ORACLE University

Integrated Cloud Applications & Platform Services

Oracle Database 19c: New Features for Administrators

Student Guide

D105110GC10 | D106790

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

ORACLE

Authors

Dominique Jeunot

Virginia Beecher

Technical Contributors and Reviewers

James Spiller

Nigel Bayliss

Padmaja Potineni

Pat Huey

Prakash Jashnani

Sanjay Kulhari

Sean Kim

Gwen Lazenby

Graphic Designer

James Hans

Lokesh Sahal

Editors

Nikita Abraham

Raj Kumar

Smita Kommini

Aju Kumar

Publishers

Veena Narasimhan

Asief Baig

Srividya Rameshkumar

1007022019

Contents

1

Using General Database Overall Enhancements

Overview

- This module focuses on the enhancements of Oracle Database 19c.
- It complements the general database topics covered in:
  - The 5-day *Oracle Database 12c: 12.2 New Features for 12.1 Administrators Ed 1* course
  - The 5-day *Oracle Database 12c R2: New Features for Administrators Ed 1* course *Part 1*
  - The 3-day *Oracle Database 18c: New Features for Administrators Ed 1* course
- Previous experience with Oracle Database 12*c* and Oracle Database 18c is required for a full understanding of database enhancements.

Refer to *Oracle Database New Features Guide 19c, Oracle Database Administrator's Guide 19c, Oracle Database Multitenant Administrator's Guide 19c, and Oracle Database Utilities 19c.*

This module follows either the 5-day *Oracle Database 12c: 12.2 New Features for 12.1 Administrators Ed 1* course or the 5-day *Oracle Database 12c Release 2 (12.2)* course Part 1 or the 3-day *Oracle Database 18c: New Features for Administrators Ed 1*. These courses are designed to introduce the enhancements of Oracle Database 12*c* Release 2 (12.2.0.1) and Oracle Database 18c that are applicable to work that is usually performed by database administrators and related personnel.

The module is designed to introduce the major enhancements of Oracle Database 19c related to database in general.

The module consists of an instructor-led lesson and practices that enable you to see how certain new functionalities behave.

Objectives

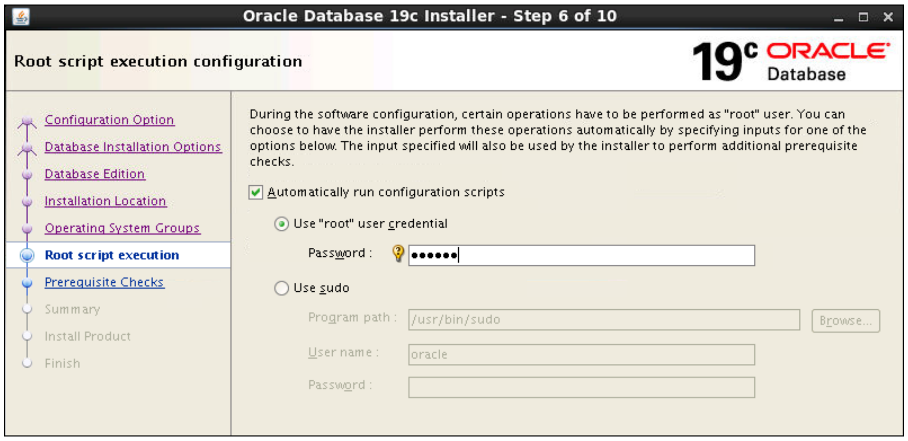
After completing this lesson, you should be able to:

- Use user-provided `root` or `sudo` credentials for automatic failed prerequisite checks, for automatic execution of fixups scripts and `root.sh`
- Use DBCA to clone or relocate a remote PDB, and to duplicate a database
- Create a PDB snapshot from a PDB
- Use data pump enhancements
- Improve the handling of very long lists of values generated by the `LISTAGG` function

ORACLE

Automatic Execution of Fixups Scripts and `root.sh`

Database Installer has an option to automate root script execution.



Oracle Database 19c Installer - Step 6 of 10

Root script execution configuration

During the software configuration, certain operations have to be performed as "root" user. You can choose to have the installer perform these operations automatically by specifying inputs for one of the options below. The input specified will also be used by the installer to perform additional prerequisite checks.

[✓] Automatically run configuration scripts

(●) Use "root" user credential

Password : ••••••

( ) Use sudo

Program path : /usr/bin/sudo

User name : oracle

Password :

Oracle Database 19c supports automatic execution of the `orainstRoot.sh` and `root.sh` shell scripts using the user-provided `root` or `sudo` credentials.

If the `root` option is selected, enter the password for the `root` user. If the `sudo` option is selected, enter the password for the user defined in the username field. To enable sudo execution, you must ask the system administrator to enter the username as a member of the sudoers list in the `/etc/sudoers` file.

The automation is supported for both Oracle Database and Oracle Grid Infrastructure installations, single-instance and RAC instances.

Silent Mode Installation

In silent mode installation, the user specifies values for the following variables in the response file used by the `runInstaller` executable:

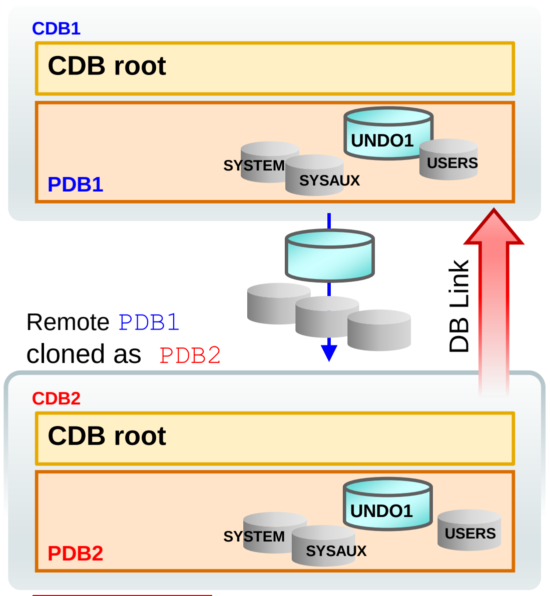
- `oracle.install.db.rootconfig.executeRootScript=TRUE/FALSE`
- `oracle.install.db.rootconfig.configMethod=SUDO/ROOT`
- `oracle.install.db.rootconfig.sudoPath=<`*`path of the sudo program`*`>`
- `oracle.install.db.rootconfig.sudoUserName=<`*`sudo user name`*`>`

The `root` or `sudo` password cannot be specified in the response file. The installer prompts for the passwords on the console. The user has to provide the passwords to the installer using these prompts.

Cloning a Remote PDB

18c

Remote source PDB1



Remote source PDB still up and fully functional:

1. Connect to the local target **CDB2** root to create the database link to **CDB1**.

```
SQL> CONNECT sys@CDB2 AS SYSDBA
SQL> CREATE PUBLIC DATABASE LINK link_pdb1
     CONNECT TO system IDENTIFIED BY password
            USING 'PDB1';
```

2. Clone the remote **PDB1** from **CDB1** to **PDB2**.

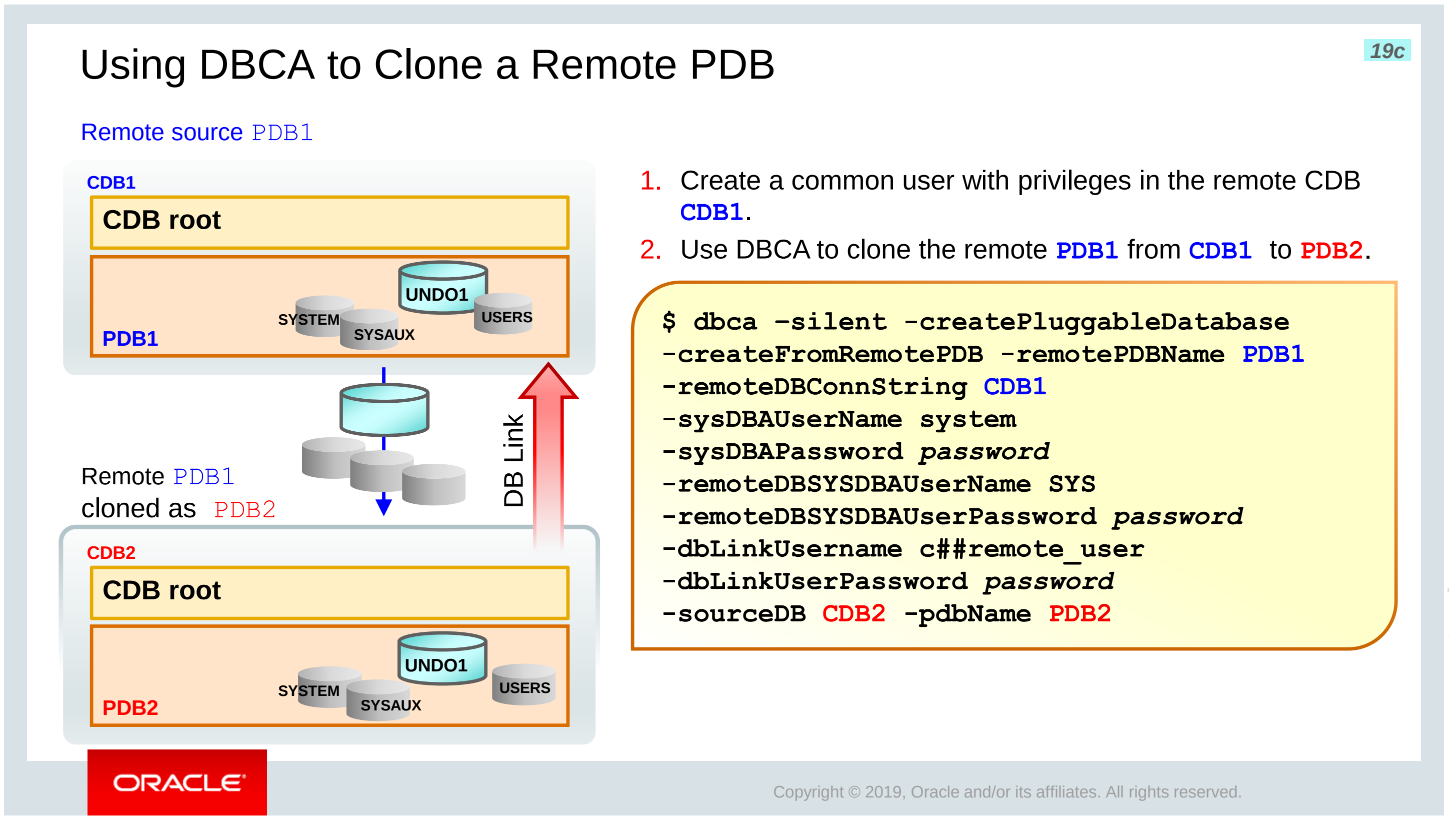
```
SQL> ALTER SESSION SET
           DB_CREATE_FILE_DEST='/…/CDB2/PDB2';
SQL> CREATE PLUGGABLE DATABASE pdb2
            FROM pdb1@link_pdb1;
```

3. Open **PDB2** in read-only or read/write mode.

```
SQL> ALTER PLUGGABLE DATABASE pdb2 OPEN;
```

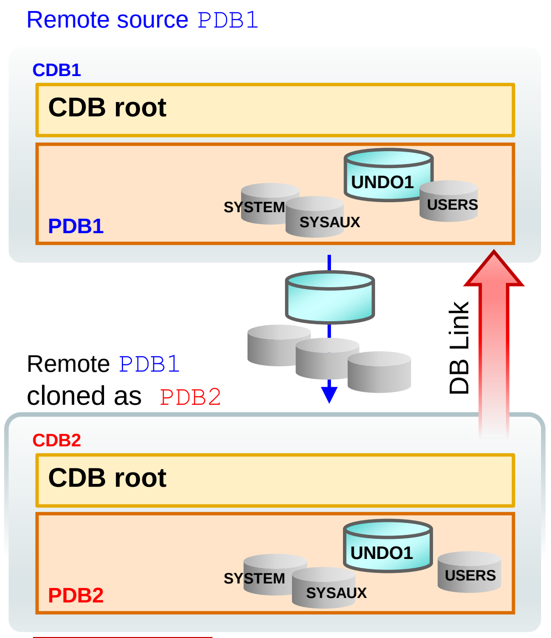
In Oracle Database 18c, cloning a production PDB to create a test PDB copies the remote production PDB into a CDB while the remote production PDB is still up and fully functional. Hot remote cloning requires both CDBs to switch from shared UNDO mode to local UNDO mode, which means that each PDB uses its own local UNDO tablespace.

To complete the operation, you must execute several SQL commands.



Using DBCA to Clone a Remote PDB

19c



1. Create a common user with privileges in the remote CDB **CDB1**.
2. Use DBCA to clone the remote **PDB1** from **CDB1** to **PDB2**.

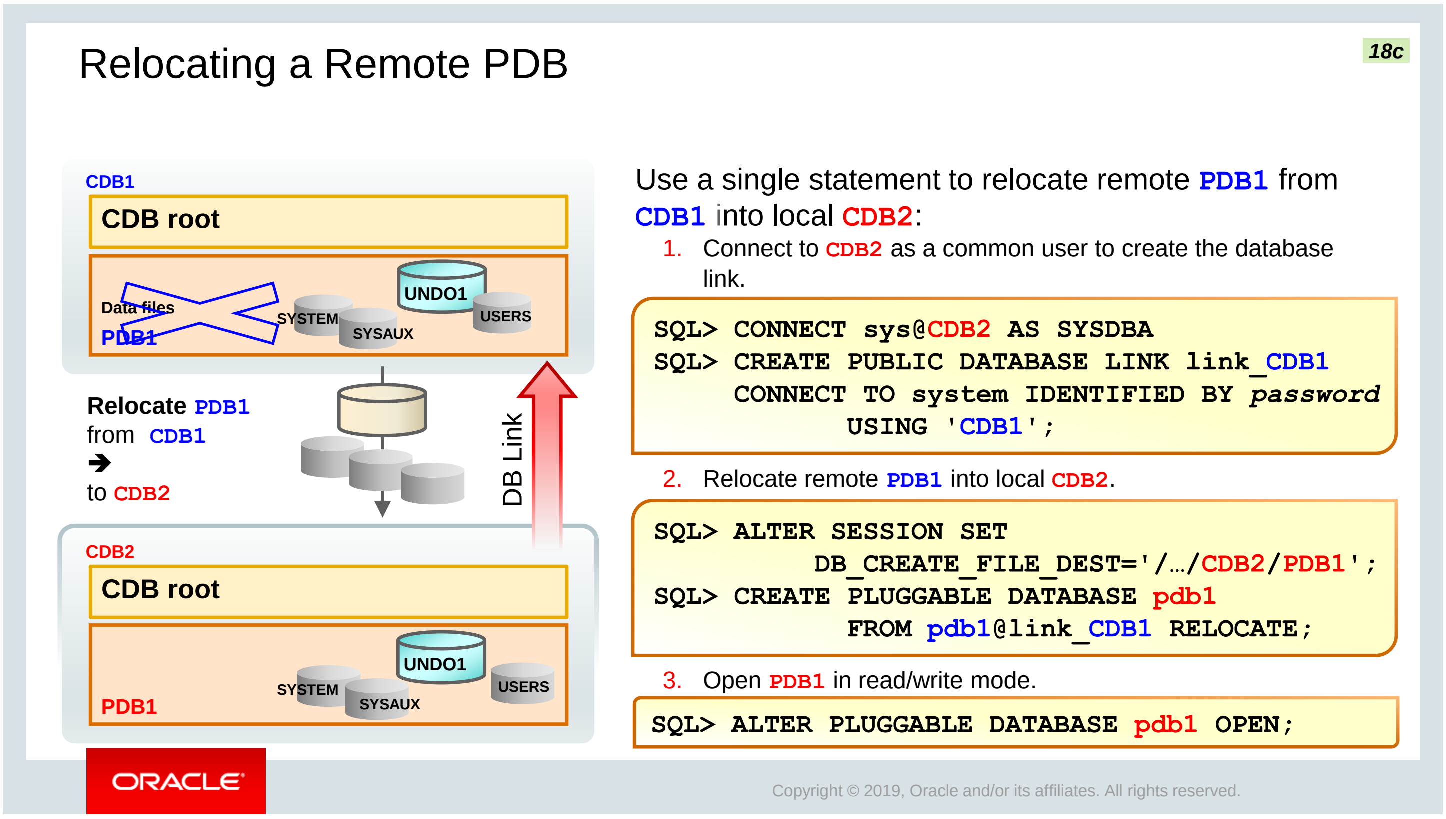
```
$ dbca -silent -createPluggableDatabase
-createFromRemotePDB -remotePDBName PDB1
-remoteDBConnString CDB1
-sysDBAUserName system
-sysDBAPassword password
-remoteDBSYSDBAUserName SYS
-remoteDBSYSDBAUserPassword password
-dbLinkUsername c##remote_user
-dbLinkUserPassword password
-sourceDB CDB2 -pdbName PDB2
```

Oracle Database 19c offers to complete the same operation by using DBCA in silent mode. The DBCA operation executes the following steps:

1. Checks the presence of the database link. If the database link exists, DBCA drops it.
2. Creates the database link
3. Creates the PDB from the remote PDB
4. Checks the status of the cloned PDB to verify that it is in mounted mode
5. Opens the cloned PDB

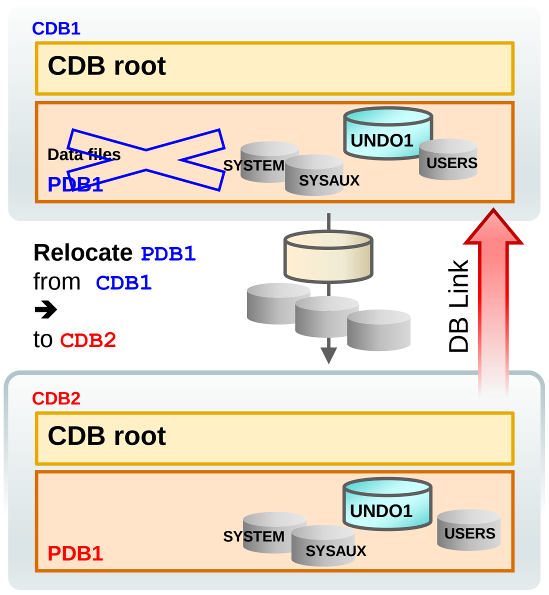
As in Oracle Database 18c, the user in the local target CDB must have the `CREATE PLUGGABLE DATABASE` privilege in the CDB root.

The remote CDB must use local undo mode. The remote CDB must be in archivelog mode. The common user in the remote PDB that the database link connects to must have the `CREATE PLUGGABLE DATABASE` and `CREATE SESSION` privileges.



Relocating a Remote PDB

18c



Use a single statement to relocate remote **PDB1** from **CDB1** into local **CDB2**:

1. Connect to **CDB2** as a common user to create the database link.

```
SQL> CONNECT sys@CDB2 AS SYSDBA
SQL> CREATE PUBLIC DATABASE LINK link_CDB1
     CONNECT TO system IDENTIFIED BY password
            USING 'CDB1';
```

2. Relocate remote **PDB1** into local **CDB2**.

```
SQL> ALTER SESSION SET
          DB_CREATE_FILE_DEST='/…/CDB2/PDB1';
SQL> CREATE PLUGGABLE DATABASE pdb1
            FROM pdb1@link_CDB1 RELOCATE;
```

3. Open **PDB1** in read/write mode.

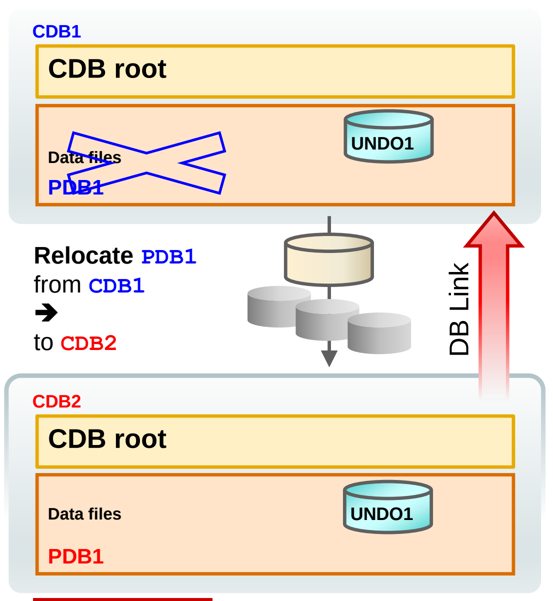
```
SQL> ALTER PLUGGABLE DATABASE pdb1 OPEN;
```

ORACLE

In Oracle Database 18c, to complete a PDB relocation, you must execute several SQL commands. The “pull” mode, connected to the CDB where the PDB will be relocated, pulls the source PDB from the CDB where the PDB exists, draining existing connections and migrating new connections without requiring any changes to the application.

Using DBCA to Relocate a Remote PDB

19c



1. Use DBCA to relocate the remote **PDB1** from **CDB1** into **CDB2**.

```
$ export ORACLE_SID=CDB2
```

```
$ dbca -silent -relocatePDB
-remotePDBName PDB1 -remoteDBConnString CDB1
-sysDBAUserName system
-sysDBAPassword password
-remoteDBSYSDBAUserName SYS
-remoteDBSYSDBAUserPassword password
-dbLinkUsername c##remote_user
-dbLinkUserPassword password
-sourceDB CDB2 -pdbName PDB1
```

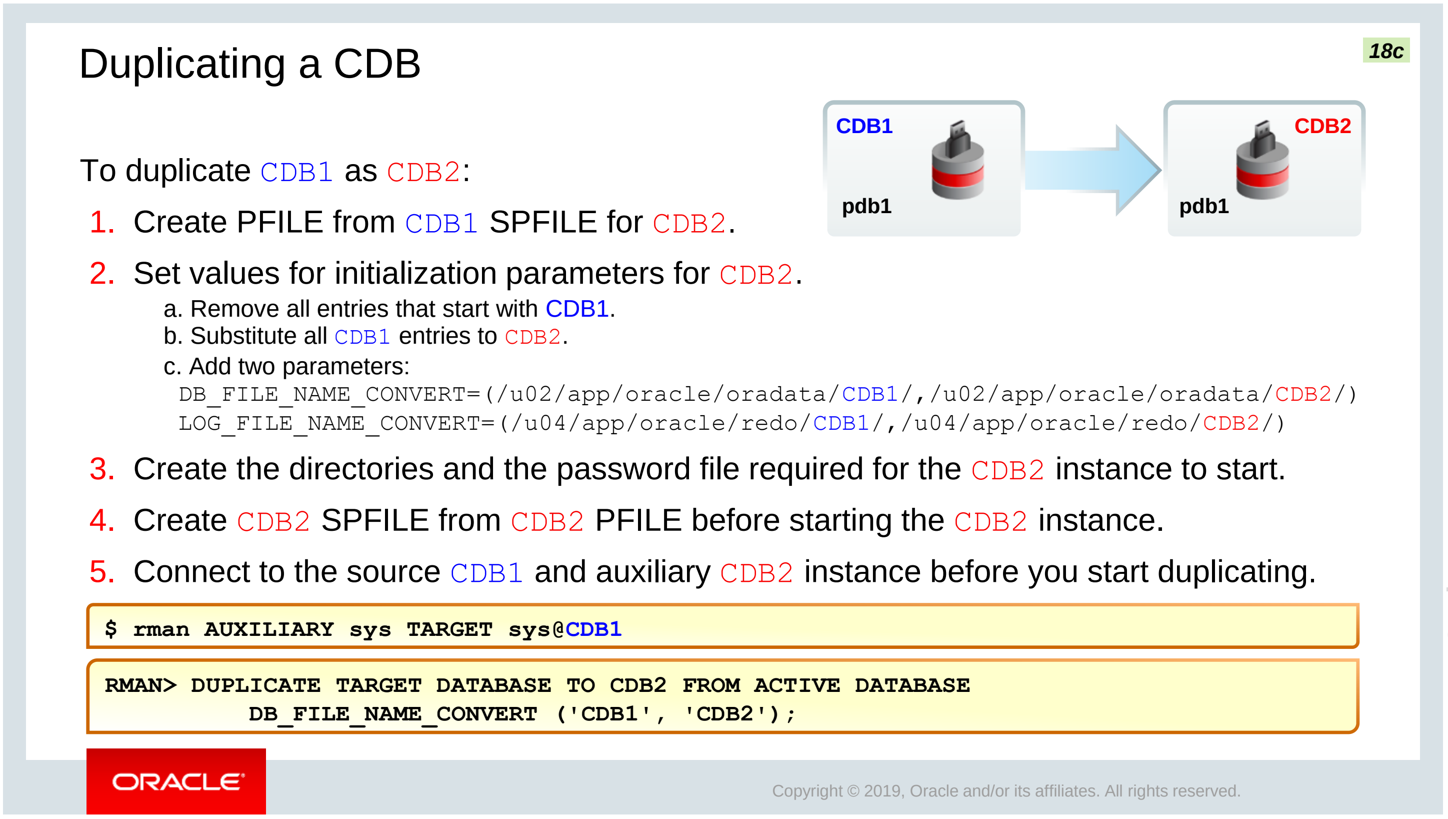
ORACLE

Oracle Database 19c offers to complete the same operation by using DBCA in silent mode. The DBCA operation executes the following steps:

1. Checks the presence of the database link. If the database link exists, DBCA drops it.
2. Creates the database link
3. Creates the PDB from the remote PDB
4. Checks the status of the cloned PDB to verify that it is in mounted mode
5. Opens the relocated PDB

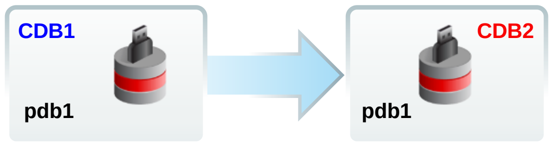
The user in the local database must have the `CREATE PLUGGABLE DATABASE` privilege in the CDB root container. The remote and local databases must in archivelog mode. The common user in the remote database that the database link connects to must have the `CREATE PLUGGABLE DATABASE`, `SESSION`, and `SYSOPER` privileges.

The local and remote databases must either have the same options installed, or the remote database must have a subset of those present on the local database.



Duplicating a CDB

18c



To duplicate CDB1 as CDB2:

1. Create PFILE from CDB1 SPFILE for CDB2.
2. Set values for initialization parameters for CDB2.
   a. Remove all entries that start with CDB1.
   b. Substitute all CDB1 entries to CDB2.
   c. Add two parameters:
   ```
   DB_FILE_NAME_CONVERT=(/u02/app/oracle/oradata/CDB1/,/u02/app/oracle/oradata/CDB2/)
   LOG_FILE_NAME_CONVERT=(/u04/app/oracle/redo/CDB1/,/u04/app/oracle/redo/CDB2/)
   ```
3. Create the directories and the password file required for the CDB2 instance to start.
4. Create CDB2 SPFILE from CDB2 PFILE before starting the CDB2 instance.
5. Connect to the source CDB1 and auxiliary CDB2 instance before you start duplicating.

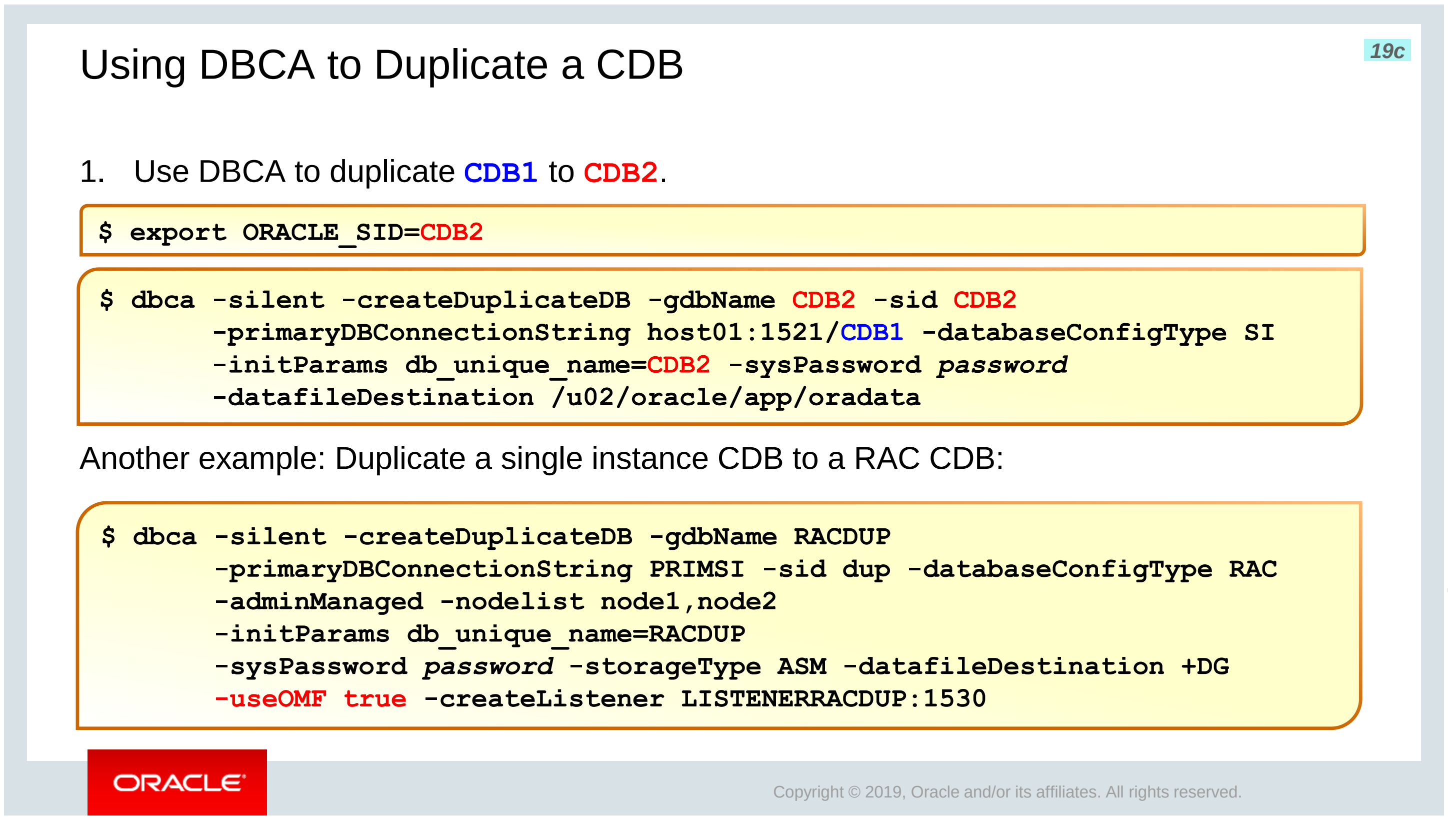
```
$ rman AUXILIARY sys TARGET sys@CDB1
```

```
RMAN> DUPLICATE TARGET DATABASE TO CDB2 FROM ACTIVE DATABASE
        DB_FILE_NAME_CONVERT ('CDB1', 'CDB2');
```

The example shows a duplication of CDB1 as CDB2 with Oracle Database 18c.

To perform this operation, connections to the source (TARGET) CDB1 and to the destination (AUXILIARY) CDB2 are required.

There are a lot of steps to perform before starting the duplicate operation itself.



Using DBCA to Duplicate a CDB

19c

1. Use DBCA to duplicate CDB1 to CDB2.

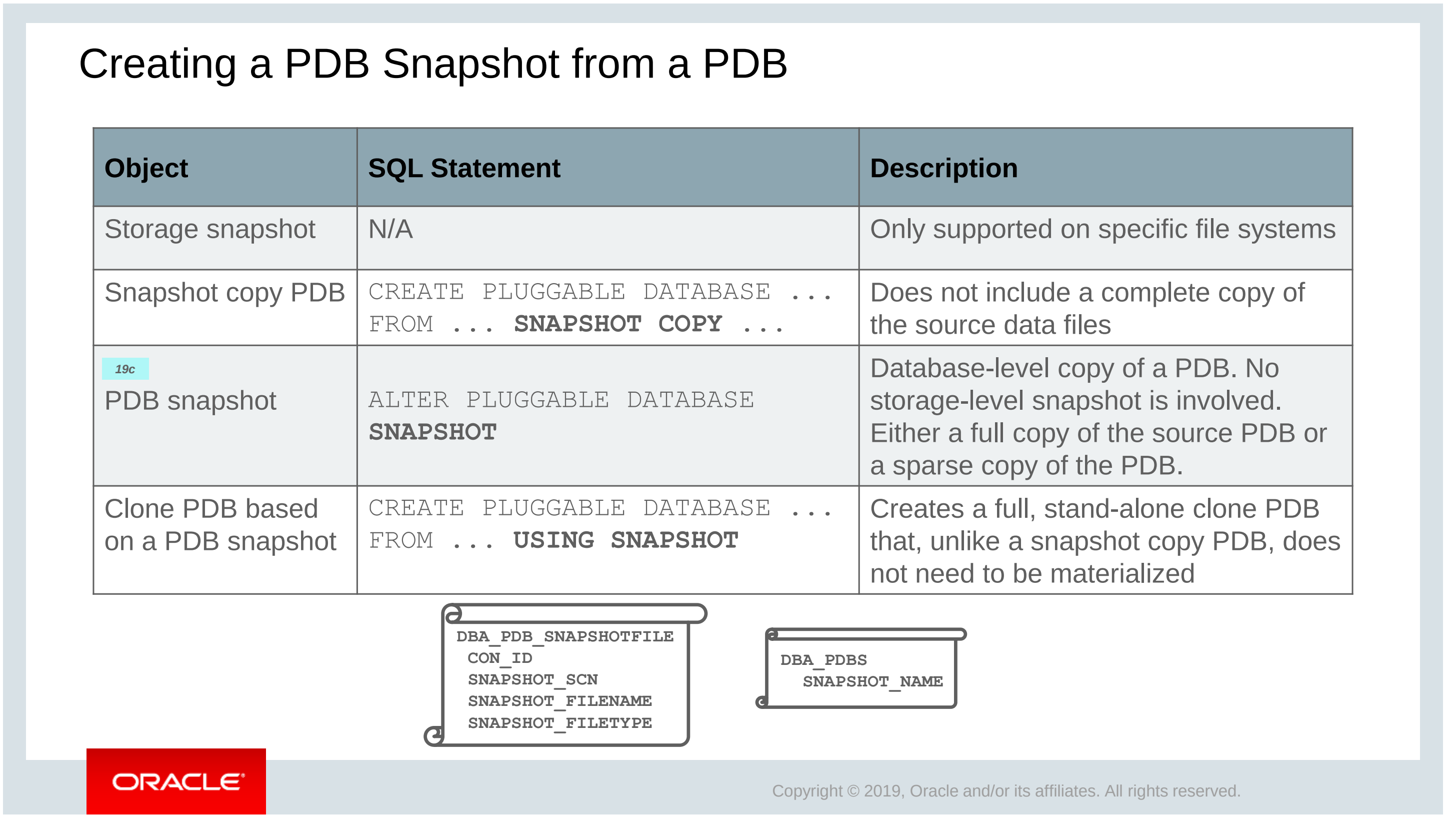
```
$ export ORACLE_SID=CDB2
```

```
$ dbca -silent -createDuplicateDB -gdbName CDB2 -sid CDB2
       -primaryDBConnectionString host01:1521/CDB1 -databaseConfigType SI
       -initParams db_unique_name=CDB2 -sysPassword password
       -datafileDestination /u02/oracle/app/oradata
```

Another example: Duplicate a single instance CDB to a RAC CDB:

```
$ dbca -silent -createDuplicateDB -gdbName RACDUP
       -primaryDBConnectionString PRIMSI -sid dup -databaseConfigType RAC
       -adminManaged -nodelist node1,node2
       -initParams db_unique_name=RACDUP
       -sysPassword password -storageType ASM -datafileDestination +DG
       -useOMF true -createListener LISTENERRACDUP:1530
```

Oracle Database 19c offers to complete the same operation by using DBCA in silent mode. In Oracle Database 19c, the use case of creating a non-OMF duplicate database of an OMF primary database is supported.



Creating a PDB Snapshot from a PDB

Object	SQL Statement	Description
Storage snapshot	N/A	Only supported on specific file systems
Snapshot copy PDB	`CREATE PLUGGABLE DATABASE ... FROM ... SNAPSHOT COPY ...`	Does not include a complete copy of the source data files
19c PDB snapshot	`ALTER PLUGGABLE DATABASE SNAPSHOT`	Database-level copy of a PDB. No storage-level snapshot is involved. Either a full copy of the source PDB or a sparse copy of the PDB.
Clone PDB based on a PDB snapshot	`CREATE PLUGGABLE DATABASE ... FROM ... USING SNAPSHOT`	Creates a full, stand-alone clone PDB that, unlike a snapshot copy PDB, does not need to be materialized

```
DBA_PDB_SNAPSHOTFILE
  CON_ID
  SNAPSHOT_SCN
  SNAPSHOT_FILENAME
  SNAPSHOT_FILETYPE
```

```
DBA_PDBS
    SNAPSHOT_NAME
```

In Oracle Multitenant, the term snapshot means different things depending on context.

- Storage-level snapshot, which is used to create a snapshot copy PDB
- PDB-level snapshot, which is used to create a standard clone PDB. A PDB-level snapshot does not involve storage-level snapshots.

Different Types of Snapshots

- A storage-level snapshot is only supported on specific file systems. The storage and security credential requirements depend on the setting of the `CLONEDB` initialization parameter.
- A snapshot copy PDB does not include a complete copy of the source data files. Rather, Oracle Database creates a storage-level snapshot of the underlying file system, and then creates the clone PDB from the snapshot. Unlike a standard clone PDB, the snapshot copy PDB is dependent on the storage snapshot. Therefore, you cannot unplug this PDB from the CDB root or plug it in to an application root. Also, you cannot drop the storage snapshot on which the PDB is based. Instead, you must materialize the snapshot copy PDB, which converts it into a full PDB with nonsparse files.
- A PDB snapshot is a database-level copy of a PDB. No storage-level snapshot is involved. If the snapshot is created when `CLONEDB=FALSE`, then the snapshot is a full copy of the source PDB. Starting in Oracle Database 19c, if the snapshot is created when `CLONEDB=TRUE`, then the snapshot is a sparse copy of the PDB.
- A clone PDB based on a PDB snapshot creates a full, stand-alone clone PDB that, unlike a snapshot copy PDB, does not need to be materialized. You cannot create a snapshot copy PDB that is based on a PDB snapshot by including both the `USING SNAPSHOT` clause and the `SNAPSHOT COPY` clause. However, you can create a stand-alone clone PDB with `USING SNAPSHOT`, and then create a snapshot copy PDB from the stand-alone PDB.

Data Pump Enhancements

Decreasing the time of data pump TTS export and import operations by bypassing steps:

- Skips the rebuild of the bitmap to reclaim free space that is left unused
- Leaves the tablespace read-only
  - Bitmaps are not rebuilt.
  - If the time zone files are different, any table with a time zone column is dropped.
- Skips the set closure check allowing TTS or Full Transportable export to remain in read/write mode
  - Increases availability

Migrating objects involving encrypted columns into a cloud database:

- Excludes the `ENCRYPTION` clause on import

Controlling overutilization of resources when multiple users perform data pump jobs:

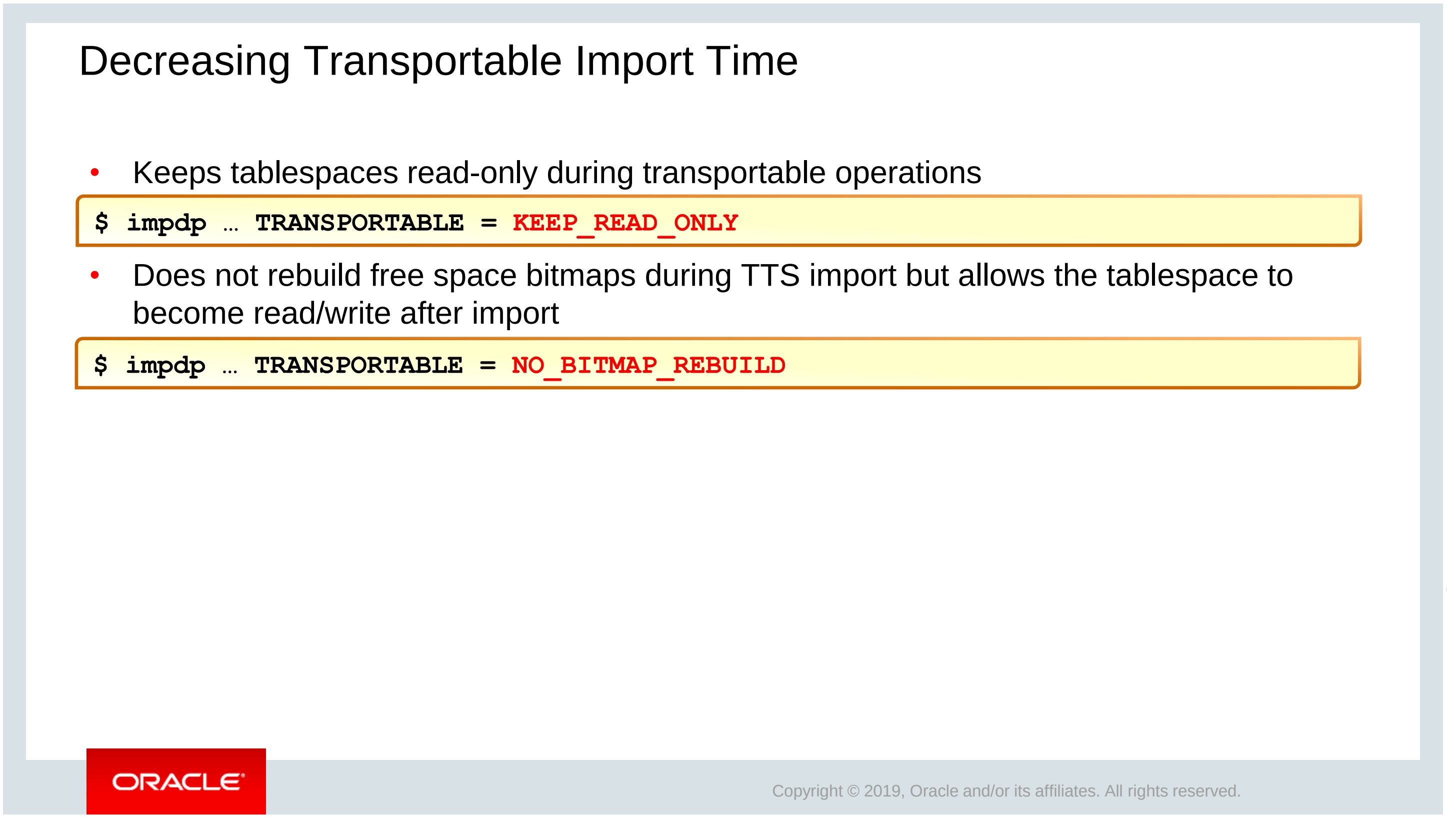
- The number of jobs that can be started in a PDB
- The number of parallel workers used for an individual data pump job

Tablespace data files can be read-only during the Transportable Tablespace (TTS) import process. However, read-only mode does not rebuild tablespace bitmaps to reclaim space and improve performance of import, or adjust `TIMESTAMP WITH TIMEZONE` columns if the source database has a different DST version than the target database. DBAs can now import tablespace files mounted on two different databases as long as the files are set as read-only.

The DBA has the ability to bypass unnecessary closure checks and obtain timing requirements for TTS operations. Test mode for TTS performs a metadata-only export test using TTS or Full Transportable export. It also removes the limitation on the source database tablespaces to be in read-only mode. DBAs can more easily determine how long an export will take and discover unforeseen issues not reported by the closure check.

A new transform parameter, `OMIT_ENCRYPTION_CLAUSE`, is introduced that causes the data pump to suppress any encryption clauses associated with objects using encrypted columns.

Two new initialization parameters, `MAX_DATAPUMP_JOBS_PER_PDB` and `MAX_DATAPUMP_PARALLEL_PER_JOB`, are introduced to give DBAs more control over the number of jobs that can be started in a PDB, and the number of parallel workers that can be used for an individual data pump job, respectively. These initialization parameters give the DBA more control over resource utilization when there are multiple users who may be performing data pump jobs in a given database environment.



Decreasing Transportable Import Time

- Keeps tablespaces read-only during transportable operations

```
$ impdp … TRANSPORTABLE = KEEP_READ_ONLY
```

- Does not rebuild free space bitmaps during TTS import but allows the tablespace to become read/write after import

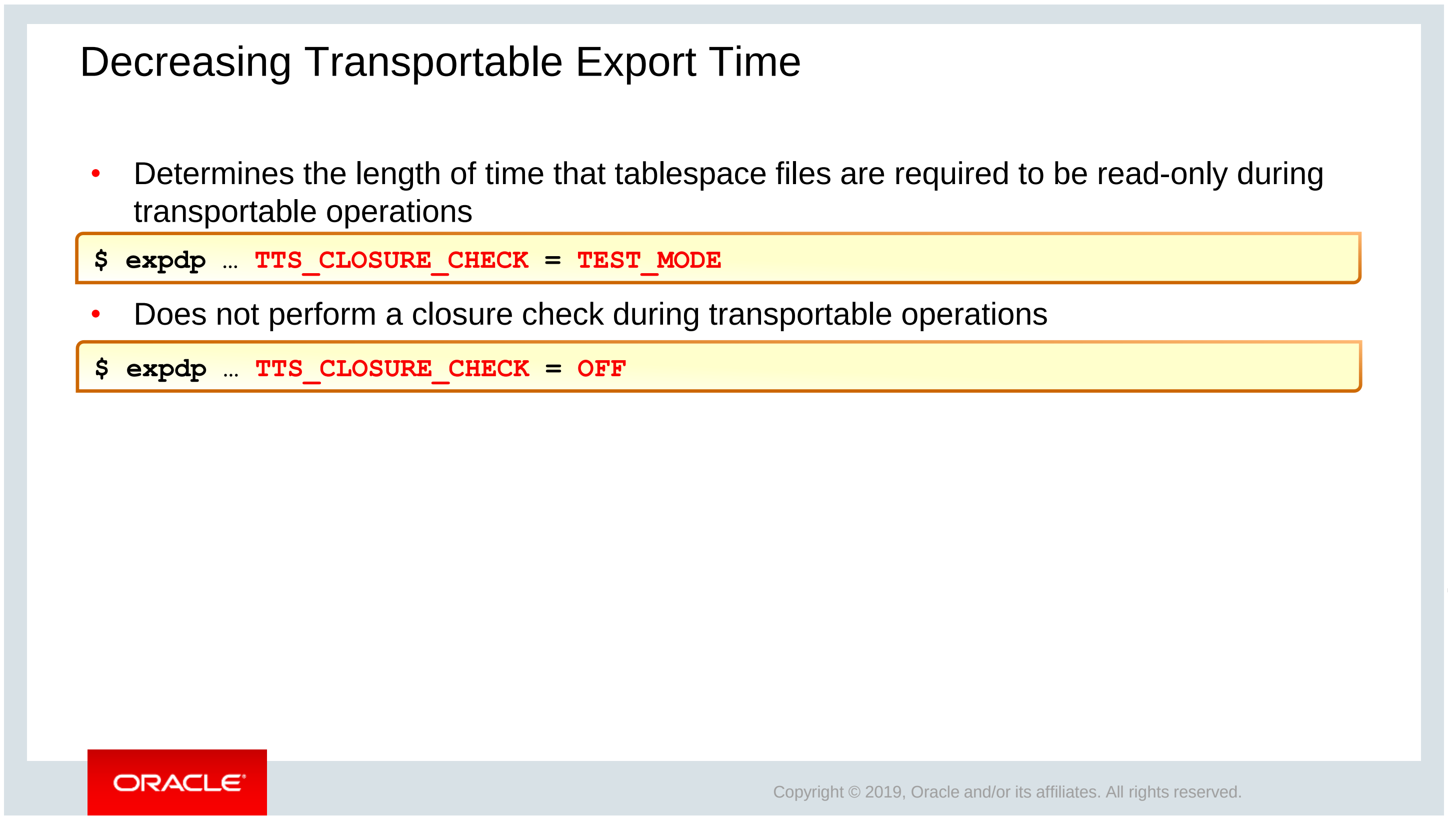
```
$ impdp … TRANSPORTABLE = NO_BITMAP_REBUILD
```

Starting in Oracle Database 12*c*, the data pump changed how import handled transportable tablespaces. Before Oracle Database 12*c*, the data pump would always leave the tablespaces as read-only, but there were two problems with this.

- First, if the version of the time zone file of the export database did not match the version of the time zone file for the import database, then all tables with time zone data types in the transportable space would be dropped because they could have incorrect values with the new time zone file.
- Second, the way the data pump tracked free space in a transportable tablespace did not scale for large tablespaces (we had customers where an export would take hours to gather information about free space). To fix these performance issues, the data pump changed to rebuilding the bitmap of free space at import time.

In Oracle Database 19c, if the DBA wants to do a read-only import, the DBA can use the `KEEP_READ_ONLY` option instead of setting OS permissions on the file. `KEEP_READ_ONLY` indicates that the transportable data files are to remain in read-only mode and allows the data files to be plugged into an additional compatible database for read-only access. Tables containing TSTZ column data cannot be updated and are dropped from the import. In this case, an `ORA-39339 "Table <schema>.<table> skipped due to transportable and TSTZ issues"` message mentions the skipped tables.

If the DBA wants to defer the rebuild of the bitmap in order to have a shorter down time for the operation, then the DBA can use the `NO_BITMAP_REBUILD` option. `NO_BITMAP_REBUILD` indicates that the transportable data files header bitmaps are not to be rebuilt. Not reclaiming unused data segments reduces the time of the import operation. The data files may be placed in read/write mode, if necessary, to update tables containing TSTZ column data. Bitmaps can be rebuilt by using the `dbms_space_admin.tablespace_rebuild_bitmaps` procedure.



Decreasing Transportable Export Time

- Determines the length of time that tablespace files are required to be read-only during transportable operations

```
$ expdp … TTS_CLOSURE_CHECK = TEST_MODE
```

- Does not perform a closure check during transportable operations

```
$ expdp … TTS_CLOSURE_CHECK = OFF
```

Oracle Database 19c introduces the ability to determine the length of time that tablespace files are required to be read-only during transportable operations.

- Running a data pump transportable operation with the `TTS_CLOSURE_CHECK` parameter in `TEST_MODE` mode is for testing purposes only, providing a timing estimation of the TTS export operation. It allows the tablespaces to remain in read/write mode. The resulting Data Pump export dump file is not available for use by data pump import.
- The time it takes to conduct the closure check can be long. The closure check can be unnecessary when the DBA knows that the transportable set is self-contained. Running a Data Pump transportable operation with the `TTS_CLOSURE_CHECK` parameter in `OFF` mode decreases the time required for data pump TTS to complete.
- Running a data pump transportable operation with the `TTS_CLOSURE_CHECK` parameter in `ON` mode indicates that the closure check step is completed to ensure that the TTS set contains no references outside the set.
- Running a data pump transportable operation with the `TTS_CLOSURE_CHECK` parameter in `FULL` mode indicates that a full multidirectional closure check is performed to ensure that there are no remote references out of or into the transportable tablespace set.

`ON`, `OFF`, and `FULL` are mutually exclusive.

Omitting the Column Encryption Attribute During Import

- Suppresses any encryption attributes when data pump export generates the `create object` DDL statement
  - Applies to materialized views, tables, and tablespace objects
  - Enables objects which are utilizing encrypted columns in the source to get created in the target database environment where encryption attributes are not supported

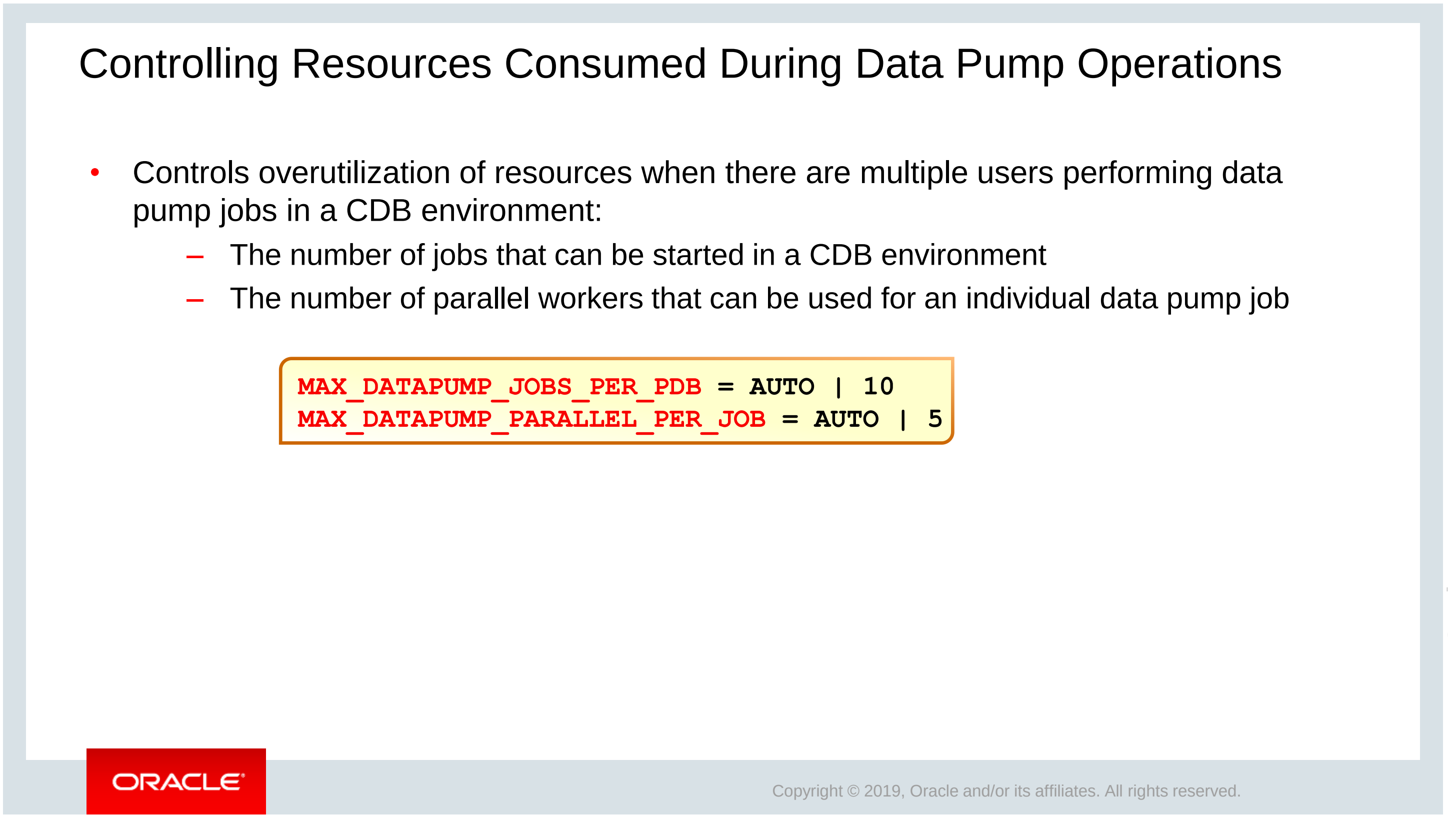
```
$ impdp … TRANSFORM = OMIT_ENCRYPTION_CLAUSE: [ Y | N ]
```

When using data pump to migrate a source database that has objects with encrypted columns, data pump import always includes any associated encryption attribute to create objects in the target database. Consequently, these objects may fail to get created in the target database due to invalid and unsupported encryption syntax if the target database uses Transparent Data Encryption (TDE) at the tablespace level. Allowing users to suppress encryption-related syntax during import facilitates the creation of objects such as tables and materialized views. This is particularly true when using data pump to migrate an on-premises database that has objects which have encrypted columns into the cloud. In the Oracle Public Cloud environment, data is encrypted by default using TDE and the encrypted tablespace feature. However, this environment does not support the encrypted column feature.

Oracle Database 19c introduces a new transform parameter, `OMIT_ENCRYPTION_CLAUSE`, which directs Data Pump to suppress any encryption clause associated with objects using encrypted columns.

- `OMIT_ENCRYPTION_CLAUSE:N`: Generates a create object DDL that includes the related encryption attribute clauses. This is the default behavior.
- `OMIT_ENCRYPTION_CLAUSE:Y`: Generates a create object DDL that does not include any encryption attribute clause

The `OMIT_ENCRYPTION_CLAUSE` parameter is not allowed during transportable import jobs.



Controlling Resources Consumed During Data Pump Operations

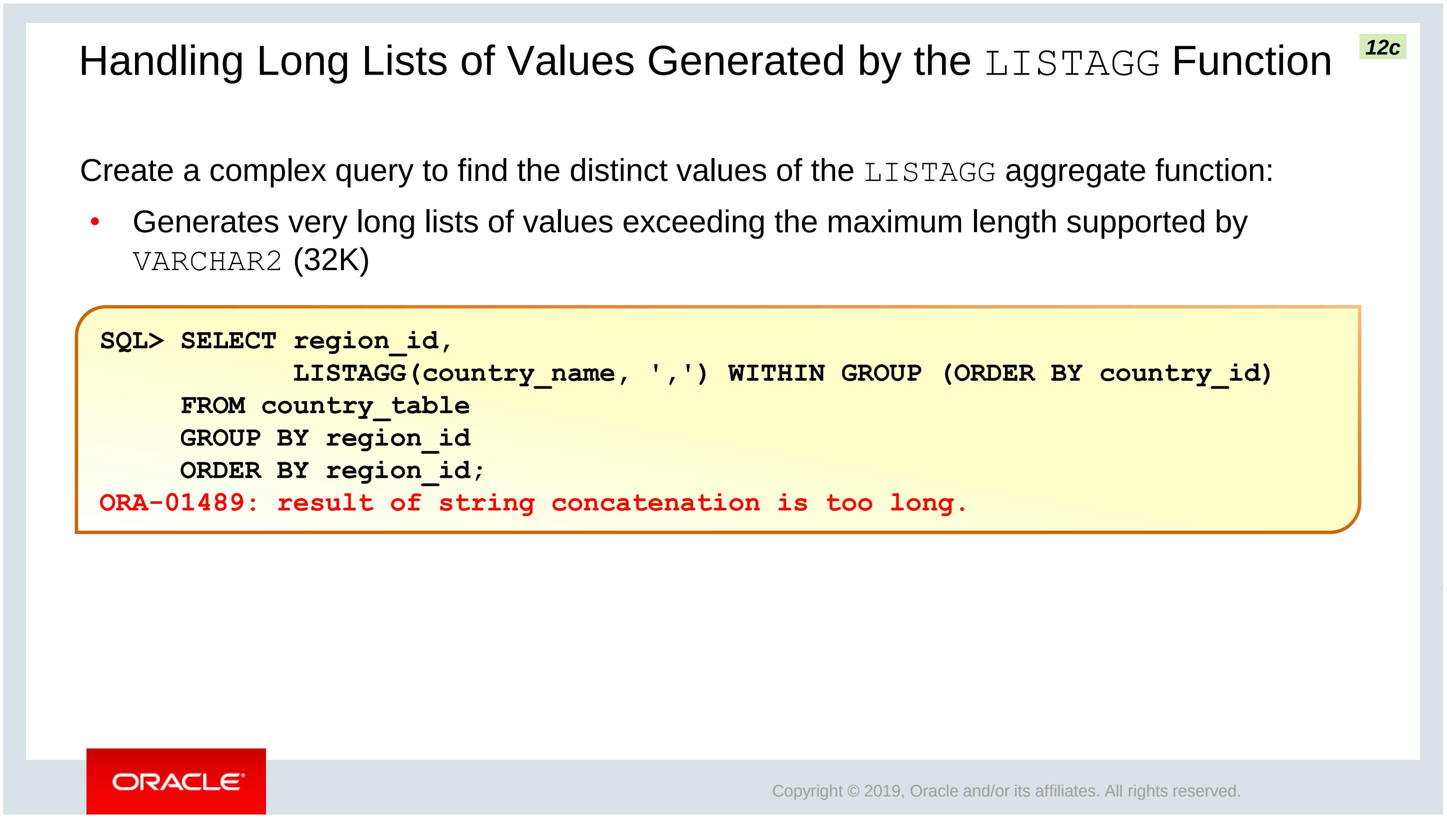
- Controls overutilization of resources when there are multiple users performing data pump jobs in a CDB environment:
  - The number of jobs that can be started in a CDB environment
  - The number of parallel workers that can be used for an individual data pump job

```
MAX_DATAPUMP_JOBS_PER_PDB = AUTO | 10
MAX_DATAPUMP_PARALLEL_PER_JOB = AUTO | 5
```

ORACLE

Oracle Database 19c introduces two new initialization parameters, `MAX_DATAPUMP_JOBS_PER_PDB` and `MAX_DATAPUMP_PARALLEL_PER_JOB`, to give DBAs more control over the number of data pump jobs that can be started in each PDB in a CDB, and over the number of parallel workers that can be used for an individual data pump job, respectively. These initialization parameters give the DBA more control over resource utilization when there are multiple users who may be performing data pump jobs in a given CDB.

- `MAX_DATAPUMP_JOBS_PER_PDB`: The default value will not work for all databases. Database administrators will have to determine if the default value works well for their database.
  When this parameter has a value of `AUTO`, Oracle Data Pump derives its actual value to be 50% of the `SESSIONS` initialization parameter.
  A value that is too large could cause Oracle Data Pump to consume too many system resources, while a value that is too small could prevent users from performing their Oracle Data Pump tasks.
- `MAX_DATAPUMP_PARALLEL_PER_JOB`: When this parameter has a value of `AUTO`, Oracle Data Pump derives its value to be 50 percent of the value of the `SESSIONS` initialization parameter.



Handling Long Lists of Values Generated by the LISTAGG Function

12c

Create a complex query to find the distinct values of the LISTAGG aggregate function:

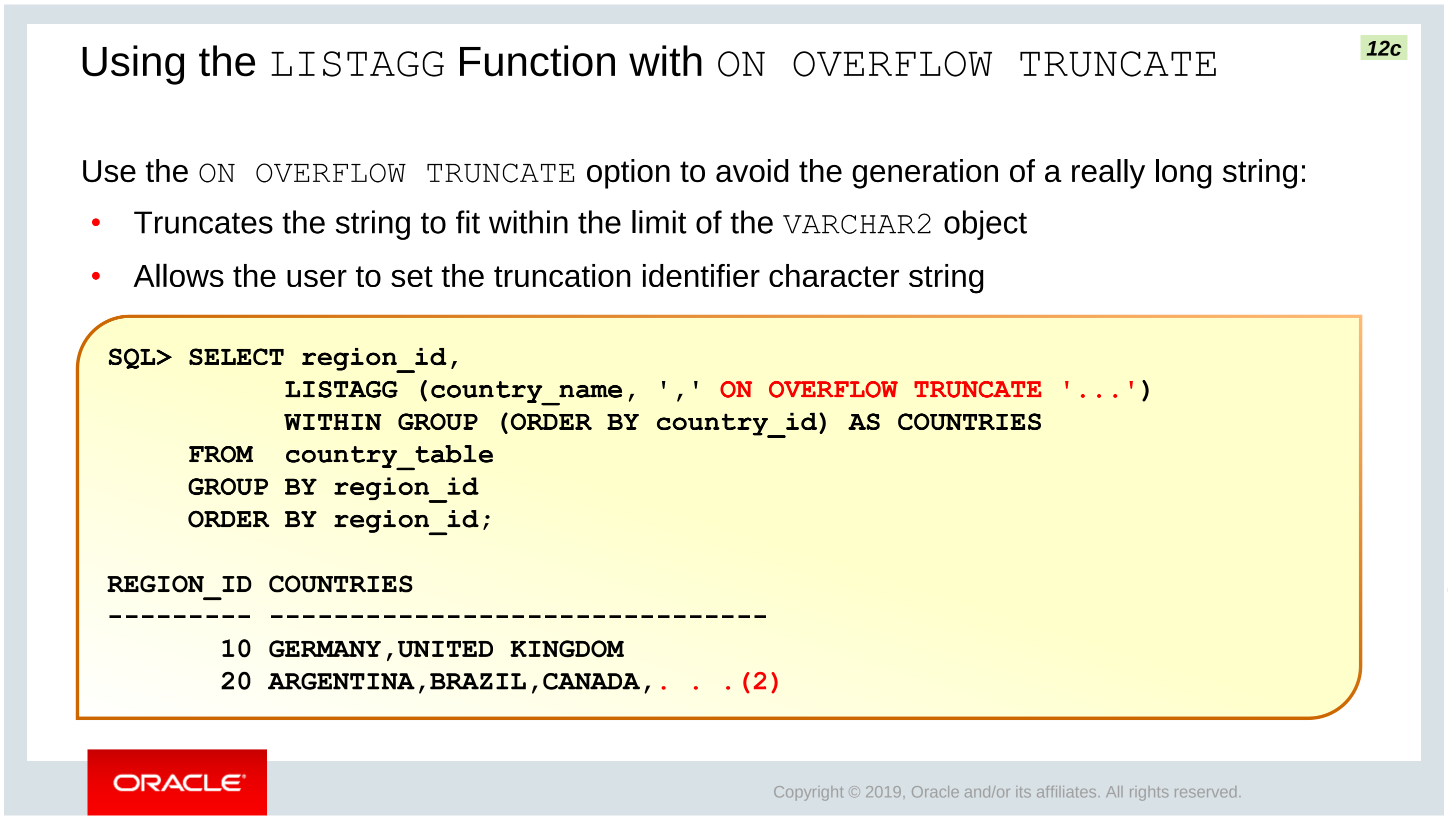
- Generates very long lists of values exceeding the maximum length supported by VARCHAR2 (32K)

```
SQL> SELECT region_id,
            LISTAGG(country_name, ',') WITHIN GROUP (ORDER BY country_id)
     FROM country_table
     GROUP BY region_id
     ORDER BY region_id;
ORA-01489: result of string concatenation is too long.
```

In Oracle Database pre-12.2, the LISTAGG aggregate function performs the following steps:

1. Orders the rows for each group in a query according to the ORDER BY expression
2. Concatenates the values into a single string

If the string generated contains too many characters, exceeding the limits of VARCHAR2, an error is raised.



Using the LISTAGG Function with ON OVERFLOW TRUNCATE

12c

Use the ON OVERFLOW TRUNCATE option to avoid the generation of a really long string:

- Truncates the string to fit within the limit of the VARCHAR2 object
- Allows the user to set the truncation identifier character string

```
SQL> SELECT region_id,
            LISTAGG (country_name, ',' ON OVERFLOW TRUNCATE '...')
            WITHIN GROUP (ORDER BY country_id) AS COUNTRIES
     FROM   country_table
     GROUP BY region_id
     ORDER BY region_id;

REGION_ID COUNTRIES
--------- -------------------------------
       10 GERMANY,UNITED KINGDOM
       20 ARGENTINA,BRAZIL,CANADA,. . .(2)
```

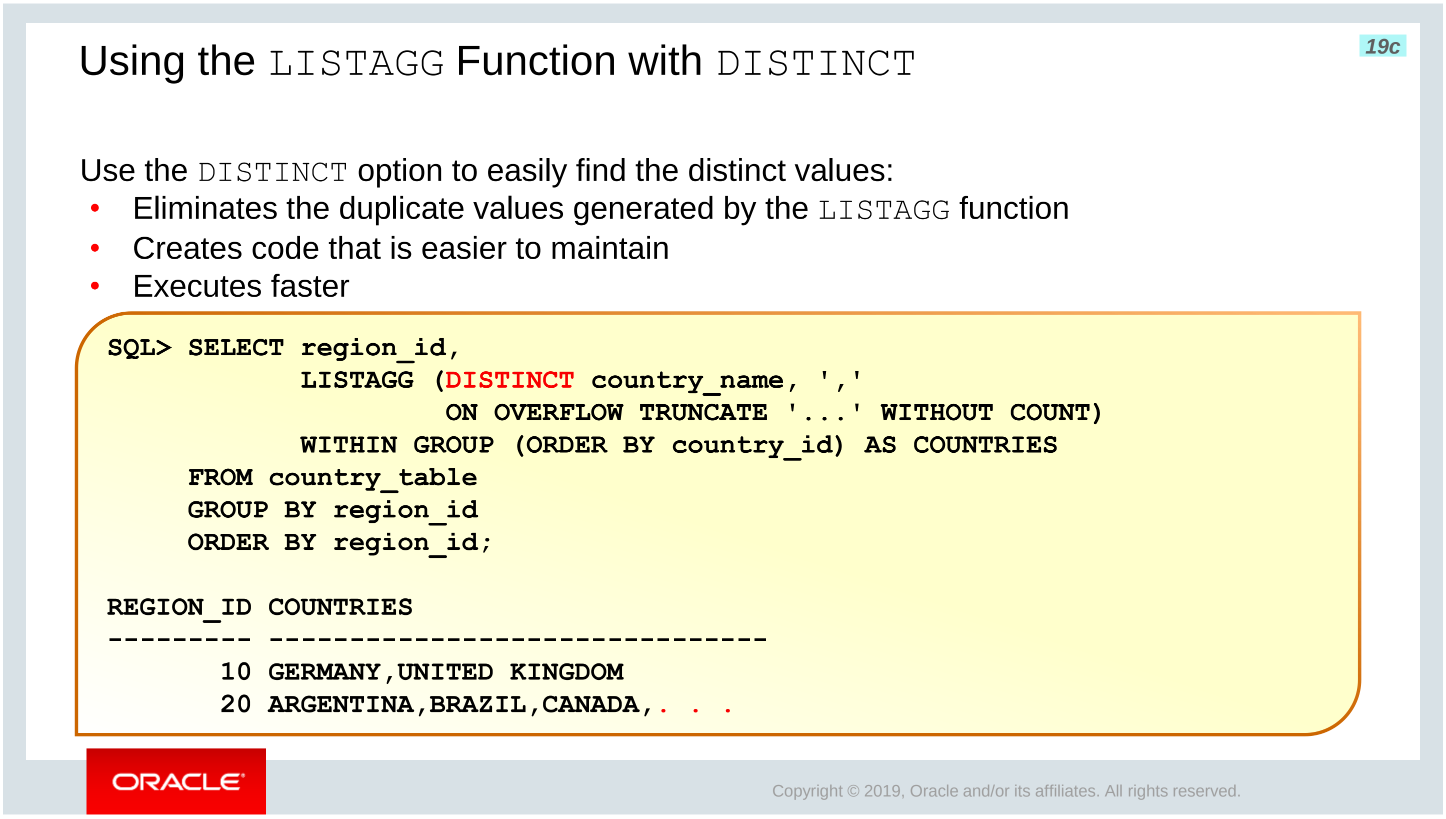
The LISTAGG aggregate function with the ON OVERFLOW TRUNCATE option performs the following steps:

1. Orders the rows for each group in a query according to the ORDER BY expression
2. Concatenates the values into a single string
3. Truncates the string to fit within the limit of the VARCHAR2 object

How can the user control the size of the string generated?

The user can set the overflow/truncation identifier character string as part of the definition of the LISTAGG function. Thus, if the string does not fit within the limit of the VARCHAR2 object, the string is automatically truncated to fit within the limit of the VARCHAR2 object. The overflow/truncation identified character string is a constant literal character string that is appended to the end of a list of values, after the last separator occurrence of the delimiter character where the list is too long.

- The clause is optional and the default value is NULL.
- A value from a row cannot be partially truncated. It is either completely removed or completely included. A value cannot be partially truncated to fit within the specified limits of the list returned by the LISTAGG function.
- When truncation occurs, a literal value replaces the missing values and the number of missing values is displayed. The literal value is defined within the ON OVERFLOW TRUNCATE clause. In the example in the slide, the concatenated string is truncated and replaced by '…' and there are two missing values. If you do not want to display the number of missing values, use the WITHOUT COUNT clause in the ON OVERFLOW TRUNCATE clause.



Using the `LISTAGG` Function with `DISTINCT`

19c

Use the `DISTINCT` option to easily find the distinct values:

- Eliminates the duplicate values generated by the `LISTAGG` function
- Creates code that is easier to maintain
- Executes faster

```
SQL> SELECT region_id,
            LISTAGG (DISTINCT country_name, ','
                     ON OVERFLOW TRUNCATE '...' WITHOUT COUNT)
            WITHIN GROUP (ORDER BY country_id) AS COUNTRIES
     FROM country_table
     GROUP BY region_id
     ORDER BY region_id;

REGION_ID COUNTRIES
--------- ------------------------------
       10 GERMANY,UNITED KINGDOM
       20 ARGENTINA,BRAZIL,CANADA,. . .
```

ORACLE

The `LISTAGG` aggregate function with the `DISTINCT` option performs the following steps:

1. Orders the rows for each group in a query according to the `ORDER BY` expression
2. Eliminates duplicate values from the specified expression before concatenating the values into a single string
3. Concatenates the values into a single string

How are duplicate values identified?

Two identical strings with trailing spaces are treated as duplicates. For example, strings `'a'` and `'a  '` are treated as duplicates. `LISTAGG` uses the first value encountered. Oracle uses blank-padded comparison semantics only when both values in the comparison are either expressions of data type `CHAR`, `NCHAR`, text literals, or values returned by the `USER` function.

`DISTINCT` and `ON OVERFLOW TRUNCATE` can be used together.

Summary

In this lesson, you should have learned how to:

- Use user-provided `root` or `sudo` credentials for automatic failed prerequisite checks, for automatic execution of fixups scripts and `root.sh`
- Use DBCA to clone or relocate a remote PDB, and to duplicate a database
- Create a PDB snapshot from a PDB
- Use data pump enhancements
- Improve the handling of very long lists of values generated by the `LISTAGG` function

Practices Environment - 1

Each student has one virtual machine (VM1) to use during the class.

- Oracle Database 19c installed
- One CDB named **`ORCL`** and its PDB named **`PDB1`**

Practices Environment - 2

Pre-created databases and instances with their respective PDBs:

- Data files in `/u02/app/oracle/oradata/`
- All backup files in `/u03/app/oracle/fast_recovery_area/ORCL`
- Password and init files in `$ORACLE_HOME/dbs`
- Diagnostics files in `/u01/app/oracle/diag/rdbms/orcl`
- TDE wallet in `/u01/app/oracle/admin/orcl/tde_wallet`
- Net files in `$ORACLE_HOME/network/admin`

ORACLE

The configuration of the environment used for practices of the course matches the configuration used on an Oracle Cloud Virtual Machine for the pre-created `ORCL` database of Oracle Database Cloud Service. Oracle Database Cloud Service gives you the ability to deploy Oracle databases in the cloud, with each database deployment containing a single Oracle database.

This is a good way to get familiar with the Oracle Database cloud environment that is preconfigured for cloud customers.

Practice 1: Overview

- 1-1: Discovering the Practices Environment
- 1-2: Installing Oracle Database 19c with Automatic `root.sh` Execution
- 1-3: Cloning a PDB by Using DBCA in Silent Mode
- 1-4: Relocating a PDB by Using DBCA in Silent Mode
- 1-5: Duplicating a CDB by Using DBCA in Silent Mode
- 1-6: Decreasing TTS Import Time
- 1-7: Decreasing TTS Export Time
- 1-8: Omitting the Column Encryption Attribute During Import
- 1-9: Avoiding Errors Due to Values Generated by `LISTAGG`

2

Using Security Enhancements

ORACLE

Overview

- This module focuses on the security enhancements of Oracle Database 19c.
- It complements the topics covered in:
  - The 5-day *Oracle Database 12c: 12.2 New Features for 12.1 Administrators Ed 1* course
  - The 5-day *Oracle Database 12c R2: New Features for Administrators Ed 1* course *Part 2*
  - The 3-day *Oracle Database 18c: New Features for Administrators Ed 1* course
- Previous experience with Oracle Database 12*c* and 18c is required for a full understanding of security enhancements.

Refer to *Oracle Database New Features Guide 19c, Oracle Database Security Guide 19c, Oracle Database Advanced Security Guide 19c, and Oracle Database Vault Administrator's Guide 19c.*

This module follows either the 5-day *Oracle Database 12c: 12.2 New Features for 12.1 Administrators Ed 1* course or the 5-day *Oracle Database 12c R2: New Features for Administrators Part 2 Ed 1* course or the 3-day *Oracle Database 18c: New Features for Administrators Ed 1* course. These courses are designed to introduce the enhancements of Oracle Database 12*c* Release 2 (12.2.0.1) and Oracle Database 18c that are applicable to work that is usually performed by database administrators and related personnel.

The module is designed to introduce the major enhancements of Oracle Database 19c related to security.

The module consists of an instructor-led lesson and practices that enable you to see how certain new functionalities behave.

Objectives

After completing this lesson, you should be able to:

- Observe that Oracle-supplied schemas are schema-only accounts
- Protect application data by using Database Vault Operations Control
- Constrain `AUDIT POLICY` and `NOAUDIT POLICY` SQL commands for individual unified audit policies in database vault command rules
- Audit direct user activities in the database without collecting indirect user activity audit data
- Use privilege analysis with Oracle Database Enterprise Edition
- Handle operations on Oracle-managed and user-managed tablespaces encrypted with TDE
- Inform the database of the location change of the password file

Schema-Only Accounts

18c

Ensures that a user cannot log in to the instance:

- Enforces data access through the application
- Secures schema objects
  - Prevents objects from being dropped by the connected schema
- A schema-only account cannot be:
  - Granted administrator privileges
  - Used in database links

```
SQL> CREATE USER schema_noauth NO AUTHENTICATION;
```

19c

- Keeps Oracle-supplied schemas created with no authentication as schema-only accounts
- Grants and revokes administrator privileges to and from schema-only accounts

In Oracle Database 18c, an account can be created with the `NO AUTHENTICATION` clause to ensure that the account is not permitted to log in to the instance. Removing the password and the ability to log in essentially just leaves a schema. The schema account can be altered to allow login, but can then have the password removed. The `ALTER USER` statement can be used to disable or re-enable the login capability.

The `DBA_USERS` view has a new column, `AUTHENTICATION_TYPE`, which displays `NONE` when `NO AUTHENTICATION` is set, and `PASSWORD` when a password is set.

Oracle Database 19c creates most of the Oracle-supplied schemas as schema-only accounts. These schemas have their passwords removed to prevent users from authenticating to these accounts. This is the case for the Database Vault–supplied schemas such as `DVSYS`, `DVF`, and the Oracle Label Security–supplied `LBACSYS` schema. The benefit of this functionality is that administrators no longer have to periodically rotate the passwords for these Oracle Database–provided schemas. This functionality also reduces the security risk of attackers using default passwords to hack into these accounts.

Be aware that sample schemas such as `HR` still exist. They should not remain on production databases.

Accounts with administrator privileges such as `SYSOPER` or `SYSBACKUP` can now be schema-only accounts. Schema-only accounts can be granted administrator privileges.

Database Vault Operations Control

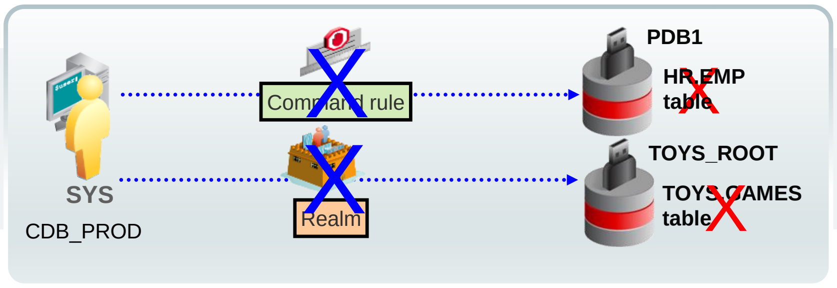
19c

```
DBA_DV_STATUS
DV_APP_PROTECTION
```

Protection in addition to Database Vault realms and command rules:

- Implements separation of duty and access controls without PDB owners having to do anything at all
- Blocks DBAs (managing the database) and/or application DBAs (updating and tuning applications) from accessing PDB local data
- Requires Database Vault to be configured and enabled in the CDB root, not in PDBs

There is no need to create realms or command rules to protect PDB data from DBA common users.



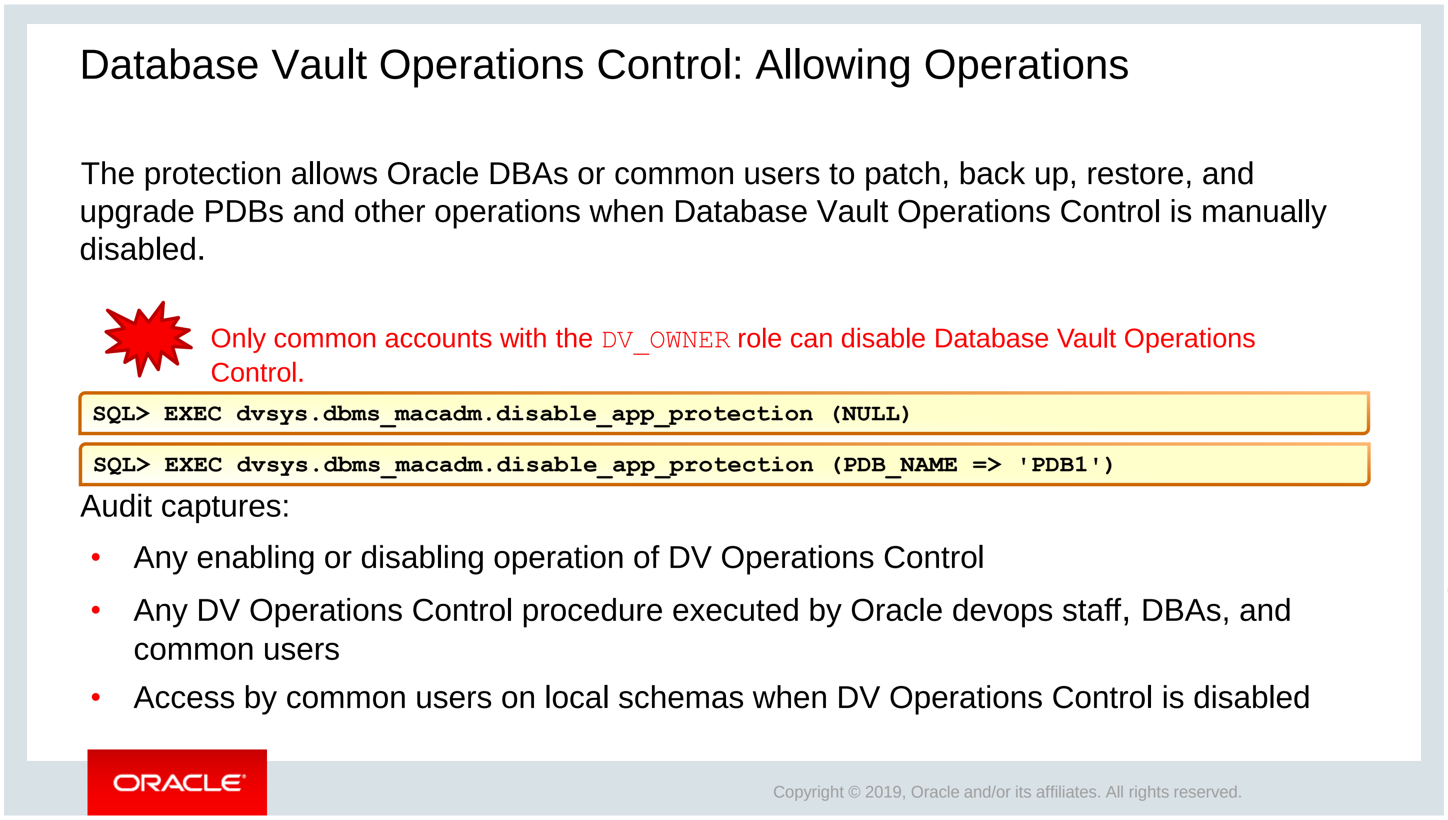
Oracle Database Vault provides command rules and realms to protect sensitive data from users with system privileges and object privileges (mandatory realms). A command rule controls whether a particular SQL command can be executed for the given circumstance. A realm can prevent a user from accessing protected objects if the user is not authorized in the realm.

With Oracle Database 19c, Database Vault provides an extra layer of protection on the database objects in addition to the default RDBMS security. Database Vault Operations Control creates a wall between common users in databases and the customer database data that resides in the associated PDBs. Database Vault Operations Control prevents common users from accessing application local data that resides in regular PDBs, application roots, or application PDBs. Application common schemas are treated the same as PDB local schemas because they are part of the customer data.

The capability allows you to store sensitive data for your business applications and manage the database without having to access the sensitive data in PDBs.

Because the user managing Database Vault Operations Control needs to log in as a highly privileged user for maintenance purposes, there needs to be a control in place to prevent him/her from accidentally or maliciously accessing PDB customer data. When users are accessing the database as a highly privileged user, such as `SYS` or common users with the `DBA` role or with powerful system privileges, they can easily access customer data in PDBs.

To prevent highly privileged users from accessing customer data in PDBs, Database Vault Operations Control is enabled on the CDB root for controlling CDB-level access. However, Database Vault is not required to be configured or enabled in PDBs, but must be configured and enabled at the CDB root level. Database Vault separation of duty is still not enforced in the customer PDB unless the customer enables Database Vault in the PDB. This allows customers to operate the database in a way they are familiar with, either with or without Database Vault, and are still protected by Database Vault Operations Control.



Database Vault Operations Control: Allowing Operations

The protection allows Oracle DBAs or common users to patch, back up, restore, and upgrade PDBs and other operations when Database Vault Operations Control is manually disabled.

Only common accounts with the `DV_OWNER` role can disable Database Vault Operations Control.

```
SQL> EXEC dvsys.dbms_macadm.disable_app_protection (NULL)
```

```
SQL> EXEC dvsys.dbms_macadm.disable_app_protection (PDB_NAME => 'PDB1')
```

Audit captures:

- Any enabling or disabling operation of DV Operations Control
- Any DV Operations Control procedure executed by Oracle devops staff, DBAs, and common users
- Access by common users on local schemas when DV Operations Control is disabled

Because database patching, backing up, restoring, and upgrading do not touch customer schemas, with Database Vault Operations Control protecting the customer data, DBAs and privileged common users can still run tools as `SYSDBA` to patch the database (with `DV_PATCH_ADMIN` granted commonly in the CDB root). They can also update Oracle dictionaries in both CDB and PDBs to back up, restore, and upgrade the database without the capability of accessing customer data.

The behavior of local Database Vault policies against local users' access still work as before in a PDB.

Database external users that are using external database authentication methods, such as by network or other third party services, are not impacted by the DBAs control. The capability only concerns users that have gone through database authentication and are mapped to a native database user.

Only common users with the `DV_OWNER` role granted locally can use the `DVSYS.DBMS_MACADM.DISABLE_APP_PROTECTION` procedure to disable Database Vault Operations Control at different levels in the CDB. Local Database Vault owners in PDBs cannot enable or disable Database Vault Operations Control. This way, customers cannot turn on or off Database Vault Operations Control in their PDBs. The `DV_OWNER` role is granted locally in the CDB root so that `DV_OWNER` in the CDB root cannot interfere with Database Vault enabled by customers in PDBs. Only common users with the locally granted `DV_OWNER` role in the CDB root can run the Database Vault Operations Control APIs, the `DVSYS.DBMS_MACADM` new procedures.

- Set `NULL` as the parameter value to disable control for the whole CDB
- Set the `PDB_NAME` parameter to a PDB name to disable control for a specific PDB

A common user locally granted the `DV_OWNER` role can use the `DBMS_MACADM.ENABLE_APP_PROTECTION` procedure to enable Database Vault Operations Control at different levels in the CDB. This is the default configuration.

All the above auditing records go to a unified audit trail in the current container. If the access happens in a PDB, the audit records go to the PDB audit trail. DV Operations Control mandatory audit goes to the CDB root unified audit trail because the operation can only be done on the CDB root. When common users access local schemas when DV Operations Control is disabled, audit policies need to be created.

Database Vault Operations Control: Granting the DV_OWNER Role

- Only common users with the DV_OWNER role granted locally in the CDB root can execute Database Vault Operations Control procedures.
  - sys.configure_dv
  - dvsys.dbms_macadm.disable_app_protection / enable_app_protection
  - dvsys.dbms_macadm.add_app_exception / delete_app_exception
- Grants the DV_OWNER role locally in the CDB root:

```
SQL> CONNECT / AS SYSDBA
SQL> EXEC sys.congifure_dv (DVOWNER_UNAME => 'c##dvo', -
                             DVACCTMGR_UNAME => 'c##dvacctmgr',-
                             FORCE_LOCAL_DVOWNER => TRUE)
```

- Still grants the DV_ACCTMGR role commonly in the CDB root or application roots 18c

How is a common user granted the DV_OWNER role locally in the CDB root?

The FORCE_LOCAL_DVOWNER parameter in the SYS.CONFIGURE_DV procedure allows you to define the scope of the DV_OWNER role grant operation. The parameter is not applicable in PDBs such as application roots, and application and regular PDBs.

- TRUE: The DV_OWNER role is granted locally regardless of the current container. Even in the CDB root, the role is granted locally to the common Database Vault owner.
- FALSE: The DV_OWNER role is granted to the common Database Vault owner with the same scope as the container:
  - In the CDB root, the role is granted commonly to the common Database Vault owner.
  - In application roots, the role is granted commonly to the common Database Vault owner.
  - In regular and application PDBs, the role is granted locally to the common Database Vault owner.
- Not specified: The default behavior is the same as when it is set to FALSE.

The grant operation of the DV_ACCTMGR role to the common Database Vault account manager is not affected by the parameter. The SYS.CONFIGURE_DV procedure grants:

- The DV_ACCTMGR role commonly to the common Database Vault account manager in the CDB root
- The DV_ACCTMGR role commonly to the common Database Vault account manager in the application root
- The DV_ACCTMGR role locally to the common Database Vault account manager in the application PDB or regular PDB

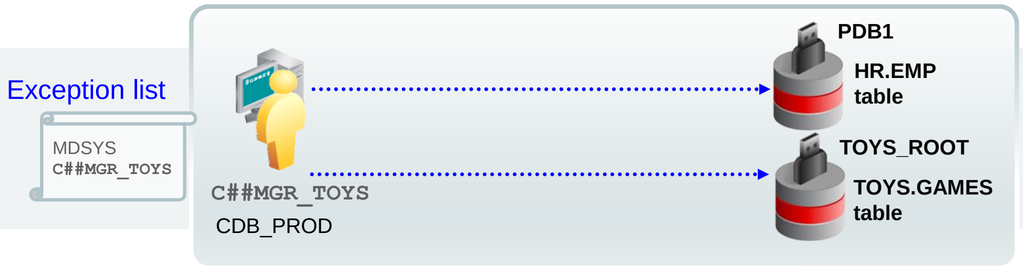
The DV_ACCTMGR role needs to be granted commonly in the CDB root and in application roots. Otherwise, the Database Vault account manager will not be able to create common users in the CDB root and in application roots.

Database Vault Operations Control: Setting an Exception List

DVSYS.DBA_DV_APP_EXCEPTION

Exception list:

- Database features such as Oracle TEXT and Oracle Spatial use preconfigured common users to access local schemas in PDBs.
- The DV_OWNER in the CDB root maintains the list of common users and packages so that applications and options can work as usual.



```
SQL> EXEC dvsys.dbms_macadm.add_app_exception (owner        => 'MDSYS', -
                                               package_name => 'PACK_APP1')
```

Some database features such as Oracle TEXT and Oracle Spatial might use preconfigured common users to access PDBs. It is also possible that Oracle-provided features use a common user to access local schemas.

To help in the smooth adoption of Database Vault Operations Control, the DV_OWNER (the Database Vault user granted the DV_OWNER role) configures an exception list of common users and packages on the CDB root so that applications can work as usual. With this in place, SQL statements generated from these packages are not blocked by Database Vault Operations Control when they access local customer data.

Ideally, this exception list is empty after Oracle Database teams update their features to stop accessing local user data. If required, Oracle Database features or options users, such as MDSYS of the Spatial option, need to be in the exception list so that the feature or option can work.

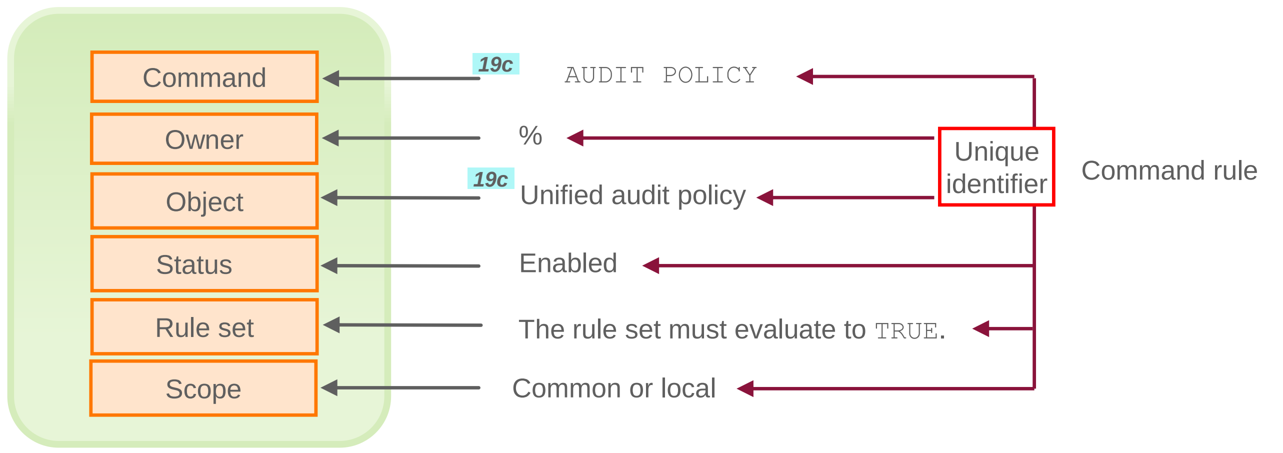
The DV_OWNER uses the DBMS_MACADM.ADD_APP_EXCEPTION procedure to allow a common user or a package to access local schemas. The exception list is effective for the whole CDB. When the exception is for a package, owner statements from the given package can access local schemas. The package must be owned by a common user defined in the exception, and its procedures must be definer's rights procedures.

The DBMS_MACADM.DELETE_APP_EXCEPTION procedure is used to remove the exception for a common user or a package to access local schemas.

The common schemas and packages names that are in the exception list are stored in the DVSYS.DBA_DV_APP_EXCEPTION view in the CDB root. A query from the view in a PDB results in no rows.

The DVSYS.DBA_DV_STATUS view provides the Database Vault Operations Control status for the current container. A query from the view works in the CDB root and in PDBs.

Creating Database Vault Command Rules to Protect Audit Policies

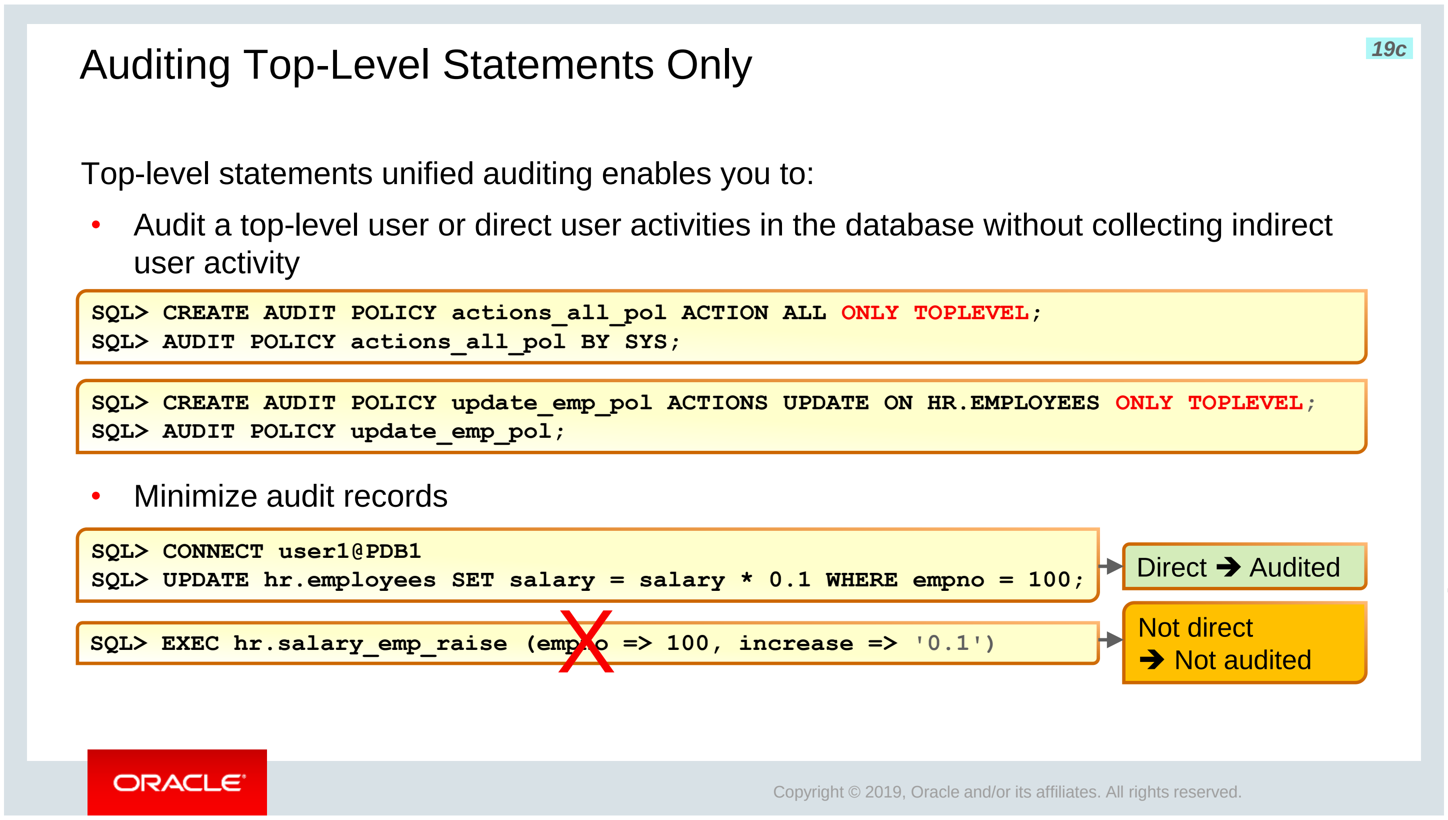


```
SQL> EXEC dvsys.dbms_macadm.create_command_rule(-
              command      => 'AUDIT POLICY', rule_set_name => 'RS_TRUE',-
              object_owner => '%', object_name    => 'AUDPOL1',-
              enabled => DBMS_MACUTL.G_YES, scope   => DBMS_MACUTL.G_SCOPE_LOCAL)
```

Until Oracle Database 19c, you can use Database Vault command rules to protect a wide range of SQL statements, in addition to basic Oracle Database DDL and DML statements. Command rules do not have a name. They are known by the unique combination of a command, an object owner, and an object name. Therefore, you can create multiple `SELECT` command rules (for example, as long as the object owner and object name attributes are different in these command rules).

- **Command:** The command that is being protected against. This includes:
  - Most DDL (`CREATE`, `ALTER`, `DROP`, `TRUNCATE`, `AUDIT`, and `NOAUDIT`)
  - `ALTER SYSTEM`
  - `EXECUTE`
  - `SELECT`, `INSERT`, `UPDATE`, and `DELETE`
- **Status:** Indicates whether the command rule is currently in effect or not. To disable the effects of a command rule, set this to Disabled.
- **Object Owner:** The owner of the object or objects that this rule applies to. This is not applicable in some cases, such as with the `FLASHBACK DATABASE` command. It must be “%” because an audit policy is a non-schema object.
- **Object Name:** The object within the Object Owner schema that is being protected.
- **Rule Set:** The name of the rule set that protects these objects from this command. If the rule set evaluates to `TRUE` at the time of attempting the command, the command succeeds. Otherwise, a rule set violation error is returned.
- **Scope:** The scope defines whether the command rule is applied commonly in all PDBs or locally within a PDB.

Database Vault command rules are not able to differentiate which users can enable and disable different unified audit policies. Oracle Database 19c enables you to directly specify a unified policy name as part of a command rule. The command rule applies to both `AUDIT` and `NOAUDIT` commands on the unified audit policy. In the example in the slide, `AUDPOL1` is the name of a unified audit policy.



Auditing Top-Level Statements Only

19c

Top-level statements unified auditing enables you to:

- Audit a top-level user or direct user activities in the database without collecting indirect user activity

```
SQL> CREATE AUDIT POLICY actions_all_pol ACTION ALL ONLY TOPLEVEL;
SQL> AUDIT POLICY actions_all_pol BY SYS;
```

```
SQL> CREATE AUDIT POLICY update_emp_pol ACTIONS UPDATE ON HR.EMPLOYEES ONLY TOPLEVEL;
SQL> AUDIT POLICY update_emp_pol;
```

- Minimize audit records

```
SQL> CONNECT user1@PDB1
SQL> UPDATE hr.employees SET salary = salary * 0.1 WHERE empno = 100;
```

Direct ➔ Audited

```
SQL> EXEC hr.salary_emp_raise (empno => 100, increase => '0.1')
```

Not direct ➔ Not audited

ORACLE

Oracle Database 19c enables unified auditing to audit only top-level user directly issued events, without the overhead of indirect SQL statements. Top-level statements are SQL statements that users directly issue. These statements can be important for both security and compliance. SQL statements run from within PL/SQL procedures or functions are not considered top level because they may be less relevant for auditing purposes.

The `CREATE AUDIT POLICY` statement can include or exclude top-level statement audit records in the unified audit trail for any user by using the `ONLY TOPLEVEL` clause.

The first example in the slide shows how to define an audit policy that captures all top-level statements executed by the `SYS` user.

The second example shows how to limit the audit trail to top-level instances of the `UPDATE` statement on the `HR.EMPLOYEES` table. When a user executes an `UPDATE` command on the `HR.EMPLOYEES` table, the action is audited, whereas when the user executes the same command through a procedure, the action is not audited.

Privilege Analysis

Privilege analysis runs a dynamic analysis of users and applications:

- Finds privileges and roles that are used and unused
- Helps you revoke unnecessary privileges and roles granted

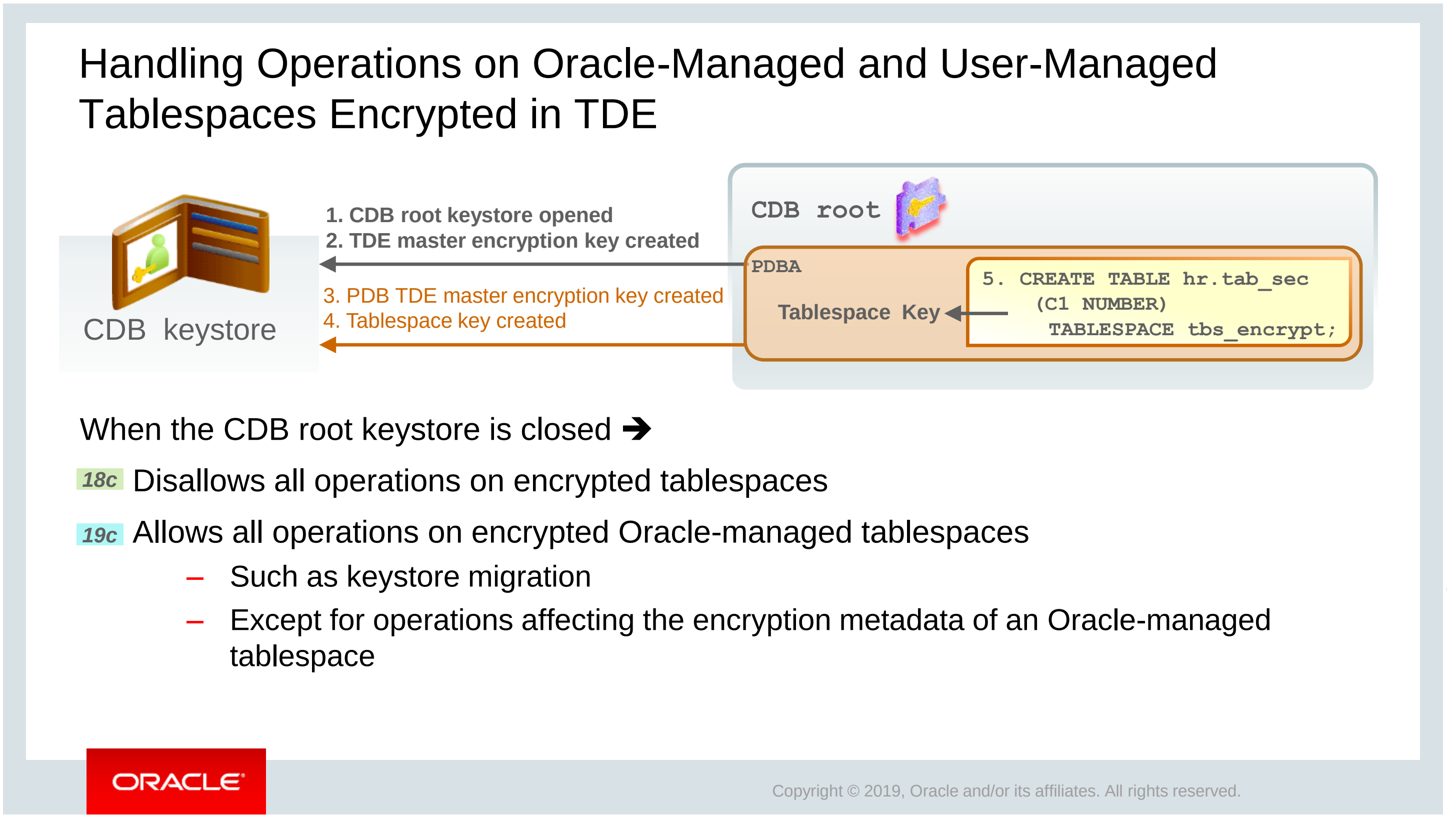
18c Privilege analysis is available as part of Oracle Database Vault.

19c Privilege analysis is available as part of Oracle Database Enterprise Edition.

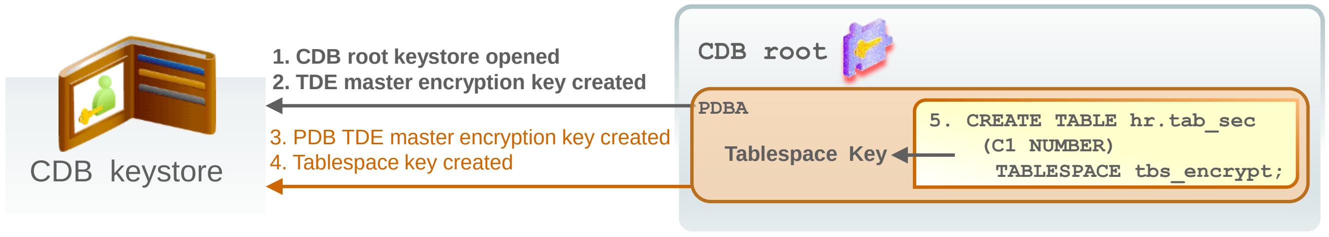
ORACLE

Privilege analysis reduces the work to implement the least privileges best practices by showing exactly what is used and unused by each account. It is highly effective and is designed to work in production, test, and development databases.

Privilege analysis is now available as part of Oracle Database 19c Enterprise Edition. The Database Vault option is not required to use the feature starting with Oracle Database 19c.



Handling Operations on Oracle-Managed and User-Managed Tablespaces Encrypted in TDE


CDB keystore
1. CDB root keystore opened
2. TDE master encryption key created
3. PDB TDE master encryption key created
4. Tablespace key created
CDB root
PDBA
Tablespace Key

```
5. CREATE TABLE hr.tab_sec
     (C1 NUMBER)
       TABLESPACE tbs_encrypt;
```

When the CDB root keystore is closed ➔

18c Disallows all operations on encrypted tablespaces

19c Allows all operations on encrypted Oracle-managed tablespaces

- Such as keystore migration
- Except for operations affecting the encryption metadata of an Oracle-managed tablespace

Using a TDE keystore to encrypt data requires the keystore to be closed for certain keystore management operations:

- When you have to rotate the password of the Oracle Key Vault endpoint using the `okvutil changepwd` command
- When you migrate the keystore type from being a file-based wallet to using a Hardware Security module

In Oracle Database 18c, users also close the TDE keystore to disallow operations on encrypted tablespaces. The behavior when the keystore is closed does not depend on the type of tablespace being accessed. Operations on user-managed tablespaces or Oracle-managed tablespaces, like `SYSTEM`, `SYSAUX`, `UNDO`, and `TEMP` tablespaces, raise the `ORA-28365 "wallet is not open"` error.

In Oracle Database 19c, the behavior depends on what kind of tablespace is being accessed:

- Operations on user-managed tablespaces still raise the `ORA-28365 "wallet is not open"` error.
- Operations on the data of Oracle-managed tablespaces are unaffected by the closing of the TDE keystore and continue to work, including customer-issued SQL. Nevertheless, operations which affect the encryption metadata of an Oracle-managed tablespace as opposed to the data it contains are prevented with an `ORA-28365 "wallet is not open"` error because the TDE master encryption key is not available when the TDE keystore is closed.
- While the TDE keystore is unavailable, operations on the data of Oracle-managed tablespaces are unaffected and continue to work if the instance initialization parameter `COMPATIBLE` is set to a value greater than 19.0.0.0.

If TDE is used, user-managed tablespaces are most likely encrypted, and there could be Oracle-managed tablespaces also encrypted. Closing the keystore should be temporary, just to be able to perform key management operations that require the keystore to be closed and reopened internally. The keystore needs to stay open in general if there are encrypted user-managed tablespaces so that the applications can access user data.

Informing the Database of the Location Change of the Password File

```
SQL> SELECT file_name FROM v$passwordfile_info;

FILE_NAME
----------------------------------------------------------------------------
/scratch/oracledb/product/19c/db_home1/dbs/orapwORCL
```

The password file location changes:

```
$ orapwd file='+data/ORCL/orapwdb' dbuniquename = 'ORCL'
```

➔ Does not inform the database of the change

```
SQL> ALTER SYSTEM FLUSH PASSWORDFILE_METADATA_CACHE;
```

➔ Updates `FILE_NAME` in the `V$PASSWORDFILE_INFO` view

```
SQL> SELECT file_name FROM v$passwordfile_info;

FILE_NAME
----------------------------------------------------------------------------
+DATA/ORCL/orapwdb
```

If the password file location changes, the `ALTER SYSTEM FLUSH PASSWORDFILE_METADATA_CACHE` command should be executed in the database to inform the database of the change. This command flushes the metadata cache and the subsequent logons use the new password file. The flush command also clears the database cache across RAC instances. There could be some delay in propagating the change. Until the flush is fully propagated to all the instances, there could be instances which would continue to use the old password file.

If the `V$PASSWORDFILE_INFO.FILE_NAME` columns do not show the password file location, use the `ALTER SYSTEM FLUSH PASSWORDFILE_METADATA_CACHE` command to flush the database cache to make the database refresh the password file location and point to the new password file. The command is useful when you store the password file on an ASM disk group, when the password file is used for SYSASM authentication on ASM instances, or when the Oracle ASM password file is restored from a backup.

Summary

In this lesson, you should have learned how to:

- Observe that Oracle-supplied schemas are schema-only accounts
- Protect application data by using Database Vault Operations Control
- Constrain `AUDIT POLICY` and `NOAUDIT POLICY` SQL commands for individual unified audit policies in database vault command rules
- Audit direct user activities in the database without collecting indirect user activity audit data
- Use privilege analysis with Oracle Database Enterprise Edition
- Handle operations on Oracle-managed and user-managed tablespaces encrypted with TDE
- Inform the database of the location change of the password file

Practices Environment - 1

Each student has one virtual machine (VM1) to use during class.

- Oracle Database 19c installed
- One CDB named **`ORCL`** and its PDB named **`PDB1`**

ORACLE

Practices Environment - 2

Pre-created databases and instances with their respective PDBs:

- Data files in `/u02/app/oracle/oradata/`
- All backup files in `/u03/app/oracle/fast_recovery_area/ORCL`
- Password and init files in `$ORACLE_HOME/dbs`
- Diagnostics files in `/u01/app/oracle/diag/rdbms/orcl`
- TDE wallet in `/u01/app/oracle/admin/orcl/tde_wallet`
- Net files in `$ORACLE_HOME/network/admin`

The configuration of the environment used for the practices of the course matches the configuration used on the Oracle Cloud Virtual Machine for the pre-created `ORCL` database of Oracle Database Cloud Service. Oracle Database Cloud Service gives you the ability to deploy Oracle databases in the cloud, with each database deployment containing a single Oracle database.

This is a good way to get familiar with the Oracle Database Cloud environment that is preconfigured for cloud customers.

Practice 2: Overview

- 2-1: Exploring Oracle-Supplied Schema-Only Accounts
- 2-2: Protecting Application Data by Using Database Vault Operations Control
- 2-3: Constraining `AUDIT POLICY` and `NOAUDIT POLICY` SQL Commands with Oracle Database Vault Command Rules
- 2-4: Auditing Direct User Activities
- 2-5: Handling Operations on Oracle-Managed and User-Managed Tablespaces Encrypted in TDE

3

Using Availability Enhancements

ORACLE

Overview

- This module focuses on RMAN and flashback enhancements of Oracle Database 19c.
- It complements the RMAN topics covered in:
  - The 5-day *Oracle Database 12c: 12.2 New Features for 12.1 Administrators Ed 1* course
  - The 5-day *Oracle Database 12c R2: New Features for Administrators Part 1 Ed 1* course
  - The 3-day *Oracle Database 18c: New Features for Administrators Ed 1*
- Previous experience with Oracle Database 12*c* or 18c is required for a full understanding of RMAN and flashback enhancements.

Refer to *Oracle Database New Features Guide 19c* and *Oracle Database Backup and Recovery User's Guide 19c.*

ORACLE

This module follows either the 5-day *Oracle Database 12c: 12.2 New Features for 12.1 Administrators Ed 1* course or the 5-day *Oracle Database 12cR2: New Features for Administrators Part 1 Ed 1* course or the 3-day *Oracle Database 18c: New Features for Administrators Ed 1* course. These courses are designed to introduce the enhancements of Oracle Database 12*c* Release 2 (12.2.0.1) and Oracle Database 18c that are applicable to work that is usually performed by database administrators and related personnel.

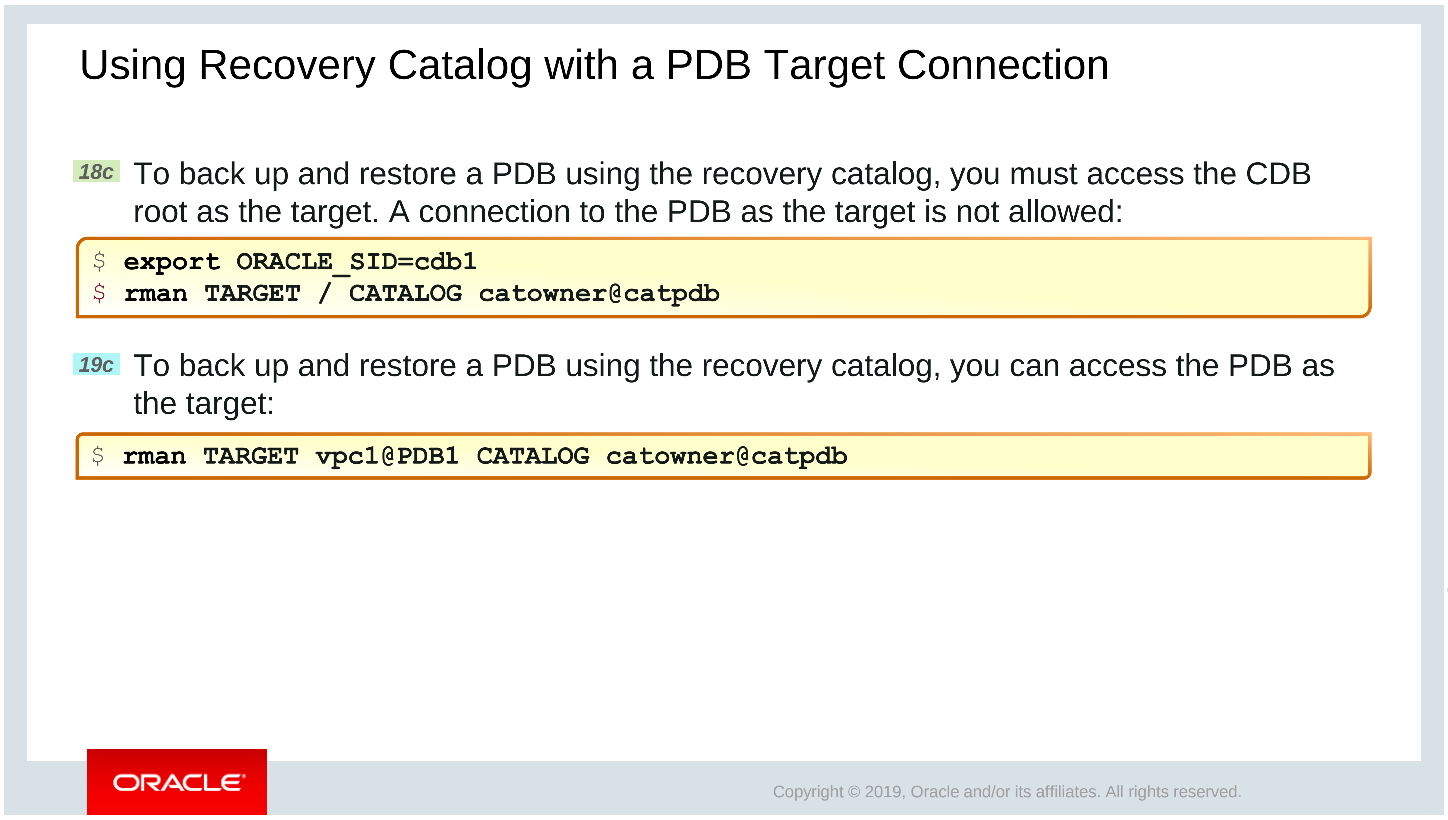
The module is designed to introduce the major enhancements of Oracle Database 19c related to RMAN and flashback operations.

The module consists of an instructor-led lesson and practices that enable you to see how certain new functionalities behave.

Objectives

After completing this lesson, you should be able to:

- Connect to the recovery catalog and to the PDB target database
- Use the Zstandard compression method as a real-time compression algorithm providing high compression ratios
- Reclaim flashback logs proactively
- Propagate restore points from a primary to standby site
- Automatically flash back a physical standby database when a primary database is flashed back



Using Recovery Catalog with a PDB Target Connection

18c To back up and restore a PDB using the recovery catalog, you must access the CDB root as the target. A connection to the PDB as the target is not allowed:

```
$ export ORACLE_SID=cdb1
$ rman TARGET / CATALOG catowner@catpdb
```

19c To back up and restore a PDB using the recovery catalog, you can access the PDB as the target:

```
$ rman TARGET vpc1@PDB1 CATALOG catowner@catpdb
```

Until Oracle Database 19c, there are two basic ways a PDB can be backed up or restored using RMAN:

- Connect to the CDB root as the target database, and then back up and restore the PDB
- Connect to the PDB as the target database, and then back up and restore the PDB

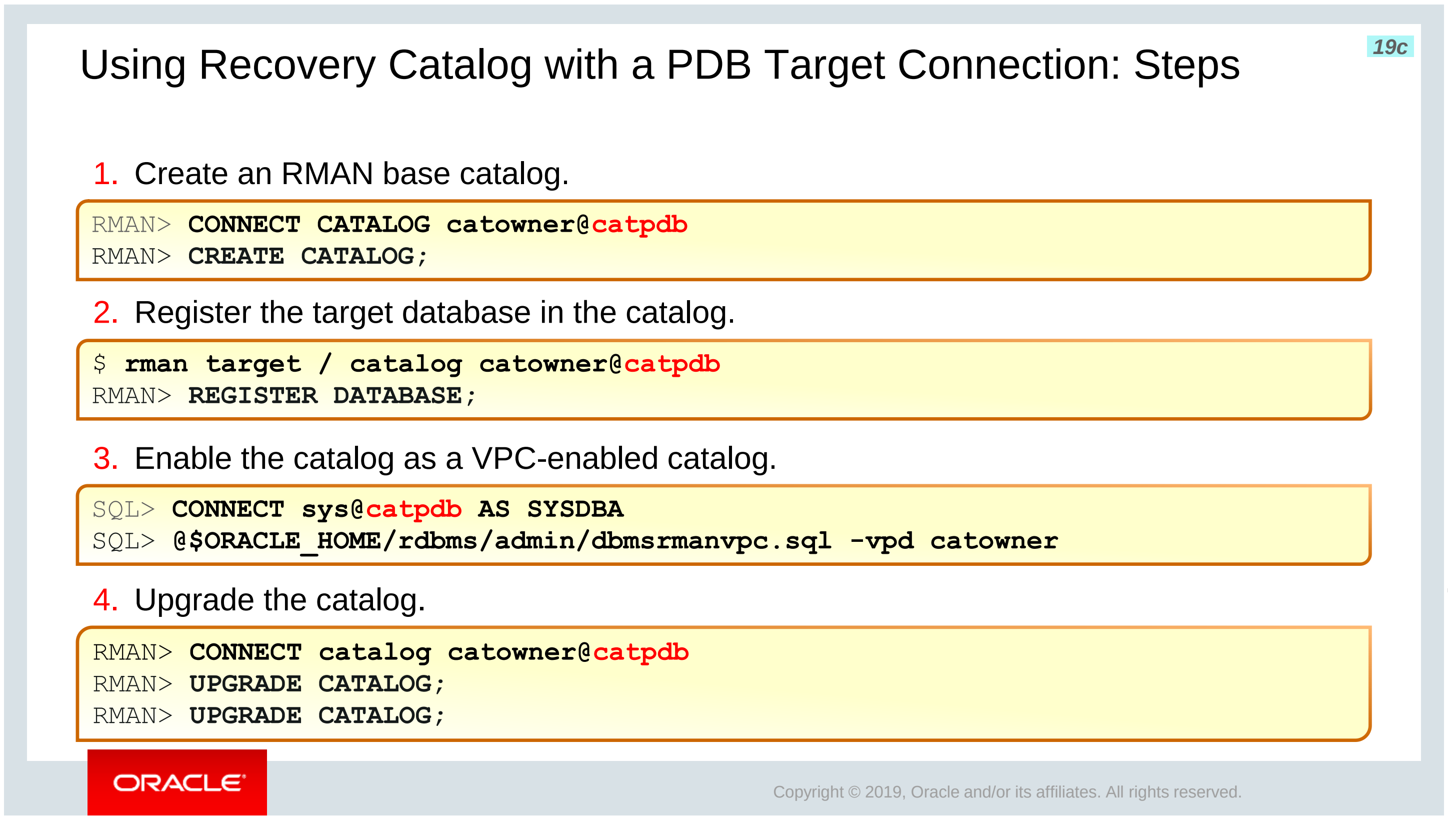
If recovery catalog is used, only the first option is supported.

Oracle Database 19c allows the recovery catalog connection when the user is connected to the PDB as the target database.

The recovery catalog hides the rows of other PDBs metadata when the target database is a PDB. The recovery catalog user has to be a Virtual Private Catalog (VPC) user or a VPC owner. Only a VPC-enabled catalog allows a connection to a PDB as the target database. A catalog can be VPC-enabled when Virtual Private Database (VPD)–related grants are done, and only if the catalog user is explicitly defined.

```
sqlplus sys@CATPDB
@$ORACLE_HOME/rdbms/admin/dbmsrmanvpc.sql -vpd catowner

Checking the operating user... Passed
Granting VPD privileges to the owner of the base catalog schema CATOWNER
========================================
VPD SETUP STATUS:
VPD privileges granted successfully!
Connect to RMAN base catalog and perform UPGRADE CATALOG.
```

In the example, `CATOWNER` is the catalog owner in the catalog PDB, `CATPDB`.



Using Recovery Catalog with a PDB Target Connection: Steps

19c

1. Create an RMAN base catalog.

```
RMAN> CONNECT CATALOG catowner@catpdb
RMAN> CREATE CATALOG;
```

2. Register the target database in the catalog.

```
$ rman target / catalog catowner@catpdb
RMAN> REGISTER DATABASE;
```

3. Enable the catalog as a VPC-enabled catalog.

```
SQL> CONNECT sys@catpdb AS SYSDBA
SQL> @$ORACLE_HOME/rdbms/admin/dbmsrmanvpc.sql -vpd catowner
```

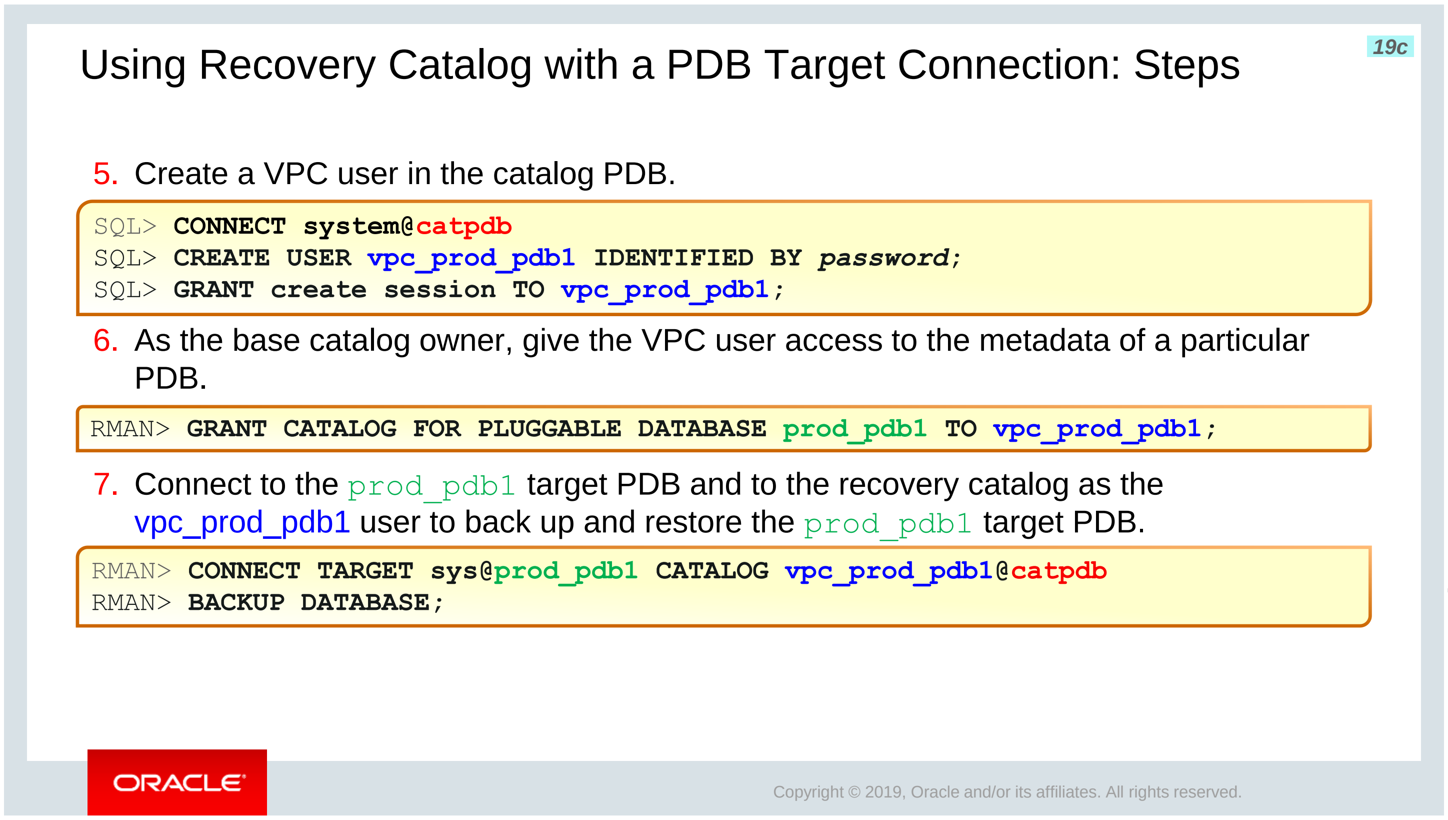
4. Upgrade the catalog.

```
RMAN> CONNECT catalog catowner@catpdb
RMAN> UPGRADE CATALOG;
RMAN> UPGRADE CATALOG;
```

ORACLE

To be able to connect to the recovery catalog and to the PDB as the target database, create virtual private RMAN catalogs (VPC) for groups of databases and users.

1. The catalog owner creates the catalog in the catalog PDB. Do not use the target PDB to be backed up as the PDB for the recovery catalog. The recovery catalog must be protected if the target PDB (or CDB) is lost.
2. The catalog owner registers the target PDB in the catalog.
3. The `SYS` user in the catalog PDB enables the catalog to be VPC-enabled. Virtual Private Database (VPD)–related grants are done to the user explicitly defined as the catalog owner.
4. Upgrade the catalog.



Using Recovery Catalog with a PDB Target Connection: Steps

19c

5. Create a VPC user in the catalog PDB.

```
SQL> CONNECT system@catpdb
SQL> CREATE USER vpc_prod_pdb1 IDENTIFIED BY password;
SQL> GRANT create session TO vpc_prod_pdb1;
```

6. As the base catalog owner, give the VPC user access to the metadata of a particular PDB.

```
RMAN> GRANT CATALOG FOR PLUGGABLE DATABASE prod_pdb1 TO vpc_prod_pdb1;
```

7. Connect to the `prod_pdb1` target PDB and to the recovery catalog as the vpc_prod_pdb1 user to back up and restore the `prod_pdb1` target PDB.

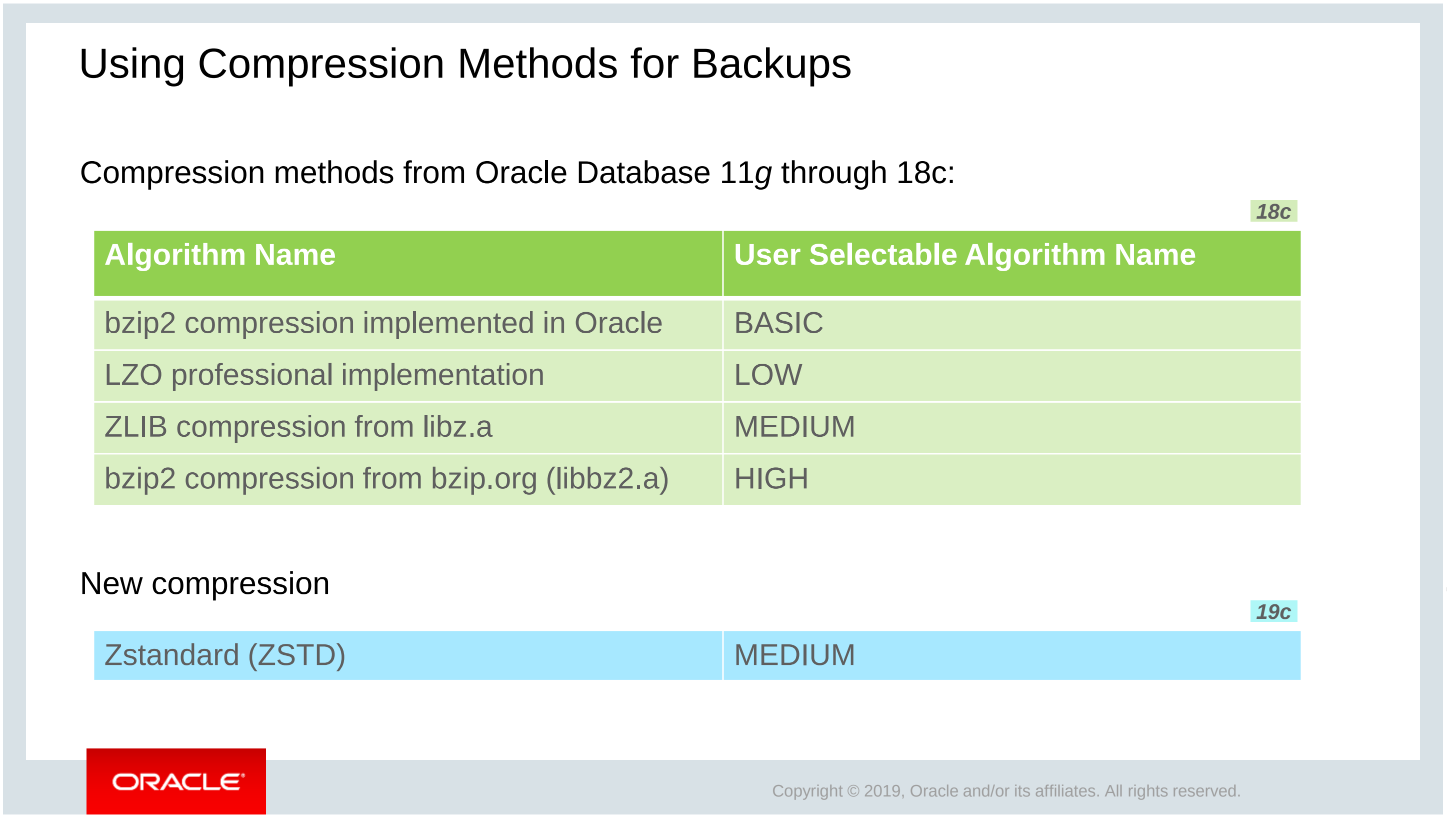
```
RMAN> CONNECT TARGET sys@prod_pdb1 CATALOG vpc_prod_pdb1@catpdb
RMAN> BACKUP DATABASE;
```

ORACLE

5. The DBA in the catalog PDB creates the users that will have access to the virtual private catalogs (VPC).
6. The catalog owner grants VPC users the `CATALOG FOR PLUGGABLE DATABASE` privilege for each of the PDBs that VPC users must have access to. In the example in the slide, the `vpc_prod_pdb1` VPC user is created in the target PDB. In the `catpdb` recovery catalog PDB, the user is granted the appropriate privilege to access the metadata of the backups for the `prod_pdb1` PDB.
7. The VPC user can then connect to the recovery catalog PDB for a particular target PDB, and thus complete RMAN operations. The VPC user can see only those target PDBs that have been granted. For most RMAN operations, you additionally need the `SYSDBA` or `SYSOPER` privileges on the target PDB.

If the catalog owner can grant VPC users the `CATALOG FOR PLUGGABLE DATABASE` privilege for each of the PDBs that VPC users need to access to, the catalog owner can also revoke the `CATALOG FOR PLUGGABLE DATABASE` privilege on these PDBs from VPC users.

```
RMAN> REVOKE CATALOG FOR PLUGGABLE DATABASE prod_pdb1
                          FROM vpc_prod_pdb1;
```



Using Compression Methods for Backups

Compression methods from Oracle Database 11*g* through 18c:

18c

Algorithm Name	User Selectable Algorithm Name
bzip2 compression implemented in Oracle	BASIC
LZO professional implementation	LOW
ZLIB compression from libz.a	MEDIUM
bzip2 compression from bzip.org (libbz2.a)	HIGH

New compression

19c

Zstandard (ZSTD)	MEDIUM

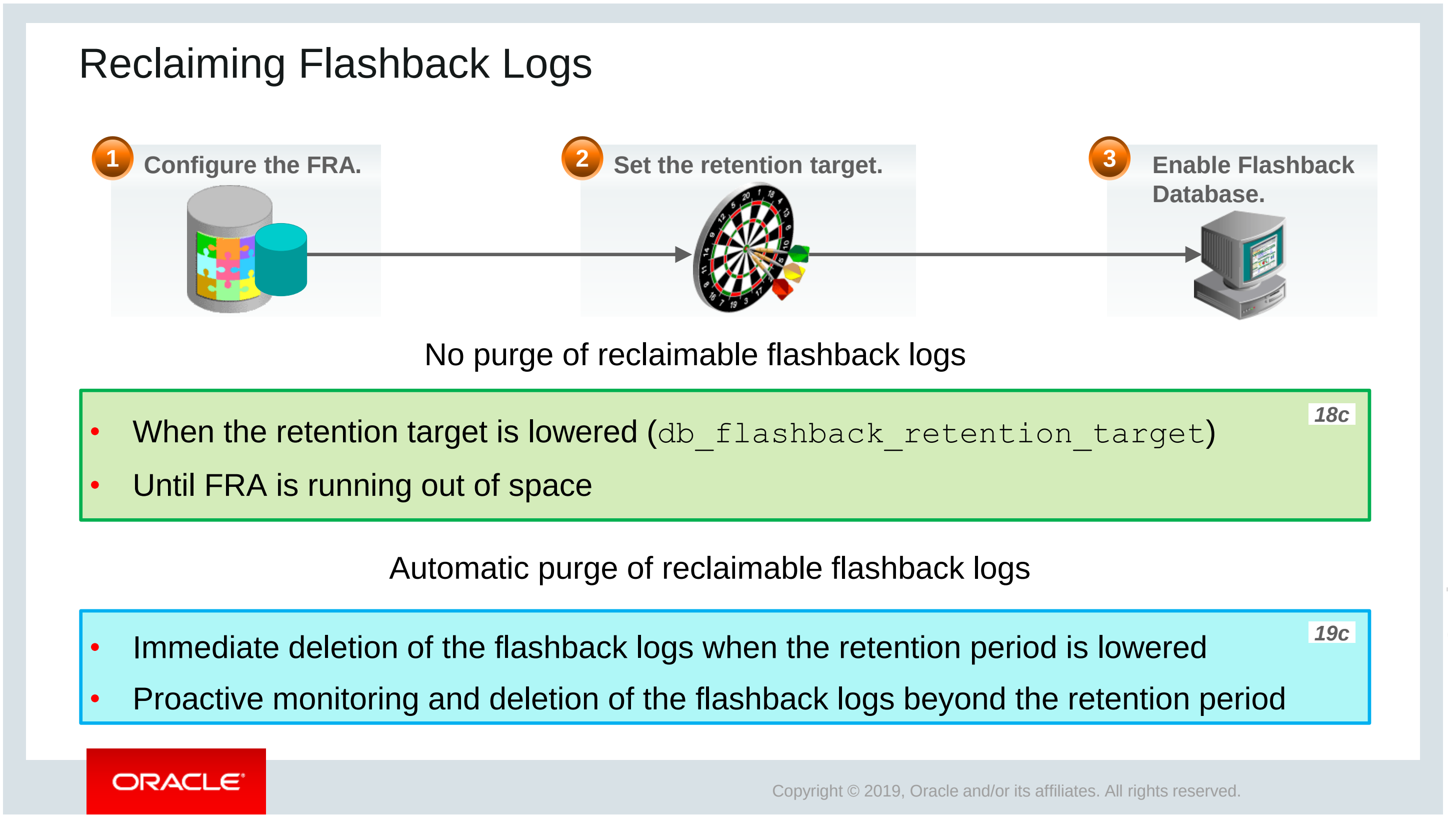
RMAN has four modes of compression starting from Oracle Database 11*g* through Oracle Database 18c for the user to select.

- `BASIC`
- `LOW`
- `MEDIUM`
- `HIGH`

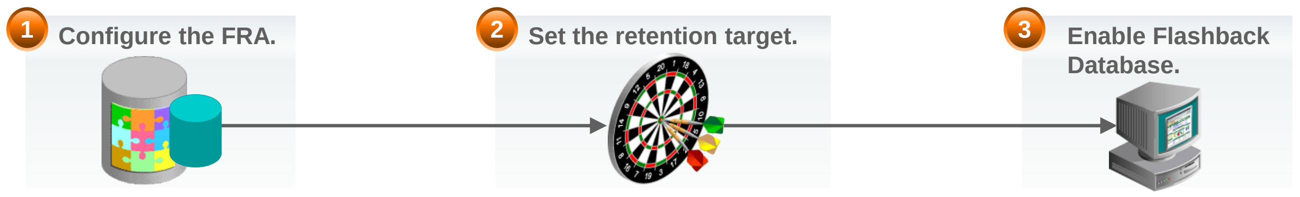
The table in the slide shows the mapping between the compression mode selected by the user in RMAN for backups and the actual algorithm used to compress. When a user tries to restore from an old backup, RMAN selects the appropriate algorithm to decompress it based on the Oracle database version and the algorithm that is used for compression.

Oracle Database 19c introduces a new compression algorithm for backups. Zstandard is a real-time compression algorithm providing high compression ratios. It offers a very wide range of compression and speed trade-off, while being backed by a very fast decoder. Zstandard library is provided as open source software using a BSD license. RMAN supports the Zstandard compression for backing up data files and archived log files. The current option of `MEDIUM` uses the ZSTD method of compression for backing up a data file or archived log file.

The usage of ZSTD compression requires the `COMPATIBLE` initialization parameter be set to 19.0 or greater because prior versions of Oracle do not know how to decompress ZSTD compressed data.



Reclaiming Flashback Logs



No purge of reclaimable flashback logs

18c

- When the retention target is lowered (`db_flashback_retention_target`)
- Until FRA is running out of space

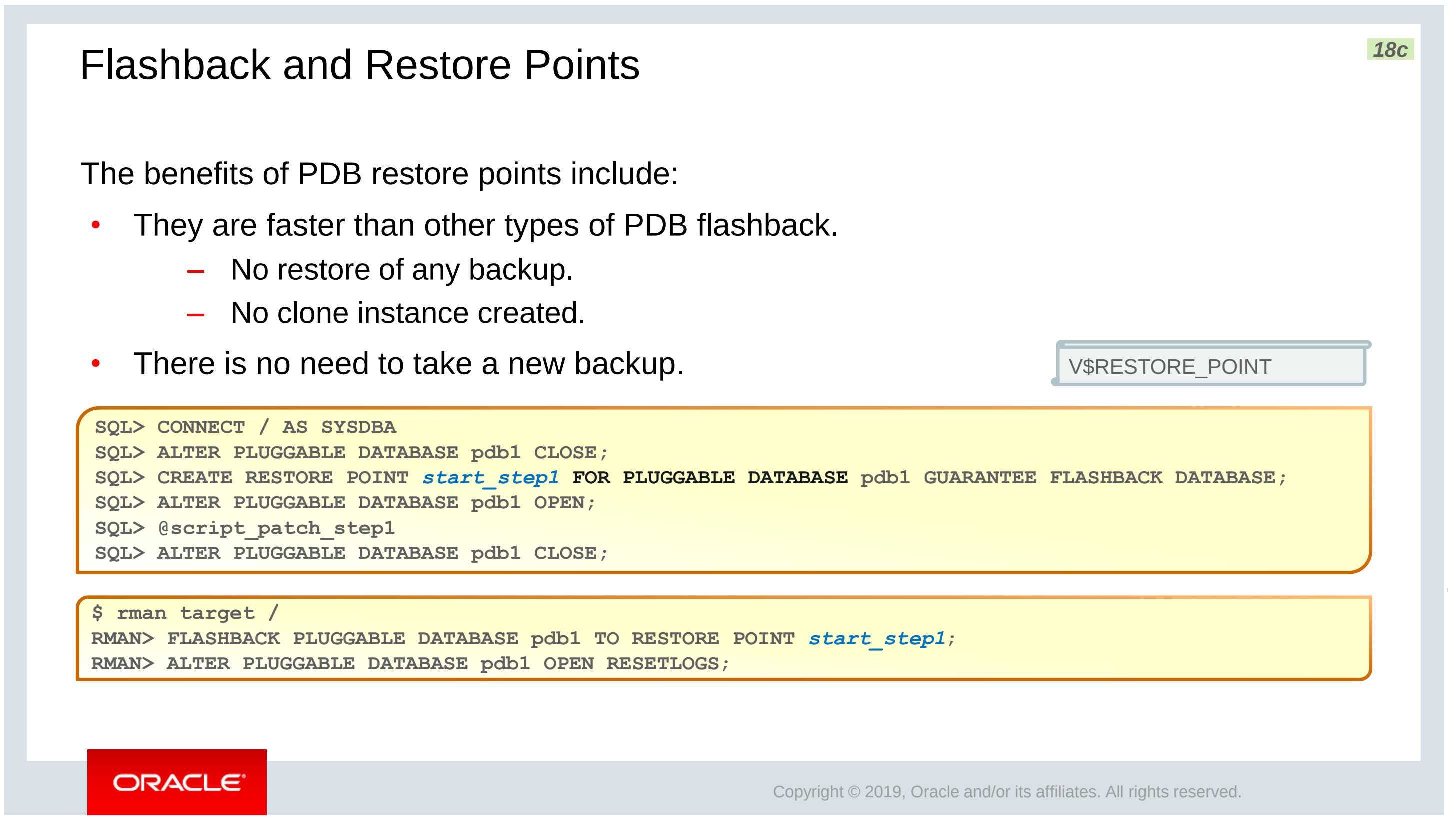
Automatic purge of reclaimable flashback logs

19c

- Immediate deletion of the flashback logs when the retention period is lowered
- Proactive monitoring and deletion of the flashback logs beyond the retention period

Until Oracle Database 19c, flashback logs that are beyond the flashback retention period set through the `db_flashback_retention_target` initialization parameter are marked as reclaimable but are not purged until there is space pressure. The reclaimable flashback logs are deleted only when the fast recovery area is running out of space. The DBA cannot manually delete the flashback logs. This means that when a backup needs to be completed and the FRA is 100 % full, the backup operation must wait until the reclaimable flashback logs are purged to free space in the FRA.

With Oracle Database 19c, flashback logs beyond the retention period are proactively deleted without degrading the flashback performance and before there is space pressure. The flashback log space is tracked and monitored proactively to provide flashback log retention in a transparent manner.



18c

Flashback and Restore Points

The benefits of PDB restore points include:

- They are faster than other types of PDB flashback.
  - No restore of any backup.
  - No clone instance created.
- There is no need to take a new backup.

V$RESTORE_POINT

```
SQL> CONNECT / AS SYSDBA
SQL> ALTER PLUGGABLE DATABASE pdb1 CLOSE;
SQL> CREATE RESTORE POINT start_step1 FOR PLUGGABLE DATABASE pdb1 GUARANTEE FLASHBACK DATABASE;
SQL> ALTER PLUGGABLE DATABASE pdb1 OPEN;
SQL> @script_patch_step1
SQL> ALTER PLUGGABLE DATABASE pdb1 CLOSE;
```

```
$ rman target /
RMAN> FLASHBACK PLUGGABLE DATABASE pdb1 TO RESTORE POINT start_step1;
RMAN> ALTER PLUGGABLE DATABASE pdb1 OPEN RESETLOGS;
```

ORACLE

Normal and Guaranteed Restore Points

PDB normal and guaranteed restore points are restore points that pertain to only a specific PDB and are visible within the PDB only.

- A normal PDB restore point is essentially a bookmark for a past point for that PDB.
- A guaranteed PDB restore point guarantees a PDB flashback to that restore point.

PDB flashback uses the PDB point-in-time recovery to recover the shared undo. RMAN automatically restores the shared undo and certain tablespaces in a clone instance, and recovers the data to the same point in time.

Clean Restore Points

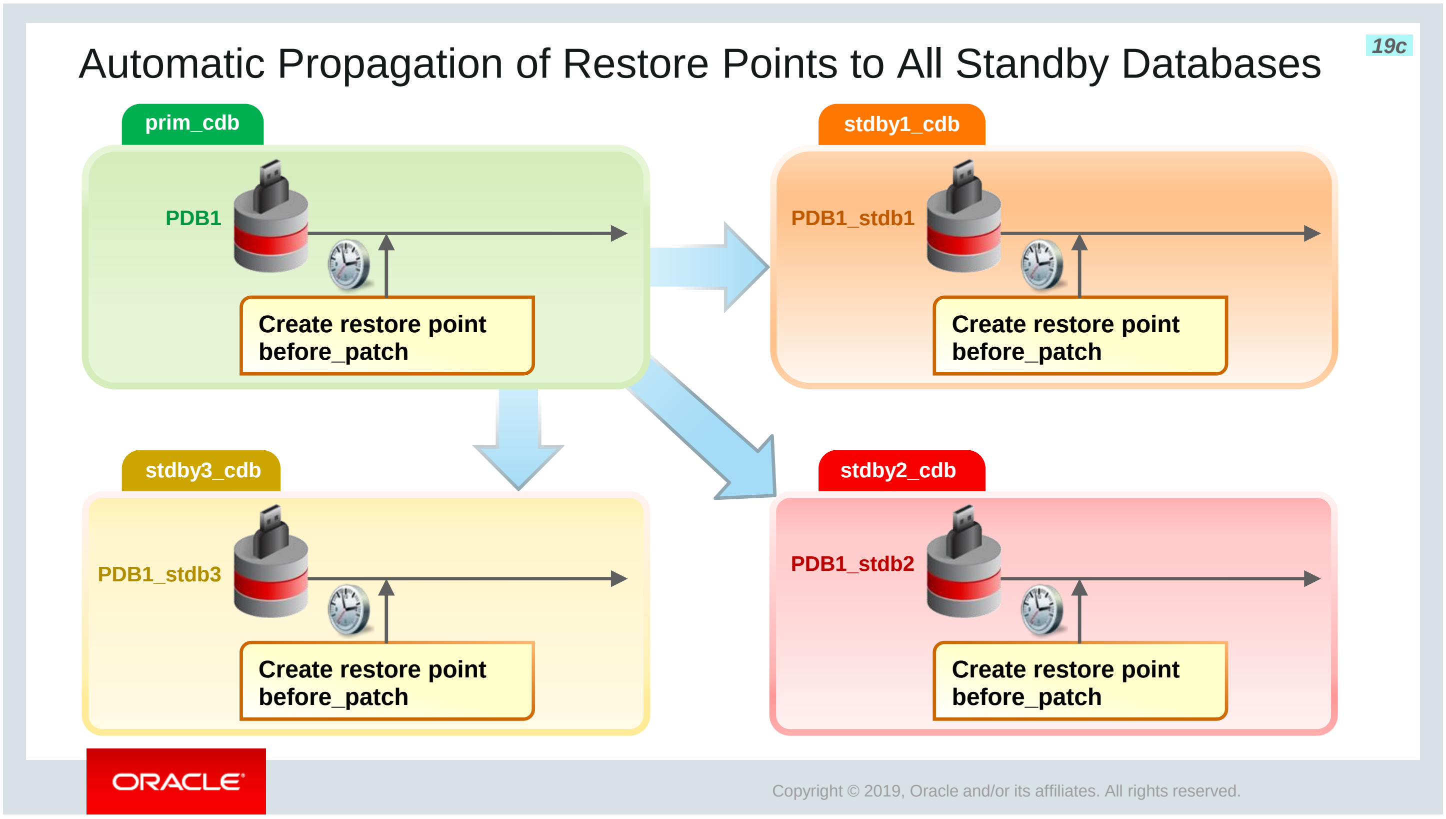
A clean PDB restore point can be created after a PDB is closed with no outstanding transactions and ONLY in shared undo mode. Thus, flashing back a PDB to a clean restore point is faster than other types of flashback because it does not require restoring backups or creating a clone instance. For example, a DBA expecting to roll back an application upgrade should consider creating a clean PDB guaranteed restore point before the application upgrade.

A restore point that is created while being connected to the CDB root without specifying the new `FOR PLUGGABLE DATABASE` clause is called a CDB restore point.

In case the PDB is the application root of an application container, all restore points are “clean” with local undo, because the availability of local undo means that the effects of active transactions at the time of the restore point can be undone.

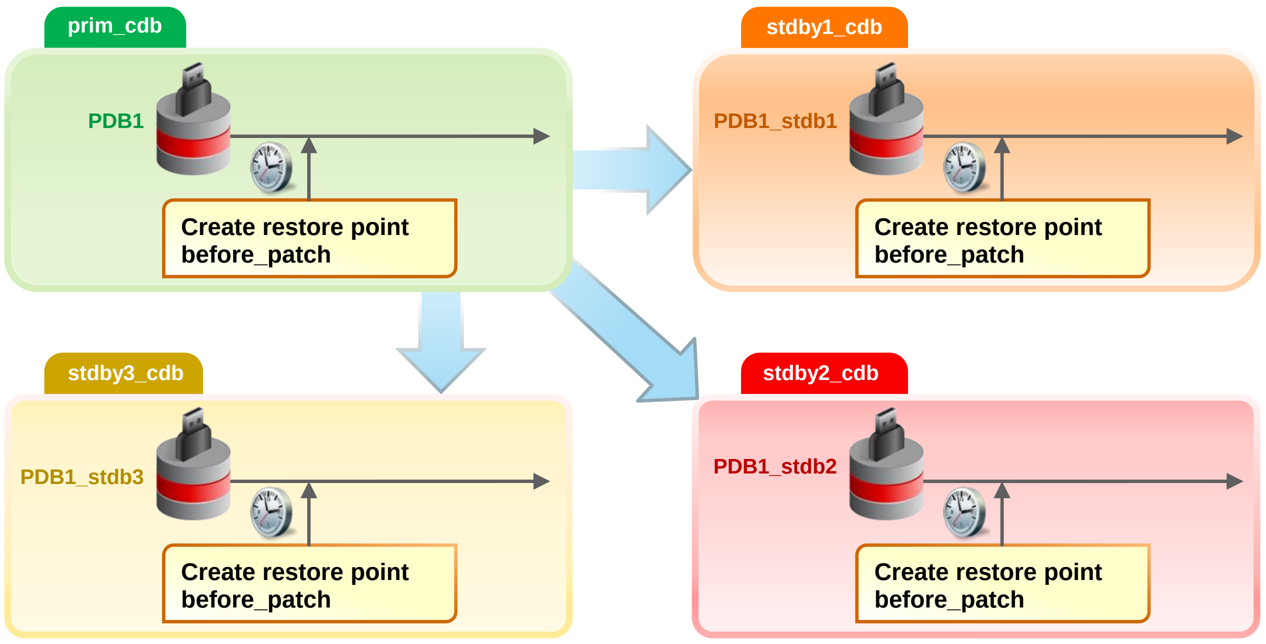
Caution: PDB-guaranteed restore points can potentially result in Oracle running out of space in the FRA. The DBA should remove PDB-guaranteed restore points when the PDB flashback operations are completed.

Be aware that flashback mode requires archivelog mode to be enabled.



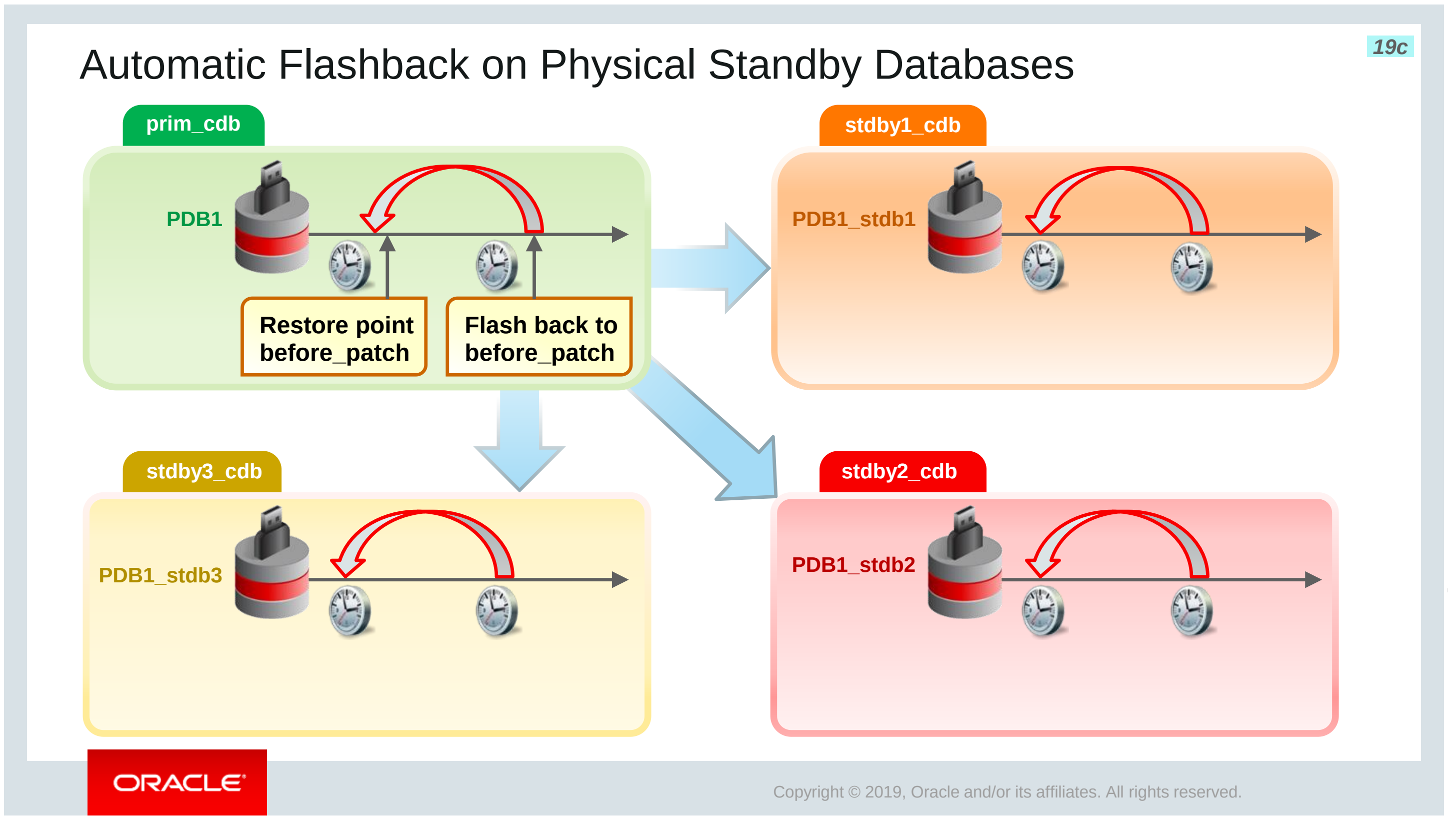
Automatic Propagation of Restore Points to All Standby Databases

19c


prim_cdb
PDB1
Create restore point before_patch
stdby1_cdb
PDB1_stdb1
Create restore point before_patch
stdby3_cdb
PDB1_stdb3
Create restore point before_patch
stdby2_cdb
PDB1_stdb2
Create restore point before_patch

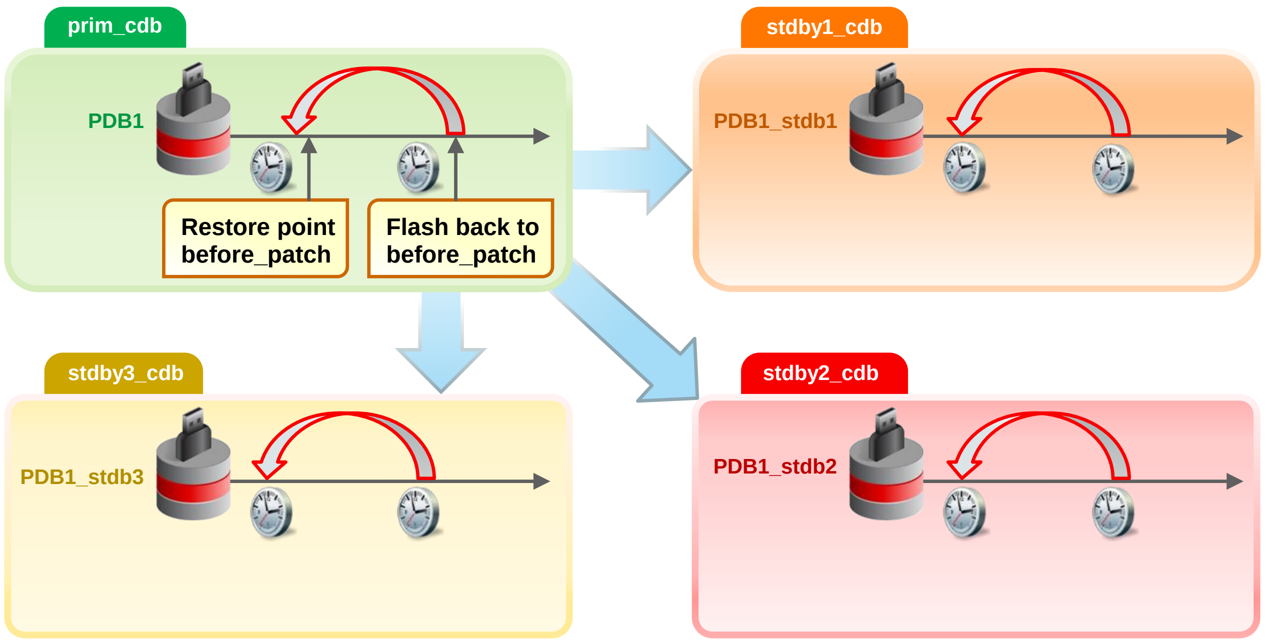
Prior to Oracle Database 19c, after a user creates a restore point on the primary database, the physical standby databases do not receive any information regarding the created restore point on the primary database. In case the user wants the restore point on the physical standby databases to have the same properties as those created on the primary database, the user has to do it manually.

Oracle Database 19c automatically propagates the restore point creation or deletion on the primary database to all the physical standby databases, thereby providing complete management of restore point propagation.



Automatic Flashback on Physical Standby Databases

19c



Prior to Oracle Database 19c, when a user issues a flashback on a primary database and opens the primary database in `RESETLOGS`, the Managed Recovery Process (MRP) on the physical standby just stops. This means that the standby database is no longer following the primary database.

The DBA is thus responsible for manually issuing a flashback or restoring the data files prior to the `RESETLOGS` and restarting the MRP on the standby database. This is also true whenever the DBA performs a PDB point-in-time recovery (PITR) or PDB flashback and then a PDB `RESETLOGS` on the primary CDB. The corresponding PDB on the standby CDB is no longer following the PDB on the primary CDB until the user manually flashes back the PDB on the standby and restarts the recovery.

Oracle Database 19c allows automatic flashback on the standby database or the PDB on the standby CDB, without any DBA intervention whenever the primary database performs database `RESETLOGS` or PDB `RESETLOGS`, provided the standby database or CDB is in mount mode and not open, and standby recovery is active.

The DBA can avoid the automatic flashback of the standby database or PDB by either keeping the standby in open mode or by stopping the MRP process in case the standby is closed.

Summary

In this lesson, you should have learned how to:

- Connect to the recovery catalog and to the PDB target database
- Use the Zstandard compression method as a real-time compression algorithm providing high compression ratios
- Reclaim flashback logs proactively
- Propagate restore points from a primary to standby site
- Automatically flash back a physical standby database when a primary database is flashed back

Practices Environment - 1

Each student has one virtual machine (VM1) to use during class.

- Oracle Database 19c installed
- One CDB named **`ORCL`** and its PDB named **`PDB1`**

Practices Environment - 2

Pre-created databases and instances with their respective PDBs:

- Data files in `/u02/app/oracle/oradata/`
- All backup files in `/u03/app/oracle/fast_recovery_area/ORCL`
- Password and init files in `$ORACLE_HOME/dbs`
- Diagnostics files in `/u01/app/oracle/diag/rdbms/orcl`
- TDE wallet in `/u01/app/oracle/admin/orcl/tde_wallet`
- Net files in `$ORACLE_HOME/network/admin`

The configuration of the environment used for the practices of the course matches the configuration used on the Oracle Cloud Virtual Machine for the pre-created `ORCL` database of Oracle Database Cloud Service. Oracle Database Cloud Service gives you the ability to deploy Oracle databases in the cloud, with each database deployment containing a single Oracle CDB.

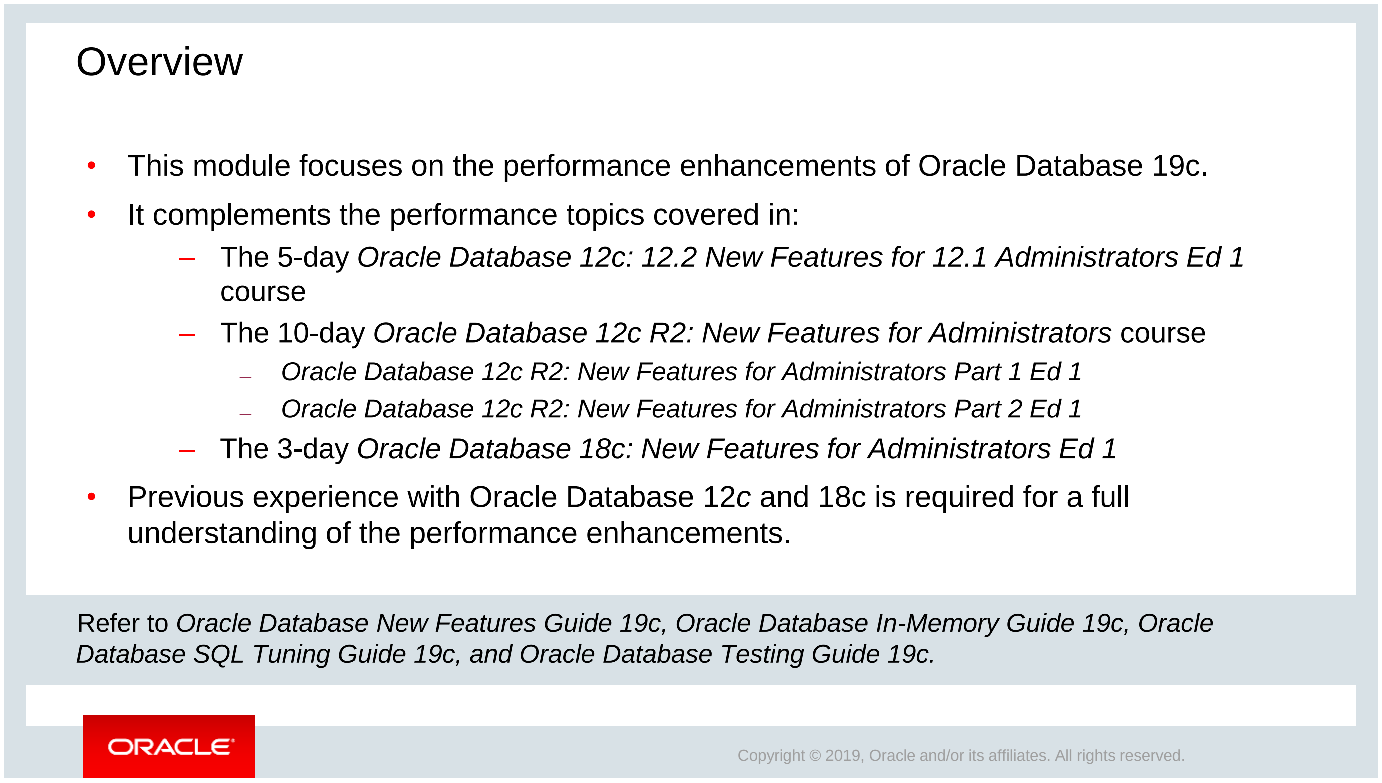
This is a good way to get familiar with the Oracle Database Cloud environment that is preconfigured for cloud customers.

Practice 3: Overview

- 3-1: Using RMAN to Connect to a PDB to Use the Recovery Catalog
- 3-2: Exploring Automatic Deletion of Flashback Logs

4

Using Performance Enhancements



Overview

- This module focuses on the performance enhancements of Oracle Database 19c.
- It complements the performance topics covered in:
  - The 5-day *Oracle Database 12c: 12.2 New Features for 12.1 Administrators Ed 1* course
  - The 10-day *Oracle Database 12c R2: New Features for Administrators* course
    - *Oracle Database 12c R2: New Features for Administrators Part 1 Ed 1*
    - *Oracle Database 12c R2: New Features for Administrators Part 2 Ed 1*
  - The 3-day *Oracle Database 18c: New Features for Administrators Ed 1*
- Previous experience with Oracle Database 12*c* and 18c is required for a full understanding of the performance enhancements.

Refer to *Oracle Database New Features Guide 19c, Oracle Database In-Memory Guide 19c, Oracle Database SQL Tuning Guide 19c, and Oracle Database Testing Guide 19c.*

This module follows either the 5-day *Oracle Database 12c: 12.2 New Features for 12.1 Administrators Ed 1* course or the 10-day *Oracle Database 12c Release 2 (12.2)* course or the 3-day *Oracle Database 18c: New Features for Administrators Ed 1*. These courses are designed to introduce the enhancements of Oracle Database 12*c* Release 2 (12.2.0.1) and Oracle Database 18c that are applicable to work that is usually performed by database administrators and related personnel.

The module is designed to introduce the major enhancements of Oracle Database 19c related to performance.

The module consists of an instructor-led lesson and practices that enable you to see how certain new functionalities behave.

Objectives

After completing this lesson, you should be able to:

- Configure and use deferred writes with Memoptimized Rowstore - Fast Ingest
- Configure PDB-level ADDM analysis
- Enable SQL developers to use Real-Time SQL Monitor
- Use automatic SQL plan management with the AUTO option
- Wait until all in-memory objects with a nondefault priority are populated
- Configure and use DB Replay at the PDB level
- Use a view to effectively utilize the PL/SQL extended mode of DB Replay
- Reduce the scope and overhead of supplemental logging

Memoptimized Rowstore – Fast Lookup

18c

Fast ingest and query rates for thousands of devices from the Internet requires:

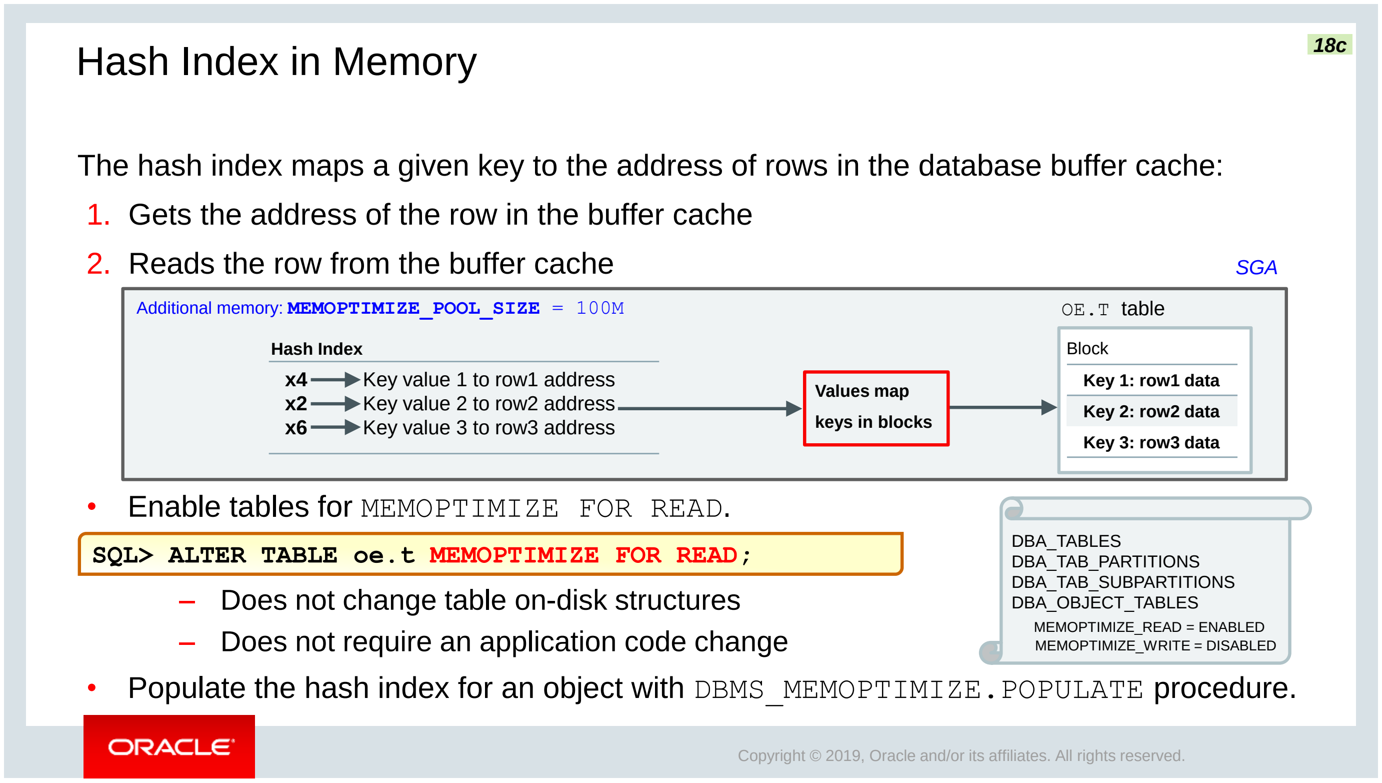
- High-speed streaming of single-row inserts
- Very fast lookups of key-value type data in the database buffer cache
  - Querying data with the `PRIMARY KEY` integrity constraint enabled
  - Using a new in-memory hash index structure
  - Accessing table rows permanently pinned in the buffer cache
- Aggregated and streamed data to the database through trusted clients

Smart devices connected to the Internet that have the ability to send and receive data require support for fast ingest and query rates for thousands of devices. The Memoptimized Rowstore feature is meant to provide high-speed streaming of single-row inserts and very fast lookups of key–value type data. The feature works only on tables that have the `PRIMARY KEY` integrity constraint enabled.

To provide the speed necessary to service thousands of devices, the data is aggregated and streamed to the database through trusted clients.

The fast query part of the Memoptimized Rowstore feature allows access to existing rows through a new hash index structure and pinned database blocks.

Oracle Database supports the ingest and access of row-based data in a fraction of the time that it takes for conventional SQL transactions. With the ability to ingest high-speed streaming of input data, and the use of innovative protocols and hash indexing of key–value pairs for lookups, the Memoptimized Rowstore feature significantly reduces transaction latency and overhead, and enables businesses to deploy thousands of devices to monitor and control all aspects of their business.



Hash Index in Memory

18c

The hash index maps a given key to the address of rows in the database buffer cache:

1. Gets the address of the row in the buffer cache
2. Reads the row from the buffer cache

SGA
Additional memory: MEMOPTIMIZE_POOL_SIZE = 100M
OE.T table
Hash Index
x4 Key value 1 to row1 address
x2 Key value 2 to row2 address
x6 Key value 3 to row3 address
Values map keys in blocks
Block
Key 1: row1 data
Key 2: row2 data
Key 3: row3 data

- Enable tables for `MEMOPTIMIZE FOR READ`.

```
SQL> ALTER TABLE oe.t MEMOPTIMIZE FOR READ;
```

  - Does not change table on-disk structures
  - Does not require an application code change

DBA_TABLES
DBA_TAB_PARTITIONS
DBA_TAB_SUBPARTITIONS
DBA_OBJECT_TABLES
MEMOPTIMIZE_READ = ENABLED
MEMOPTIMIZE_WRITE = DISABLED

- Populate the hash index for an object with `DBMS_MEMOPTIMIZE.POPULATE` procedure.

A hash table in-memory mapping a given key to the location of corresponding rows enables quick access of the Oracle data block storing the row.

The hash table in memory is indexed with a user-specified primary key, very similar to hash clusters containing tables with the `PRIMARY KEY` constraint enabled. This memory structure is called a hash index, although the underlying data structure is a hash table. The data structure resides in the instance memory, requiring additional space in SGA. You can set the `MEMOPTIMIZE_POOL_SIZE` initialization parameter to reserve static SGA allocation at instance startup.

To build a fast code path, having a hash index data structure in memory is not sufficient. Rows of tables are stored in disk blocks and, when row data is queried, the database buffer cache caches tables blocks in the SGA of the database instance. Given that the blocks are aged out based on the replacement policy used by the buffer cache, the blocks have to be permanently pinned in the buffer cache to avoid disk I/O. This is the reason for setting the `MEMOPTIMIZE FOR READ` attribute to a table that does not change the on-disk structure.

Using the DDL command `ALTER TABLE t MEMOPTIMIZE FOR READ` cascades the attribute to all existing partitions (and subpartitions). Use the `NO MEMOPTIMIZE FOR READ` clause to disable the feature on an object. By default, tables are `MEMOPTIMIZE FOR READ` disabled.

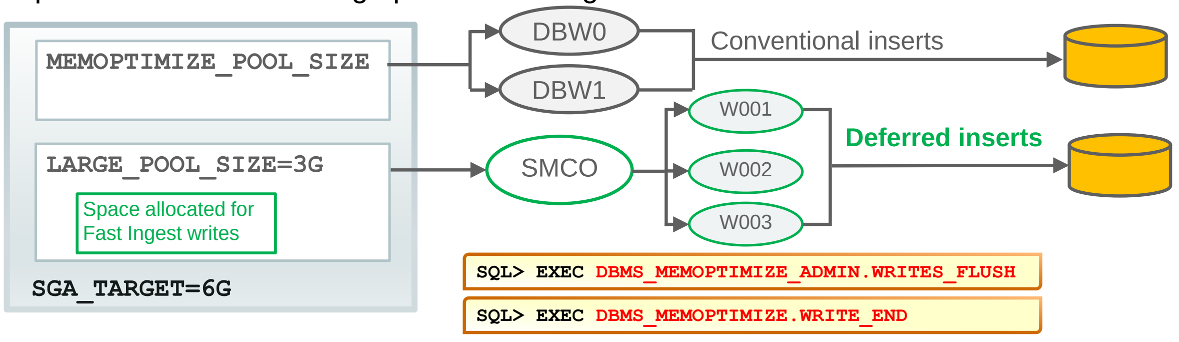
Setting the `MEMOPTIMIZE FOR WRITE` attribute to a table gives a primary key, and inserts the key, corresponding data, and metadata in the hash index structure during row inserts. The hash index structure is updated during other write operations such as delete and update.

Use the `DBMS_MEMOPTIMIZE.POPULATE` procedure to populate the hash index for an object, table, partition, or subpartition.

```
SQL> exec DBMS_MEMOPTIMIZE.POPULATE (SCHEMA_NAME => 'SH',
          table_name => 'SALES', PARTITION_NAME => 'SALES_Q3_2003')
```

Memoptimized Rowstore - Fast Ingest

- 18c The RDBMS insert infrastructure may not be able to process single-row inserts at the ingest speeds required.
- 19c Memoptimized Rowstore - Fast Ingest supports small, high volume transactions of data from IoT type devices with a minimal amount of transactional overhead.
  - **Tradeoff:** Data can be lost and, therefore, inserts might have to be retried.
- Space allocated in the large pool for fast ingest writes:



```
SQL> EXEC DBMS_MEMOPTIMIZE_ADMIN.WRITES_FLUSH
SQL> EXEC DBMS_MEMOPTIMIZE.WRITE_END
```

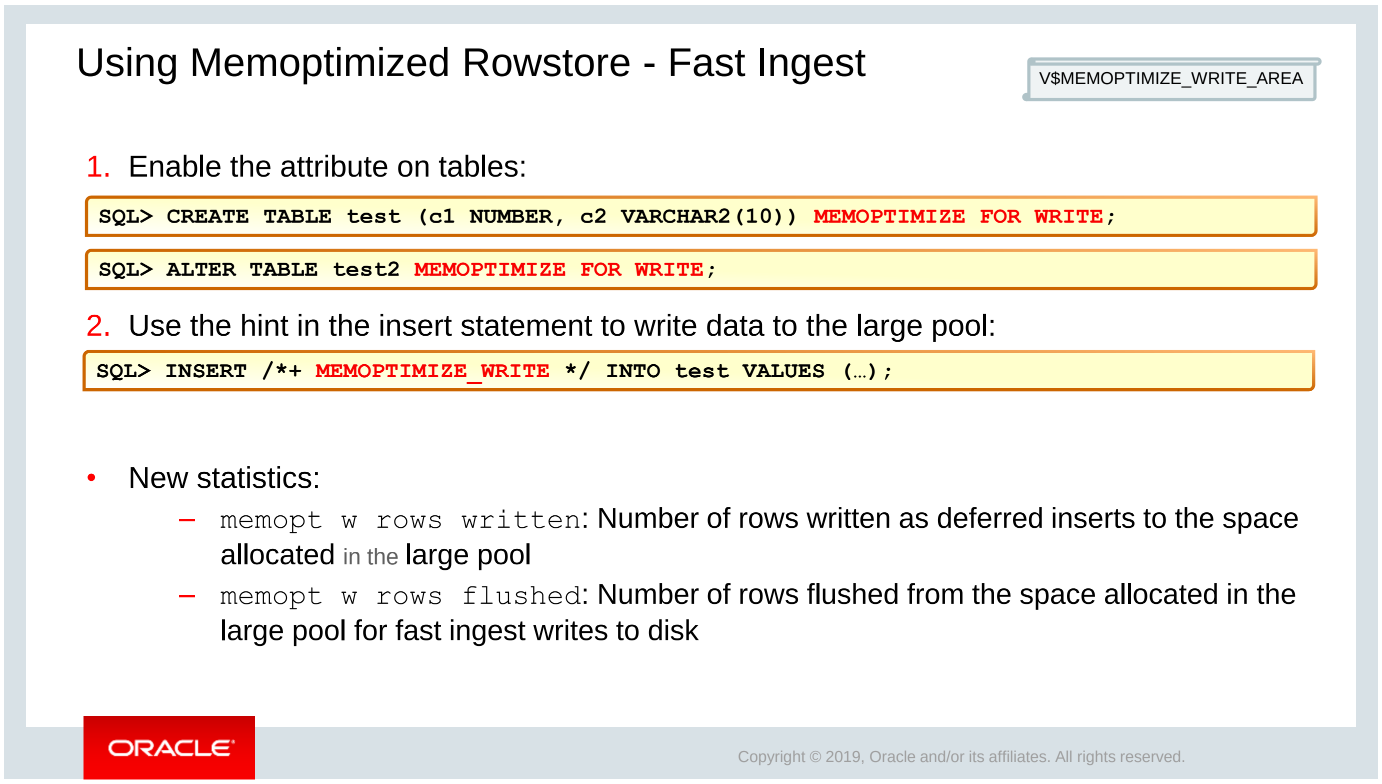
Until Oracle Database 19c, the RDBMS insert infrastructure does not handle the speeds at which single-row inserts must be ingested.

The motivation for the Memoptimized Rowstore - Fast Ingest in Oracle Database 19c has arisen from the explosion of IoT data, such as smart meter readings or data transmitted from any number of different sensors. Such data is typically small in size and transmitted frequently. The Memoptimized Rowstore - Fast Ingest feature allows streaming single-row inserts from a set of very specific concurrent clients.

The inserted row is persisted to disk in a deferred, asynchronous manner, different from the conventional inserts, through the Space Management Coordinator (SMCO) and W*xxx* slave background processes after 1MB worth of writes per session per object or after 60 seconds. You can flush all deferred writes made by the session with the `DBMS_MEMOPTIMIZE.WRITE_END` procedure. Flushing all data from the space allocated in the large pool for deferred writes to disk can also be forced by using the `DBMS_MEMOPTIMIZE_ADMIN.WRITES_FLUSH` procedure at the instance level.

- Data residing in the space allocated in the large pool for deferred writes, although assumed committed, can be lost in the event of crashes and instance failure.
- Any buffered data in the space allocated in the large pool for deferred writes cannot be read by any session, including the writer, until the background process sweep. Reading data directly from the space allocated in the large pool for deferred writes is not supported.
- Committed changes are not visible to any reader until the background process sweep.
- Clients that must avoid data loss should keep a local copy of data after writing to the space allocated in the large pool for deferred writes. After the payload has been committed to disk, the client can destroy its local copy of the data. In the event of data loss or instance restart, the client is able to replay the writes to the space allocated in the large pool for deferred writes.

Constraints are supported just as for regular inserts. Any constraints that can be evaluated without looking at the existing data on disk are still honored in the foreground process, such as `CHECK` and `NOT NULL` constraints. Any constraint that needs to be evaluated when the insert is written to disk is deferred to time of drainage to disk, such as `UNIQUE` and `PRIMARY KEY` constraints. As such, deferred inserts may appear as successful at the time of write to the space allocated in the large pool for deferred writes, and may error when written to disk from the space allocated in the large pool for deferred writes. Similarly, all index maintenance is evaluated at the time of draining rows from the space allocated in the large pool for deferred writes to disk. For best performance, it is recommended to disable constraints. In the absence of an error logging, erroneous rows are written to a trace file.



Using Memoptimized Rowstore - Fast Ingest

V$MEMOPTIMIZE_WRITE_AREA

1. Enable the attribute on tables:

```
SQL> CREATE TABLE test (c1 NUMBER, c2 VARCHAR2(10)) MEMOPTIMIZE FOR WRITE;
```

```
SQL> ALTER TABLE test2 MEMOPTIMIZE FOR WRITE;
```

2. Use the hint in the insert statement to write data to the large pool:

```
SQL> INSERT /*+ MEMOPTIMIZE_WRITE */ INTO test VALUES (…);
```

- New statistics:
  - `memopt w rows written`: Number of rows written as deferred inserts to the space allocated in the large pool
  - `memopt w rows flushed`: Number of rows flushed from the space allocated in the large pool for fast ingest writes to disk

To have rows inserts work as deferred inserts, the following conditions must be satisfied:

1. A table must be enabled for `MEMOPTIMIZE FOR WRITE` before data for that table can be written to the space allocated in large pool for fast ingest writes. This is solely a top-level attribute, and cannot be specified at the (sub)partition level.
2. To enable an insert to write data to the space allocated in large pool for fast ingest writes, the insert statement must use the `MEMOPTIMIZE_WRITE` hint.

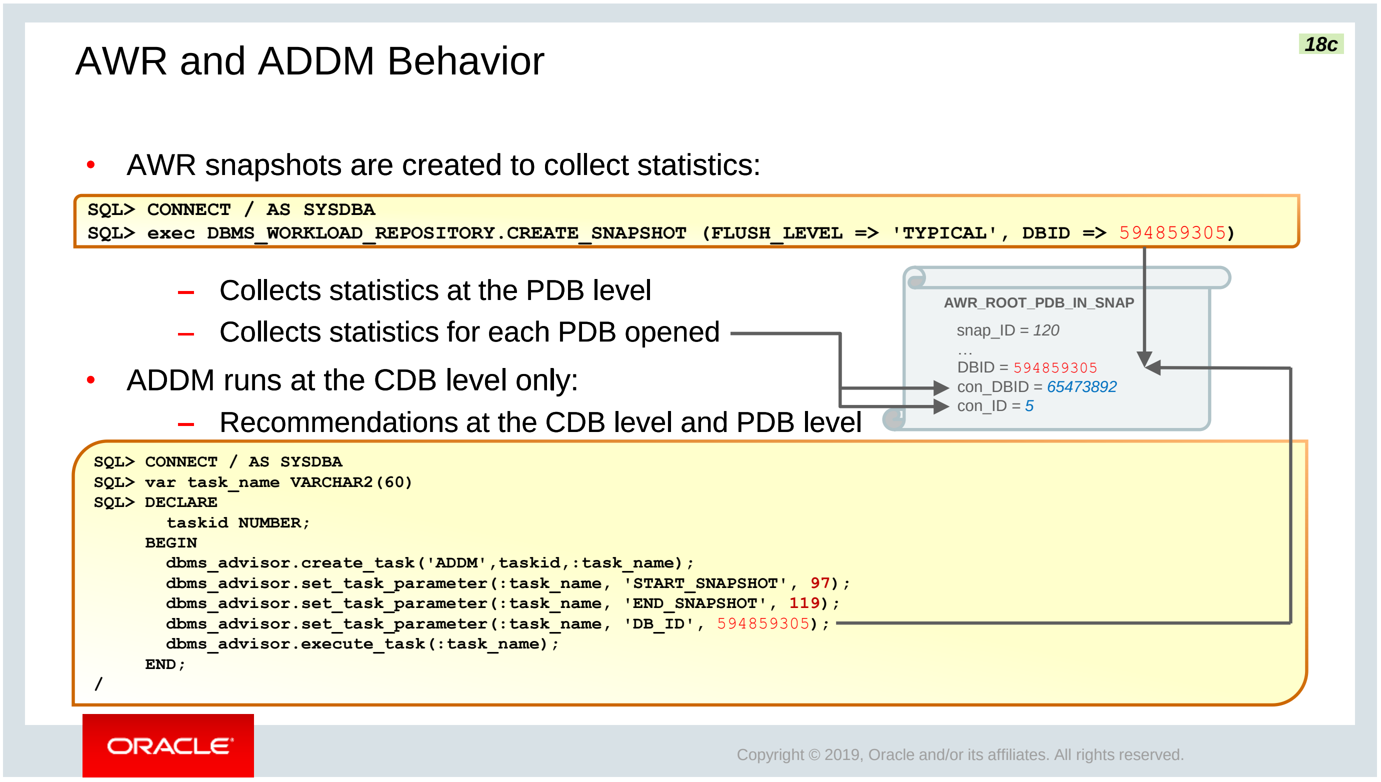
Statistics added all begin with the prefix “`memopt w`”.

- "`memopt w rows written`”: Number of rows written via Memoptimized Rowstore - Fast Ingest as deferred inserts
- "`memopt w rows flushed`”: Number of rows flushed

The `V$MEMOPTIMIZE_WRITE_AREA` exposes the state of the space allocated in the large pool for fast ingest writes. This view has the following fields:

- `TOTAL_SIZE`: Total size of the space allocated in the large pool for fast ingest writes
- `USED_SPACE`: Total space used (nondrained writes) in the space allocated in the large pool for fast ingest writes
- `FREE_SPACE`: Total space free for writes in the space allocated in the large pool for fast ingest writes
- `NUM_WRITES`: Number of current nondrained writes in the space allocated in the large pool for fast ingest writes
- `NUM_WRITERS`: Number of current writers to the space allocated in the large pool for fast ingest writes

You can view the free video about this feature in the course under the module name “On - Premises Videos”.



AWR and ADDM Behavior

18c

- AWR snapshots are created to collect statistics:

```
SQL> CONNECT / AS SYSDBA
SQL> exec DBMS_WORKLOAD_REPOSITORY.CREATE_SNAPSHOT (FLUSH_LEVEL => 'TYPICAL', DBID => 594859305)
```

  – Collects statistics at the PDB level
  – Collects statistics for each PDB opened
- ADDM runs at the CDB level only:
  – Recommendations at the CDB level and PDB level

```
SQL> CONNECT / AS SYSDBA
SQL> var task_name VARCHAR2(60)
SQL> DECLARE
       taskid NUMBER;
     BEGIN
       dbms_advisor.create_task('ADDM',taskid,:task_name);
       dbms_advisor.set_task_parameter(:task_name, 'START_SNAPSHOT', 97);
       dbms_advisor.set_task_parameter(:task_name, 'END_SNAPSHOT', 119);
       dbms_advisor.set_task_parameter(:task_name, 'DB_ID', 594859305);
       dbms_advisor.execute_task(:task_name);
     END;
/
```

Since Oracle Database 12*c*, PDB-level snapshots—a collection of statistics that matter at the PDB level— are created when the PDB is opened. Even if the snapshot is created while being connected to the CDB root, the snapshot contains statistics that are collected for each opened PDB. A PDB-level report contains information that is specific only to a PDB.

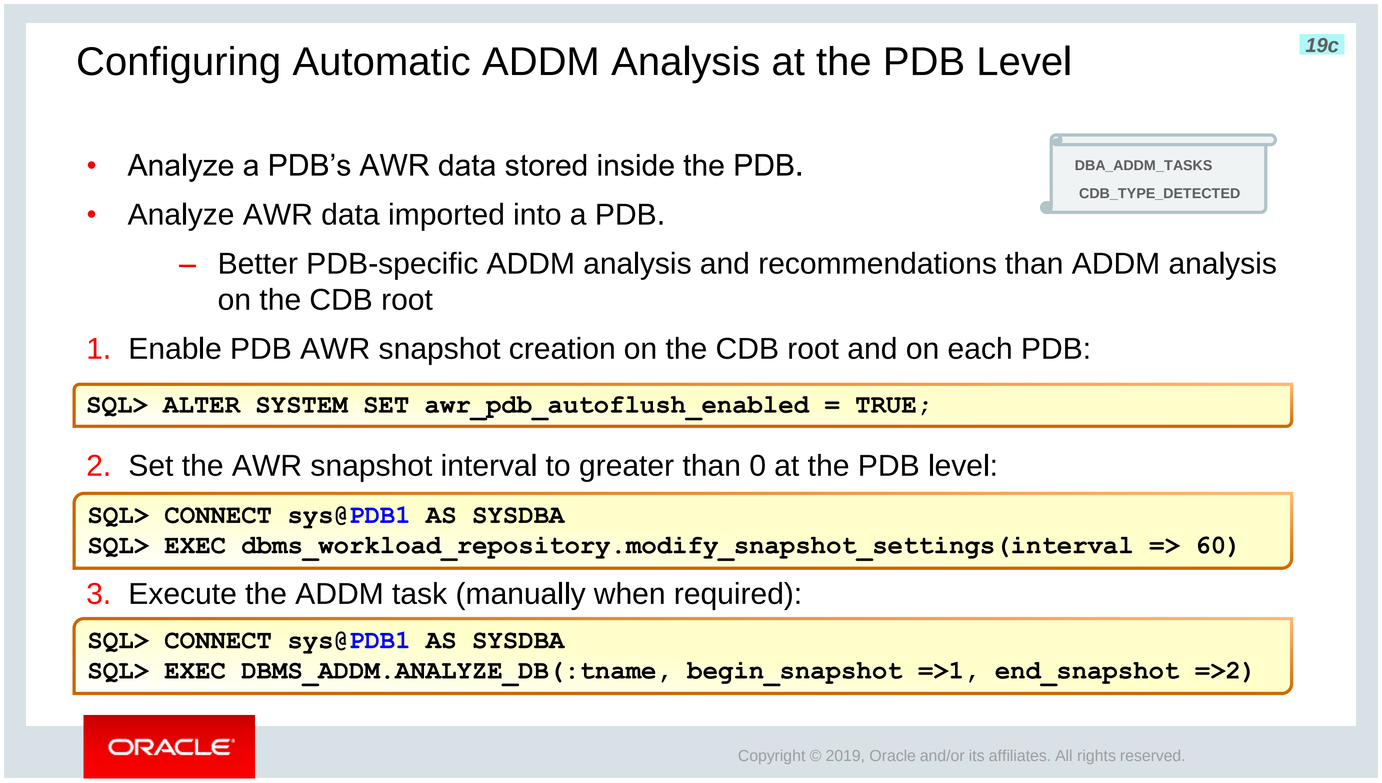
The `AWR_ROOT_PDB_IN_SNAP` view captures the list of opened PDBs at the time of Automatic Workload Repository (AWR) snapshot creation. The `CON_ID` column displays the ID of the PDB to which the data pertains. The same information is kept in the `DBA_HIST_PDB_IN_SNAP` view.

```
SQL> select * from AWR_ROOT_PDB_IN_SNAP;

   SNAP_ID       DBID INSTANCE_NUMBER   CON_DBID       FLAG CON_ID

---------- ---------- --------------- ---------- ---------- ------

       120   594859305                1   16956870          0         4

       120   594859305                1   65473892          0         5

       121   594859305                1  216956870          0         4

       121   594859305                1   65473892          0         5
```

You cannot request an ADDM task at the PDB level. All ADDM runs must be performed in the CDB root and all ADDM results are stored in the CDB root.

However, ADDM tasks provide recommendations on statements executed in any container—in the CDB root, in application roots, in regular PDBs, and in application PDBs. However, ADDM results cannot be viewed when the current container is a PDB.

ADDM analyzes activity in a PDB within the context of the current analysis target, but ADDM does not analyze only one PDB at a time.



Configuring Automatic ADDM Analysis at the PDB Level

19c

DBA_ADDM_TASKS
CDB_TYPE_DETECTED

- Analyze a PDB's AWR data stored inside the PDB.
- Analyze AWR data imported into a PDB.
  - Better PDB-specific ADDM analysis and recommendations than ADDM analysis on the CDB root

1. Enable PDB AWR snapshot creation on the CDB root and on each PDB:

```
SQL> ALTER SYSTEM SET awr_pdb_autoflush_enabled = TRUE;
```

2. Set the AWR snapshot interval to greater than 0 at the PDB level:

```
SQL> CONNECT sys@PDB1 AS SYSDBA
SQL> EXEC dbms_workload_repository.modify_snapshot_settings(interval => 60)
```

3. Execute the ADDM task (manually when required):

```
SQL> CONNECT sys@PDB1 AS SYSDBA
SQL> EXEC DBMS_ADDM.ANALYZE_DB(:tname, begin_snapshot =>1, end_snapshot =>2)
```

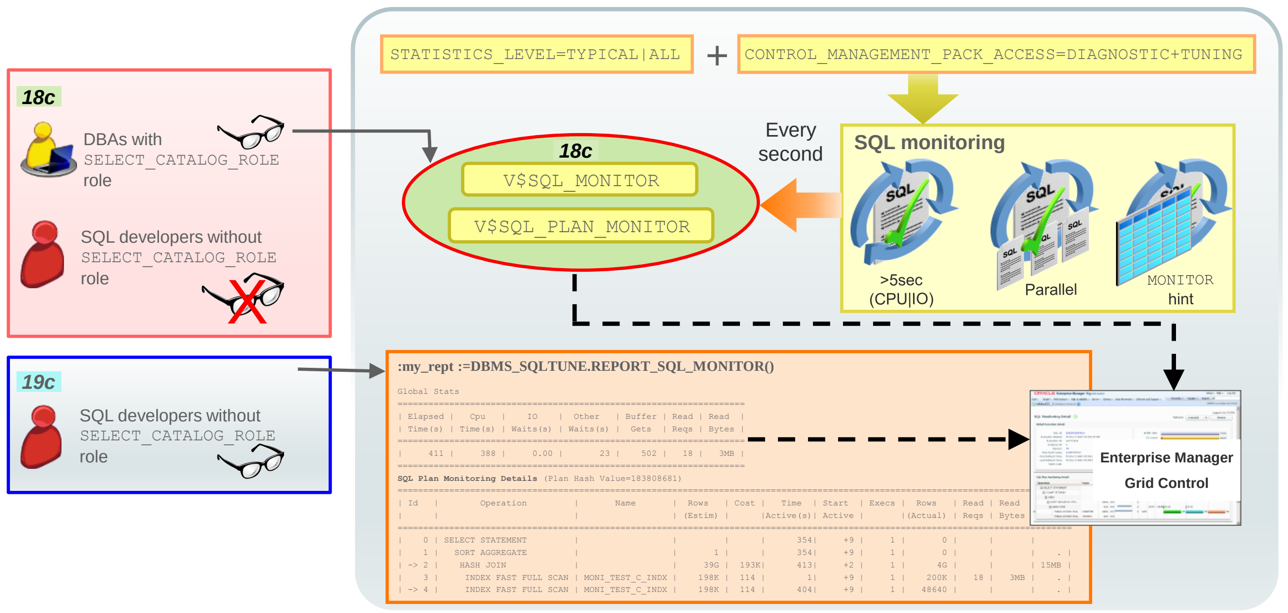
In Oracle Database 19c, ADDM can run automatically on a PDB after the AWR snapshots are enabled in the PDB. AWR snapshots are not enabled by default on a PDB.

When AWR snapshots are enabled in a PDB and an ADDM analysis runs in the PDB, some parts of the ADDM report and recommendations differ from the result of an ADDM execution in the CDB root. A PDB analysis means that the ADDM task is analyzing the AWR data of a PDB, and only PDB-specific findings and recommendations are produced. Enabling AWR snapshots on a PDB does not change the ADDM report on a CDB root.

AWR snapshots on a PDB are stored in the PDB, whichever tablespace is defined. No one can view PDB snapshots from the CDB root for security purposes.

The new `CDB_TYPE_DETECTED` column in the `DBA_ADDM_TASKS` view displays the type of level analyzed by ADDM: CDB or PDB.

SQL Developers and Users Enabled for SQL Monitor Usage


18c
DBAs with SELECT_CATALOG_ROLE role
SQL developers without SELECT_CATALOG_ROLE role
19c
SQL developers without SELECT_CATALOG_ROLE role
STATISTICS_LEVEL=TYPICAL|ALL
+
CONTROL_MANAGEMENT_PACK_ACCESS=DIAGNOSTIC+TUNING
18c
V$SQL_MONITOR
V$SQL_PLAN_MONITOR
Every second
SQL monitoring
>5sec (CPU|IO)
Parallel
MONITOR hint
Enterprise Manager Grid Control

```
:my_rept :=DBMS_SQLTUNE.REPORT_SQL_MONITOR()

Global Stats
====================================================================
| Elapsed |   Cpu   |    IO    |  Other   | Buffer | Read | Read  |
| Time(s) | Time(s) | Waits(s) | Waits(s) |  Gets  | Reqs | Bytes |
====================================================================
|     411 |     388 |     0.00 |       23 |    502 |   18 |   3MB |
====================================================================
SQL Plan Monitoring Details (Plan Hash Value=183808681)
=================================================================================================================================
| Id   |       Operation           |       Name        |  Rows   | Cost |   Time    | Start  | Execs |   Rows   | Read | Read
|      |                           |                   | (Estim) |      |Active(s)| Active |       | (Actual) | Reqs | Bytes
=================================================================================================================================
|    0 | SELECT STATEMENT          |                   |         |      |      354|     +9 |     1 |        0 |      |       |
|    1 |  SORT AGGREGATE           |                   |       1 |      |      354|     +9 |     1 |        0 |      |       |     . |
| -> 2 |   HASH JOIN               |                   |     39G | 193K|      413|     +2 |     1 |       4G |      |       |  15MB |
|    3 |    INDEX FAST FULL SCAN | MONI_TEST_C_INDX |    198K |  114 |        1|     +9 |     1 |     200K |   18 |   3MB |     . |
| -> 4 |    INDEX FAST FULL SCAN | MONI_TEST_C_INDX |    198K |  114 |      404|     +9 |     1 |    48640 |      |       |     . |
```

The Real-Time SQL Monitoring feature on Oracle Database 11*g* enables DBAs to access the views which are only accessible by users who have been granted the `SELECT_CATALOG_ROLE` role to monitor the performance of SQL statements while they are executing.

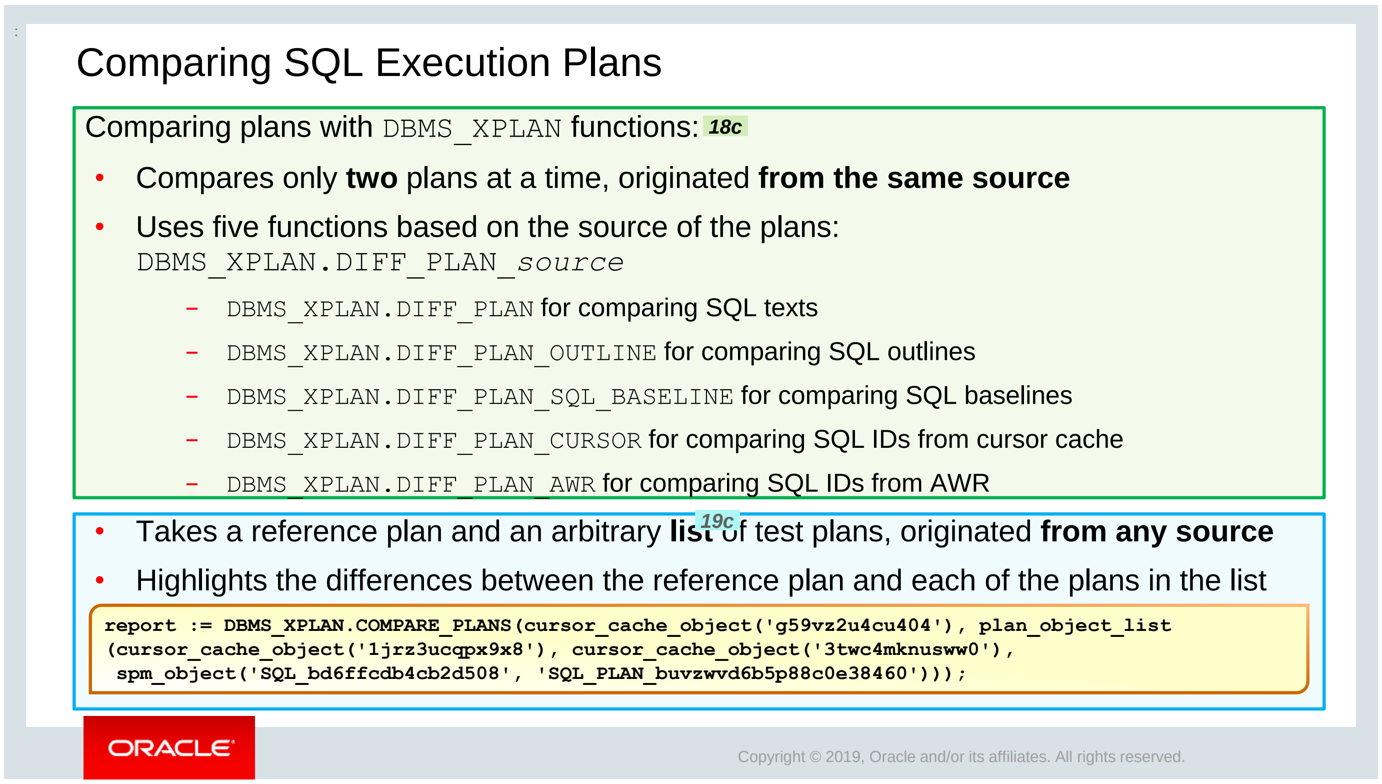
The Real-Time SQL Monitoring feature is enabled by default when the `STATISTICS_LEVEL` initialization parameter is either set to `ALL` or to `TYPICAL`, which is the default value. Additionally, the `CONTROL_MANAGEMENT_PACK_ACCESS` parameter must be set to `DIAGNOSTIC+TUNING` (the default value) because Real-Time SQL Monitoring is a feature of the Oracle Database Tuning Pack.

By default, Real-Time SQL Monitoring is started when a SQL command runs in parallel, or when it has consumed at least five seconds of the CPU or I/O time in a single execution.

As mentioned, Real-Time SQL Monitoring is active by default. However, two command-level hints are available to force or prevent a SQL command from being monitored. To force real-time SQL monitoring, use the `MONITOR` hint. To prevent the hinted SQL command from being monitored, use the `NO_MONITOR` hint.

A regular user, such as an application developer or, in general, a low-privileged user without the `SELECT CATALOG ROLE` and `SELECT` privilege on the SQL monitor fixed views, can write a SQL statement, execute it, and see the SQL result set and its SQL plan using the `EXPLAIN PLAN` command. However, he or she cannot see its execution plan because it is stored in `V$SQL_PLAN` and not its SQL monitor statistics because they are maintained in `V$` views.

Oracle Database 19c enables SQL developers to use Real-Time SQL Monitor to allow them to analyze the performance of their SQL statement without requiring super-user privileges or roles. Reports are available to application developers to view and analyze monitoring statistics for their SQL executions.



Comparing SQL Execution Plans

Comparing plans with `DBMS_XPLAN` functions: *18c*

- Compares only **two** plans at a time, originated **from the same source**
- Uses five functions based on the source of the plans: `DBMS_XPLAN.DIFF_PLAN_`*`source`*
  - `DBMS_XPLAN.DIFF_PLAN` for comparing SQL texts
  - `DBMS_XPLAN.DIFF_PLAN_OUTLINE` for comparing SQL outlines
  - `DBMS_XPLAN.DIFF_PLAN_SQL_BASELINE` for comparing SQL baselines
  - `DBMS_XPLAN.DIFF_PLAN_CURSOR` for comparing SQL IDs from cursor cache
  - `DBMS_XPLAN.DIFF_PLAN_AWR` for comparing SQL IDs from AWR

19c

- Takes a reference plan and an arbitrary list of test plans, originated **from any source**
- Highlights the differences between the reference plan and each of the plans in the list

```
report := DBMS_XPLAN.COMPARE_PLANS(cursor_cache_object('g59vz2u4cu404'), plan_object_list
(cursor_cache_object('1jrz3ucqpx9x8'), cursor_cache_object('3twc4mknusww0'),
 spm_object('SQL_bd6ffcdb4cb2d508', 'SQL_PLAN_buvzwvd6b5p88c0e38460')));
```

In Oracle Database 18c, SQL plan management (SPM) uses `DBMS_XPLAN` functions to compare plans:

- `DBMS_XPLAN.DIFF_PLAN`
- `DBMS_XPLAN.DIFF_PLAN_SQL_BASELINE`
- `DBMS_XPLAN.DIFF_PLAN_CURSOR`
- `DBMS_XPLAN.DIFF_PLAN_AWR`
- `DBMS_XPLAN.DIFF_PLAN_OUTLINE`

The functions provided have the following limitations:

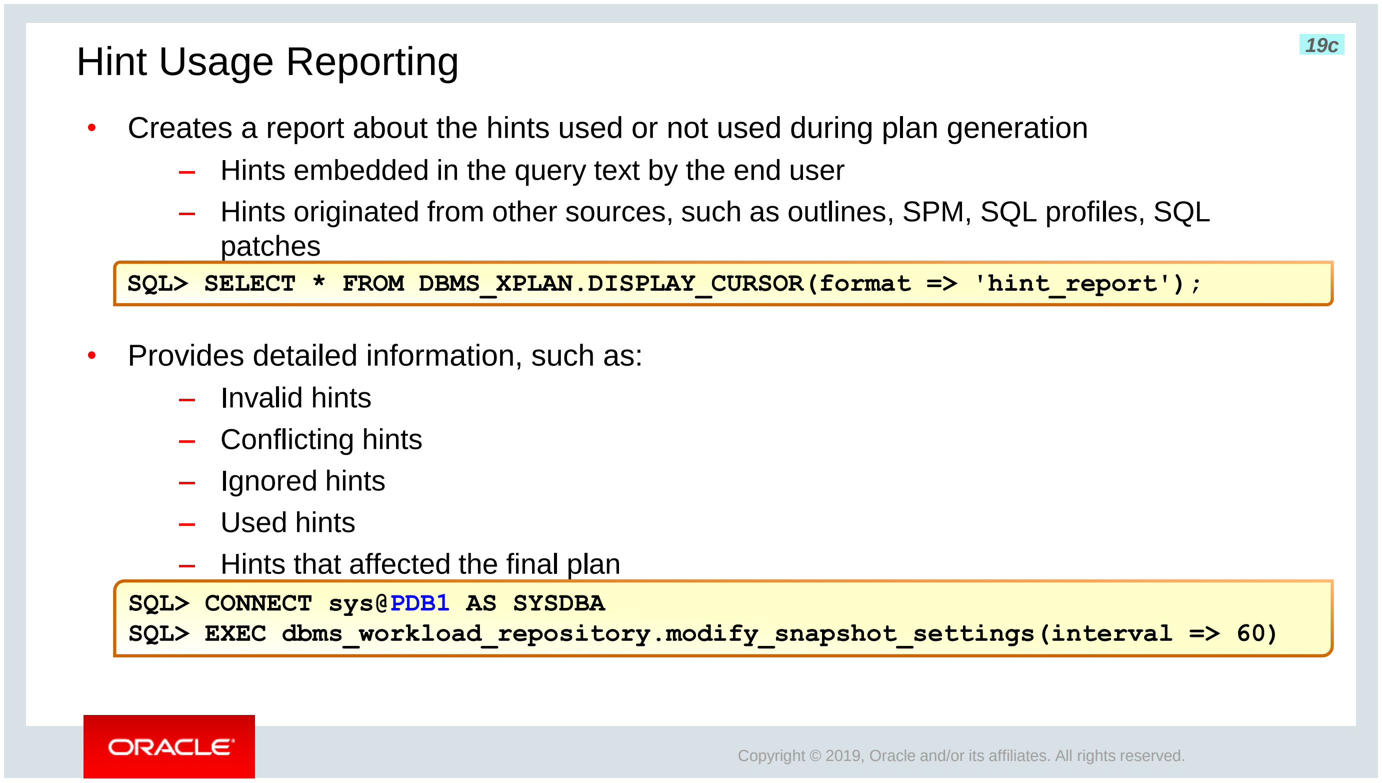
- It can compare only two plans at a time.
- Both the plans that are being compared must originate from the same source. For example, it can only compare two AWR plans or two SQL baseline plans. You must use one of the five different functions to compare plans based on different sources.

Oracle Database 19c introduces the plan comparison tool that provides a logical comparison of a reference plan with a set of test plans. The source of each plan provided as input to the plan comparison tool can be different. For example, you can compare two different plans stored in AWR, a plan in cursor cache with a plan stored in SPM plan baselines. The differences in the plans are highlighted in a plan comparison report.

Use Cases

- **New plans:** When a plan change occurs, users want to compare it with an old plan stored in AWR.
- **SQL Performance Analyzer (SPA), SQL Tune:** How does a plan generated based on a new SQL profile or plan generated through SPA differ from the original plan?

- **SPM:** When an accepted baseline plan is not reproduced by SPM, how does the original plan stored in the baseline differ from the generated plan? What are the differences between the different plan baselines captured for the same query?
- **Plan generated based on the effect of hints or parameters:** How does a plan change when a particular hint is added, or the value of a parameter is changed?



Hint Usage Reporting

19c

- Creates a report about the hints used or not used during plan generation
  - Hints embedded in the query text by the end user
  - Hints originated from other sources, such as outlines, SPM, SQL profiles, SQL patches

```
SQL> SELECT * FROM DBMS_XPLAN.DISPLAY_CURSOR(format => 'hint_report');
```

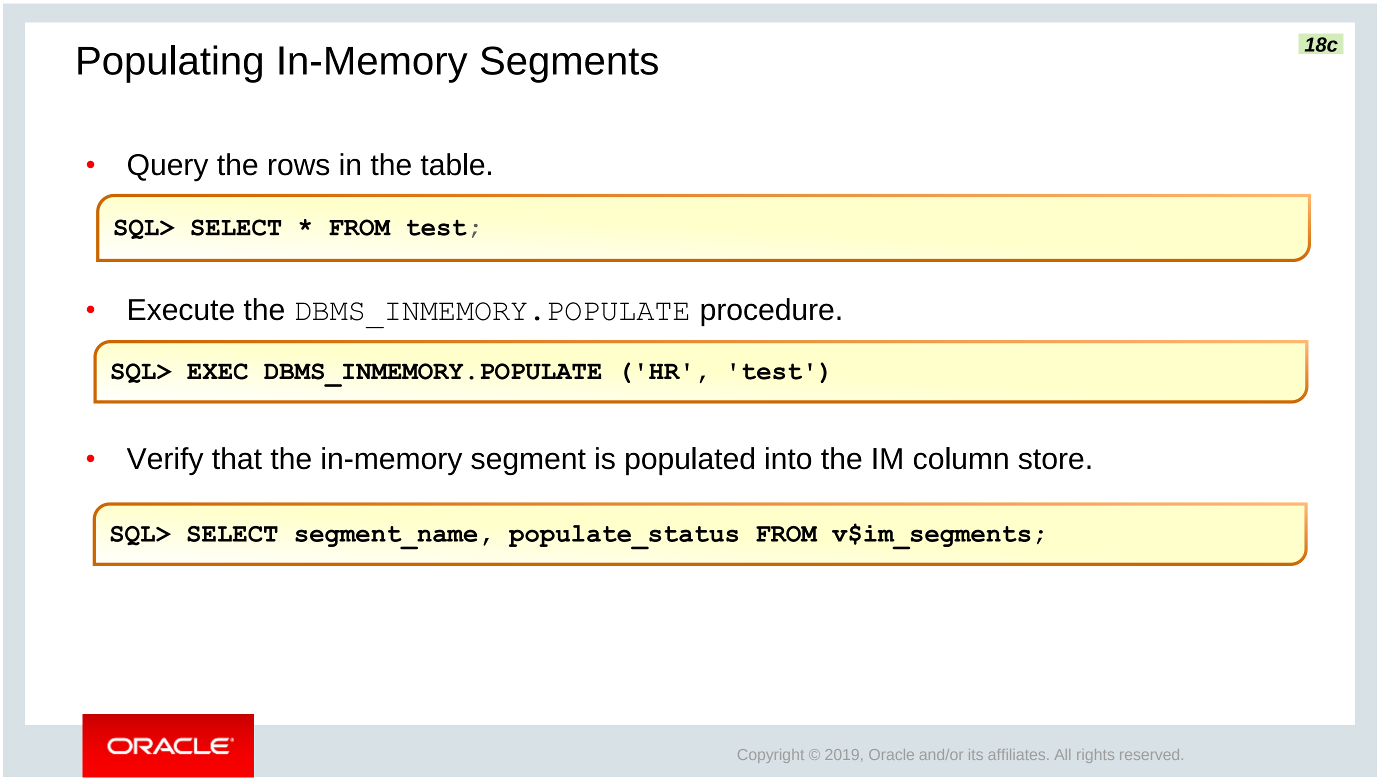
- Provides detailed information, such as:
  - Invalid hints
  - Conflicting hints
  - Ignored hints
  - Used hints
  - Hints that affected the final plan

```
SQL> CONNECT sys@PDB1 AS SYSDBA
SQL> EXEC dbms_workload_repository.modify_snapshot_settings(interval => 60)
```

In Oracle Database 19c, a hint usage reporting mechanism creates a report about the hints that were used or not used during plan generation. When a query that uses hints is processed, it is useful for analysis and diagnosis to determine which hints were obeyed by the optimizer and what were not. This report can help users and developers. The hints can be hints embedded in the query text by the user or originate from other sources such as outlines, SQL plan baselines, SQL profiles, and SQL patches. The hint report provides detailed information such as:

- Invalid hints
- Conflicting hints
- Ignored hints because of the query block location specified
- Used hints
- Hints that affected the final plan

The report provides valuable information to users as well as developers to identify issues with plans and the features that use plan information.



Populating In-Memory Segments

18c

- Query the rows in the table.

```
SQL> SELECT * FROM test;
```

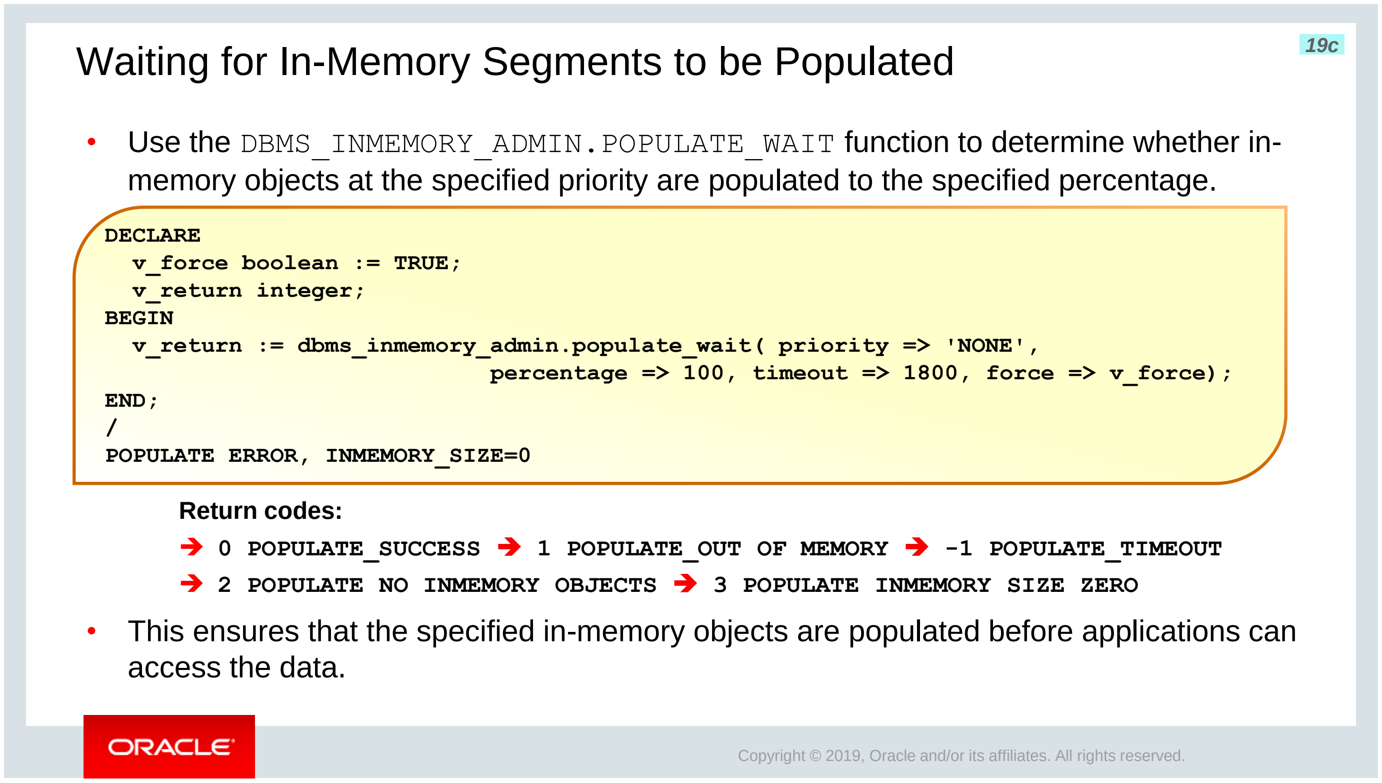
- Execute the `DBMS_INMEMORY.POPULATE` procedure.

```
SQL> EXEC DBMS_INMEMORY.POPULATE ('HR', 'test')
```

- Verify that the in-memory segment is populated into the IM column store.

```
SQL> SELECT segment_name, populate_status FROM v$im_segments;
```

To populate in-memory segments into the IM column store, either query the table data or execute the `DBMS_INMEMORY.POPULATE` procedure.



Waiting for In-Memory Segments to be Populated

19c

- Use the `DBMS_INMEMORY_ADMIN.POPULATE_WAIT` function to determine whether in-memory objects at the specified priority are populated to the specified percentage.

```
DECLARE
  v_force boolean := TRUE;
  v_return integer;
BEGIN
  v_return := dbms_inmemory_admin.populate_wait( priority => 'NONE',
                           percentage => 100, timeout => 1800, force => v_force);
END;
/
POPULATE ERROR, INMEMORY_SIZE=0
```

Return codes:

➔ `0 POPULATE_SUCCESS` ➔ `1 POPULATE_OUT OF MEMORY` ➔ `-1 POPULATE_TIMEOUT`

➔ `2 POPULATE NO INMEMORY OBJECTS` ➔ `3 POPULATE INMEMORY SIZE ZERO`

- This ensures that the specified in-memory objects are populated before applications can access the data.

ORACLE

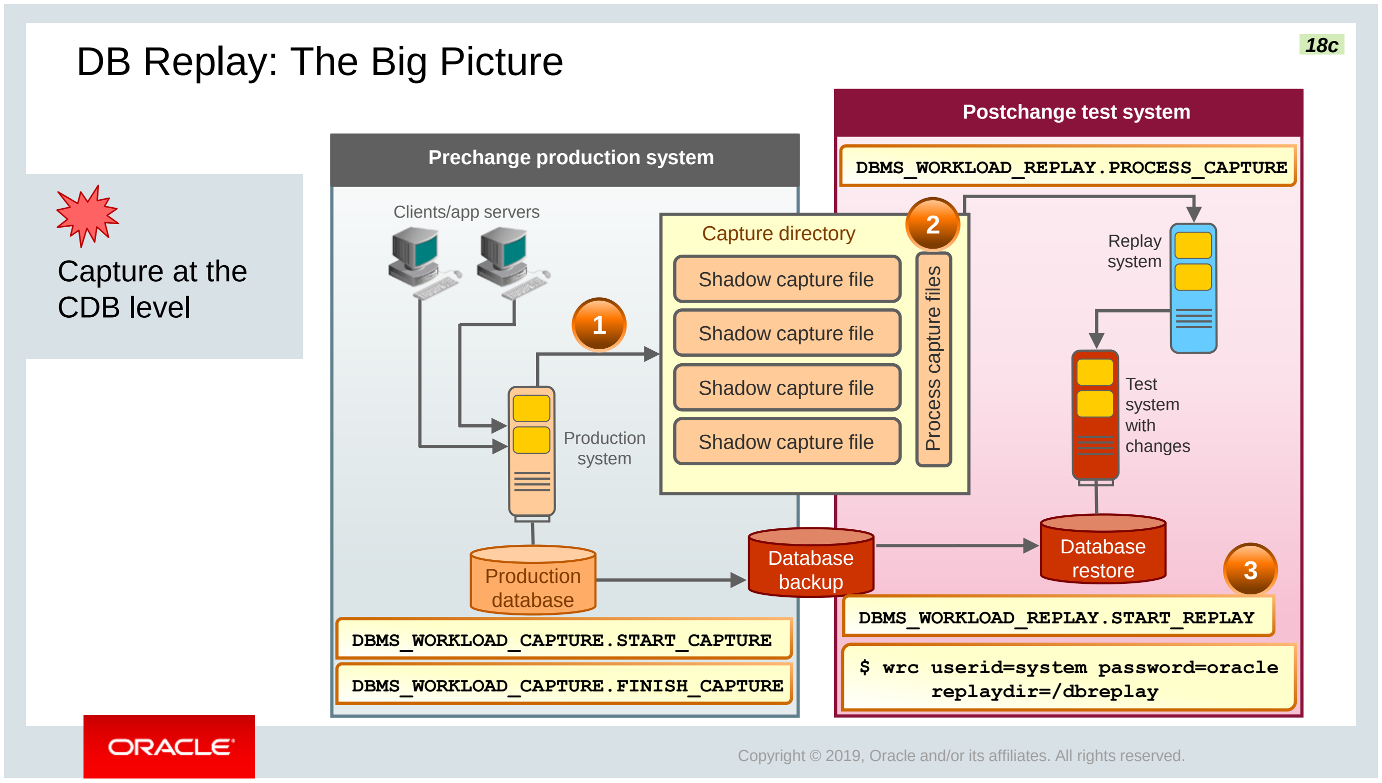
Oracle Database 19c introduces the `DBMS_INMEMORY_ADMIN.POPULATE_WAIT` function. Executing this function returns a value that indicates the status of the eligible object's population. You can write a wrapper package that invokes the function at startup. Based on the value returned, the package can start the application's services, send some messages to the application tier, or even open the database if it was opened in restricted mode.

The example in the slide populates all in-memory objects regardless of `PRIORITY` setting, specifying that the function will wait until all objects are 100% populated, and it will time out with an error if success is not achieved within 1800 seconds (30 minutes). The default value for the priority is `LOW`, for the percentage, `100`, for the timeout, `9999999`, and for the force, `FALSE`.

The message returned by the function means that the population cannot be successful because the IMCS size is set to 0.

The return code can be one of the five following values:

- `POPULATE_TIMEOUT := -1`
- `POPULATE_SUCCESS := 0`
- `POPULATE_OUT_OF_MEMORY := 1`
- `POPULATE_NO_INMEMORY_OBJECTS := 2`
- `POPULATE_INMEMORY_SIZE_ZERO := 3`



DB Replay: The Big Picture

18c

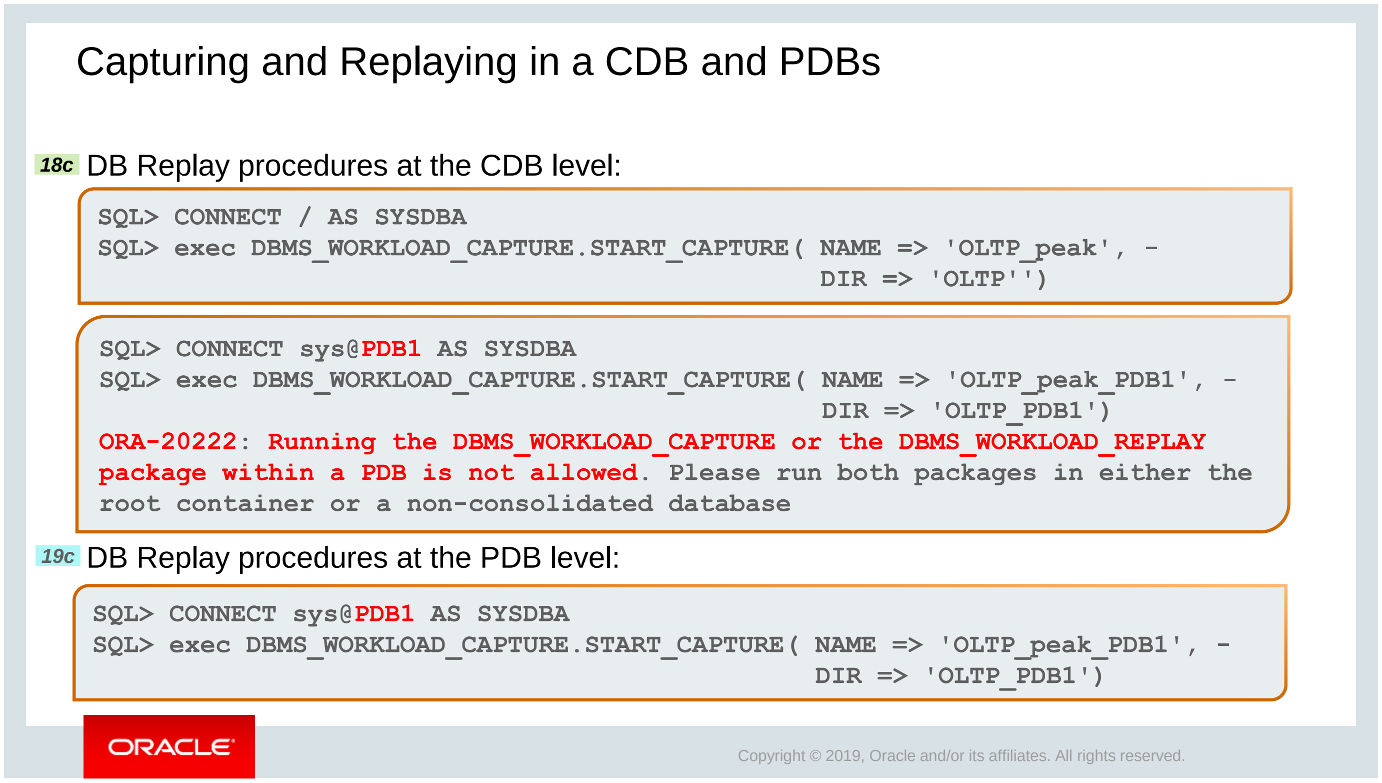
Capture at the CDB level

```
DBMS_WORKLOAD_REPLAY.PROCESS_CAPTURE
DBMS_WORKLOAD_CAPTURE.START_CAPTURE
DBMS_WORKLOAD_CAPTURE.FINISH_CAPTURE
DBMS_WORKLOAD_REPLAY.START_REPLAY
$ wrc userid=system password=oracle
      replaydir=/dbreplay
```

Since Oracle Database 11*g* managing system changes, a significant benefit is the added confidence to the business in the success of performing the change. The record-and-replay functionality offers an accurate method to test the impact of a variety of system changes, including database upgrade, configuration changes, and hardware upgrade.

A useful application of Database Replay is to test the performance of a new server configuration. Consider a customer who is utilizing a single instance database and wants to move to a RAC setup. The customer records the workload of an interesting period and then sets up a RAC test system for replay. During replay, the customer is able to monitor the performance benefit of the new configuration by comparing the performance to the recorded system. Using DB replay can convince the customer to move to a RAC
configuration after seeing the benefits.

Another application is debugging. You can record and replay sessions emulating an environment to make bugs more reproducible. Manageability feature testing is another benefit. Self-managing and self-healing systems need to implement this advice automatically (“autonomic computing model”). Multiple replay iterations allow testing and fine-tuning of the control strategies’ effectiveness and stability. The database administrator, or a user with special privileges granted by the DBA, initiates the record-and-replay cycle and has full control over the entire procedure.



Capturing and Replaying in a CDB and PDBs

18c DB Replay procedures at the CDB level:

```
SQL> CONNECT / AS SYSDBA
SQL> exec DBMS_WORKLOAD_CAPTURE.START_CAPTURE( NAME => 'OLTP_peak', -
                                               DIR => 'OLTP'')
```

```
SQL> CONNECT sys@PDB1 AS SYSDBA
SQL> exec DBMS_WORKLOAD_CAPTURE.START_CAPTURE( NAME => 'OLTP_peak_PDB1', -
                                               DIR => 'OLTP_PDB1')
ORA-20222: Running the DBMS_WORKLOAD_CAPTURE or the DBMS_WORKLOAD_REPLAY
package within a PDB is not allowed. Please run both packages in either the
root container or a non-consolidated database
```

19c DB Replay procedures at the PDB level:

```
SQL> CONNECT sys@PDB1 AS SYSDBA
SQL> exec DBMS_WORKLOAD_CAPTURE.START_CAPTURE( NAME => 'OLTP_peak_PDB1', -
                                               DIR => 'OLTP_PDB1')
```

ORACLE

In Oracle Database 18c, capturing and replaying the workload can only be executed from the CDB root. This means that the capture and the replay works only at the CDB level. Connect first in the CDB root to start capturing by using the `DBMS_WORKLOAD_CAPTURE.START_CAPTURE` procedure. The other procedures to stop capturing when workload is sufficient for testing, and then to replay can only be done at the CDB level in a connection to the CDB root. It captures workload for all PDBs including the CDB root.

With the popularity of Oracle cloud services, the capture and replay functionality is extended to cloud users to allow each tenant in the CDB to complete a capture and replay for his own data, and, therefore, at the PDB level. This is also available on the on-premises configuration.

Oracle Database 19c allows capturing and replaying workload at the PDB level. Only workload at that PDB is recorded. Only one PDB-level capture or replay can be executed at a time. Replay reporting can be generated at the PDB-level. If a session switches to another PDB, the user calls from that PDB are not captured. If a session that is currently connected to another PDB switches to the current PDB, the user calls are captured. All the replay sessions are required to connect to the current PDB.

Oracle Database 19c also enhances the reporting data, helping to monitor and diagnose performance problems during replay in a multitenant environment. The replay report provides general information about capture and replay, such as workload statistics, errors encountered during the replay, capture and replay duration, replay divergence, and a statistical comparison between the capture phase and the replay phase. The replay report monitors a live replay and helps to provide more information to understand or debug a slow replay.

Reporting

19c

New information added in reporting:

- Replay Sessions
- Synchronization
- Tracked Commits
- Session Failures

The Replay Sessions section includes statistics about the ongoing replay sessions and the finished replay sessions. However, the statistics related to the ongoing replay sessions are shown only if the replay is in progress. This section shows statistics about the replay sessions, including:

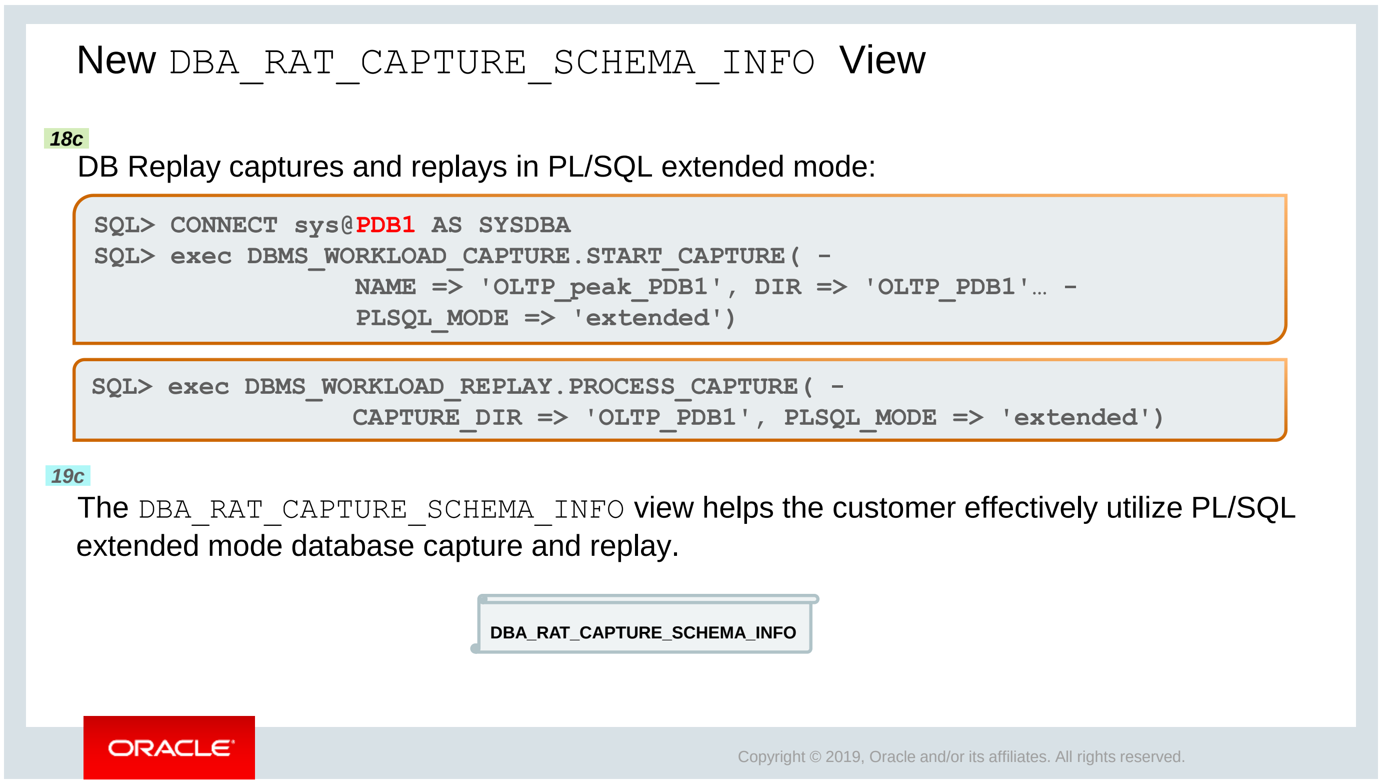
- Top Events
- Top Slowest Replay Sessions
- Top Fastest Replay Sessions

This information allows the user to understand the top events for the ongoing and finished replay sessions, which replay sessions were slower than capture, and a comparison of the session elapsed time between capture and replay.

Database Replay is able to reproduce the captured workload using the exact concurrency. This means the captured SQL coordinate their execution to preserve the execution order. If the execution of one of these SQLs is delayed or blocked during replay, this could result in the replay being slower. The Synchronization section contains information about the sessions blocking the execution of one of these synchronized SQLs. However, this information is shown only if the execution of one of these SQLs is blocked. This allows the user to understand why the replay is blocked or moving slower than capture. This section also includes the Top Events and the Top SQL with Top Events related to those sessions running one of the synchronized SQLs.

Database Replay keeps track of the commits executed during capture, as well as the time between their executions. Based on this, Database Replay is able to detect slow commits and drill down to parts of the workload that slowed down the replay. The Tracked Commits section includes information about the number of captured commits and the time when the commit was executed. Based on this information, the report is able to show if the replay is moving slower than capture.

The replay of a user session might fail and the replay of the remaining calls in the session are skipped. The session failure could be caused by several reasons, including improper system configuration and process crash. The Session Failures section includes information about the SQL executed when the session failed, the file ID and the call counter of the failed session, as well as the number of calls that were not executed.



New DBA_RAT_CAPTURE_SCHEMA_INFO View

18c

DB Replay captures and replays in PL/SQL extended mode:

```
SQL> CONNECT sys@PDB1 AS SYSDBA
SQL> exec DBMS_WORKLOAD_CAPTURE.START_CAPTURE( -
                 NAME => 'OLTP_peak_PDB1', DIR => 'OLTP_PDB1'… -
                 PLSQL_MODE => 'extended')
```

```
SQL> exec DBMS_WORKLOAD_REPLAY.PROCESS_CAPTURE( -
                 CAPTURE_DIR => 'OLTP_PDB1', PLSQL_MODE => 'extended')
```

19c

The DBA_RAT_CAPTURE_SCHEMA_INFO view helps the customer effectively utilize PL/SQL extended mode database capture and replay.

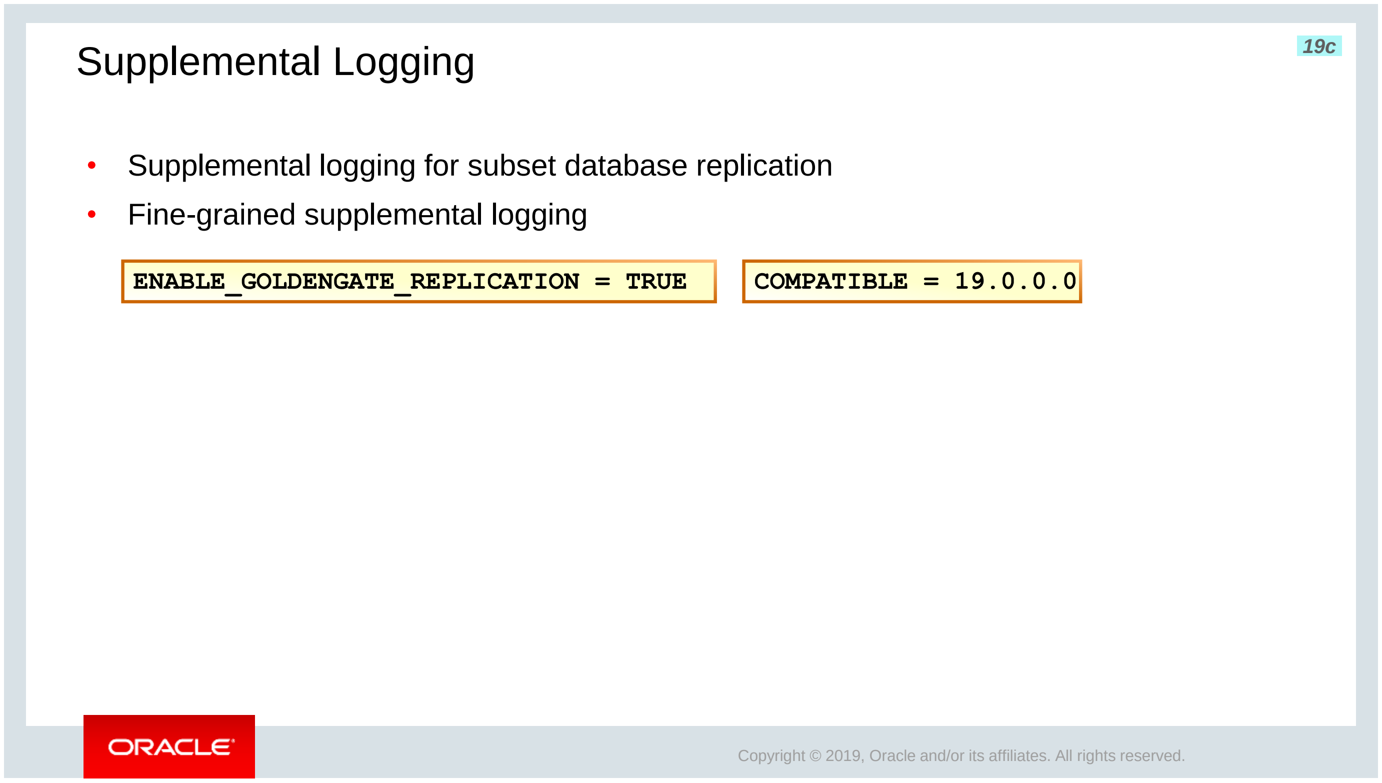
DBA_RAT_CAPTURE_SCHEMA_INFO

ORACLE

In Oracle Database 18c, by allowing the customer to capture and replay database workload at the granular level in PL/SQL, customers can capture and replay their workload more completely and with less divergence. The optional `PLSQL_MODE` parameter in the `DBMS_WORKLOAD_CAPTURE.START_CATPURE` procedure determines how PL/SQL is handled by DB Replay during capture and replays.

The value can be set to `top_level`, where only top-level PL/SQL calls are captured and replayed, which is how DB Replay handled PL/SQL before Oracle Database 12*c* Release 2. This is the default value. The other value is `extended`, where both top-level PL/SQL calls and SQL called from PL/SQL are captured. When the workload is replayed, the replay can be done at either top-level or extended level, but not both.

Oracle Database 19c provides the new `DBA_RAT_CAPTURE_SCHEMA_INFO` view to help the customer effectively utilize PL/SQL extended mode database capture and replay.



Supplemental Logging

19c

- Supplemental logging for subset database replication
- Fine-grained supplemental logging

```
ENABLE_GOLDENGATE_REPLICATION = TRUE
```

```
COMPATIBLE = 19.0.0.0
```

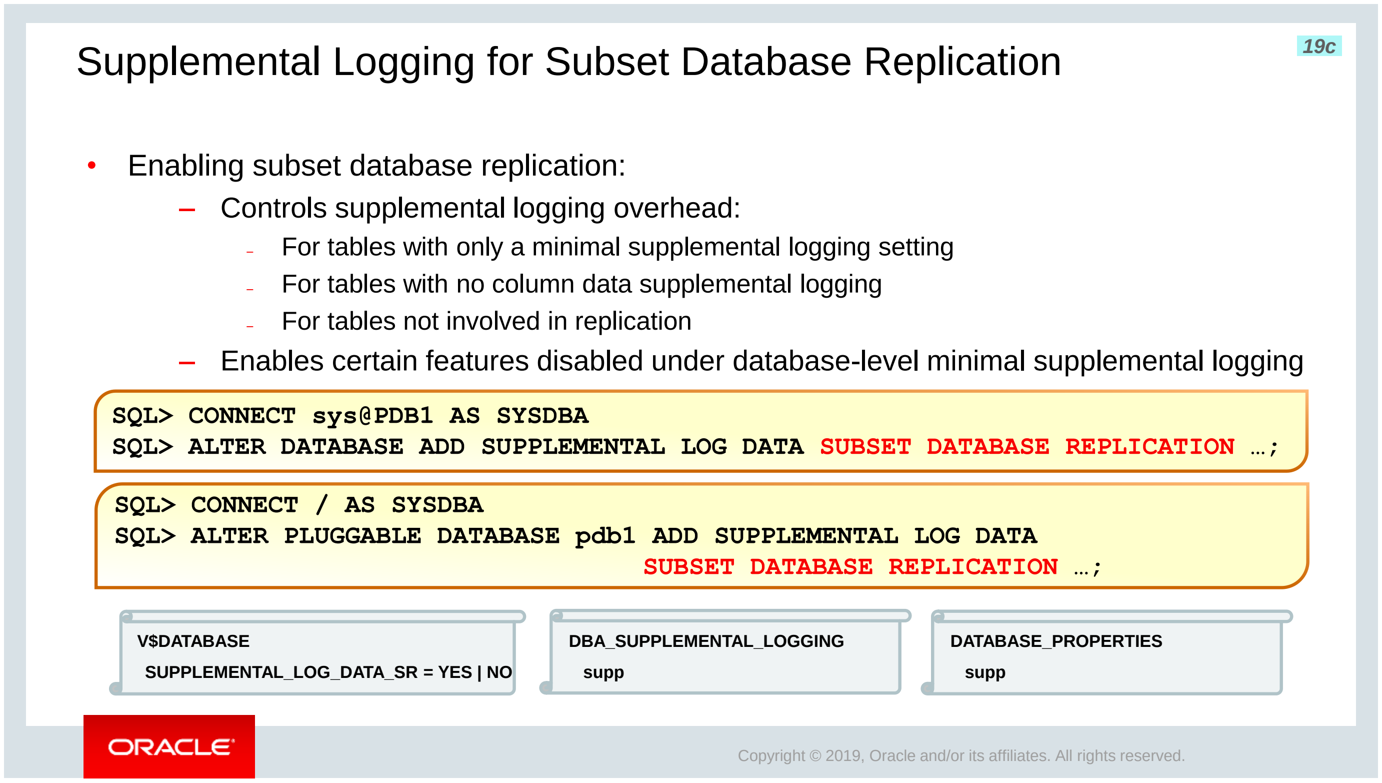
Supplemental logging results in nontrivial overhead for Logminer and log-based replication customers. It adds additional information to redo log and restricts some features in order to make recovery redo self-describing for logical replication.

Oracle Database 19c introduces enhancements to reduce the scope and overhead of supplemental logging:

- Subset database replication supplemental logging
- Fine-grained supplemental logging (or logical replication)

Two instance parameters need to be set to allow the usage of both capabilities:

- `ENABLE_GOLDENGATE_REPLICATION=TRUE`
- `COMPATIBLE=19.0.0.0`



19c

Supplemental Logging for Subset Database Replication

- Enabling subset database replication:
  - Controls supplemental logging overhead:
    - For tables with only a minimal supplemental logging setting
    - For tables with no column data supplemental logging
    - For tables not involved in replication
  - Enables certain features disabled under database-level minimal supplemental logging

```
SQL> CONNECT sys@PDB1 AS SYSDBA
SQL> ALTER DATABASE ADD SUPPLEMENTAL LOG DATA SUBSET DATABASE REPLICATION …;
```

```
SQL> CONNECT / AS SYSDBA
SQL> ALTER PLUGGABLE DATABASE pdb1 ADD SUPPLEMENTAL LOG DATA
                                     SUBSET DATABASE REPLICATION …;
```

V$DATABASE
SUPPLEMENTAL_LOG_DATA_SR = YES | NO

DBA_SUPPLEMENTAL_LOGGING
supp

DATABASE_PROPERTIES
supp

Database-level minimal supplemental logging is the prerequisite for any given redo to be log-minable, and therefore, must be enabled for container level, schema level, or table level column data supplemental logging to take effect. However, it results in redo overhead and feature restrictions such as:

- Only singleton `UPDATE` redoes are generated for multi-row update DML
- Array delete is disabled for compressed rows
- SQL*Loader direct path load to tables with XML/ADT columns error out
- Exclusive table-level lock held for DDL

The overhead of minimal supplemental logging is inevitable for full database replication. However, for partial database replication such as with Oracle Golden Gate, Oracle Database 19c introduces subset database replication supplemental logging to reduce the redo overhead for tables with only a minimal supplemental logging setting, with no column data supplemental logging or involvement in replication.

By default, subset database replication supplemental logging is disabled. When subset database replication supplemental logging is disabled, database-wide supplemental logging behavior is the same as the behavior in Oracle Database 18c. When subset database replication supplemental logging is enabled, redo are not generated for tables with only minimal supplemental logging setting or with no column data supplemental logging or no involvement in replication. Certain features currently disabled under database-level minimal supplemental logging can be enabled. A property is set in the `DATABASE_PROPERTIES` view to indicate whether subset database replication supplemental logging is enabled or not.

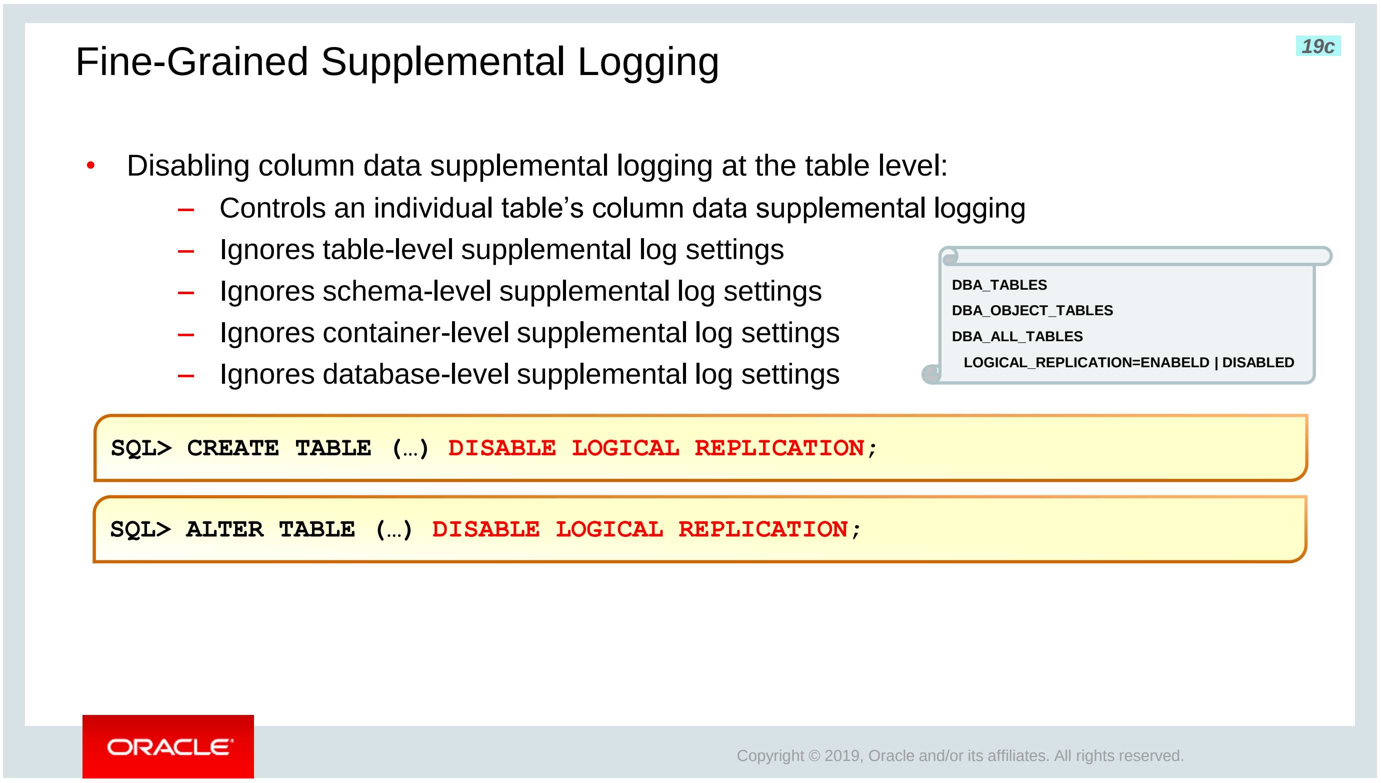
The `ALTER DATABASE ADD SUPPLEMENTAL LOG DATA SUBSET DATABASE REPLICATION` command is only allowed when the `ENABLE_GOLDENGATE_REPLICATION` parameter is set to `TRUE`.

In case the `ALTER DATABASE ADD SUPPLEMENTAL LOG DATA SUBSET DATABASE REPLICATION` command is executed from the CDB root, subset database replication supplemental logging is enabled for the entire database, including all PDBs, regardless of the PDB-level setting for subset database replication supplemental logging.

If subset database replication supplemental logging needs to be enabled for a specific PDB, you have two possibilities:

- Execute the `ALTER PLUGGABLE DATABASE` *`pdb_name`* `ADD SUPPLEMENTAL LOG DATA SUBSET DATABASE REPLICATION` command from a connection to the CDB root.
- Execute the `ALTER DATABASE ADD SUPPLEMENTAL LOG DATA SUBSET DATABASE REPLICATION` command from a connection to the PDB.

To disable subset database replication supplemental logging, use the `ALTER DATABASE DROP SUPPLEMENTAL LOG DATA SUBSET DATABASE REPLICATION` or `ALTER PLUGGABLE DATABASE` *`pdb_name`* `DROP SUPPLEMENTAL LOG DATA SUBSET DATABASE REPLICATION` command.



Fine-Grained Supplemental Logging

19c

- Disabling column data supplemental logging at the table level:
  – Controls an individual table’s column data supplemental logging
  – Ignores table-level supplemental log settings
  – Ignores schema-level supplemental log settings
  – Ignores container-level supplemental log settings
  – Ignores database-level supplemental log settings

DBA_TABLES
DBA_OBJECT_TABLES
DBA_ALL_TABLES
LOGICAL_REPLICATION=ENABELD | DISABLED

```
SQL> CREATE TABLE (…) DISABLE LOGICAL REPLICATION;
```

```
SQL> ALTER TABLE (…) DISABLE LOGICAL REPLICATION;
```

Schema-level supplemental logging in partial database replication eliminates the unsafe window where supplemental log data is missing for DMLs done between table creation and supplemental logging addition. However, schema-level supplemental logging imposes huge overhead when only a small subset of tables in schemas need to be replicated.

Oracle Database 19c introduces the fine-grained supplemental logging capability to reduce the overhead, enabling and disabling column data supplemental logging at the table level, so that an individual table’s column data supplemental logging can be turned on or off regardless of its upper-level supplemental log settings. If supplemental logging (or logical replication) is disabled, a bit is set in the `DATABASE_PROPERTIES` view. The property bit is used during the semantic analysis phase to indicate that minimal supplemental logging is done for this table regardless of its schema- or container- or database-level supplemental setting.

Fine-grained supplemental logging allows partial database replication to disable column data supplemental logging for uninteresting tables so that even when supplemental logging is enabled at the database- or container- or schema-level, the overhead of supplemental logging is greatly reduced for uninteresting tables.

The `ALTER TABLE … DISABLE LOGICAL REPLICATION` command disables logical replication at the table level. By default, table logical replication is enabled. When logical replication is enabled for a table, the table’s supplemental log data is determined by the union of database-, container-, schema-, and table-level supplemental log setting. This is the same behavior as in Oracle Database 18c.

When logical replication is disabled for a table, the table only has minimal supplemental logging, with all the supplemental logging settings at the database-, container-, and schema-levels being ignored. You can only disable logical replication for a table when no supplemental logging setting is specified at the table level to avoid losing supplemental logging data by mistake. When table-level supplemental logging is added, logical replication is implicitly enabled for this table.

Summary

In this lesson, you should have learned how to:

- Configure and use deferred writes with Memoptimized Rowstore - Fast Ingest
- Configure PDB-level ADDM analysis
- Enable SQL developers to use Real-Time SQL Monitor
- Use automatic SQL plan management with the AUTO option
- Wait until all in-memory objects with a nondefault priority are populated
- Configure and use DB Replay at the PDB level
- Use a view to effectively utilize the PL/SQL extended mode of DB Replay
- Reduce the scope and overhead of supplemental logging

Practices Environment: 1

Each student has one virtual machine (VM1) to use during class.

- Oracle Database 19c installed
- One CDB named **`ORCL`** and its PDB named **`PDB1`**

Practices Environment: 2

Pre-created databases and instances with their respective PDBs:

- Data files in `/u02/app/oracle/oradata/`
- All backup files in `/u03/app/oracle/fast_recovery_area/ORCL`
- Password and init files in `$ORACLE_HOME/dbs`
- Diagnostics files in `/u01/app/oracle/diag/rdbms/orcl`
- TDE wallet in `/u01/app/oracle/admin/orcl/tde_wallet`
- Net files in `$ORACLE_HOME/network/admin`

The configuration of the environment used for the practices of the course matches the configuration used on the Oracle Cloud Virtual Machine for the pre-created `ORCL` database of Oracle Database Cloud Service. Oracle Database Cloud Service gives you the ability to deploy Oracle databases in the cloud, with each database deployment containing a single Oracle database.

This is a good way to get familiar with the Oracle Database Cloud environment that is preconfigured for cloud customers.

Practice 4: Overview

- 4-1: Using Memoptimized Rowstore - Fast Ingest
- 4-2: Completing an ADDM Analysis Inside a PDB
- 4-3: Using Real-Time SQL Monitoring as a SQL Developer
- 4-4: Waiting for In-Memory Objects to be Populated
- 4-5: Configuring and Using Database Replay at the PDB Level

5

Using Big Data and Data Warehousing Enhancements

ORACLE

Overview

- This module focuses on data warehousing enhancements of Oracle Database 19c.
- It complements the data warehousing topics covered in:
  - The 5-day *Oracle Database 12c: 12.2 New Features for 12.1 Administrators Ed 1* course
  - The 5-day *Oracle Database 12c R2: New Features for Administrators Part 2 Ed 1* course
  - The 3-day *Oracle Database 18c: New Features for Administrators Ed 1* course
- Previous experience with Oracle Database 12*c* or 18c is required for a full understanding of data warehousing enhancements.

Refer to *Oracle Database New Features Guide 19c, Oracle Database VLDB and Partitioning Guide 19c, and Oracle Database In-Memory Guide 19c.*

This module follows either the 5-day *Oracle Database 12c: 12.2 New Features for 12.1 Administrators Ed 1* course or the 5-day *Oracle Database 12c R2: New Features for Administrators Part 2 Ed 1* course or the 3-day *Oracle Database 18c: New Features for Administrators Ed 1* course. These courses are designed to introduce the enhancements of Oracle Database 12*c* Release 2 (12.2.0.1) and Oracle Database 18c that are applicable to work that is usually performed by database administrators and related personnel.

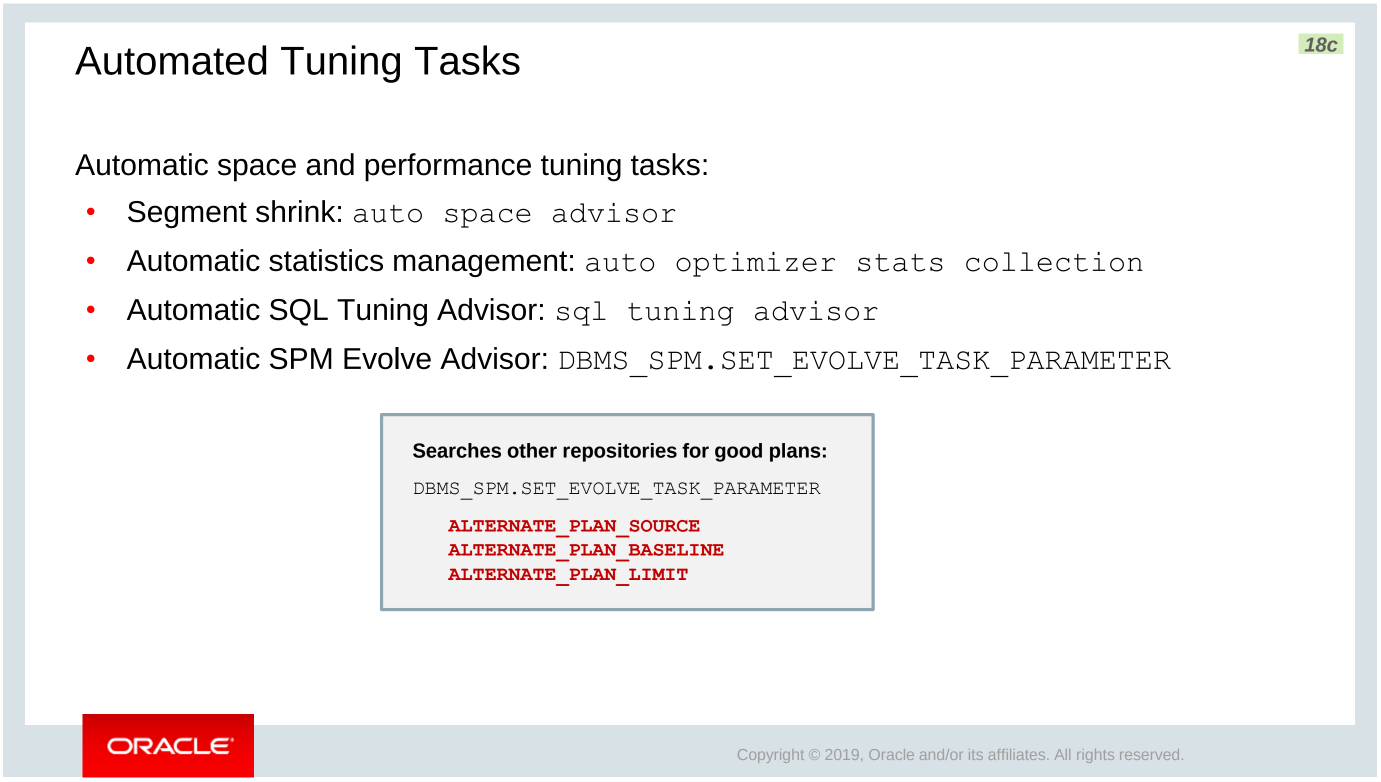
The module is designed to introduce the major enhancements of Oracle Database 19c related to big data and data warehousing.

The module consists of an instructor-led lesson and practices that enable you to see how certain new functionalities behave.

Objectives

After completing this lesson, you should be able to:

- Configure and use the automatic indexing task
- Configure and use hybrid partitioned tables with internal and external partitions
- Populate in-memory external tables
- Execute a parallel query on in-memory external tables
- Use the `ORACLE_HIVE` and `ORACLE_BIGDATA` driver types for in-memory external tables



18c

Automated Tuning Tasks

Automatic space and performance tuning tasks:

- Segment shrink: `auto space advisor`
- Automatic statistics management: `auto optimizer stats collection`
- Automatic SQL Tuning Advisor: `sql tuning advisor`
- Automatic SPM Evolve Advisor: `DBMS_SPM.SET_EVOLVE_TASK_PARAMETER`

Searches other repositories for good plans:

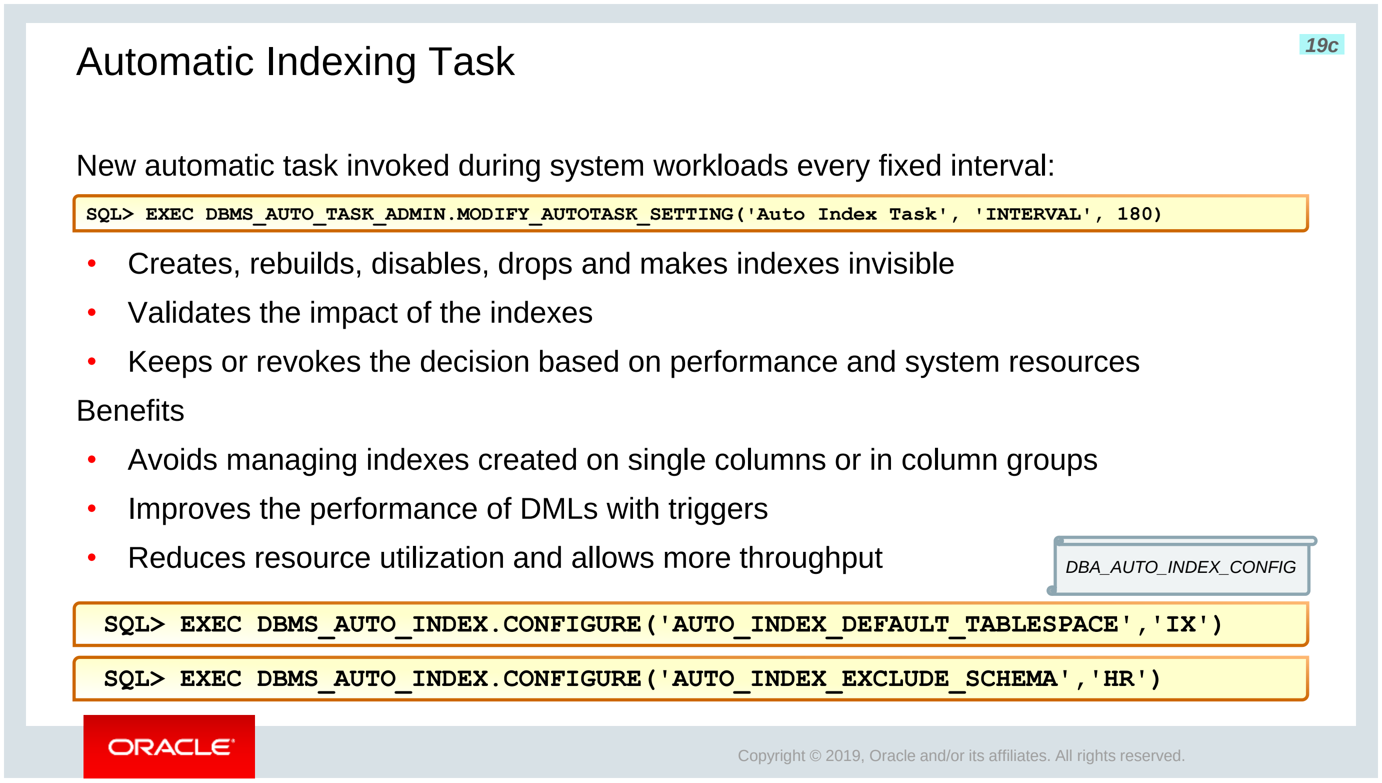
```
DBMS_SPM.SET_EVOLVE_TASK_PARAMETER
   ALTERNATE_PLAN_SOURCE
   ALTERNATE_PLAN_BASELINE
   ALTERNATE_PLAN_LIMIT
```

ORACLE

To control the automatic segment shrink task, the automatic statistics collection, or the SQL Tuning Advisor automated task, use the `DBMS_AUTO_TASK_ADMIN.ENABLE` procedure:

```
BEGIN
  DBMS_AUTO_TASK_ADMIN.ENABLE (
     client_name => 'auto space advisor',
     operation   => NULL, window_name => NULL);
END;
BEGIN
  DBMS_AUTO_TASK_ADMIN.ENABLE (
     client_name => 'auto optimizer stats collection',
     operation   => NULL, window_name => NULL);
END;
BEGIN
  DBMS_AUTO_TASK_ADMIN.ENABLE (
     client_name => 'sql tuning advisor',
     operation   => NULL, window_name => NULL);
END;
```

In Oracle Database 12.2, the automatic SPM Evolve Advisor can search for additional plans that are not yet in the SQL Management Base (SMB) in other sources like AWR, STS, and cursor cache. If such plans are found, the advisor tries to evolve them by using its existing algorithm. If their performance is satisfactory, the advisor adds them to the SMB and marks them as accepted plans. SPM also needs the alternate plans to be test-executed. Specify the task parameters using the `DBMS_SPM.SET_EVOLVE_TASK_PARAMETER` procedure.



Automatic Indexing Task

19c

New automatic task invoked during system workloads every fixed interval:

```
SQL> EXEC DBMS_AUTO_TASK_ADMIN.MODIFY_AUTOTASK_SETTING('Auto Index Task', 'INTERVAL', 180)
```

- Creates, rebuilds, disables, drops and makes indexes invisible
- Validates the impact of the indexes
- Keeps or revokes the decision based on performance and system resources

Benefits

- Avoids managing indexes created on single columns or in column groups
- Improves the performance of DMLs with triggers
- Reduces resource utilization and allows more throughput

DBA_AUTO_INDEX_CONFIG

```
SQL> EXEC DBMS_AUTO_INDEX.CONFIGURE('AUTO_INDEX_DEFAULT_TABLESPACE','IX')
```

```
SQL> EXEC DBMS_AUTO_INDEX.CONFIGURE('AUTO_INDEX_EXCLUDE_SCHEMA','HR')
```

Oracle Database 19c introduces the Automatic Indexing task, a very important performance feature for the Autonomous Transaction Processing Cloud (ATP) and also for the deployment of the data warehouse application in Autonomous Data Warehouse Cloud (ADWC). The automated task establishes a set of candidate indexes for creation during workloads in order to:

- Create single and concatenated indexes that are based on:
  - Single column usage tracking, enabled by default
  - Column group usage
- Validate the impact of the indexes
- Make decisions on whether to keep or drop the indexes
- Monitor the usage of the indexes and drop the ones that are not used after a predefined and configurable time period

The Automatic Indexing task is not limited to creating indexes, but includes all activities related to indexes, such as disabling (unusable), rebuilding, making them invisible, and dropping. For example, the index is automatically rebuilt following a large number of DMLs that cause a decay in the index structure, making it less efficient during query processing. Another example is partition maintenance operations like `ALTER TABLE MOVE`, which make the indexes unusable. These indexes are automatically rebuilt by the task.

The Automatic Indexing task is configured by using the `DBMS_AUTO_INDEX.CONFIGURE` procedure to set the preferences that control the execution and behavior of tasks.

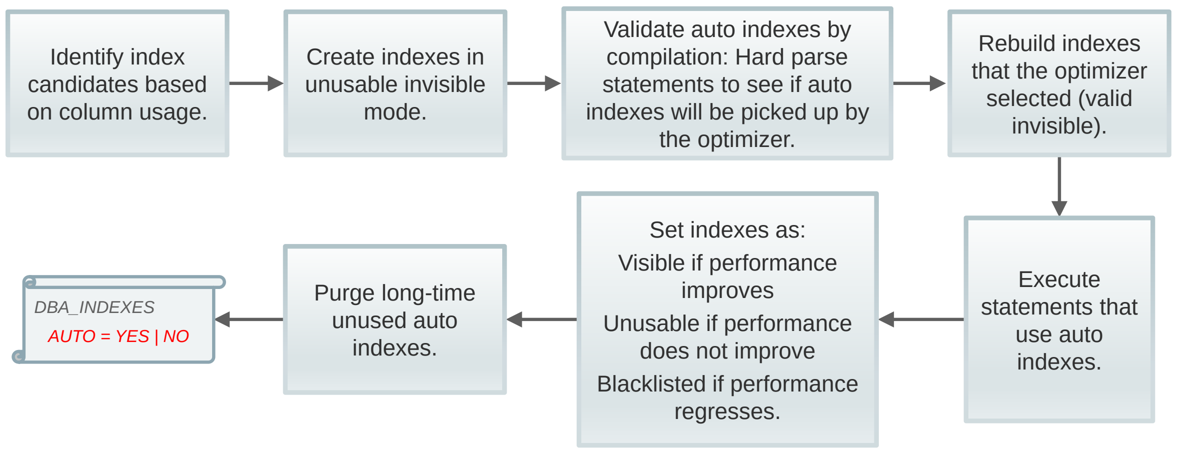
- `AUTO_INDEX_EXCLUDE_SCHEMA`: Specifies a schema name to exclude from creating indexes
  automatically. The function can be called multiple times to specify multiple schemas.
- `AUTO_INDEX_REPORT_RETENTION`: Defines the number of days auto indexing logs are retained before they are purged. Auto index report is generated based on the logs, and it cannot generate a report for the period beyond the specified retention. The default value is 31 days.
- `AUTO_INDEX_RETENTION_FOR_AUTO`: Defines the number of days auto indexes are retained after their last used date before they are purged. The default value is 373 days.
- `AUTO_INDEX_RETENTION_FOR_MANUAL`: Defines the number of days user-created indexes are retained after their last used date before they are purged. NULL value means user-created indexes are not purged by the auto index task. The default value is NULL.
- `AUTO_INDEX_DEFAULT_TABLESPACE`: Defines the tablespace name where all auto indexes are stored. The default value is NULL and interpreted as the user default tablespace.
- `AUTO_INDEX_TEMP_TABLESPACE`: Defines the tablespace name where you can allocate temporary space when building an index. The default value is NULL and interpreted as the user default temporary tablespace.
- `AUTO_INDEX_MODE`: Defines the different modes that auto index can operate.
  - `IMPLEMENT`: Auto indexes are available for users to use in their workload. This is the default value.
  - `REPORT ONLY`: Auto indexes are not made available for the user to use.
- `AUTO_INDEX_SPACE_BUDGET`: Defines the percentage of user default tablespace that can be used for auto indexes. This value is ignored if `AUTO_INDEX_DEFAULT_TABLESPACE` has been set to a valid value. The default value is NULL.

The `REPORT_ACTIVITY` and `REPORT_LAST_ACTIVITY` functions report the results of the task.

The task runs automatically and independently at each container level, PDB or CDB root.

Workflow

The task performs the following steps every time it is invoked or every *n* minutes:


Identify index candidates based on column usage.
Create indexes in unusable invisible mode.
Validate auto indexes by compilation: Hard parse statements to see if auto indexes will be picked up by the optimizer.
Rebuild indexes that the optimizer selected (valid invisible).
Execute statements that use auto indexes.
Set indexes as:
Visible if performance improves
Unusable if performance does not improve
Blacklisted if performance regresses.
Purge long-time unused auto indexes.
DBA_INDEXES
AUTO = YES | NO

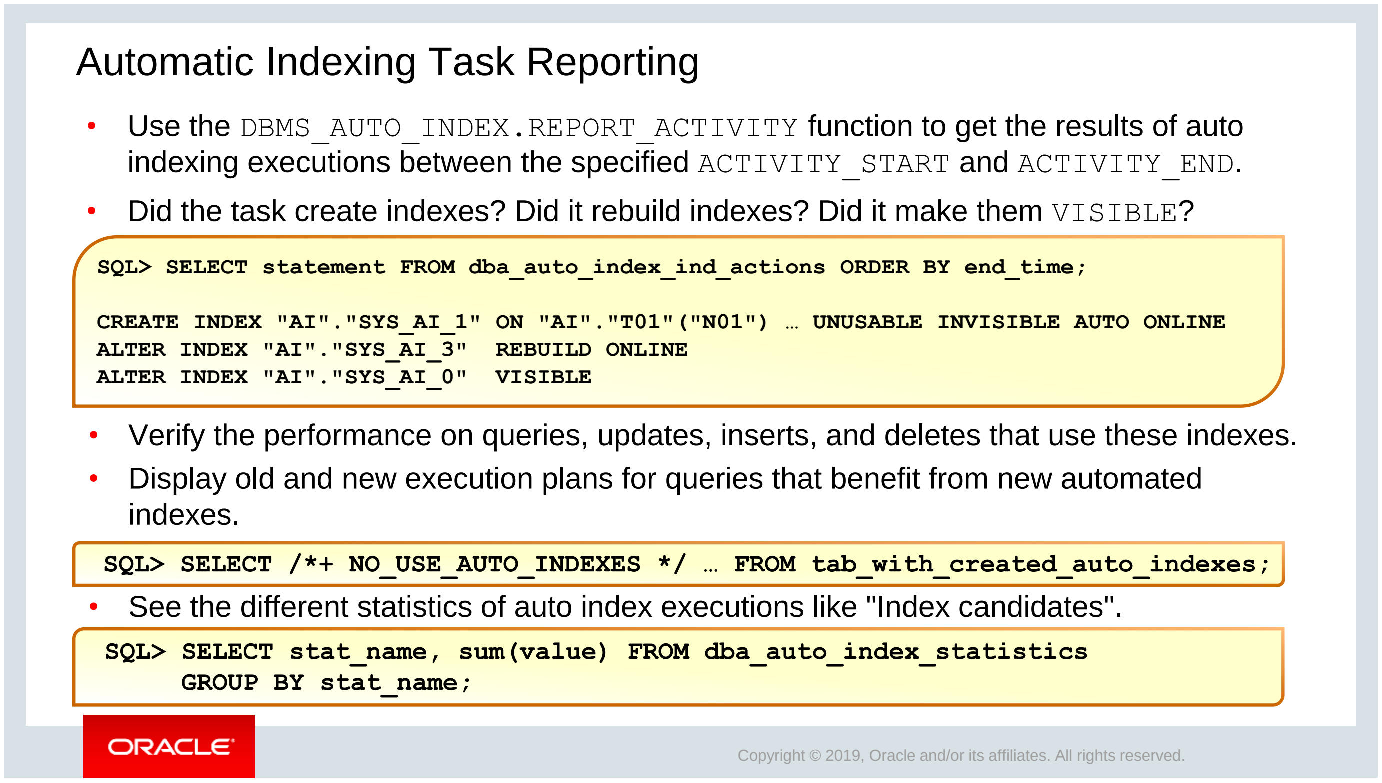
Every time the task is invoked, it performs the following steps:

1. **Identify index candidates:** In each iteration, the identification is based on column usage and column group usage.
2. **Create indexes based on merged column groups in unusable or invisible mode:** Unusable indexes do not have segments and there is no maintenance cost. It is created in invisible mode so that it is not visible to user workload until the impact of creating it is validated. Only B-Tree indexes are created. Only local indexes are created on partitioned tables. Indexes are created on tables with statistics.
3. **Validate auto indexes by compilation:** Use the top N past executed user statements to validate if the candidate indexes are worth rebuilding. The top N statements could be based on CPU and I/O utilization and new statements seen after the last iteration. The validation consists of two steps:
   a) Compilation to check whether the optimizer would choose the new indexes
   b) Execution to check the performance of the SQL statements after rebuilding the indexes

   After the validation, the possible outcomes are:
   - **At least one of the statements improves above some threshold:** All the indexes rebuilt in the current iteration are marked as visible.
   - **None of the statements improve significantly:** The new indexes are marked as unusable. This reduces the number of auto indexes maintained in the system.
   - Statements that regress are blacklisted to not use auto indexes. A SQL patch is provided to not use these indexes.
4. **Purge auto indexes:** Drop auto indexes that are not used for a long time (for example, 53 weeks).

The new `AUTO` column in the `DBA_INDEXES` view indicates whether the index is created by the auto index task.

You can view the free video about this feature in the course under the module name “On - Premises Videos”.



Automatic Indexing Task Reporting

- Use the `DBMS_AUTO_INDEX.REPORT_ACTIVITY` function to get the results of auto indexing executions between the specified `ACTIVITY_START` and `ACTIVITY_END`.
- Did the task create indexes? Did it rebuild indexes? Did it make them `VISIBLE`?

```
SQL> SELECT statement FROM dba_auto_index_ind_actions ORDER BY end_time;

CREATE INDEX "AI"."SYS_AI_1" ON "AI"."T01"("N01") … UNUSABLE INVISIBLE AUTO ONLINE
ALTER INDEX "AI"."SYS_AI_3"  REBUILD ONLINE
ALTER INDEX "AI"."SYS_AI_0"  VISIBLE
```

- Verify the performance on queries, updates, inserts, and deletes that use these indexes.
- Display old and new execution plans for queries that benefit from new automated indexes.

```
SQL> SELECT /*+ NO_USE_AUTO_INDEXES */ … FROM tab_with_created_auto_indexes;
```

- See the different statistics of auto index executions like "Index candidates".

```
SQL> SELECT stat_name, sum(value) FROM dba_auto_index_statistics
     GROUP BY stat_name;
```

The `DBMS_AUTO_INDEX.REPORT_ACTIVITY` function reports the results of auto index executions between the specified `ACTIVITY_START` and `ACTIVITY_END` periods.

You can configure the following parameters for the function:

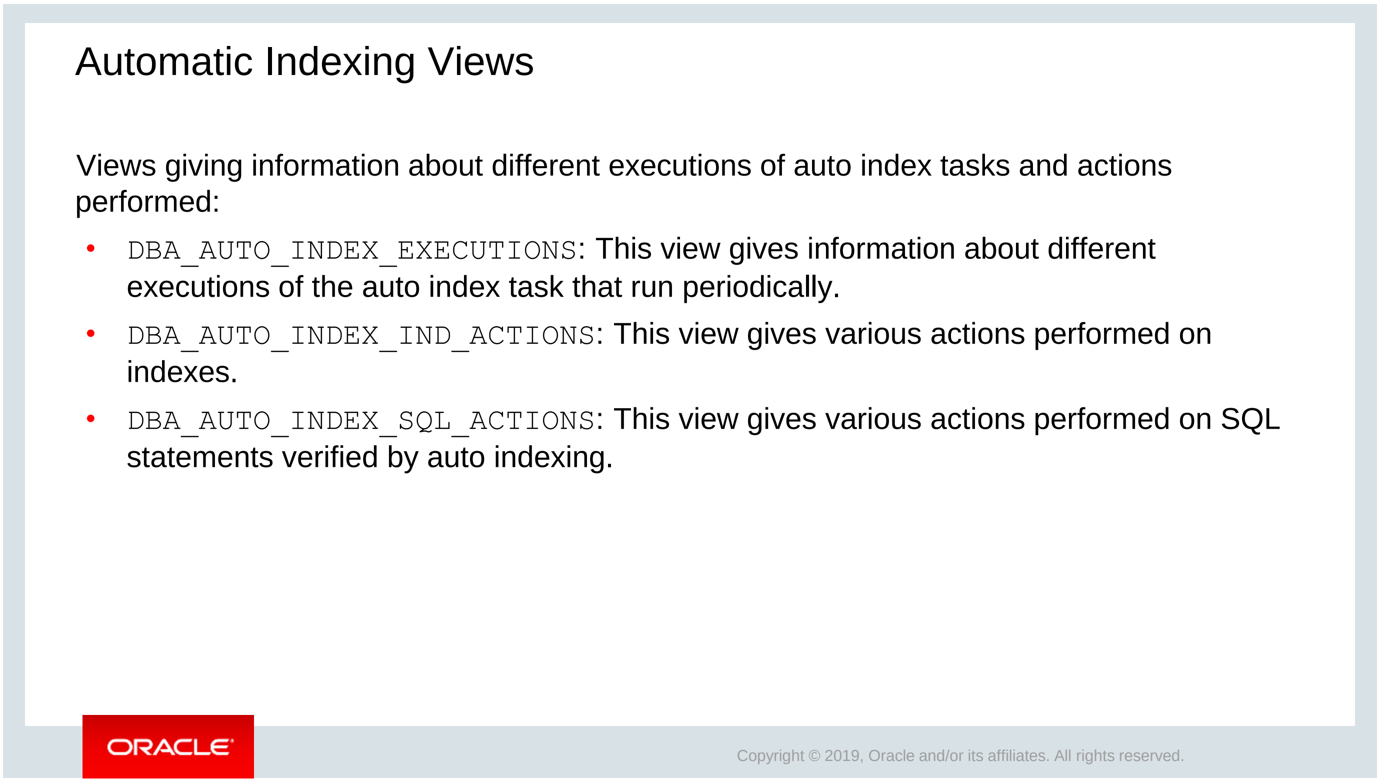
- `ACTIVITY_START:` Time stamp from which auto index executions are observed for the report. NULL shows the report for the last execution. The default is current time - 1 day, `SYSTIMESTAMP - 1`.
- `ACTIVITY_END:` Time stamp until which auto index executions are observed for the report. The default is current time, `SYSTIMESTAMP`.
- `TYPE:` Type of the report. Possible values are `TEXT`, `HTML`, `XML`. The default is `TEXT`.
- `SECTION:` Particular section in the report. Possible values are `SUMMARY`, `INDEX_DETAILS`, `VERIFICATION_DETAILS`, `ERRORS`, `ALL`. Combinations of different values can be specified. For example: '`SUMMARY+INDEX_DETAILS+ERRORS`', '`ALL-ERRORS`'. The default is `ALL`.
- `LEVEL:` Format of the report. Possible values are `BASIC`, `TYPICAL`, `ALL`. The default is `TYPICAL`.

The following example reports the results of auto index executions of the last five days.

```
DECLARE
  report clob := null;
BEGIN
  report := DBMS_AUTO_INDEX.REPORT_ACTIVITY ( ACTIVITY_START => systimestamp,
                                              ACTIVITY_END => systimestamp - 5);
END;
```

The `DBMS_AUTO_INDEX.REPORT_LAST_ACTIVITY` function reports the results of auto index execution of the last day in text format, with all sections at the typical level.

You can compare the performance results on tables on which the auto indexing task implemented indexes by first querying the tables with the `NO_USE_AUTO_INDEXES` hint and then without the `NO_USE_AUTO_INDEXES` (default is `USE_AUTO_INDEXES`).

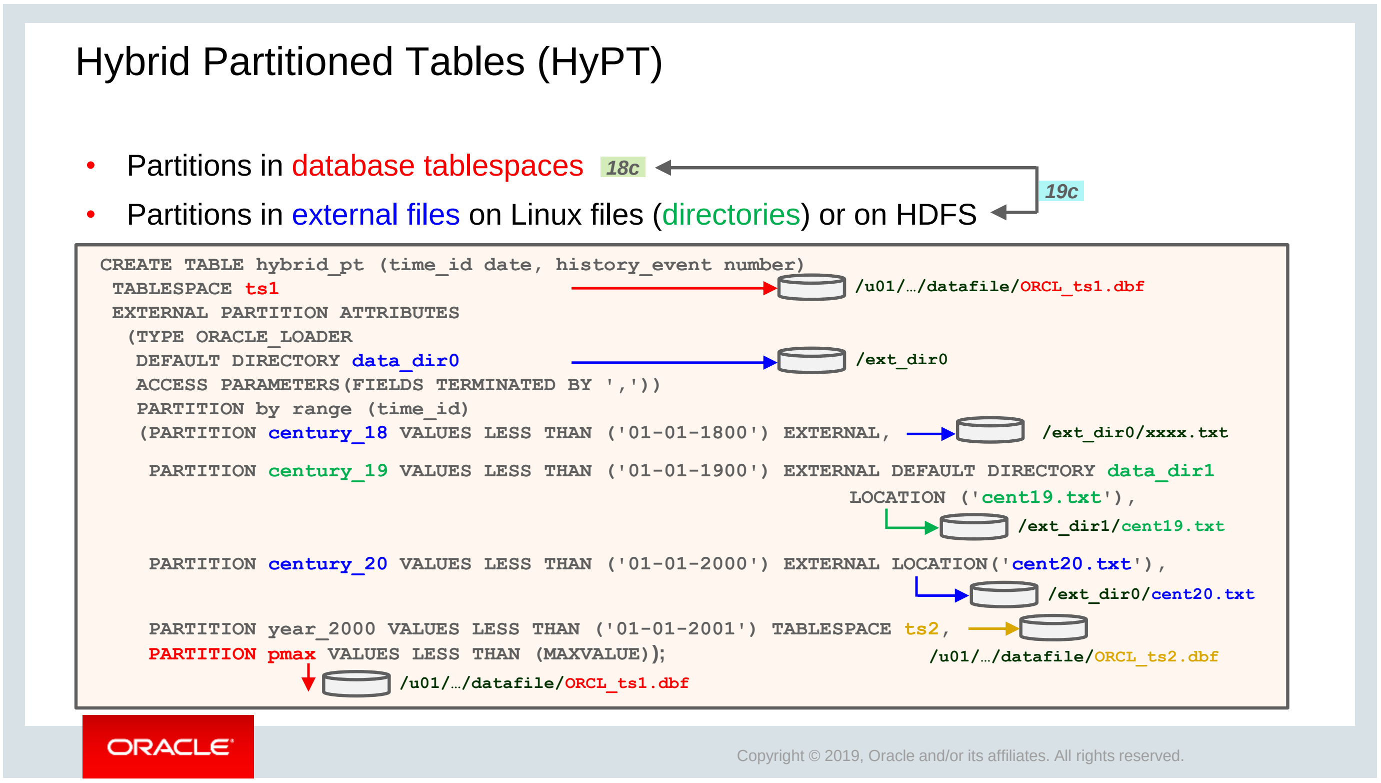


Automatic Indexing Views

Views giving information about different executions of auto index tasks and actions performed:

- DBA_AUTO_INDEX_EXECUTIONS: This view gives information about different executions of the auto index task that run periodically.
- DBA_AUTO_INDEX_IND_ACTIONS: This view gives various actions performed on indexes.
- DBA_AUTO_INDEX_SQL_ACTIONS: This view gives various actions performed on SQL statements verified by auto indexing.

- DBA_AUTO_INDEX_EXECUTIONS:
  - EXECUTION_NAME: Name of an execution of the auto index task
  - EXECUTION_START/EXECUTION_END: Start/end time of the execution
  - STATUS: Status of the execution
- DBA_AUTO_IND_ACTIONS:
  - EXECUTION_NAME: Name of an execution of the auto index task
  - ACTION_ID: An ID for the action
  - INDEX_NAME/INDEX_OWNER: Index name/owner
  - TABLE_NAME/TABLE_OWNER: Index belongs to this table/owner
  - COMMAND: Command used
  - STATEMENT: Statement used for performing the action
  - START_TIME/END_TIME: Action start/end time
  - ERROR#: Any error (0 means no error)
- DBA_AUTO_INDEX_SQL_ACTIONS:
  - EXECUTION_NAME: Name of an execution of auto index task
  - ACTION_ID: An ID for the action
  - SQL_ID: SQL_ID for the statement verified
  - PLAN_HASH_VALUE: Plan hash value of the statement verified
  - COMMAND: Command used
  - STATEMENT: Statement used for performing the action
  - START_TIME /END_TIME: Action start/end time
  - ERROR#: Any error (0 means no error)



Hybrid Partitioned Tables (HyPT)

- Partitions in database tablespaces 18c
- Partitions in external files on Linux files (directories) or on HDFS 19c

```
CREATE TABLE hybrid_pt (time_id date, history_event number)
 TABLESPACE ts1
 EXTERNAL PARTITION ATTRIBUTES
  (TYPE ORACLE_LOADER
   DEFAULT DIRECTORY data_dir0
   ACCESS PARAMETERS(FIELDS TERMINATED BY ','))
   PARTITION by range (time_id)
   (PARTITION century_18 VALUES LESS THAN ('01-01-1800') EXTERNAL,
    PARTITION century_19 VALUES LESS THAN ('01-01-1900') EXTERNAL DEFAULT DIRECTORY data_dir1
                                                       LOCATION ('cent19.txt'),
    PARTITION century_20 VALUES LESS THAN ('01-01-2000') EXTERNAL LOCATION('cent20.txt'),
    PARTITION year_2000 VALUES LESS THAN ('01-01-2001') TABLESPACE ts2,
    PARTITION pmax VALUES LESS THAN (MAXVALUE));
```

/u01/…/datafile/ORCL_ts1.dbf
/ext_dir0
/ext_dir0/xxxx.txt
/ext_dir1/cent19.txt
/ext_dir0/cent20.txt
/u01/…/datafile/ORCL_ts2.dbf
/u01/…/datafile/ORCL_ts1.dbf

ORACLE

Oracle Database 12.2 allowed partitions of a partitioned table to be created either as internal partitions in the database in Oracle data files or in external sources as external partitions.

Oracle Database 19c introduces the Hybrid Partitioned Tables (HyPT). It allows partitions of a partitioned table to reside in the database in Oracle data files (internal partitions) and also in external sources (external partitions).

`RANGE` and `LIST` partitioning methods are supported in a HyPT.

The `EXTERNAL PARTITION ATTRIBUTES` clause of the `CREATE TABLE` statement is defined at the table level for specifying table-level external parameters in the HyPT, such as:

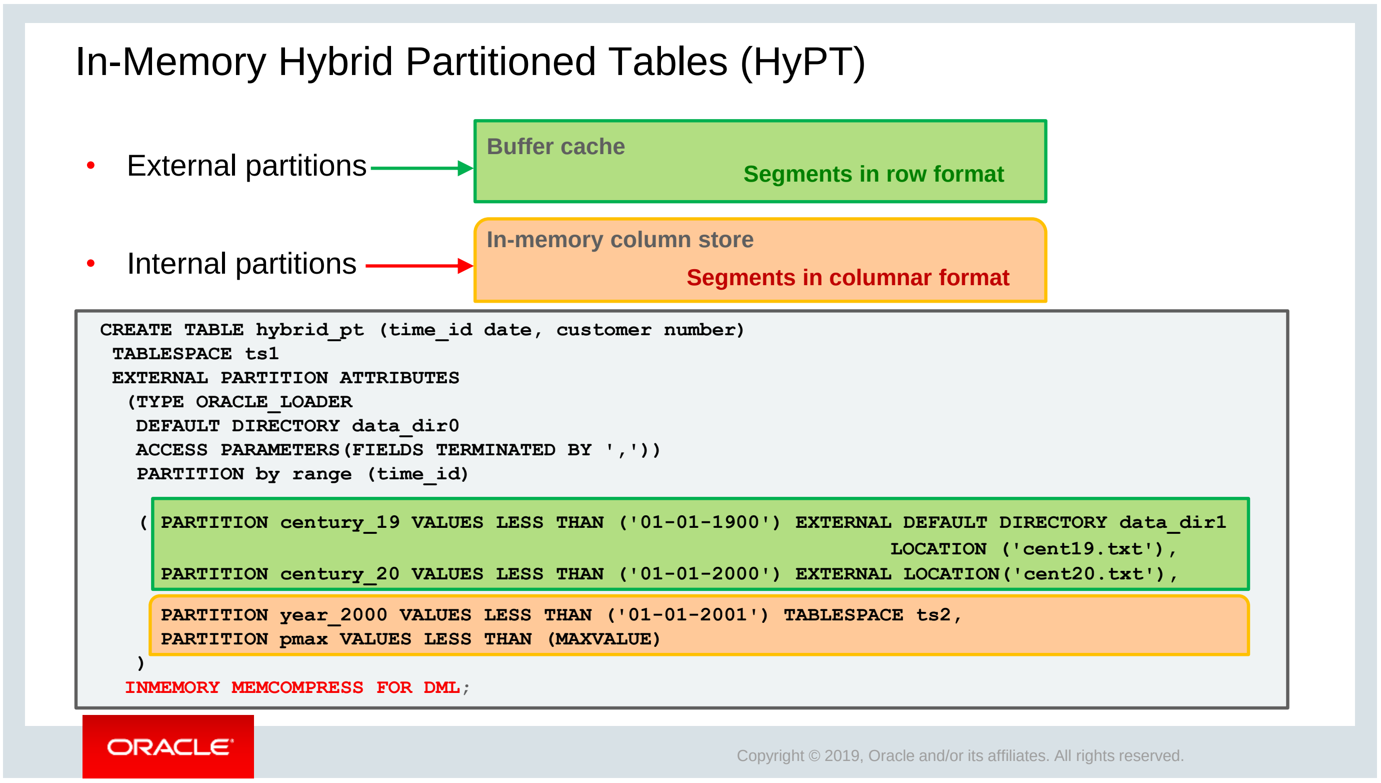
- The access driver type (`ORACLE_LOADER`, `ORACLE_DATAPUMP`, `ORACLE_HDFS`, `ORACLE_HIVE`)
- The `DEFAULT DIRECTORY` for all external partition files
- The `ACCESS PARAMETERS`

The `EXTERNAL` clause of the `PARTITION` clause defines the partition as an external partition. When there is no `EXTERNAL` clause, the partition is an internal partition. You can specify for each external partition different attributes than the default attributes defined at the table level, such as the directory.

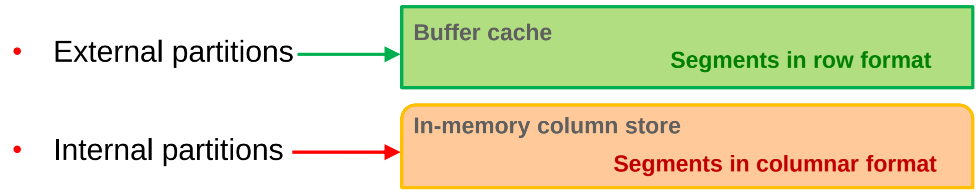
When there is no external file defined for an external partition, the external partition is empty. It can be populated with an external file by using an `ALTER TABLE MODIFY PARTITION` statement.

In the example in the slide, the table is a hybrid range partitioned table with five partitions. There are three external partitions and two internal partitions. The external files for the `century_18` and `century_20` external partitions are stored in the `data_dir0` directory defined in the `DEFAULT DIRECTORY` of the `EXTERNAL PARTITION ATTRIBUTES` clause. The external file for the `century_19` external partition is stored in another directory, `data_dir1`, defined at the partition level. The `century_18` external partition is an empty external partition with no external file at the moment.

The last two partitions are internal partitions: the `year_2000` partition stores its data in the `ts2` tablespace and not in `ts1` defined at the table level, whereas the `pmax` partition stores its data in the `ts1` tablespace defined at the table level.



In-Memory Hybrid Partitioned Tables (HyPT)

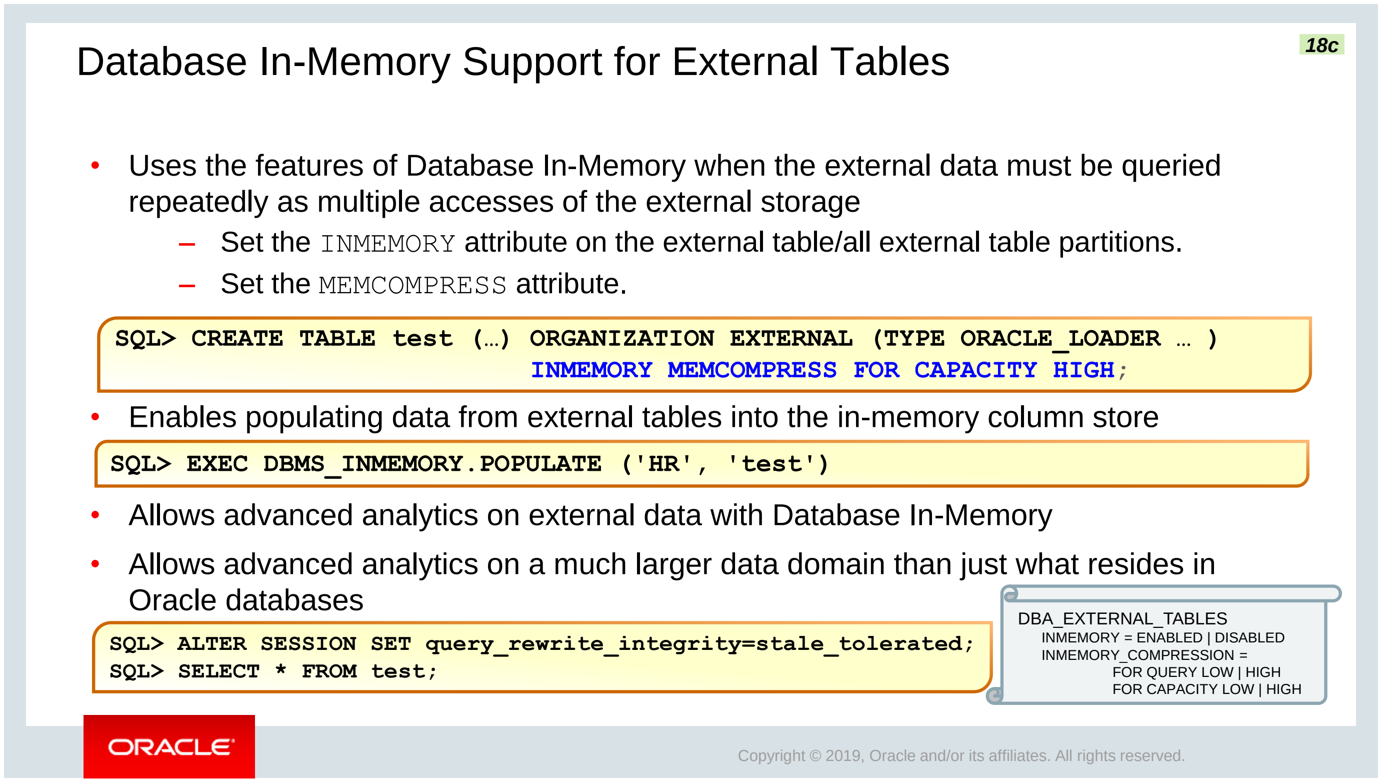


```
CREATE TABLE hybrid_pt (time_id date, customer number)
 TABLESPACE ts1
 EXTERNAL PARTITION ATTRIBUTES
  (TYPE ORACLE_LOADER
   DEFAULT DIRECTORY data_dir0
   ACCESS PARAMETERS(FIELDS TERMINATED BY ','))
   PARTITION by range (time_id)

    ( PARTITION century_19 VALUES LESS THAN ('01-01-1900') EXTERNAL DEFAULT DIRECTORY data_dir1
                                                                 LOCATION ('cent19.txt'),
      PARTITION century_20 VALUES LESS THAN ('01-01-2000') EXTERNAL LOCATION('cent20.txt'),
      PARTITION year_2000 VALUES LESS THAN ('01-01-2001') TABLESPACE ts2,
      PARTITION pmax VALUES LESS THAN (MAXVALUE)
    )
  INMEMORY MEMCOMPRESS FOR DML;
```

A HyPT can be defined as an in-memory table, but only the internal partitions inherit the in-memory attributes.

In the example in the slide, the table is a hybrid range partitioned table with four partitions. Only the two internal partitions inherit the in-memory attributes.

If you then add an external partition, it will not inherit the in-memory attributes, whereas if you add an internal partition, it will inherit the in-memory attributes.



Database In-Memory Support for External Tables

18c

- Uses the features of Database In-Memory when the external data must be queried repeatedly as multiple accesses of the external storage
  - Set the `INMEMORY` attribute on the external table/all external table partitions.
  - Set the `MEMCOMPRESS` attribute.

```
SQL> CREATE TABLE test (…) ORGANIZATION EXTERNAL (TYPE ORACLE_LOADER … )
                           INMEMORY MEMCOMPRESS FOR CAPACITY HIGH;
```

- Enables populating data from external tables into the in-memory column store

```
SQL> EXEC DBMS_INMEMORY.POPULATE ('HR', 'test')
```

- Allows advanced analytics on external data with Database In-Memory
- Allows advanced analytics on a much larger data domain than just what resides in Oracle databases

```
SQL> ALTER SESSION SET query_rewrite_integrity=stale_tolerated;
SQL> SELECT * FROM test;
```

```
DBA_EXTERNAL_TABLES
    INMEMORY = ENABLED | DISABLED
    INMEMORY_COMPRESSION =
                FOR QUERY LOW | HIGH
                FOR CAPACITY LOW | HIGH
```

ORACLE

Oracle Database 18c enables the population of data from external tables into the in-memory column store.

This allows population of data that is not stored in Oracle Database. This can be valuable when you have other external data stores and you want to perform advanced analytics on that data with Database In-Memory. This can be particularly valuable when the external data needs to be queried repeatedly. You can avoid multiple accesses of the external storage, and the queries can use the features of Database In-Memory multiple times.

Data from external sources with `ORACLE_LOADER` and `ORACLE_DATAPUMP` access types can be summarized and populated into the in-memory column store where repeated, ad hoc analytic queries can be run that might be too expensive to run on the source data.

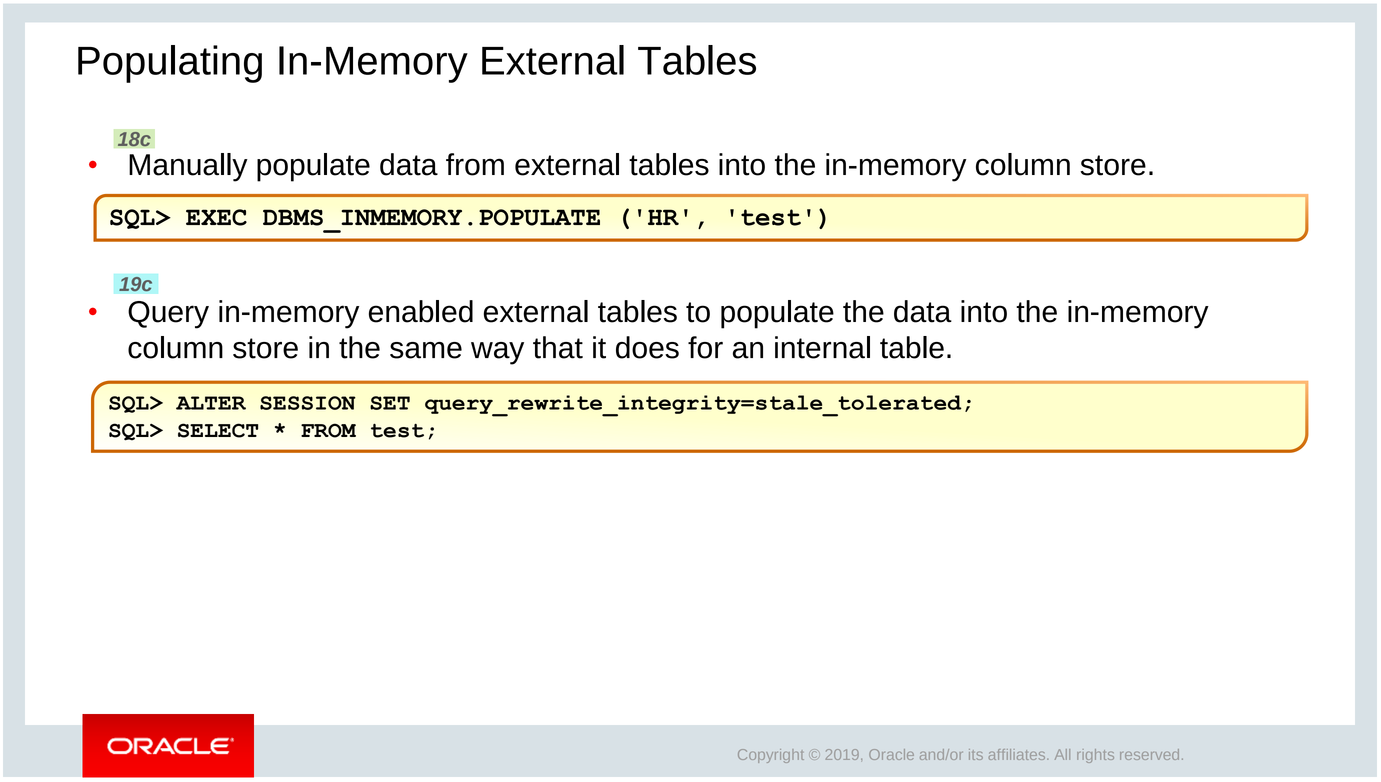
The in-memory external tables also benefit from in-memory expressions.

You can set the `INMEMORY` attribute and its correlated `MEMCOMPRESS` attribute when creating and altering an external table. If the external table is partitioned, all individual partitions are defined as in-memory segments. The ability to exclude certain columns is not yet implemented.

Querying an in-memory external table requires the `QUERY_REWRITE_INTEGRITY` parameter in the session to be set as `STALE_TOLERATED`. If updates of the external file occur, either repopulation of the in-memory segment with the `DBMS_INMEMORY.REPOPULATE` procedure or altering the in-memory table as `NO INMEMORY` and resetting it as `MEMORY` is required.

Statistics are collected on in-memory external tables because they are on in-memory heap tables like `IM populate external table read time (ms).`

If Oracle Database 18c allows heat maps and makes Automatic Data Optimization (ADO) manage eviction of cooling segments from the in-memory area, this is not the case on external tables and external table partitions.



Populating In-Memory External Tables

18c
- Manually populate data from external tables into the in-memory column store.

```
SQL> EXEC DBMS_INMEMORY.POPULATE ('HR', 'test')
```

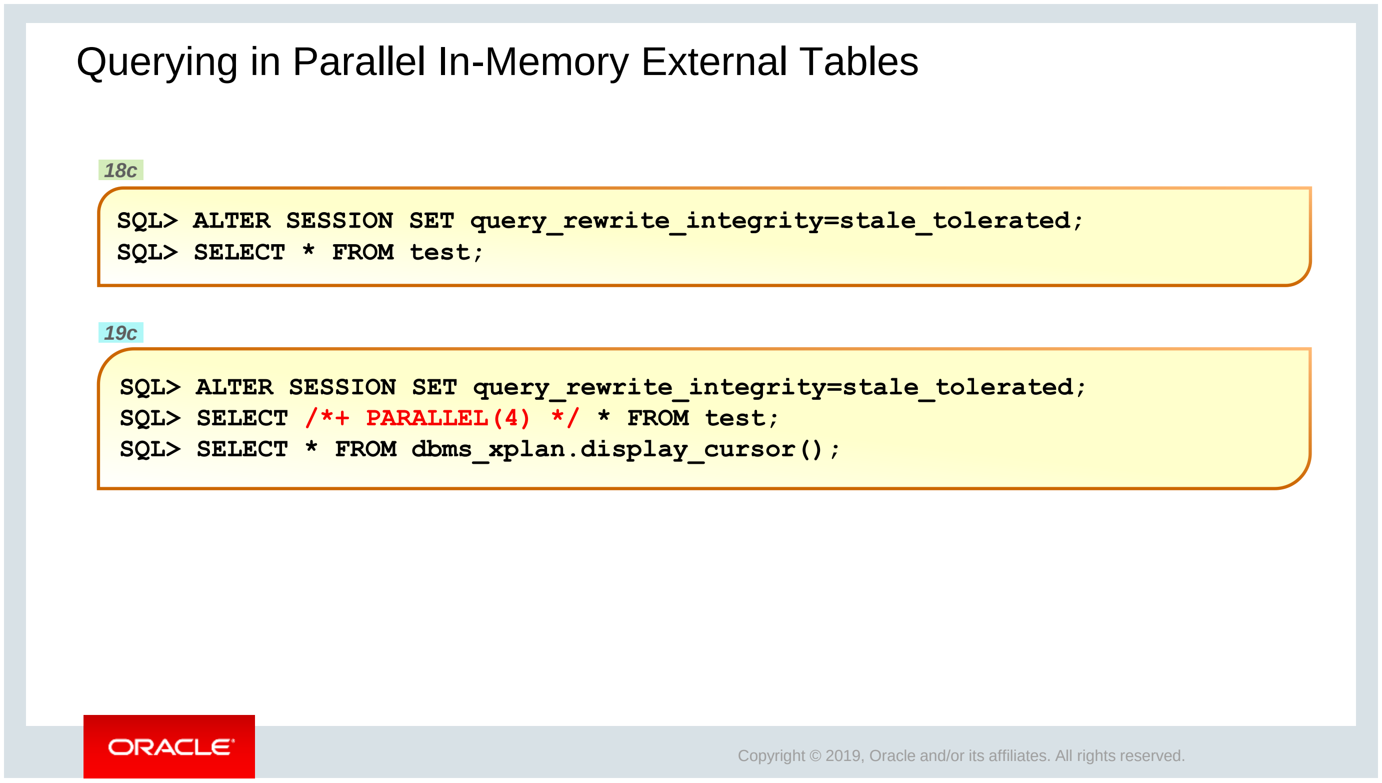
19c
- Query in-memory enabled external tables to populate the data into the in-memory column store in the same way that it does for an internal table.

```
SQL> ALTER SESSION SET query_rewrite_integrity=stale_tolerated;
SQL> SELECT * FROM test;
```

Oracle Database 18c enables the population of data from external tables into the in-memory column store. The population into the In-Memory column store has to be completed manually by executing the `DBMS_INMEMORY.POPULATE` procedure.

In Oracle Database 19c, querying an in-memory enabled external table initiates the population into the In-Memory column store in the same way that it does for an internal table.

You can view the free video about this feature in the course under the module name “On - Premises Videos”.



Querying in Parallel In-Memory External Tables

18c

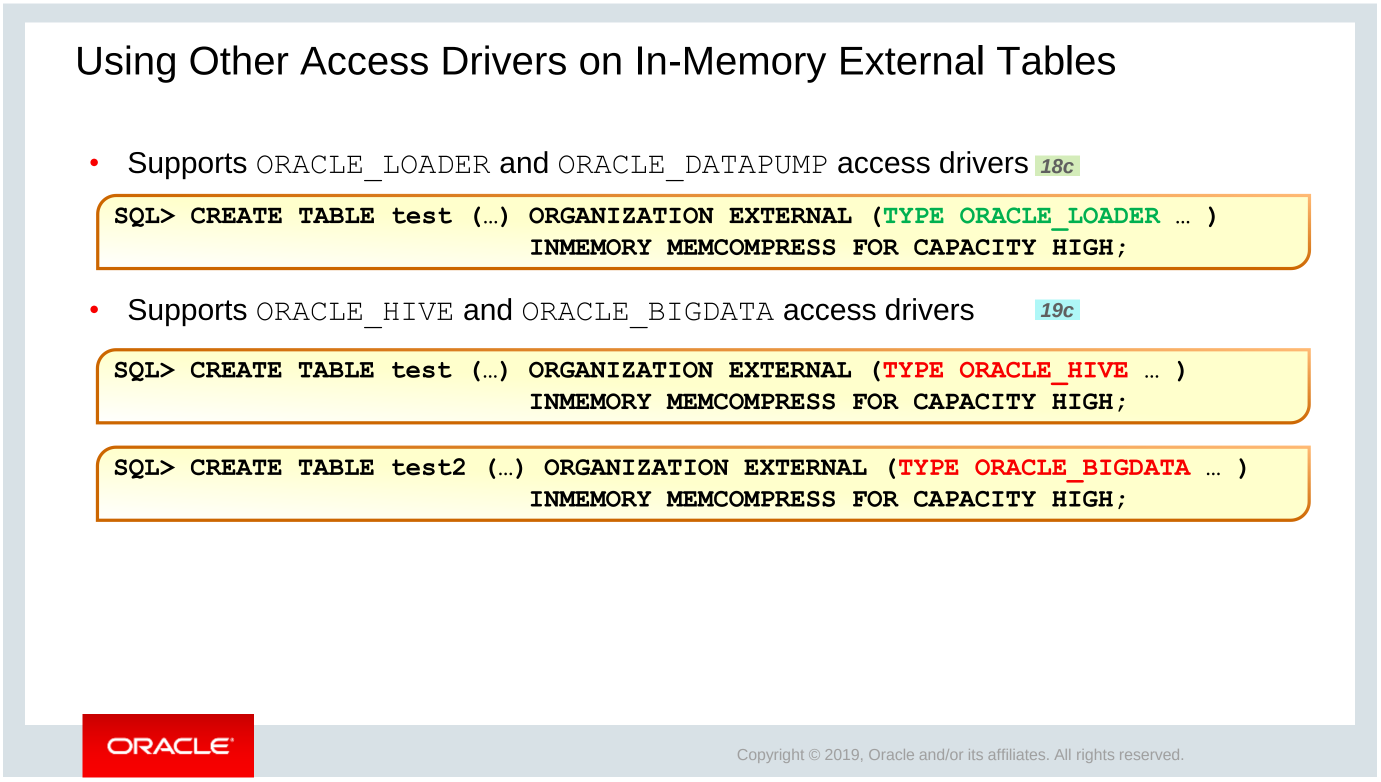
```
SQL> ALTER SESSION SET query_rewrite_integrity=stale_tolerated;
SQL> SELECT * FROM test;
```

19c

```
SQL> ALTER SESSION SET query_rewrite_integrity=stale_tolerated;
SQL> SELECT /*+ PARALLEL(4) */ * FROM test;
SQL> SELECT * FROM dbms_xplan.display_cursor();
```

Oracle Database 18c enables the population of data from external tables into the in-memory column store. Nevertheless, in-memory external tables cannot be accessed by parallel execution.

Oracle Database 19c lifts the serial query–only restriction and normal parallel query is fully supported on in-memory external tables.



Using Other Access Drivers on In-Memory External Tables

- Supports `ORACLE_LOADER` and `ORACLE_DATAPUMP` access drivers 18c

```
SQL> CREATE TABLE test (…) ORGANIZATION EXTERNAL (TYPE ORACLE_LOADER … )
                           INMEMORY MEMCOMPRESS FOR CAPACITY HIGH;
```

- Supports `ORACLE_HIVE` and `ORACLE_BIGDATA` access drivers 19c

```
SQL> CREATE TABLE test (…) ORGANIZATION EXTERNAL (TYPE ORACLE_HIVE … )
                           INMEMORY MEMCOMPRESS FOR CAPACITY HIGH;
```

```
SQL> CREATE TABLE test2 (…) ORGANIZATION EXTERNAL (TYPE ORACLE_BIGDATA … )
                           INMEMORY MEMCOMPRESS FOR CAPACITY HIGH;
```

ORACLE

In Oracle Database 18c, only the `ORACLE_LOADER` and `ORACLE_DATAPUMP` access drivers are supported for in-memory external tables.

Oracle Database 19c lifts the restriction to allow `ORACLE_HIVE` and `ORACLE_BIGDATA` access drivers.

Summary

In this lesson, you should have learned how to:

- Configure and use the automatic indexing task
- Configure and use hybrid partitioned tables with internal and external partitions
- Populate in-memory external tables
- Execute a parallel query on in-memory external tables
- Use the `ORACLE_HIVE` and `ORACLE_BIGDATA` driver types for in-memory external tables

ORACLE

Practices Environment - 1

Each student has one virtual machine (VM1) to use during class.

- Oracle Database 19c installed
- One CDB named **`ORCL`** and its PDB named **`PDB1`**

Practices Environment - 2

Pre-created databases and instances with their respective PDBs:

- Data files in `/u02/app/oracle/oradata/`
- CDB root data files in `/u02/app/oracle/oradata/ORCL`
- PDB data files in `/u02/app/oracle/oradata/ORCL/PDB1`
- All backup files in `/u03/app/oracle/fast_recovery_area/ORCL`
- Password and init files in `$ORACLE_HOME/dbs`
- Diagnostics files in `/u01/app/oracle/diag/rdbms/orcl/ORCL/…`
- TDE wallet in `/u02/app/oracle/admin/ORCL/tde_wallet`
- Net files in `$ORACLE_HOME/network/admin`

The configuration of the environment used for the practices of the course matches the configuration used on the Oracle Cloud Virtual Machine for the pre-created `ORCL` database of Oracle Database Cloud Service. Oracle Database Cloud Service gives you the ability to deploy Oracle databases in the cloud, with each database deployment containing a single Oracle CDB.

This is a good way to get familiar with the Oracle Database Cloud environment that is preconfigured for cloud customers.

Practice 5: Overview

- 5-1: Managing Hybrid Partitioned Tables
- 5-2: Creating and Populating In-Memory Hybrid Partitioned Tables
- 5-3: Creating and Populating In-Memory External Tables

6

Using Diagnosability Enhancements

Overview

- This module focuses on the diagnosability enhancements of Oracle Database 19c and Trace File Analyzer 18c (18.3.0).
- It complements the diagnosability topics covered in the 5-day *Oracle Database 12c R2: New Features for Administrators Part 1 Ed 1* course.
- Previous experience with Oracle Database 12*c* and Oracle Database 18c is required for a full understanding of diagnosability enhancements.

Refer to *Oracle Database New Features Guide 19c, Oracle Database PL/SQL Packages and Types Reference 19c, Oracle Database SQL Tuning Guide 19c, Oracle Trace File Analyzer User's Guide 18.3.0, and Autonomous Health Framework User's Guide 19c.*

This module follows the 5-day *Oracle Database 12c Release 2 (12.2)* course Part 2. The course is designed to introduce the new features and enhancements of Oracle Database 12*c* Release 2 (12.2.0.1) that are applicable to work that is usually performed by database administrators and related personnel.

The module is designed to introduce the major new enhancements of Oracle Database 19c related to diagnosability as well as the enhancements of Trace File Analyzer.

The module consists of an instructor-led lesson and practices that enable you to see how certain new functionalities behave.

Objectives

After completing this lesson, you should be able to:

- Use automated SQL diagnosibility to diagnose and repair SQL statements
- Benefit from the new service request data collections
- Upload data collections directly to your service request
- Redact data in data collections to upload

End-to-End Automated Service for Incidents

19c

Fault prediction, detection, and fixation:

- Avoid or minimize human intervention.
- Provide a quick turnaround for any issue.

➔ Incident creation

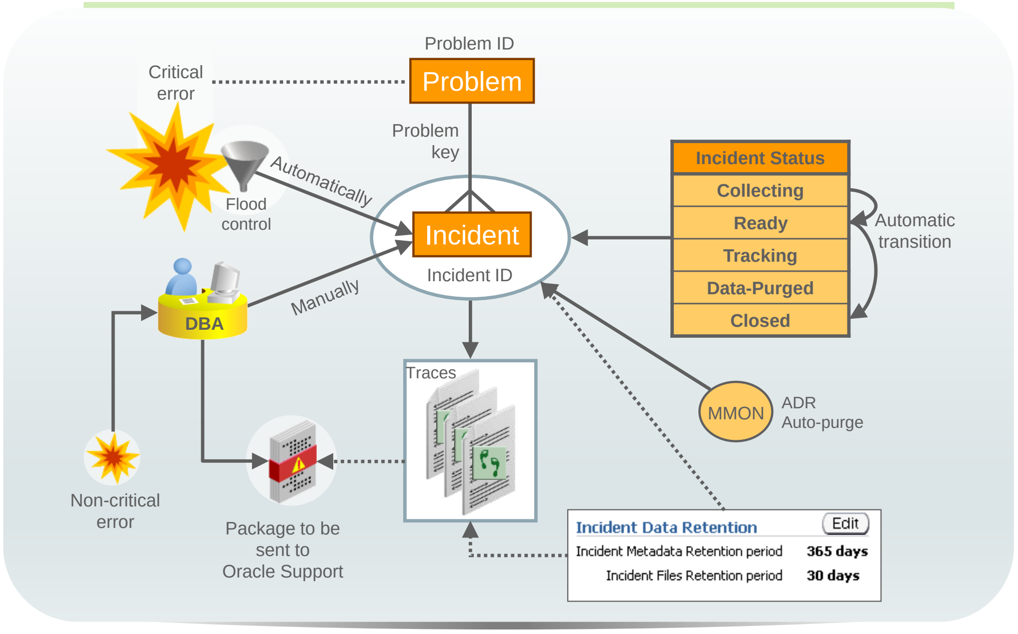
➔ SQL diagnosibility and repair

In Oracle Database 18c, the Automatic Diagnostic Repository Command Interpreter (ADRCI) utility is a command-line tool that you use to manage Oracle Database diagnostic data. The SQL Test Case Builder and SQL Diagnostic Advisor are available for improving SQL diagnosibility.

Oracle Database 19c provides a full set of end-to-end automated service for fault prediction, detection, and fixation to avoid or minimize human intervention and provide a quick turnaround for any issue. SQL diagnosibility is one of the areas that requires automation in this direction. The capability is required for the fully-managed Autonomous Data Warehouse Cloud (ADWC) service.

Automatic Diagnosability Repository Incidents

18c



Incident Data Retention	
Incident Metadata Retention period	365 days
Incident Files Retention period	30 days

ORACLE

Since Oracle Database 11*g*, an always-on, in-memory tracing facility enables database components to capture diagnostic data upon first failure for critical errors such as ORA-07445 (Operating System exception), ORA-00600 (internal errors), ORA-4020 (Deadlock on library object), ORA-8103 (Object no longer exists), ORA-1410 (Invalid ROWID), ORA-1578 (Data block corrupted), ORA-29740 (Node eviction), ORA-255 (Database is not mounted), ORA-376 (File cannot be read at this time), ORA-4030 (Out of process memory), ORA-4031 (Unable to allocate more bytes of shared memory), ORA-355 (The change numbers are out of order), ORA-356 (Inconsistent lengths in the change description), and ORA-353 (Log corruption). A special repository, called Automatic Diagnostic Repository (ADR), is automatically maintained to hold diagnostic information about critical error events. This information can be used to create incident packages to be sent to Oracle Support Services for investigation.

ADRCI is a command-line tool that is part of the fault diagnosability infrastructure introduced in Oracle Database Release 11g. ADRCI enables you to:

- View diagnostic data within Automatic Diagnostic Repository (ADR).
- Package incident and problem information into a zip file for transmission to Oracle Support. This is done using a service called Incident Package Service (IPS).

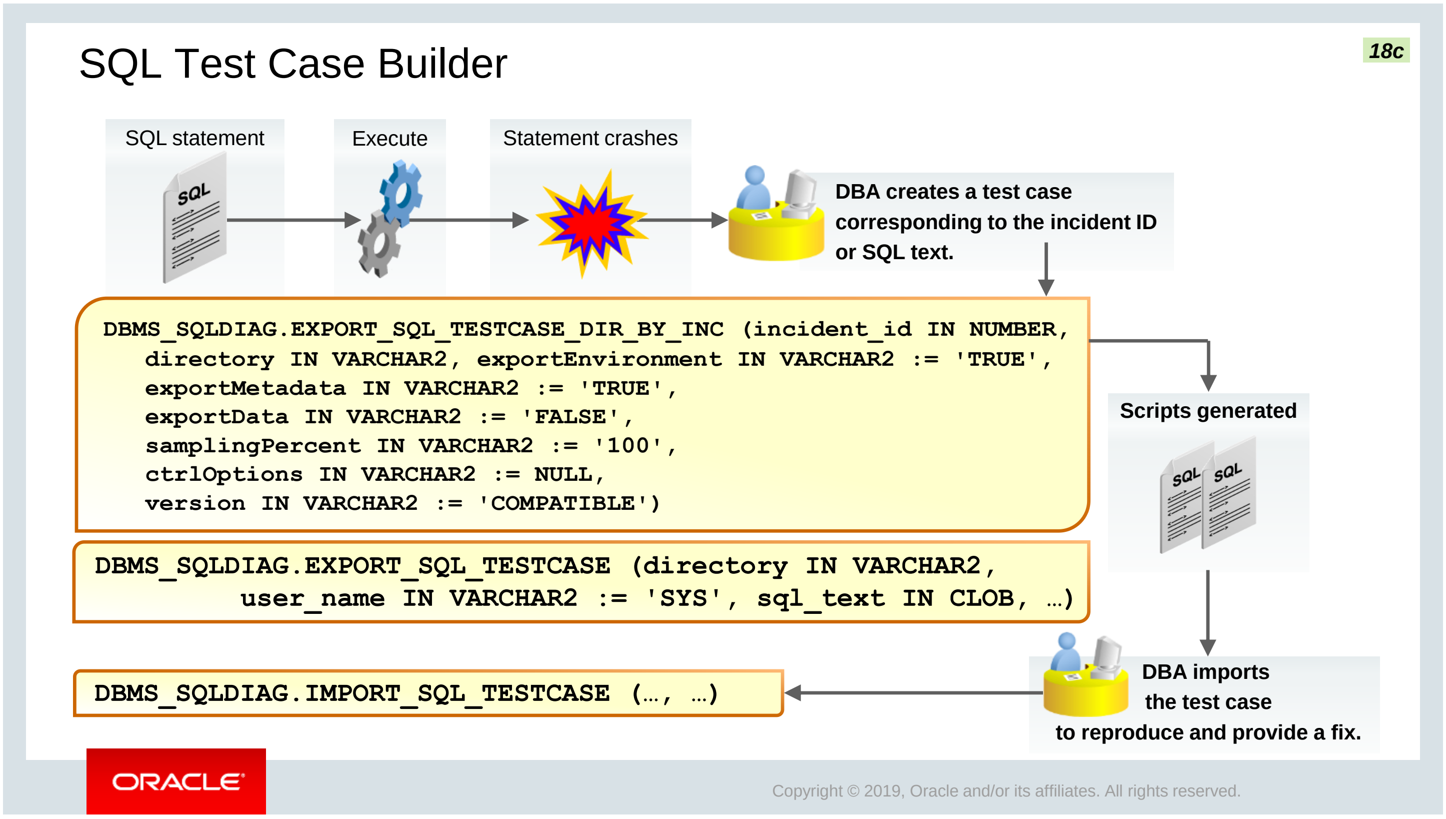
For each incident, there are two alert files generated. One is textual and the other is an XML file. You can view the alert log in text format with Enterprise Manager and the ADRCI utility.

The INCIDENT directory contains multiple subdirectories, where each subdirectory is named for a particular incident, and where each contains dumps pertaining only to that incident.

The HM directory contains the checker run reports generated by the Health Monitor.

There is also a METADATA directory that contains important files for the repository itself. You can compare this to a database dictionary. This dictionary can be queried using ADRCI.

ADR Command Interpreter (ADRCI) is a utility that you can use to perform all the tasks permitted by the Support Workbench, but in a command-line environment. The ADRCI utility also enables you to view the names of the trace files in ADR, and to view the alert log with XML tags stripped, with and without content filtering.



SQL Test Case Builder

18c
SQL statement
Execute
Statement crashes
DBA creates a test case corresponding to the incident ID or SQL text.
Scripts generated
DBA imports the test case to reproduce and provide a fix.

```
DBMS_SQLDIAG.EXPORT_SQL_TESTCASE_DIR_BY_INC (incident_id IN NUMBER,
   directory IN VARCHAR2, exportEnvironment IN VARCHAR2 := 'TRUE',
   exportMetadata IN VARCHAR2 := 'TRUE',
   exportData IN VARCHAR2 := 'FALSE',
   samplingPercent IN VARCHAR2 := '100',
   ctrlOptions IN VARCHAR2 := NULL,
   version IN VARCHAR2 := 'COMPATIBLE')
```

```
DBMS_SQLDIAG.EXPORT_SQL_TESTCASE (directory IN VARCHAR2,
          user_name IN VARCHAR2 := 'SYS', sql_text IN CLOB, …)
```

```
DBMS_SQLDIAG.IMPORT_SQL_TESTCASE (…, …)
```

For most SQL components, obtaining a reproducible test case is the single most important factor in bug resolution speed. This is also the longest and most painful step for the customer. The goal of the SQL test case builder (TCB) is to gather as much information related to an SQL incident as possible, and package it in a way that allows a developer at Oracle to reproduce the problem.

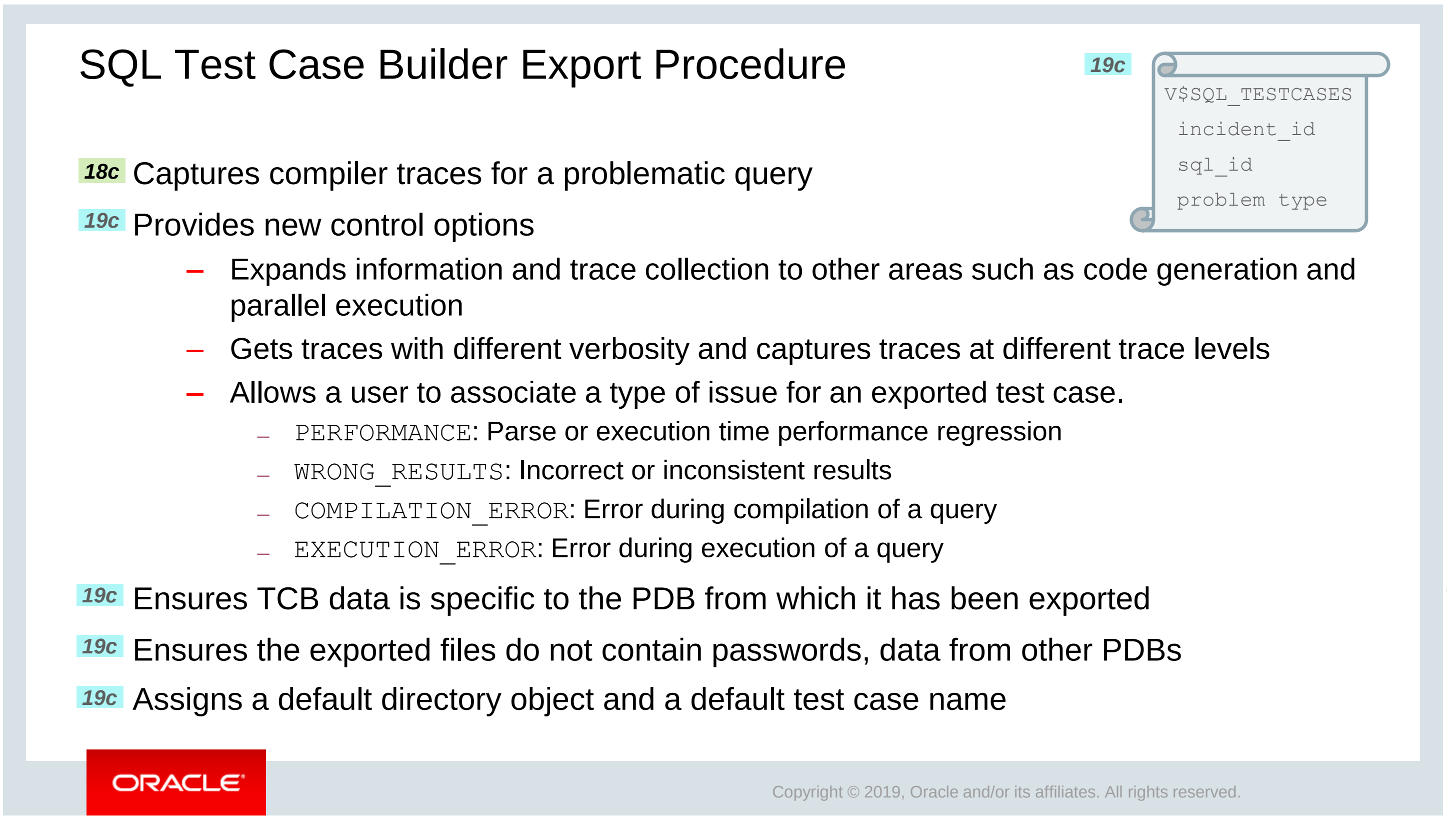
Building a test case at the customer site helps support or developers reproduce the problem on a different system. The SQL TCB gathers all the information needed to reproduce the problem in another Oracle instance. The scripts generated contain the commands required to re-create all the necessary objects and the environment. After the test case is ready, you can create a ZIP file of the directory and move it to another database, or upload the file to Oracle Support or your developers. Oracle Support or the developers will import the test case to reproduce the issue and provide a fix for the failing SQL.

You can invoke the SQL TCB by using:

- `DBMS_SQLDIAG.EXPORT_SQL_TESTCASE_DIR_BY_INC`: The function generates scripts for the SQL test case corresponding to the incident ID passed as an argument.
- `DBMS_SQLDIAG.EXPORT_SQL_TESTCASE_DIR_BY_TXT`: This function generates scripts for the SQL test case corresponding to the SQL passed as an argument.
- `DBMS_SQLDIAG.EXPORT_SQL_TESTCASE`: This procedure exports a SQL test case to a directory.

For all of them, `ctrlOptions` provides opaque control parameters.

- `CAPTURE`: This parameter defines the mode of TCB capture. `BASIC` captures all the information in that release as well as AWR reports, and SQL monitors reports and parameter information. `WITH_RUNTIME_INFO` captures runtime information for the SQL, such as dynamic sampling data, list of binds, and Dynamic Plan info, along with information captured under `BASIC` mode.
- `NAME=mexec_count`: Value is any positive number (N). This parameter tells TCB to execute the statement for N time and capture runtime info at the end of each execution. To execute three times, set `ctrlOptions` with the following string: `'<parameter NAME="mexec_count">3</parameter>'`.
- `NAME=stat_history_since`: Value is date. The object statistics history is exported using this parameter. Statistics history after date specified are exported.



SQL Test Case Builder Export Procedure

19c

```
V$SQL_TESTCASES
 incident_id
 sql_id
 problem type
```

18c Captures compiler traces for a problematic query

19c Provides new control options

- Expands information and trace collection to other areas such as code generation and parallel execution
- Gets traces with different verbosity and captures traces at different trace levels
- Allows a user to associate a type of issue for an exported test case.
  - `PERFORMANCE`: Parse or execution time performance regression
  - `WRONG_RESULTS`: Incorrect or inconsistent results
  - `COMPILATION_ERROR`: Error during compilation of a query
  - `EXECUTION_ERROR`: Error during execution of a query

19c Ensures TCB data is specific to the PDB from which it has been exported

19c Ensures the exported files do not contain passwords, data from other PDBs

19c Assigns a default directory object and a default test case name

ORACLE

In Oracle Database 18c, TCB captures only compiler traces for a problematic query.

Oracle Database 19c expands the information and trace collection to other areas such as code generation and parallel execution, and enables the user to get traces with different verbosity by allowing them to capture traces at different trace levels.

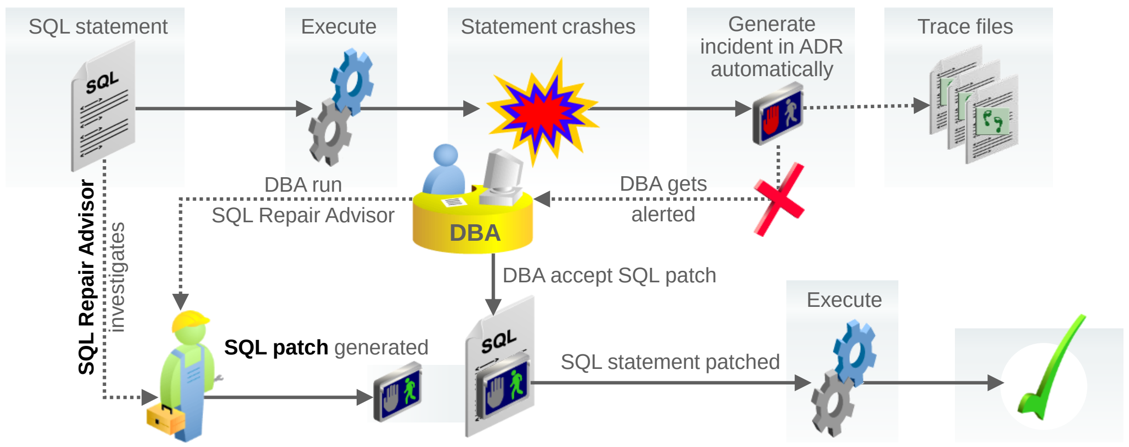
The directory argument and test case name are optional. The default test case name is in the `oratcb_`*`con_id_sql_id_seq_num_session_id`* format.

The `ctrlOptions` parameter provides new opaque control parameters.

- `COMPRESS`: Can be used to compress the exported test case files into a zip file. Example: `<parameter name="compress">yes</parameter>`
- `DIAG_EVENT`: Component traces are a crucial piece of information in diagnosing the problem. The parameter improves the quality and amount of diagnostic tracing.
  Example: `<parameter name="diag_event">SQLEXEC_MEDIUM</parameter>`
  These are the possible values and their internal action:
  - ‘`ADS`‘: “trace [ADS] disk=medium”
  - ‘`COMPILER`‘: “trace [sql_compiler.*] disk=highest”
  - ‘`SQLEXEC_LOW`’ or ‘`SQLEXEC_MEDIUM`’ or ‘`SQLEXEC_HIGH`’ or ‘`SQLEXEC_HIGHEST`‘: “trace [sql_execution.*] disk=low” or disk=medium or disk=high or disk=highest
- `PROBLEM_TYPE`: Allows a user to associate a type of issue for an exported test case. The possible values for this option are:
  - `PERFORMANCE`: Parse/Execution time performance regression
  - `WRONG_RESULTS`: Incorrect or inconsistent results
  - `COMPILATION_ERROR`: Error during compilation of a query
  - `EXECUTION_ERROR`: Error during execution of a query

SQL Diagnostic Advisor

18c



1. The DBA creates a diagnostic task: `dbms_sqldiag.create_diagnosis_task`
2. The DBA sets diagnostic task preferences: `dbms_sqltune.set_tuning_task_parameter`
3. The DBA executes the diagnostic task: `dbms_sqldiag.execute_diagnosis_task`
4. The DBA accepts the SQL Patch: `dbms_sqldiag.accept_sql_patch`

You run SQL Repair Advisor after a SQL statement fails with a critical error that generates a problem in ADR. The advisor analyzes the statement and in many cases recommends a patch to repair the statement.

1. You first create a diagnostic task for a SQL using SQL text or SQL ID.
2. You set the parameter values for the diagnostic task to run.
3. You request the task execution to get the recommendations and possible solutions. SQL Repair Advisor comes up with the SQL profile and or SQL patch to fix the problem or provide a workaround for the problem.
4. You manually verify the recommendations and accept the SQL profile generated by the diagnostic task. If you implement the recommendation, the applied SQL patch circumvents the failure by causing the query optimizer to choose an alternate execution plan for future executions. This is done without changing the SQL statement itself.

Note: In case no workaround is found by SQL Repair Advisor, you are still able to package the incident files and send the corresponding diagnostic data to Oracle Support.

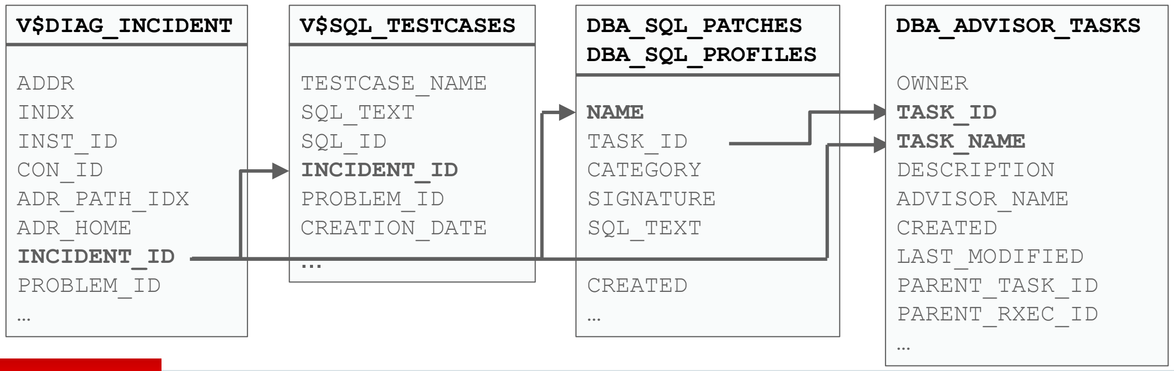
SQL Diagnostic and Repair

19c

The components store their metadata in different tables and in different forms.

- DDE Incident ➔ `V$DIAG_*` views
- SQL Test Case Builder ➔ `V$SQL_TESTCASES`
- SQL Diagnosis and Repair ➔ `DBA_SQL_PROFILES` and `DBA_SQL_PATCHES`
- SQL Advisor ➔ `DBA_ADVISOR_TASKS`

Correlation key ID



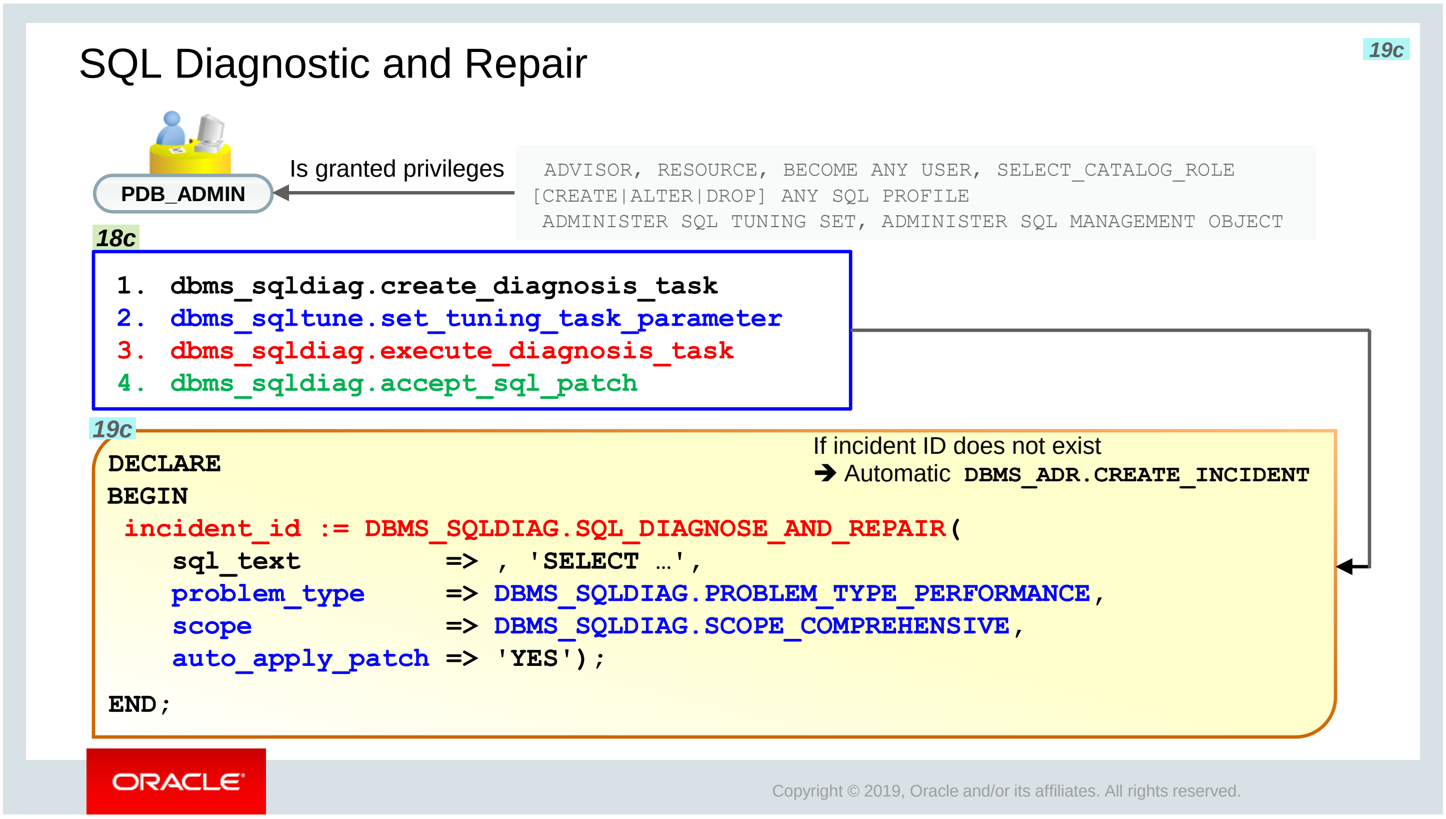
In Oracle Database 19c, for an end-to-end SQL diagnosability solution, all the problems need to be tracked with incident creation. When a user reports any soft errors, such as compilation or runtime errors, wrong results, or a performance problem, the user can create an incident and track it by using the incident ID.

The SQL Diagnostic and Repair (SDR) functionality is based on the fact that an overwhelming majority of problems of the SQL layer can be avoided by following a different code path in the SQL engine the next time the server processes the SQL. The optimizer presents unique characteristics from the diagnosability point of view. The goal of the optimizer is to explore alternative ways of executing the same statement. This puts it in a unique position to avoid generating a broken plan.

All the four components, DDE, TCB, SDR, and Advisor, store their metadata in different tables and in different forms.

- Incident metadata is stored in `V$DIAG_*` views with incident ID as a key.
- TCB stores metadata about a test case for a problem in the `V$SQL_TESTCASES` view with incident ID, SQL ID and problem type as key.
- SDR stores the SQL patch information in `DBA_SQL_PATCHES` and `DBA_SQL_PROFILES` views with task_id and task_exec_name as keys.
- Metadata for SQL Advisor can be found in the `DBA_ADVISOR_TASKS` view with task_name and task_id as key.

A correlation key ID links the four components for a single problem.



SQL Diagnostic and Repair

19c
PDB_ADMIN
Is granted privileges
18c
19c

```
ADVISOR, RESOURCE, BECOME ANY USER, SELECT_CATALOG_ROLE
[CREATE|ALTER|DROP] ANY SQL PROFILE
ADMINISTER SQL TUNING SET, ADMINISTER SQL MANAGEMENT OBJECT
```

```
1. dbms_sqldiag.create_diagnosis_task
2. dbms_sqltune.set_tuning_task_parameter
3. dbms_sqldiag.execute_diagnosis_task
4. dbms_sqldiag.accept_sql_patch
```

If incident ID does not exist
➔ Automatic `DBMS_ADR.CREATE_INCIDENT`

```
DECLARE
BEGIN
 incident_id := DBMS_SQLDIAG.SQL_DIAGNOSE_AND_REPAIR(
     sql_text         => , 'SELECT …',
     problem_type     => DBMS_SQLDIAG.PROBLEM_TYPE_PERFORMANCE,
     scope            => DBMS_SQLDIAG.SCOPE_COMPREHENSIVE,
     auto_apply_patch => 'YES');
END;
```

To consolidate the steps in the slide and automate the problem diagnosis and workaround implementation, there is a new `SQL_DIAGNOSE_AND_REPAIR` function that automatically completes the following actions:

1. Creates an incident if the incident ID does not already exist, populates incident metadata with required information, such as `SQL_ID`, `SQL_TEXT`, `INCIDENT_ID`, compilation env, and so on.
2. Creates a diagnostic task
3. Executes the diagnostic task
4. Implements the SQL patch and recommendations for a given SQL statement if the validation from the `DBA_ADVISOR_RECOMMENDATIONS` view for the generated SQL patch outputs a `SAFE` status. The SDR works in a conservative mode. The fix or workaround for performance issues needs to be verified by the user before accepting if the status is not `SAFE`.

```
 FUNCTION sql_diagnose_and_repair(
     sql_text           IN    CLOB,
     bind_list          IN    sql_binds := NULL,
     scope              IN    VARCHAR2  := SCOPE_COMPREHENSIVE,
     time_limit         IN    NUMBER    := TIME_LIMIT_DEFAULT,
     problem_type       IN    NUMBER    := PROBLEM_TYPE_PERFORMANCE,
     auto_apply_patch   IN    VARCHAR2  := YES)
   RETURN NUMBER;
```

When the problem type is `PROBLEM_TYPE_PERFORMANCE`, SDR tries to first find an alternate plan from the plan history, such as the cursor cache, from AWR, or from SQL Tuning Set. If an alternate plan is not found, then it invokes SQL Tuning to tune the given SQL. The SQL tune is invoked only when the scope is specified as `COMPREHENSIVE` with `PROBLEM_TYPE_PERFORMANCE`.

The problem type can be set to the following values:

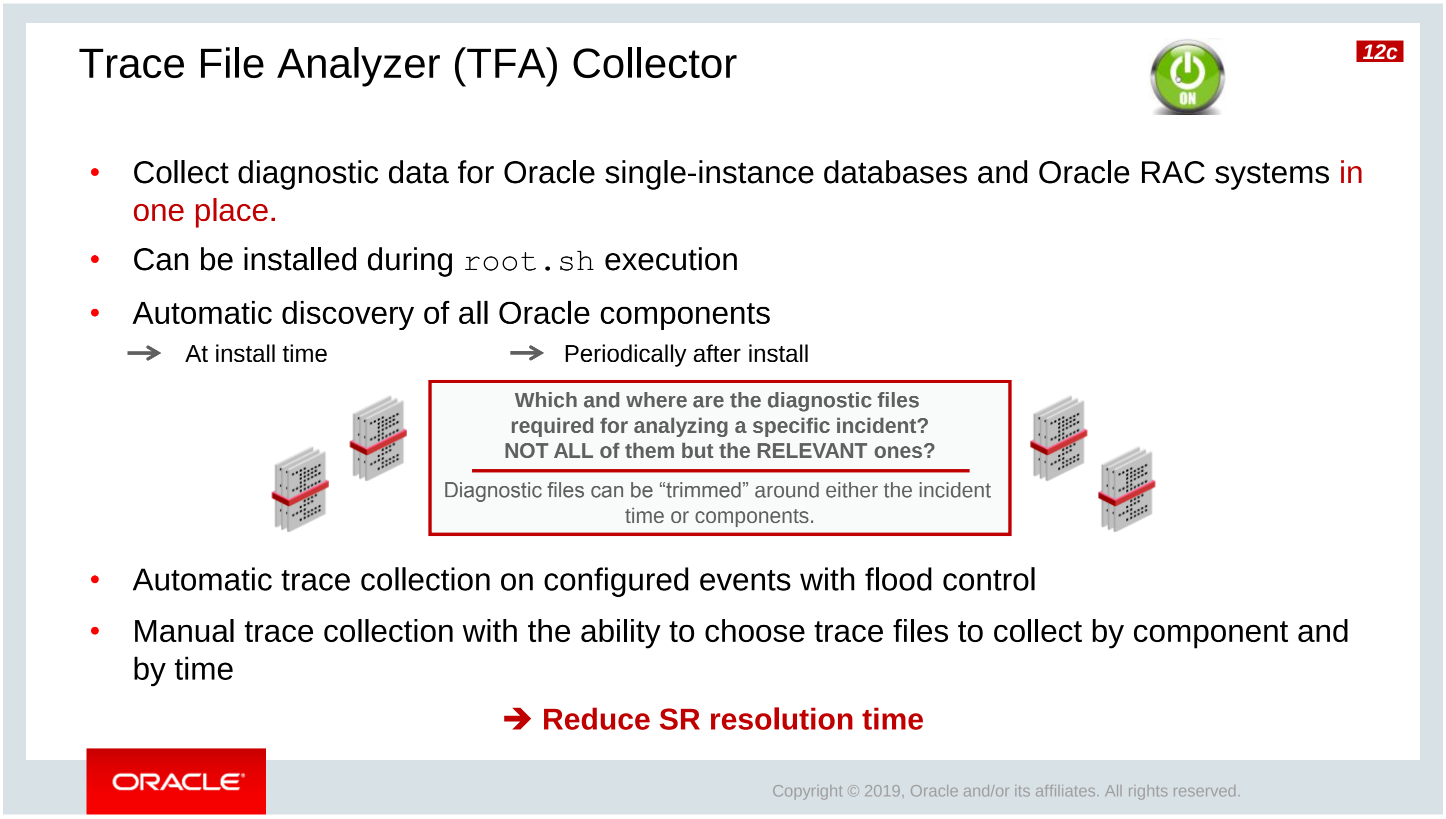
- `PROBLEM_TYPE_WRONG_RESULTS`: Diagnoses incorrect results
- `PROBLEM_TYPE_COMPILATION_ERROR`: Diagnoses the crash during compilation of the statement
- `PROBLEM_TYPE_EXECUTION_ERROR`: Diagnoses the crash during execution of the statement

This is the `SQL_DIAGNOSE_AND_REPAIR` function with the `SQL_ID` input argument:

```
FUNCTION SQL_diagnose_and_repair(
    sql_id            IN   VARCHAR2,
    plan_hash_value   IN   NUMBER   := NULL,
    scope             IN   VARCHAR2 := SCOPE_COMPREHENSIVE,
    time_limit        IN   NUMBER   := TIME_LIMIT_DEFAULT,
    problem_type      IN   NUMBER   := PROBLEM_TYPE_PERFORMANCE,
    auto_apply_patch  IN   VARCHAR2 := YES)
  RETURN NUMBER;
```

This is the `SQL_DIAGNOSE_AND_REPAIR` function with the `INCIDENT_ID` input argument:

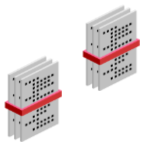
```
FUNCTION SQL_diagnose_and_repair(
    incident_id       IN   VARCHAR2,
    scope             IN   VARCHAR2 := SCOPE_COMPREHENSIVE,
    time_limit        IN   NUMBER   := TIME_LIMIT_DEFAULT,
    problem_type      IN   NUMBER   := PROBLEM_TYPE_PERFORMANCE,
    auto_apply_patch  IN   VARCHAR2 := YES)
  RETURN NUMBER;
```



Trace File Analyzer (TFA) Collector

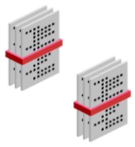
12c

- Collect diagnostic data for Oracle single-instance databases and Oracle RAC systems in one place.
- Can be installed during `root.sh` execution
- Automatic discovery of all Oracle components
  - → At install time
  - → Periodically after install



Which and where are the diagnostic files required for analyzing a specific incident? NOT ALL of them but the RELEVANT ones?

Diagnostic files can be "trimmed" around either the incident time or components.



- Automatic trace collection on configured events with flood control
- Manual trace collection with the ability to choose trace files to collect by component and by time

➔ Reduce SR resolution time

ORACLE

The Trace File Analyzer (TFA) utility has become the standard for collecting diagnostic information in the Oracle Database, including the Oracle Database single-instance and Oracle Real Application Clusters (Oracle RAC).
TFA collector is a tool for targeted trace file collection from live systems.

At install time, TFA collector automatically discovers all Oracle components. Even after installation, it still discovers new Oracle components installed. Thanks to xml files, TFA is agnostic of version/product.

The TFA collection works under two modes:

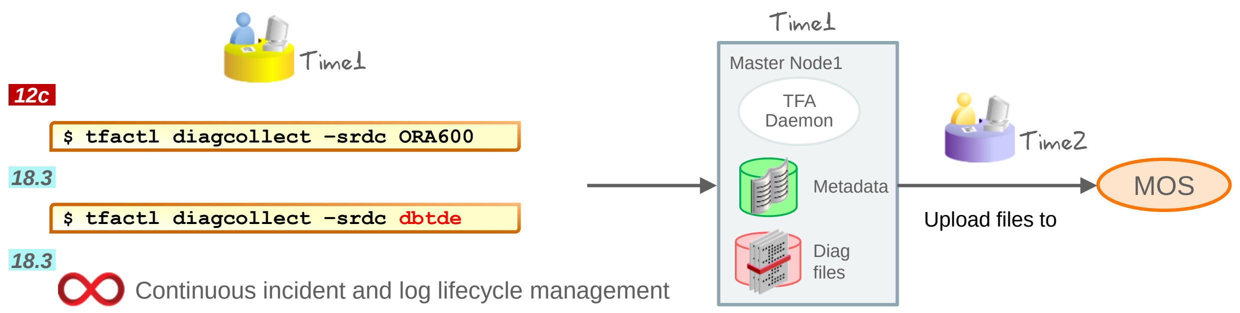
- Automatic collection:
  - On events like `ORA-600`, `ORA-7445`, `ORA-4031`, node evictions and instance termination, and other configured events
  - Periodically checking trace directories, by component (OS, ASM, DB)
- Manual collection with the ability to choose the trace files to collect

To get more information about TFA Collector, refer to the My Oracle Support note: *TFA Collector - Tool for Enhanced Diagnostic Gathering (Doc ID 1513912.1)*

When installing the Oracle Database 12.2.0.1, OUI asks you to run the `root.sh` shell script.

```
# /u01/app/oracle/product/12.2.0/dbhome_1/root.sh

Performing root user operation.

The following environment variables are set as:

ORACLE_OWNER= oracle
ORACLE_HOME=  /u01/app/oracle/product/12.2.0/dbhome_1

Enter the full pathname of the local bin directory: [/usr/local/bin]:

Copying dbhome to /usr/local/bin ...
Copying oraenv to /usr/local/bin ...
Copying coraenv to /usr/local/bin ...

Creating /etc/oratab file...
Entries will be added to the /etc/oratab file as needed by
Database Configuration Assistant when a database is created
Finished running generic part of root script.

Now product-specific root actions will be performed.

Do you want to setup Oracle Trace File Analyzer (TFA) now ? yes|[no] : yes

Installing Oracle Trace File Analyzer (TFA).

Log File: /u01/app/oracle/product/12.2.0/dbhome_1/install/root_host01_2016-08-
03_07-41-46-138282006.log

Finished installing Oracle Trace File Analyzer (TFA)
#
```

TFA Collector Process



12c

```
$ tfactl diagcollect -srdc ORA600
```

18.3

```
$ tfactl diagcollect -srdc dbtde
```

18.3 Continuous incident and log lifecycle management

12c	18.3	18.3
Errors automatically detected:	*Errors automatically detected:*	*SRDCs detected:*
• `ORA-00600`	• `ORA-00020, ORA-00060, ORA-00700`	`dbimpdpperf, dbtde,`
• `ORA-07445`	• `ORA-01031, ORA-01555, ORA-01578`	`dbsqlperf, emagentperf,`
• `ORA-04031, ORA-4030`	• `ORA-08102, ORA-08103, ORA-01628`	`emomscrash, emomsheap,`
• Instance termination	• `ORA-27300, ORA-27301, ORA-27302`	`emomshungcpu`
	• `ORA-29548, ORA-30036`	

In Oracle Database 12*c*, TFA collection is an automated process by default, periodically detecting and collecting data for predefined events like `ORA-600`, `ORA-7445`, `ORA-4031`, node evictions, and instance termination, occurring on single-instance databases (and on the all nodes of a cluster of RAC databases).

You can configure other events for which you want data to be automatically collected. You can also manually launch a Service Request Data Collection (SRDC) for a specific event.

Trace File Analyzer 18.3 includes new SRDCs like:

- `dbimpdpperf` and `dbsqlperf` for Data Pump import and SQL performance problems
- `dbtde` for Transparent Data Encryption (TDE) problems
- `emagentperf` for Enterprise Manager Agent performance problems
- `emomscrash` for Enterprise Manager crash problems
- `emomheap` for Enterprise Manager java heap usage or performance problems
- `emomhungcpu` for Enterprise Manager OMS crash, restart, or performance problems

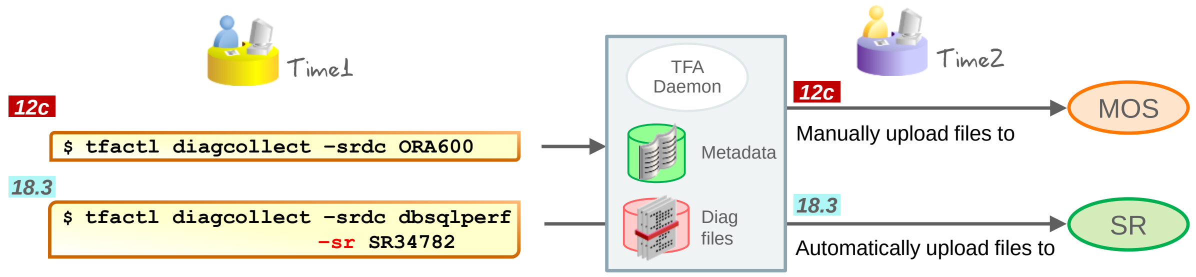
Oracle Database 18.3 automatically collects 40 top event errors in continuous incident and log lifecycle management:

`ORA-00020, ORA-00060, ORA-00700, ORA-01555, ORA-01628, ORA-04030, ORA-27300, ORA-27301, ORA-27302, ORA-30036` and other internal errors.

These issues are related to database performance, database patch installation, database patch conflict, database UNIX resources, database XDB, database installation, database upgrade, database preupgrade, database AWR space, database shutdown, database startup, database dataguard, EM tablespace metrics, EM debugging setting, EM metric alert, EM cliadd, EM cludisc, EM dbsys, EM gendisc, and EM procdisc.

18.3

Upload Metadata and Data to a Service Request



12c
```
$ tfactl diagcollect -srdc ORA600
```
18.3
```
$ tfactl diagcollect -srdc dbsqlperf
                        -sr SR34782
```

To upload a collection with metadata and data to your service request:

1. Store your My Oracle Support credentials in the Oracle Trace File Analyzer wallet.
```
# tfactl -setupmos
```
2. Collect and upload files directly to your SR.
```
$ tfactl diagcollect -srdc dbtde -sr SR34782
```
3. Upload any other file to Oracle Support.
```
$ tfactl upload -sr SR34782 -user username list_of_files_to_upload
```

Either as part of an automatic or manual diagnostic collection, TFA 18.3 provides the ability to upload the directory with the collection and data to your service request in My Oracle Support. Read the TFA Overview White Paper.

If your environment can make a connection to oracle.com, you can use `tfactl diagcollect` command so that Oracle Trace File Analyzer automatically uploads the collection to your Service Request (SR) for analysis by Oracle Support. You can also upload other files to your SR.

Prior to uploading, you must store your My Oracle Support credentials in the Oracle Trace File Analyzer wallet. This is a one-time task completed with the `tfactl -setupmos` command and must be done by root. The command prompts for the MOS credentials, verifies them, and stores them in the wallet.

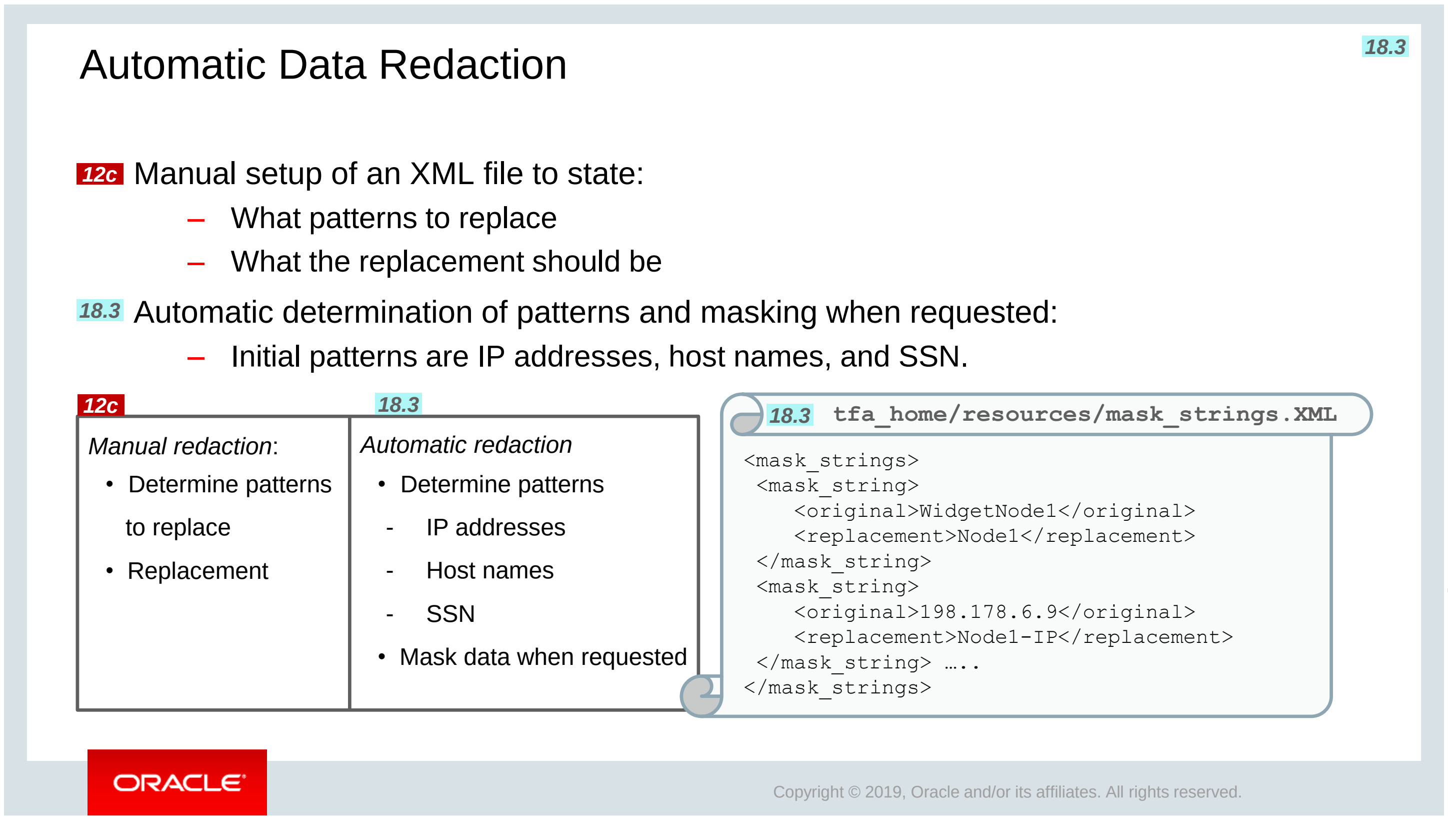
TFA collector uses `orapki` and `mkstore` commands to create the wallet and store the credentials.

The wallet is created using the `orapki` command. The wallet files are secured to be read/write by the `root` user only. All credential details are stored in the wallet using the `mkstore` command.

TFA diagcollection produces one ZIP for each host. It reads credentials from the wallet, authenticates the user, and uploads file in parts, in parallel.

You can also upload files without the wallet. When uploading without the wallet, `tfactl` prompts for the user to set up the wallet.

```
  tfactl upload -sr SR34782 -user user_id list_of_files_to_upload
```



Automatic Data Redaction

18.3

12c Manual setup of an XML file to state:
- – What patterns to replace
- – What the replacement should be

18.3 Automatic determination of patterns and masking when requested:
- – Initial patterns are IP addresses, host names, and SSN.

12c	18.3
Manual redaction:	*Automatic redaction*
• Determine patterns to replace	• Determine patterns
• Replacement	- IP addresses
	- Host names
	- SSN
	• Mask data when requested

18.3 **tfa_home/resources/mask_strings.XML**

```
<mask_strings>
 <mask_string>
    <original>WidgetNode1</original>
    <replacement>Node1</replacement>
 </mask_string>
 <mask_string>
    <original>198.178.6.9</original>
    <replacement>Node1-IP</replacement>
 </mask_string> …..
</mask_strings>
```

ORACLE

TFA 12*c* has the ability to redact data but requires manual setup of a driver XML file to state what patterns to replace and what the replacement should be.

TFA 18.3 is able to determine some patterns and mask them automatically when requested to do so. Initial patterns are IP addresses, host names, and Social Security Number (SSN).

If you want to mask sensitive data, such as host name and IP address, create the `$tfa_home/resource/mask_strings.xml` file and define all data replacements you want.

Summary

In this lesson, you should have learned how to:

- Use automated SQL diagnosibility to diagnose and repair SQL statements
- Benefit from the new service request data collections
- Upload data collections directly to your service request
- Redact data in data collections to upload

Practices Environment - 1

Each student has one virtual machine (VM1) to use during class.

- Oracle Database 19c installed
- One CDB named **`ORCL`** and its PDB named **`PDB1`**

Practices Environment - 2

Pre-created databases and instances with their respective PDBs:

- Data files in `/u02/app/oracle/oradata/`
- All backup files in `/u03/app/oracle/fast_recovery_area/ORCL`
- Password and init files in `$ORACLE_HOME/dbs`
- Diagnostics files in `/u01/app/oracle/diag/rdbms/orcl`
- TDE wallet in `/u01/app/oracle/admin/orcl/tde_wallet`
- Net files in `$ORACLE_HOME/network/admin`

The configuration of the environment used for the practices of the course matches the configuration used on the Oracle Cloud Virtual Machine for the pre-created `ORCL` database of Oracle Database Cloud Service. Oracle Database Cloud Service gives you the ability to deploy Oracle databases in the cloud, with each database deployment containing a single Oracle database.

This is a good way to get familiar with the Oracle Database Cloud environment that is preconfigured for cloud customers.

Practice 6: Overview

- 6-1: Fixing SQL Statements by Using Automatic SQL Diagnosis and Repair
- 6-2: Creating SRDCs for Specific Events

7

Sharding Enhancements

Objectives

After completing this lesson, you should be able to:

- Explain multiple table family support for system-managed sharding
- Describe support for multiple PDB shards in the same CDB
- Generate unique sequence numbers across shards
- Describe support for multi-shard query coordinators on shard catalog standbys
- Propagate parameter settings across shards

Multiple Table Family Support in System-Managed Sharding

18c Only one table family per sharded database

19c More than one table family per sharded database

- Data in the same chunks
- Data for different applications in one sharded database
- System-managed databases only
- `CONFIG TABLE FAMILY`

A **table family** is a set of related tables sharing the same sharding key.

In Oracle Database 18c, the Oracle Sharding feature supports only one table family for each sharded database.

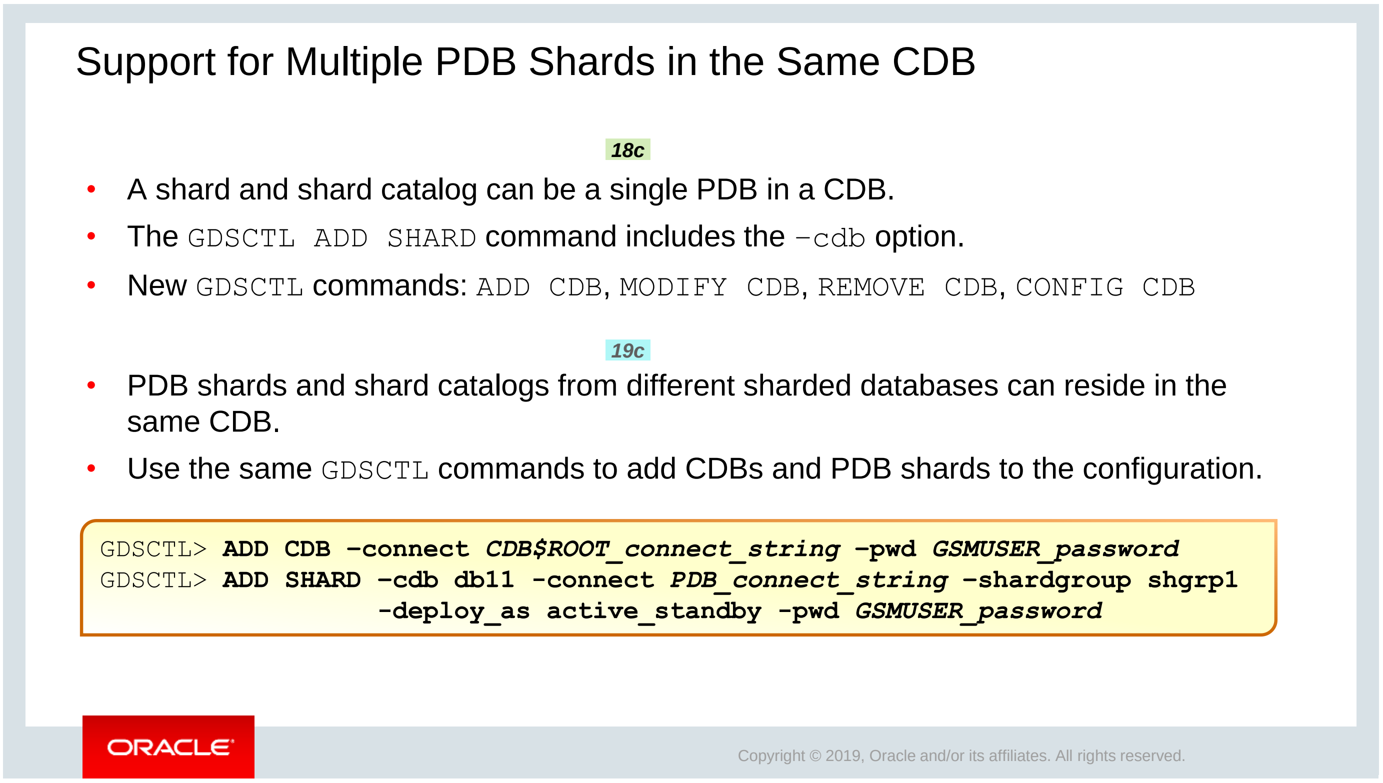
In Oracle Database 19c, Oracle Sharding includes support for multiple table families where all data from different table families reside in the same chunks, which contain partitions from different table families sharing the same hash key range. Different applications accessing different table families can be hosted on one sharded database.

This feature applies to system-managed sharded databases only. Composite and user-defined sharded databases only support one table family.

To support this enhancement, there is one new `GDSCTL` command, `CONFIG TABLE FAMILY`. Several other commands are extended to support this feature like `ADD SERVICE`, `MODIFY SERVICE`, `CONFIG SERVICE`, `CONFIG CHUNKS`, `STATUS ROUTING`, and `VALIDATE CATALOG`.

Some restrictions are changed with the use of `CREATE SHARDED TABLE` and `TABLESPACE SET`.

Cross-table family queries should be minimal and only carried out on the sharding coordinator. Each table family is associated with a different global service. Applications from different table families each have their own connection pool and service, and use their own sharding key for routing to the correct shard.



Support for Multiple PDB Shards in the Same CDB

18c

- A shard and shard catalog can be a single PDB in a CDB.
- The `GDSCTL ADD SHARD` command includes the `-cdb` option.
- New `GDSCTL` commands: `ADD CDB`, `MODIFY CDB`, `REMOVE CDB`, `CONFIG CDB`

19c

- PDB shards and shard catalogs from different sharded databases can reside in the same CDB.
- Use the same `GDSCTL` commands to add CDBs and PDB shards to the configuration.

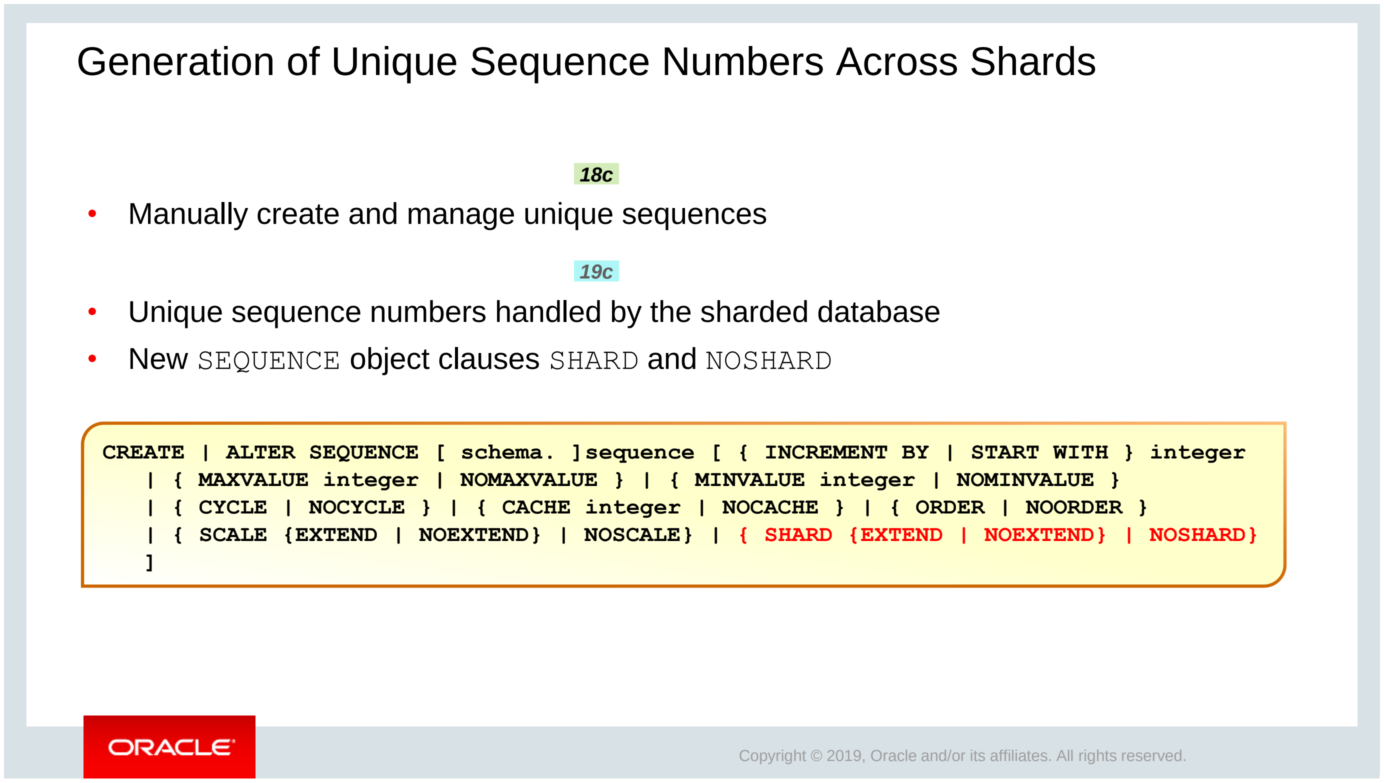
```
GDSCTL> ADD CDB -connect CDB$ROOT_connect_string -pwd GSMUSER_password
GDSCTL> ADD SHARD -cdb db11 -connect PDB_connect_string -shardgroup shgrp1
                 -deploy_as active_standby -pwd GSMUSER_password
```

In Oracle Database 18c, Oracle Sharding introduced the capability of using a single PDB in a CDB as a shard or a shard catalog database.

In Oracle Database 19c, Oracle Sharding allows the use of more than one PDB in a CDB for shards or shard catalog databases, with certain restrictions.

To support consolidation of databases on underutilized hardware, for ease of management or geographical business requirements, you can add other, non-shard or shard PDBs to the CDB containing a shard PDB. A CDB can contain shard PDBs from different sharded databases, each with their own separate catalog databases. CDBs can support multiple PDB shards from different sharded databases. However, this is limited to only one PDB shard from a given sharded database for each CDB.

No changes have been made to the `GDSCTL` commands supporting the multitenant architecture for Oracle Sharding, as shown in the example.



Generation of Unique Sequence Numbers Across Shards

18c

- Manually create and manage unique sequences

19c

- Unique sequence numbers handled by the sharded database
- New `SEQUENCE` object clauses `SHARD` and `NOSHARD`

```
CREATE | ALTER SEQUENCE [ schema. ]sequence [ { INCREMENT BY | START WITH } integer
    | { MAXVALUE integer | NOMAXVALUE } | { MINVALUE integer | NOMINVALUE }
    | { CYCLE | NOCYCLE } | { CACHE integer | NOCACHE } | { ORDER | NOORDER }
    | { SCALE {EXTEND | NOEXTEND} | NOSCALE} | { SHARD {EXTEND | NOEXTEND} | NOSHARD}
    ]
```

Before Oracle Database 19c, if you needed a unique number across shards, you had to manage it yourself.

In Oracle Database 19c, Oracle Sharding allows you to independently generate on each shard sequence numbers which are unique across all shards for non-primary key columns handled by the sharded database.

To support this feature, new `SEQUENCE` object clauses, `SHARD` and `NOSHARD`, are included in the `SEQUENCE` object DDL syntax.

A sharded sequence is created on the shard catalog but has an instance on each shard. Each instance generates monotonically increasing numbers that belong to a range which does not overlap with ranges used on other shards. Therefore, every generated number is globally unique.

A sharded sequence can be used, for example, to generate a unique order number for a table sharded by a customer ID. An application that establishes a connection to a shard using the customer ID as a key can use a local instance of the sharded sequence to generate a globally unique order number.

Note that the number generated by a sharded sequence cannot be immediately used as a sharding key for a new row being inserted into this shard because the key value may belong to another shard and the insert will result in an error. To insert a new row, the application should first generate a value of the sharding key and then use it to connect to the appropriate shard. A typical way to generate a new value of the sharding key would be by using a regular (nonsharded) sequence on the shard catalog.

If a single sharding key generator becomes a bottleneck, a sharded sequence can be used for this purpose. In this case, an application should connect to a random shard (using the global service without specifying the sharding key), get a unique key value from a sharded sequence, and then connect to the appropriate shard using the key value.

`NOSHARD` is the default for a sequence. If the `SHARD` clause is specified, this property is registered in the sequence object's dictionary table, and is shown using the `DBA_SEQUENCES`, `USER_SEQUENCES`, and `ALL_SEQUENCES` views.

When `SHARD` is specified, the `EXTEND` and `NOEXTEND` clauses define the behavior of a sharded sequence. When `EXTEND` is specified, the generated sequence values are all of length *(x+y)*, where *x* is the length of a `SHARD` offset of size 4 (corresponding to the width of the maximum number of shards, that is, 1000) affixed at the beginning of the sequence values, and *y* is the maximum number of digits in the sequence `MAXVALUE/MINVALUE`.

The default setting for the `SHARD` clause is `NOEXTEND`. With the `NOEXTEND` setting, the generated sequence values are at most as wide as the maximum number of digits in the sequence `MAXVALUE/MINVALUE`. This setting is useful for integration with existing applications where sequences are used to populate fixed width columns. On invocation of `NEXTVAL` on a sequence with `SHARD NOEXTEND` specified, a user error is thrown if the generated value requires more digits of representation than the sequence's `MAXVALUE/MINVALUE`.

If the `SCALE` clause is also specified with the `SHARD` clause, the sequence generates scalable values within a shard for multiple instances and sessions, which are globally unique. When `EXTEND` is specified with both the `SHARD` and `SCALE` keywords, the generated sequence values are all of length *(x+y+z)*, where *x* is the length of a prepended `SHARD` offset of size 4, *y* is the length of the scalable offset (default 6), and *z* is the maximum number of digits in the sequence `MAXVALUE/MINVALUE`.

Support for Multi-Shard Query Coordinators on Shard Catalog Standbys

18c

Multi-shard queries are performed on primary shard catalog database only.

19c

Active Data Guard standbys can be used to perform multi-shard queries.

- Improved scalability and availability
- Coordinator service, `GDS$COORDINATOR.<shard_name>`
- Regional affinity

ORACLE

The multi-shard query coordinator database contains the metadata of the sharded topology and provides query processing support for sharded databases. To perform multi-shard queries, the client connects to the multi-shard query coordinator using the `GDS$CATALOG` service on the shard catalog database.

Before Oracle Database 19c, only the primary shard catalog database could be used as the multi-shard query coordinator.

In Oracle Database 19c, to improve the scalability and availability of multi-shard query workloads, you can also enable the multi-shard query coordinator on Oracle Active Data Guard standbys of the shard catalog database.

Oracle Active Data Guard standby shard catalog databases in read-only mode can act as multi-shard query coordinators. For each active replica of the catalog database, a special coordinator service, `GDS$COORDINATOR.<shard_name>`, is running and registered on all shard directors. Clients can connect to this service on any of the replicas and perform multi-shard queries, allowing shard directors to distribute the multi-shard query workload with respect to runtime load balancing and decrease the load on the primary shard catalog, which is the central component of the Oracle Sharding framework. Additionally, if the database's region is set, and the client specifies the region in the connection string, a shard director routes a connection with respect to regional affinity.

Propagation of Parameter Settings Across Shards

18c

Configure parameter settings on individual shards.

19c

Centrally manage and propagate parameter settings from the shard catalog:

- Automatic propagation to the shards
- Use `ENABLE SHARD DDL` on the shard catalog

Before Oracle Database 19c, database administrators had to configure the `ALTER SYSTEM` parameter settings on each shard in a sharded database. This feature provides ease of manageability by allowing administrators to centrally manage and propagate parameter settings from the shard catalog to all the database shards. After settings are configured at the shard catalog, they are automatically propagated to all shards of the sharded database.

Propagation of system parameters happens only if done under `ENABLE SHARD DDL` on the shard catalog.

Summary

In this lesson, you should have learned how to:

- Explain multiple table family support for system-managed sharding
- Describe support for multiple PDB shards in the same CDB
- Generate unique sequence numbers across shards
- Describe support for multi-shard query coordinators on shard catalog standbys
- Propagate parameter settings across shards

Practice 7: Overview

There are no practices for this lesson.