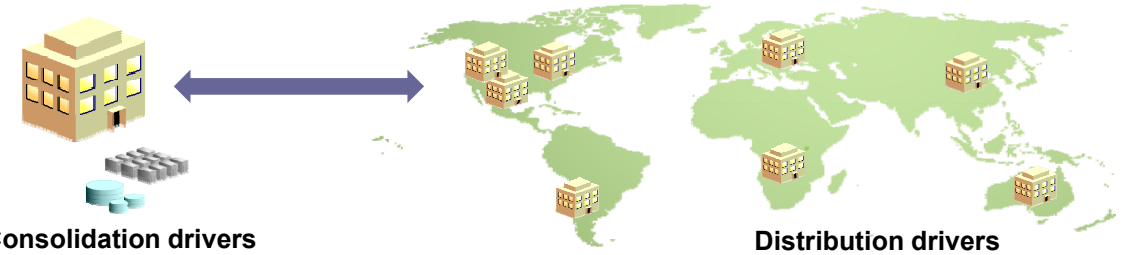
Objectives

After completing this lesson, you should be able to:

- Explain the benefits provided by Global Data Services for managing cloud-deployed distributed databases
- List the components of the Global Services Framework
- Describe the process of Global Services failover

Global Data Consolidation

- Many organizations are going through consolidation of their IT infrastructure to improve business efficiency.
  - It is impossible to achieve database consolidation solely by aggregating hundreds of databases in a few large ones.
- These organizations still need to maintain multiple replicas of their databases both locally and in geographically disparate regional data centers.



Consolidation drivers
- Efficiency / integration
- Manageability
- Cost reduction

Distribution drivers
- Performance / load balancing
- Disaster recovery
- Mergers and acquisitions

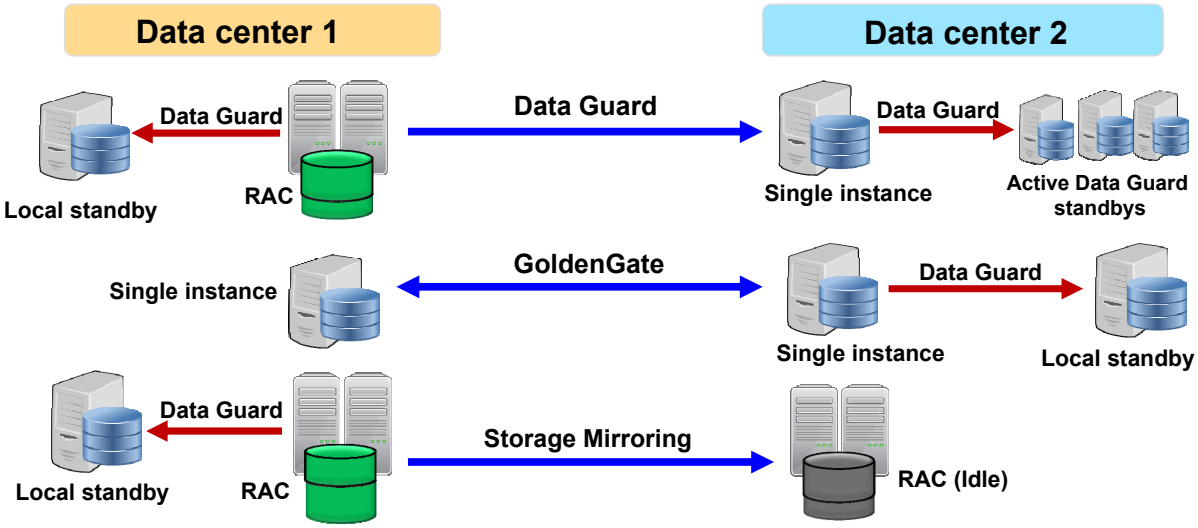
ORACLE

Many large enterprises consolidate their information technology infrastructure to improve business efficiency, and database consolidation is a critical part of this process. In today's global economy, it is impossible to achieve database consolidation only by aggregating hundreds of databases into a few large databases or collocating them in a single data center, or a combination of both solutions.

These enterprises often use technologies such as Oracle Active Data Guard and Oracle GoldenGate for their disaster recovery and replication needs. Distributing workload over multiple databases can bring data closer to clients, improve performance and scalability, and get more value out of IT assets. However, when applications are spread across multiple databases and potentially also across data centers, it can be challenging to efficiently use all your databases for best performance and availability.

Oracle Global Data Services

- Global Data Services (GDS) for database clouds applies the RAC service model to sets of globally distributed databases.
- GDS works with a single instance or RAC databases using Data Guard, GoldenGate, or other replication technologies.



Global Data Services for database clouds applies the Oracle RAC service model to sets of globally distributed, heterogeneous databases, providing dynamic load balancing, failover, and centralized service management for a set of replicated databases that offer common services. The set of databases can include Oracle RAC and noncluster Oracle databases interconnected through Oracle Data Guard, Oracle GoldenGate, or any other replication technology.

Features of Global Data Services enable you to integrate your locally and globally distributed, loosely coupled databases running on heterogeneous platforms into a scalable and highly available private database cloud that can be shared by clients around the globe.

Distributed database systems, in most cases, do not maintain absolute data consistency across replicas. Therefore, not all services that can currently run on multiple instances of an Oracle RAC database can be scaled to run in a multidatabase environment. Global Data Services is primarily intended for applications that are replication aware, use read-only services, or both. Applications that cannot be modified to work with replicated data can still benefit from improved high availability and disaster-recovery capabilities of a distributed database system.

The Global Data Services Framework

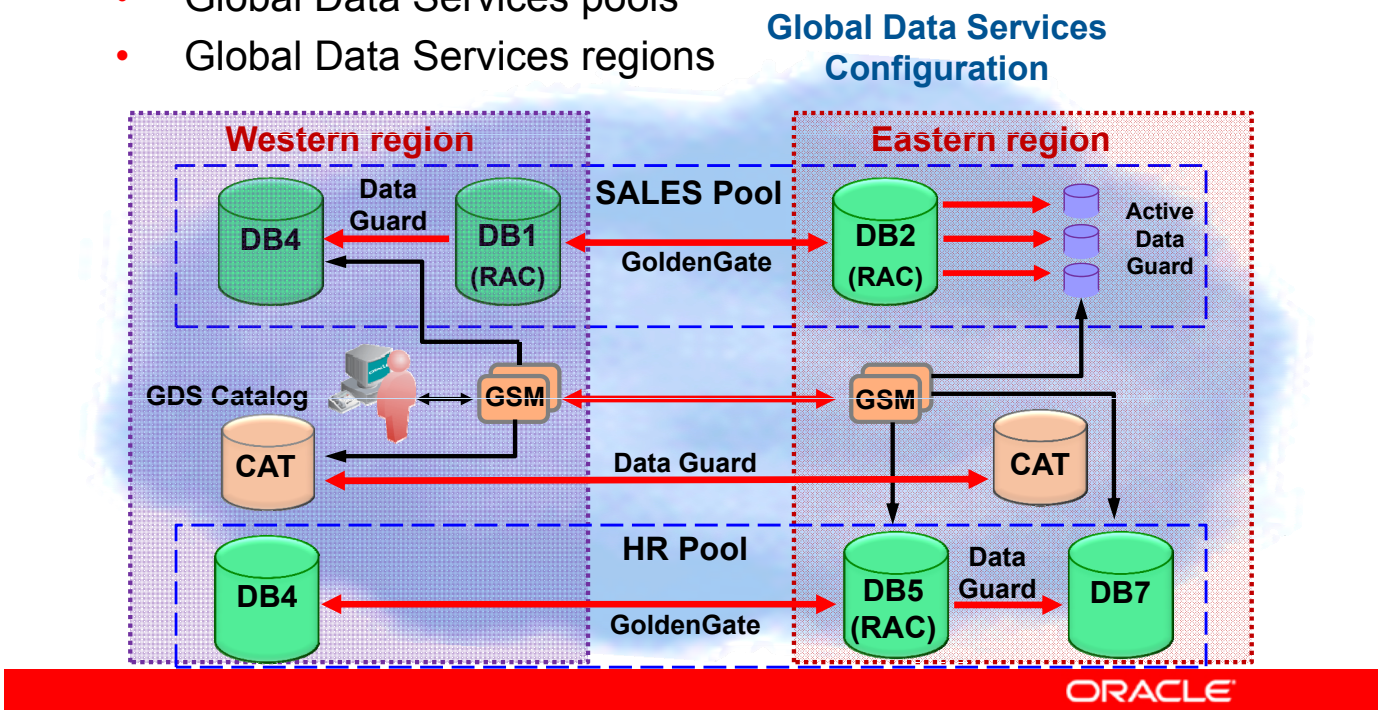
- Global Data Services employs a distributed framework, which:
  - Automates and centralizes configuration, maintenance, and monitoring of a GDS environment
  - Enables load balancing and failover for services
- The framework includes logical and physical components
  - Logical components include:
    - The Global Data Services configuration
    - Global Data Services pools
    - Global Data Services regions
  - Physical components include:
    - Global Service Manager
    - Global Data Services catalog
    - Oracle Notification Service servers
    - Global Data Services control utility (`gdsctl`)

Global Data Services uses a distributed framework that automates and centralizes configuration, maintenance, and monitoring of a Global Data Services configuration, and enables load balancing and failover for services provided by Global Data Services.

The Global Data Services framework includes logical and physical components. Logical components refer to a group of components that share the same property, including the Global Data Services configuration, Global Data Services pools, and Global Data Services regions. Physical components are the global service manager, the Global Data Services catalog, databases, Oracle Notification Service servers, and the Global Data Services control utility, `gdsctl`.

Logical Global Data Services Components

- The Global Data Services configuration
- Global Data Services pools
- Global Data Services regions



The logical components of Oracle Global Data Services include the Global Data Services configuration itself and the Global Data Services pools and Global Data Services regions contained within the GDS environment. The figure above shows the relationships between the Global Data Services framework logical components using an example of a configuration containing two Global Data Services pools (Sales and HR) and two Global Data Services regions (West and East).

Logical Global Data Services Components: The Global Data Services Configuration

- A Global Data Services configuration is a set of databases that provide global services.
- Each Global Data Services configuration has a name that the global services administrator can specify.
- The default name is **`oradbcloud`**.
- A name can contain up to 30 bytes, including valid identifiers.

A Global Data Services configuration is a set of databases that provide global services. The Global Data Services framework is a self-contained system, in which all databases contained within a Global Data Services configuration are managed by Global Data Services components that belong to the same GDS environment.

Every Global Data Services configuration has a name that global service manager can specify. The default name is `oradbcloud`. A name can contain up to 30 bytes, including valid identifiers (an alphabetical character followed by zero or more alphanumeric ASCII characters, the underscore “_”, or the number sign “#”, and possibly separated by periods if there are multiple identifiers).

Logical Global Data Services Components: Global Data Services Pool

- Each Global Data Services pool contains replicated databases that provide a common set of global services.
- A database can only belong to a single Global Data Services pool.
- It is not necessary that all databases in a pool provide the same set of global services.
  - An individual database can support only a subset of services provided by the pool.
  - All databases that provide the same global service must belong to the same pool.
- A Global Data Services pool must have a unique name within the GDS configuration.
  - The pool name can be up to 30-bytes long.
  - The default name is **`oradbpool`**.

ORACLE

A Global Data Services pool is a named subset of databases within a GDS configuration that provides a unique set of global services and belongs to the same administrative domain. Partitioning of GDS configuration databases into pools simplifies service management and provides higher security by allowing each pool to be administered by a different administrator.

A database can only belong to a single Global Data Services pool. All databases in a pool need not provide the same set of global services. However, all databases that provide the same global service must belong to the same pool.

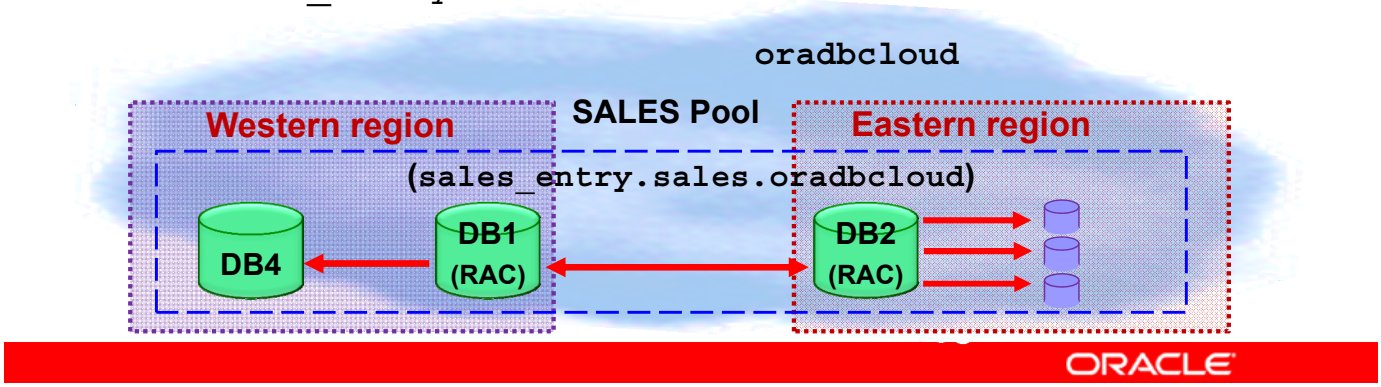
Within a Global Data Services configuration, there can be many different pools of databases that do not share anything other than Global Data Services components. Each Global Data Services pool contains replicated databases that provide a common set of global services and that belong to the same administrative domain.

A database can only belong to a single Global Data Services pool. It is not necessary that all databases in a pool provide the same set of global services; an individual database can support only a subset of services provided by the pool. However, all databases that provide the same global service must belong to the same pool.

A Global Data Services pool must have a unique name within the GDS configuration. If you do not specify a name for the pool when you create it, then the name defaults to `oradbpool`. The pool name can be up to 30-bytes long and can be any valid identifier (an alphabetical character followed by zero or more alphanumeric ASCII characters or the underscore "_").

Logical Global Data Services Components: Global Services

- A global service is provided by a set of databases that belong to the same Global Data Services pool.
- A global service must have a unique name.
- If a global service name is not fully qualified at creation, then *pool_name.cloud_name* is the default domain.
- In the example below, the service `sales_entry` would be given the fully qualified name: `sales_entry.sales.oradbcloud`



A global service is provided by a set of databases that belong to the same Global Data Services pool. However, because services in all pools of a Global Data Services configuration are managed by the same Global Data Services framework components, a global service must have a unique name within the GDS configuration. Because pool administrators create services independently, to provide uniqueness, if you attempt to create a fully qualified service name (which includes a service name and a domain) that already exists as a global service in a different pool in the same configuration, then an error results.

If a global service name is not qualified with a domain when you create it, then, instead of using the database domain as the default domain, as is normal for local services, Oracle uses *pool_name.cloud_name* as the default domain. So, a partially qualified service name gets created as the fully qualified name *service_name.pool_name.cloud_name*.

Logical Global Data Services Components: Global Data Services Region

- A Global Data Services region usually corresponds to a data center or LAN.
- A Global Data Services configuration can span one or more Global Data Services regions.
- A region can have databases belonging to different pools, but the pools should belong to the same GDS configuration.
- A Global Data Services region name should be unique within the corresponding Global Data Services configuration.
- If no name is specified at the first region creation time, the name defaults to `oraregion`.

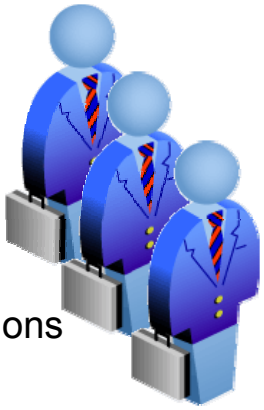
A Global Data Services region is a named subset of databases in a GDS configuration and database clients that share network proximity such that the network latency between members of a region is typically lower than between members of different regions. A region usually corresponds to a local area or metropolitan area network. For example, a data center hosting one or more GDS configuration databases and database clients in geographical proximity to the data center might belong to the same region.

A Global Data Services configuration can span one or more Global Data Services regions. A region can contain databases that belong to different Global Data Services pools, but the pools should belong to the same Global Data Services configuration.

For high-availability purposes, each region in the Global Data Services framework has a buddy region. A Global Data Services region should have a name that is unique within the corresponding Global Data Services configuration. If no name is specified at the first region creation time, the default name, `oraregion`, is given to the region. The region name can be up to 30-characters long and can be any valid identifier: an alphabetical character followed by zero or more alphanumeric ASCII characters or "_".

For high-availability purposes, each region in a GDS configuration should have a designated buddy region, which is a region that contains global service managers that can provide continued access to a GDS configuration if the global services managers in the local region become unavailable.

Physical Global Data Services Components: Global Service Manager



- The global service manager (GSM) is the central component of Global Data Services.
- The GSM performs the following tasks:
  - Acts as a regional listener connecting clients to global services
  - Measures network latency between its own Global Data Services region and all other regions
  - Performs connection load balancing
  - Monitors database instances, and generates and publishes Fast Application Notification (FAN) runtime load-balancing events
  - Maintains Global Data Services framework configuration
  - Manages cardinality and failover of global services
- Each GSM in a Global Data Services configuration manages all of the global services in that configuration.

ORACLE

The global service manager (GSM) is the central component of Global Data Services, Every global service manager in a Global Data Services configuration manages all of the global services that the configuration provides. A global service manager can be associated with only one Global Data Services configuration. There must be at least one global service manager for each Global Data Services region but, typically, you configure more than one for high availability and improved performance. If there are multiple Global Data Services pools in a GDS configuration, then they share all of the global service managers that belong to the configuration. The GSM performs the following tasks:

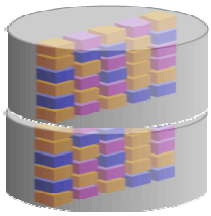
- Acts as a regional listener used by clients to connect to global services. Clients in a Global Data Services region establish database connections to local and remote databases that provide global services through regional global service managers. Client connection requests are randomly distributed among all regional GSMs. A client can connect to any of the GSMs, which redirects the client to an appropriate database according to a connection load-balancing policy. However, when all GSMs in a region are down, global service managers in the buddy region start acting as listeners for the region with failed global service managers. Therefore, the clients' connection string should contain global service managers from the local and buddy regions.
- Measures network latency between its own Global Data Services region and all other regions, and exchanges this information with global service managers in other regions.

- Performs connection load balancing by directing a client connection to the most appropriate database instance. The global service manager determines the appropriate instance based on connection load-balancing metrics that the global service manager receives from all instances, estimated network latency, and region affinity of the service.
- Monitors database instances, and generates and publishes FAN runtime load-balancing events for clients in the local Global Data Services region by combining and normalizing runtime load balancing metrics that it receives from all instances and integrating them with estimated network latency. Each global service manager also generates FAN events when it detects that other global service managers start or stop. When all global service managers in a region are down, runtime load-balancing events are generated by global service managers in the buddy region.
- Maintains Global Data Services framework configuration by making changes to configuration data in all of the Global Data Services framework components and verifying mutual consistency of the data across components.
- Manages cardinality and failover of global services by starting, stopping, and moving the services among instances and databases.

A GSM is associated with one and only one GDS configuration. Each region in the GDS configuration must have at least one GSM. It is recommended that multiple global service managers be configured in each region to improve availability and performance. Every global service manager in a GDS configuration manages all global services supported by the configuration. All GSMs in a Global Data Services region receive runtime load-balancing metrics from all databases in their Global Data Services configuration and measure network latency between regions.

Physical Global Data Services Components: Global Data Services Catalog

- The Global Data Services catalog stores:
  - Configuration data for a Global Data Services configuration
  - All global services provided by the configuration
- There can be only one catalog per GDS configuration.
- A Global Data Services catalog resides in an Oracle Database 12*c* database.



- The catalog can reside on a database outside the GDS configuration.
- You can use existing high availability (HA) technologies, such as RAC, Data Guard, and Oracle Clusterware, to protect the catalog.
- If GoldenGate is used, then ensure that the Global Data Services catalog gets replicated to a secondary database.

ORACLE

The Global Data Services catalog is a repository that stores configuration data for a Global Data Services configuration and all global services provided by the configuration. A Global Data Services catalog must be associated with a particular Global Data Services configuration, and there can only be one Global Data Services catalog per configuration. A Global Data Services catalog resides in an Oracle Database 12*c* database and can be any one of the databases within the Global Data Services configuration. The Global Data Services catalog can also reside on a database outside the Global Data Services configuration.

It is strongly recommended that high availability technologies such as Oracle RAC, Oracle Data Guard, and Oracle Clusterware be used to enhance the availability of the database where the Global Data Services catalog resides.

Physical Global Data Services Components: Databases

- A global service is provided by a set of databases residing in the same Global Data Services pool.
- The GDS framework makes the pool appear to clients as a single database with many instances.
- Each database in a Global Data Services configuration must be associated with:
  - One Global Data Services region
  - One Global Data Services pool
  - One Global Data Services configuration

A global service is provided by a set of databases residing in the same Global Data Services pool. Although a Global Data Services pool consists of multiple databases, the GDS framework makes the pool appear to clients as a single database with many instances. Each database in a Global Data Services configuration must be associated with at the most one Global Data Services region, one Global Data Services pool, and one Global Data Services configuration.

Physical Global Data Services Components: Oracle Notification Servers

- An Oracle Notification Server (ONS) runs with each GSM, delivering FAN events and runtime load-balancing metrics to clients.
- The GSM connects to each database, detects FAN events, and publishes the events to the collocated ONS server.
- RAC ONS servers are only for local services and they are not connected to the ONS servers running with the GSM.
  - Because of this, clients of global services do not subscribe to local ONS servers.
- The GSM connects to each database, detects FAN events, and publishes the events to the collocated ONS servers.
- For each GDS region, all GSMs and ONS servers are fully connected but only the primary GSM publishes FAN events.

ORACLE

In a Global Data Services configuration, an Oracle Notification Service (ONS) server runs with each global service manager, and these servers deliver FAN events and runtime load-balancing metrics to clients. The difference between these ONS servers and those running within Oracle RAC databases is that the ONS servers running in Oracle RAC are only for local services, and they are not connected to the ONS servers running with the global service managers. As a result, clients of global services do not subscribe to local ONS servers.

The global service manager connects to each database, detects FAN events, and publishes the events to the collocated ONS server. For each Global Data Services region, all global service managers and collocated ONS servers are fully connected, but only the primary global service manager publishes FAN events. Global Data Services clients connect to ONS servers of all global service managers in their region and its buddy region; however, ONS servers located in different Global Data Services regions are not connected.

Physical Global Data Services Components: The `gdsctl` Utility

- The `gdsctl` utility provides a command-line interface for configuring and managing the GDS framework.
- To execute a command, `gdsctl` may need to establish a connection to:
  - A global service manager
  - A Global Data Services catalog database
  - A database in the Global Data Services configuration
- Unless specified, `gdsctl` resolves connect strings with the current name resolution methods such as TNSNAMES.
  - The exception is the GSM name that `gdsctl` resolves by querying the `gsm.ora` file.

The `gdsctl` utility provides a command-line interface for configuring and managing the Global Data Services framework. To execute a command, `gdsctl` may need to establish a connection to a global service manager, a Global Data Services catalog database, or a database in the Global Data Services configuration.

Global Service: Overview

- For database clients, a Global Data Services configuration is represented by a set of global services.
- A GSM serving a Global Data Services configuration is aware of all global services provided by the configuration.
  - It acts as a mediator between database clients and databases in the GDS configuration.
- A client program connects to a regional global service manager and requests a connection to a global service.
- The GSM forwards the client's request to the optimal instance in the GDS configuration that offers the global service.
- The configuration and runtime status of global services are stored in the Global Data Services catalog.

ORACLE

For database clients, a Global Data Services configuration is represented by a set of global services. A global service manager serving a Global Data Services configuration is aware of all global services that the GDS configuration provides and acts as a mediator between database clients and databases in the GDS configuration. A client program connects to a regional global service manager and requests a connection to a global service. The client does not need to specify which database or instance it requires. The global service manager forwards the client's request to the optimal instance in the GDS configuration that offers the global service. Database clients that share a global service must have the same service-level requirements.

The functionality of local services is not changed by global services. Oracle Database 12*c* can provide local and global services, simultaneously. A client application can also work with global and local services, simultaneously.

The configuration and runtime status of global services are stored in the Global Data Services catalog. Each database that offers global services also maintains information about those services in a local repository (such as a system dictionary or Oracle Cluster Registry), together with data on local services. Global services that are stored in a local repository have a special flag to distinguish them from traditional local services.

Note: Databases earlier than Oracle Database 12*c* can provide local services, but only Oracle Database 12*c*, and later, can provide global services.

Global Service: Overview

- The GSM manages the global services according to the service properties you specify when you create them.
- When a database joins the GDS configuration or restarts after a shutdown, it registers with all GSMs.
- When a registration request is received, the master GSM queries the catalog and starts the specified services.
- If a global service is not up on a database, it cannot be started automatically by a database instance or Clusterware.
  - It can be started with `gsdctl`, or the GSM can automatically enable a database to offer a global service.
- For a RAC database, you can:
  - Specify a server pool where the global service will be enabled
  - Assign specific instances to Global services

ORACLE

If you are locally connected to a particular database, then you can query data on global services provided by that database, but you cannot configure, modify, start, or stop a global service using either the Server Control utility `srvctl` or the Oracle Clusterware Control utility, `crsctl`. You can perform these operations using `gdsctl` when you are connected to the Global Data Services catalog. This ensures centralized coordinated management of global services. After you configure global services for a Global Data Services configuration, the global service manager manages the global services on GDS configuration databases according to the service properties you specify when you create them.

When a database joins the GDS configuration or restarts after a shutdown, it registers with all global service managers in the GDS configuration. After the master global service manager receives the registration request, it queries the Global Data Services catalog and starts the appropriate services on the database.

If a global service is not running on a database, then it cannot be started automatically by either a database instance or Oracle Clusterware. You can manually start the service using `gdsctl`, or the global service manager can automatically enable a database to offer a global service. For an Oracle RAC database, you can specify a server pool where the global service will be enabled.

However, the Global Data Services framework does not control which particular instances within the pool offer the enabled service. This is controlled by the clusterware and the administrator of the Oracle RAC database.

When a Global Data Services database instance fails, all global service managers in the configuration get notified about it and stop forwarding requests to the instance. If this instance belongs to a noncluster database, or it is the last instance that was available in an Oracle RAC database, then, depending on the configuration, a global service manager may automatically start the service on another database in the Global Data Services pool where the service is enabled. If you decide to manually move a global service from one database to another using the appropriate `gdsctl` command, then the global service manager stops and starts the service on the corresponding databases.

Global Service Attributes

- Global services attributes control:
  - Global service startup
  - Load-balancing connections to the global services
  - Failing over those connections
- Local service attributes, including those specific to RAC and Data Guard, are also applicable to global services.
- Attributes unique to global services include:
  - Preferred or available databases
  - Replication lag
  - Region affinity
- You can enable and disable, move, and change the properties of a global service just like a local service.

ORACLE

Global services have a set of attributes associated with them that control starting the global services, load balancing connections to the global services, failing over those connections, and more. Attributes applicable to local services, including those specific to Oracle RAC and Oracle Data Guard broker environments, are also applicable to global services.

The following attributes are unique to global services:

- Preferred or available databases
- Replication lag
- Region affinity

You can modify global services as you can with local services. You can enable and disable a global service, you can move the global service to a different database, and you can change the properties of the global service.

Note: You cannot upgrade a local service to a global service.

Global Services in a RAC Database

- Some properties of a global service are only applicable to RAC databases and are unique for each RAC database.
- These properties are related to placement of global services with instances within a RAC database, including:
  - Server pools and service cardinality
  - Instance assignment (for Admin-managed databases)
  - Distributed transaction processing
- You can specify attributes for these properties using the `srvctl` utility.
- Global Data Services supports policy-managed and administer-managed RAC databases only.

Some properties of a global service are only applicable to RAC databases and are unique for each RAC database included in a GDS configuration. These properties are related to placement of global services with instances within an Oracle RAC database, including:

- Server pools and service cardinality
- Distributed transaction processing

You can specify attributes for these properties using `srvctl`; however, you must manage these properties in an Oracle RAC database. This means that all current and future service placement functionality on Oracle RAC databases will be supported for global services. Local management of database-specific service properties also provides better performance and availability. All other existing global service attributes, like load balancing, role, transparent application failover parameters, and database edition must be the same for all databases offering a global service. You must specify these attributes at the global service level.

By default, in an Oracle RAC environment, a SQL statement executed in parallel can run across all of the nodes in the cluster. The cross-node parallel execution is not intended to be used with GDS load balancing. For an Oracle RAC database in a GDS configuration, it is recommended that you restrict the scope of the parallel execution to an Oracle RAC node by setting the `PARALLEL_FORCE_LOCAL` initialization parameter to `TRUE`.

Global Services in an Data Guard Broker Configuration

- When you include a broker configuration in a GDS configuration, broker configurations are managed as a single unit.
  - Only an entire broker configuration can be added to or deleted from a Global Data Services pool.
  - A broker configuration cannot span multiple pools.
- A database is added to the GDS pool by adding it to the broker configuration using the Data Guard utility `dgmgr`.
- After adding a database to the broker configuration, run this command to synchronize GDS and Data Guard:

```
$ gdsctl sync brokerconfig
```

- Global services can be configured with a role attribute to be active in a specific role such as primary or physical standby.

Oracle Data Guard enables one primary database to be connected to up to 30 standby databases. The Oracle Data Guard broker logically groups these primary and standby databases into a broker configuration that enables the broker to manage and monitor the databases together as an integrated unit. When you include a broker configuration in a Global Data Services configuration, you manage the broker configuration as a single unit. Only an entire broker configuration can be added to or deleted from a Global Data Services pool, and a broker configuration cannot span multiple pools.

If you attempt to add or remove a database that belongs to a broker configuration to or from a Global Data Services pool, then an error occurs. You can only add a database to the Global Data Services pool by adding it to the broker configuration using the Data Guard `dgmgrl` utility. When you add a database to the broker configuration, you must run the `gdsctl sync brokerconfig` command to synchronize Global Data Services and Data Guard.

Conversely, when you remove a database from a broker configuration, it is removed from the Global Data Services pool to which this broker configuration belongs. This is the only way to remove a database from a pool that contains a broker configuration.

You can configure global services with a role attribute to be active in a specific database role, such as primary or physical standby database. If you enable fast-start failover, then the Oracle Data Guard broker automatically fails over to a standby database if the primary database fails.

The global service managers configured to work with Oracle Data Guard broker ensure that the appropriate database services are active and that the appropriate Fast Application Notification (FAN) events are published after a role change. The Global Data Services framework supports the following Oracle Data Guard broker configurations:

- The set of databases in a Global Data Services pool can be either the set of databases that belong to a single broker configuration or a set of databases that do not belong to a broker configuration. You can add a broker configuration only to an empty Global Data Services pool and, if a pool already contains a broker configuration, then, to add a database to the pool, you must add the database to the broker configuration contained in the pool.
- Role-based global services are supported only for database pools that contain a broker configuration.

Database Placement of a Global Service

- You can specify which databases will support a service.
  - These databases are referred to as preferred databases.
- GSM ensures that a global service runs on all preferred databases for which it has been specified.
- The number of preferred databases is referred to as the *database cardinality* of a global service.
- When a global service is added, a list of databases is specified for the service.
- If one of the preferred databases fails to provide a global service, the GSM relocates the service to an available database to maintain the specified database cardinality.

ORACLE

You can specify which databases will support a service. These databases are referred to as preferred databases. The GSM ensures that a global service runs on all preferred databases for which it has been specified. The number of preferred databases is referred to as the database cardinality of a global service. You must specify at least one preferred database for a global service.

When you add or modify a global service, you can specify a list of available databases for this global service. If one of the preferred databases fails to provide a global service, then the global service manager relocates the service to an available database to maintain the specified database cardinality for that service.

In a Global Data Services pool that contains an Oracle Data Guard broker configuration, a role-based global service can be started on a database only if the database is listed as preferred or available for the service and the role attribute of the database corresponds to the role attribute specified for the service. For example, a global service that can run on any database in a broker configuration (as long as the role attribute of the database is set to primary) must have primary specified for its role attribute and have all other databases in the broker configuration with role attributes set to `preferred`.

Note: If you set `preferred_all` for which databases will support a service, then you do not have to explicitly specify preferred or available databases. The `preferred_all` setting implies that all databases in the pool are preferred.

Do not confuse database cardinality of global services with their instance cardinality. Instance cardinality is specified and maintained separately for each Oracle RAC database and is not maintained across databases of a Global Data Services pool.

Global Singleton Services

- A global singleton service is offered on only one database of a Global Data Services pool at a time.
- A global singleton service has a database cardinality of 1.
- A singleton service guarantees that only a primary or master data replica can be updated.
  - This prevents data corruption caused by simultaneously updating multiple replicas.
- A global service failover to a replica not synchronized with the master replica can cause data loss.
  - To prevent automatic failover, do not specify available databases for a global singleton service that can modify data.

ORACLE

A global singleton service is a global service that is offered on only one database of a Global Data Services pool at a time. This global service has a database cardinality equal to 1. A singleton service guarantees that only a primary (master) data replica can be updated, which prevents data corruption caused by simultaneously updating multiple replicas.

In an Oracle Data Guard broker configuration, you can update the primary database or a snapshot standby database, if one exists inside your configuration. Additionally, an Oracle Data Guard broker configuration can only contain one primary database at any time.

It is important to note that a global service failover to a replica that is not synchronized with the master replica can cause data loss. To prevent an automatic failover in such cases, do not specify available databases for a global singleton service that can modify data. Such a service can only be started by the global service manager on a particular database and only a Global Data Services pool administrator can move the service to another database.

Replication Lag and Global Services

- For performance reasons, distributed environments often use asynchronous replication of data between databases.
- This increases the probability of a delay between the time an update is made on a primary and when it appears on a replica.
  - This is known as ***replication lag.***
- GDS enables apps to differentiate global services providing real-time data from those returning out-of-date data.
- For applications that can tolerate a certain degree of lag, you can configure a maximum acceptable lag value.
- A client request can only be forwarded to a replica not lagging behind the primary database longer than the configured lag time for the service.

ORACLE

For performance reasons, distributed database systems often use asynchronous replication of data between databases, which means that there is the possibility of a delay between the time an update is made to data on a primary database and the time this update appears on a replica. When this happens, the replica database lags behind its primary database.

Global Data Services enables applications to differentiate between global services that provide real-time data from services that can return out-of-date data because of replication lag. For applications that can tolerate a certain degree of lag, you can configure a maximum acceptable lag value. For applications that cannot tolerate any replication lag, you can set the lag time for global services to zero. Requests for this global service are forwarded only to a primary database, or to a replica database that is synchronized with the primary database. For applications that cannot tolerate any replication lag, you can set the lag time for global services to zero. Requests for this global service are forwarded only to a primary database, or to a replica database that is synchronized with the primary database.

For many applications, it is acceptable to read out-of-date data as long as it is consistent. Such applications can use global services running on any database replica irrespective of the length of the replication lag time. If you configure the lag time to a value other than zero, then a client request can be forwarded only to a replica database that is not lagging behind the primary database by longer than the configured lag time for the service. Specification of the maximum replication lag is only supported for Active Data Guard configurations.

Global Connection Load Balancing

- A client connecting to a RAC database using a service can take advantage of Oracle Net connection load balancing.
- Clients connecting to a global service are load balanced as necessary, across different databases and regions.
- Global connection load-balancing functionality includes:
  - Client-side load balancing
  - Server-side load balancing
  - Region affinity for global services

ORACLE

When a client connects to an Oracle RAC database using a service, the client can take advantage of the Oracle Net connection load-balancing feature to spread user connections across all the instances that support that service. Similarly, in a Global Data Services configuration, clients connecting to a global service are load balanced, as necessary, across different databases and regions. Discussion of Global connection load balancing includes:

- Client-side load balancing
- Server-side load balancing
- Region affinity for global services

Role-Based Services

- In a GDS pool with a Data Guard broker configuration, the GDS framework supports role-based global services.
- Valid roles are:
  - `PRIMARY`
  - `PHYSICAL_STANDBY`
  - `LOGICAL_STANDBY`
  - `SNAPSHOT_STANDBY`
- A global service is started *only* when the database role matches the role specified for the service.

In a Global Data Services pool that contains an Oracle Data Guard broker configuration, the Global Data Services framework supports role-based global services. Valid roles are `PRIMARY`, `PHYSICAL_STANDBY`, `LOGICAL_STANDBY`, and `SNAPSHOT_STANDBY`. The Global Data Services framework automatically starts a global service only when the database role matches the role specified for the service.

If a database switches roles or fails, then the Oracle Data Guard broker notifies the Global Data Services framework about the role change, and the global service manager ensures that services start according to the new database roles. A global service cannot fail over from a database in one Global Data Services region to a database in another region if the locality parameter is set to `LOCAL_ONLY`, and interregion failover is not enabled.

When a global service fails over, fast connection failover, if enabled on Oracle clients, provides rapid failover of the connections to that global service. The Global Data Services framework, similar to Oracle RAC, uses Fast Application Notification (FAN) to notify applications about service outages. Instead of waiting for the application to poll the database and detect a problem, clients receive FAN events and react, immediately. Sessions to the failed instance or node will be terminated, and new connections will be directed to available instances providing the global service.

All global service managers monitor service availability on all databases in a Global Data Services configuration. When a global service cannot be provided anymore because of a failure, the GSM that detects that a global service is unavailable and connects to the Global Data Services catalog database, tries to start the service on an available database.

Note: A GSM cannot automatically fail over a service if it is unable to connect to the Global Data Services catalog.

Quiz

Which statements regarding Global Data Services are true?

a. Global Data Services applies the RAC service model to sets of globally distributed, heterogeneous databases.
b. Global Data Services employs a centralized, vertical framework.
c. Global Data Services provides dynamic load balancing, failover, and centralized service management for a set of replicated databases offering common services.
d. The set of databases can include RAC and noncluster Oracle databases interconnected through Data Guard, GoldenGate, or any other replication technology.

ORACLE

Answer: a, c, and d

Summary

In this lesson, you should have learned how to:

- Explain the benefits provided by Global Data Services for managing cloud-deployed distributed databases
- List the components of the Global Services Framework
- Describe the process of Global Services failover