ORACLE UNIVERSITY

Oracle Database 12*c* R2: New Features for 12*c* R1 Administrators

Student Guide – Volume II
D93517GC10
Edition 1.0 | October 2016 | D98433

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

ORACLE

Authors

Dominique Jeunot
Jean-François Verrier
Mark Fuller
James Spiller

Technical Contributors and Reviewers

Krishnanjani Chitta
Randall Richeson
Gerlinde Frenzen
Sailaja Pasupuleti
Anita Mukundan
Harald van Breederode
Jim Stenoish
Branislav Valny

Editors

Nikita Abraham
Chandrika Kennedy
Vijayalakshmi Narasimhan

Graphic Editor

Seema Bopaiah

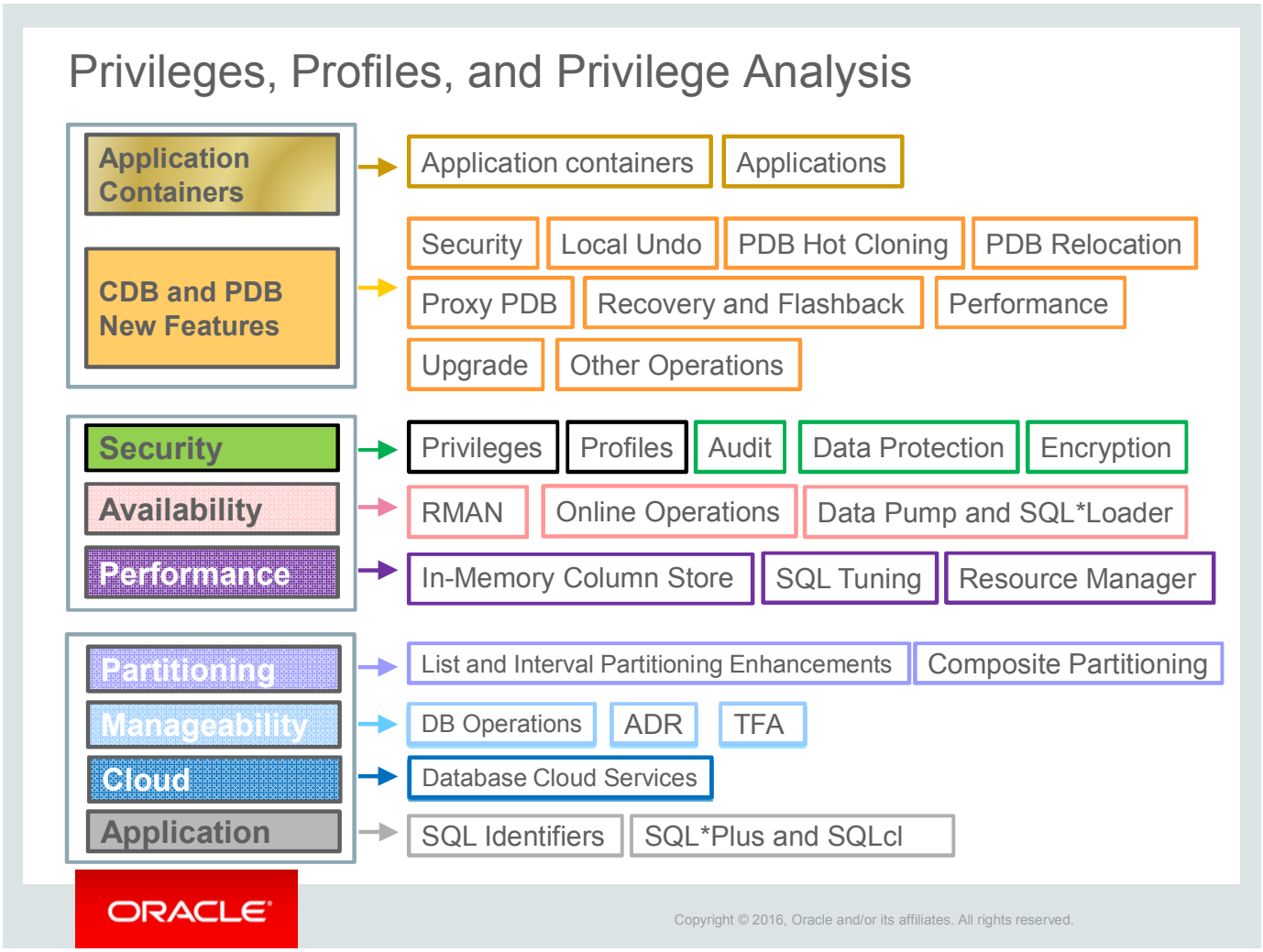
Publishers

Pavithran Adka
Syed Ali
Giri Venugopal

Contents

9

Privileges and User Profiles



This lesson describes enhancements in administrative privileges, SSL authentication for administrative privileged users, migration of the password file to 12.2, and also enhancements in user profiles and privilege analysis.

Objectives

After completing this lesson, you should be able to:

- Describe the new `SYSRAC` administrative system privilege
- Enhance security for administrative users by using the password file
- Describe the new format of the password file
- Lock inactive database accounts
- Explain the `INHERIT (ANY) REMOTE PRIVILEGES` privileges
- Capture privileges used by the CBAC and SAR roles, and at PL/SQL compilation time, with Privilege Analysis
- Compare privilege usage through Privilege Analysis runs
- Detect unused grants with Privilege Analysis

ORACLE

To get detailed information about how to perform any of the operations explained in this lesson, refer to the following guides in Oracle documentation:

- *Oracle Database Administrator's Guide 12c Release 2 (12.2)*
- *Oracle Database Security Guide 12c Release 2 (12.2)*
- *Oracle Database Vault Administrator's Guide 12c Release 2 (12.2)*

Administrative Privileged Users

Separation of duty: Task-specific privileges for administrative tasks

- 12.1 • Oracle RMAN administrator: backup and recovery
- 12.1 • Data Guard administrator
- 12.1 • Key management administrator: TDE keystore management
- 12.1 • RAC administration tasks relying on `SYSDBA`
- 12.2 • RAC administrator: RAC database management via the Clusterware agent on behalf of RAC utilities

As one of the fundamental security requirements, the principle of separation of duties dictates that completion of any critical task has an associated role. In Oracle Database 11*g*, the administration of Oracle Database depends heavily on the `SYSDBA` administrative privilege.

Oracle Database 12.1 introduced new administrative privileges that are more task-specific and least privileged to support specific administrative tasks:

- The `SYSBACKUP` administrative privilege that allows you to perform Oracle RMAN backup and recovery operations from Oracle RMAN or through SQL
- The `SYSDG` administrative privilege that allows you to perform Data Guard operations with Data Guard Broker or the DGMGRL command-line interface
- The `SYSKM` administrative privilege that allows you to manage transparent data encryption keystore operations

Oracle Database 12.2 introduces the `SYSRAC` administrative privilege.

SYSRAC Administrative Privilege

Administrative Privilege	Schema	Tasks
SYSDBA, SYSOPER	SYS / PUBLIC	These include the same operations as in 11*g*.
SYSASM	SYS	This is specific to ASM instances only.
12.1 SYSBACKUP	SYSBACKUP	Perform RMAN backup and recovery operations (RMAN or SQL*Plus).
12.1 SYSDG	SYSDG	Perform Data Guard operations (DGMGRL).
12.1 SYSKM	SYSKM	Manage transparent data encryption keystore operations.
12.2 SYSRAC	SYSRAC	Connect to the RAC database instance by using the Clusterware agent on behalf of RAC utilities (CRSCTL, SRVCTL, OCRCONFIG, and CLUVFY).

The SYSRAC privilege that is granted to database users is blocked:

- No password file support for the SYSRAC privilege
- Used only by the CRS oraagent

Oracle Database 12.2 introduces the SYSRAC administrative privilege, which is the default mode of connecting to the database instance for the Clusterware agent, on behalf of RAC utilities such as SRVCTL. With these changes to the CRS agent, no SYSDBA connections to the database instance are necessary for day-to-day administration of RAC clusters. A minimal set of privileges that are required for day-to-day administration of Oracle databases on a RAC cluster is granted by default to the SYSRAC administrative privilege. Administrative tasks such as upgrade/downgrade might still require the administrator to be granted the SYSDBA privilege temporarily.

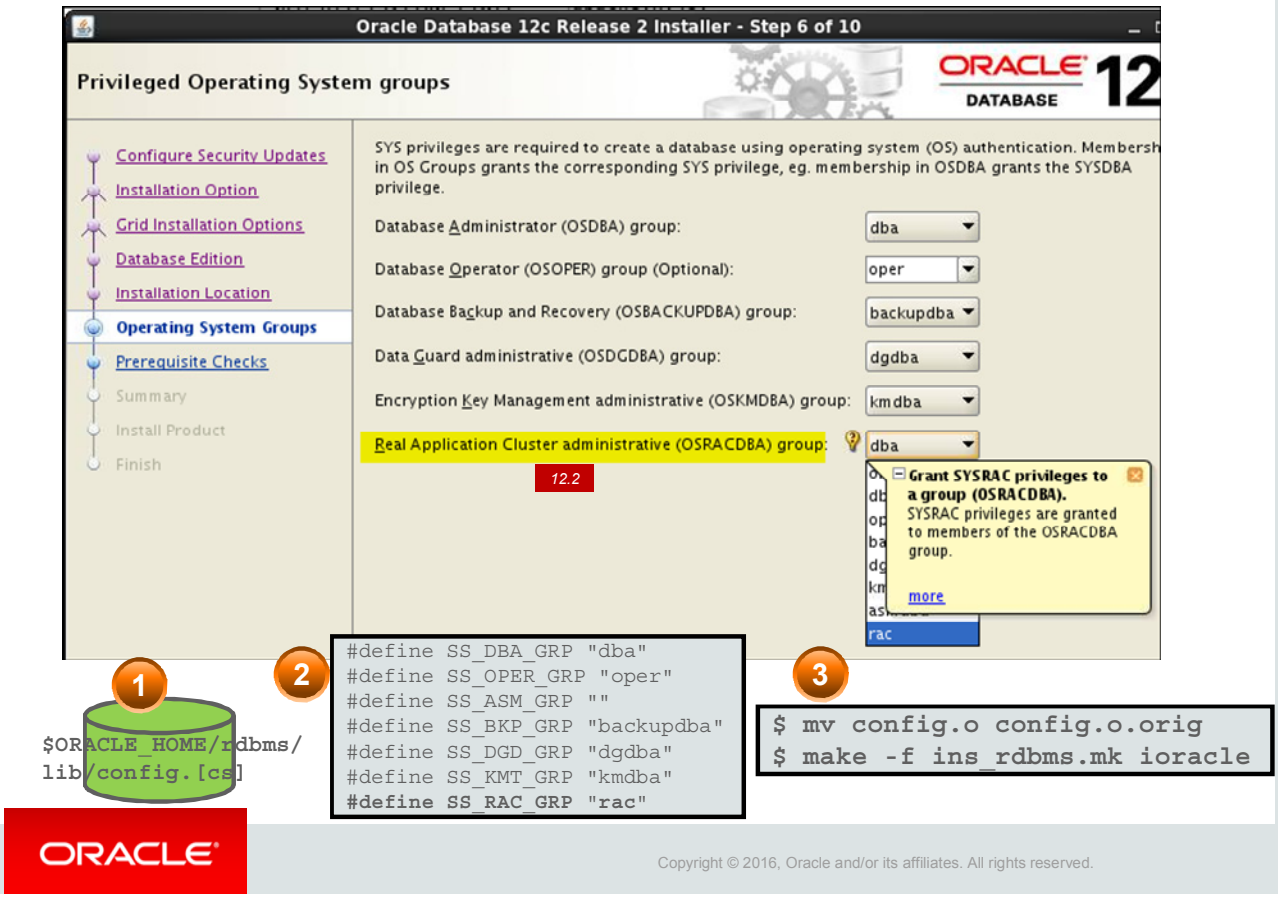
The Clusterware agent process is owned by the database installation owner. Therefore, this user is in the membership of the OSRAC group.

After connecting with the SYSRAC privilege, you are connected under the SYSRAC schema.

Unlike with other database administrative privileges, the SYSRAC administrative privilege is not available to database users either through the password file or through network authentication methods.

```
SQL> grant sysrac to  c##test;
*
ERROR at line 1:
ORA-28190: SYSRAC administrative privilege cannot be granted to other users
```

OS Authentication via the OSRACDBA Group



In OS authentication, groups are created and assigned specific names as part of the database installation process, provided that OS UNIX or Windows groups are created and accounts are assigned to the appropriate operating system–defined groups.

Membership in the OSRACDBA UNIX (or an appropriate Windows) group affects your connection to the database. In the example in the slide, if you are a member of the `RAC` OS UNIX group, and you specify `AS SYSRAC` when you connect to the database, you connect to the database with the `SYSRAC` administrative privilege under the `SYSRAC` user.

Without being a member of the OS group, users cannot connect as administrative users by using OS authentication. That is, "`CONNECT / AS SYSRAC`" will fail.

To change the OS group names, ensure that you are using the groups defined in the `$ORACLE_HOME/rdbms/lib/config.[cs]` file. Next, shut down all databases and listeners, and then relink the Oracle executable as shown in the slide.

Windows User Groups

The Windows user group is `ORA_%HOMENAME%_SYSRAC`.

Privileges of SYSRAC

System / Object Privileges

```
ALTER DATABASE MOUNT
ALTER DATABASE OPEN
ALTER DATABASE OPEN READ ONLY
ALTER DATABASE CLOSE
ALTER SESSION SET EVENTS
ALTER SESSION SET _NOTIFY_CRS
ALTER SESSION SET CONTAINER
ALTER SYSTEM REGISTER
ALTER SYSTEM SET LOCAL_LISTENER
ALTER SYSTEM SET REMOTE_LISTENER
ALTER SYSTEM SET LISTENER_NETWORKS
```

SELECT X$ tables, V$ / GV$ views

EXECUTE

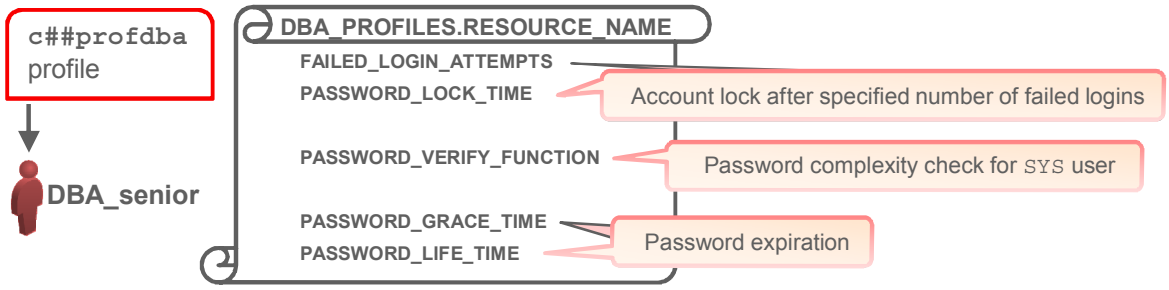
dbms_service
dbms_service_prvt
dbms_session
dbms_ha_alerts_prvt

DEQUEUE sys.sys$service_metrics

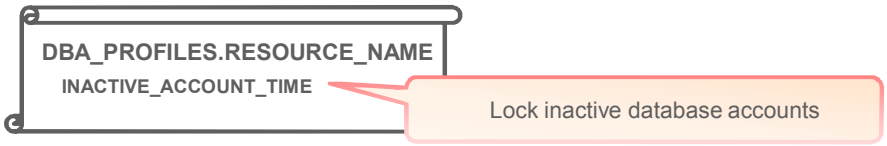
The `SYSRAC` administrative privilege allows the user that is granted this privilege to perform the specific operations described in the slide. The `SYSRAC` user has access to fixed views similar to the other administrative privileges. The access to fixed views cannot be revoked or granted because it is hardcoded similarly to the ability of a `SYSRAC` session to perform the operations specified in the slide. The `SYSRAC` user is also granted the `EXECUTE` privilege on the packages listed in the slide.

Profiles for Administrative and End Users

- Enforce profile password limits on administrative users:



- Lock inactive database accounts.



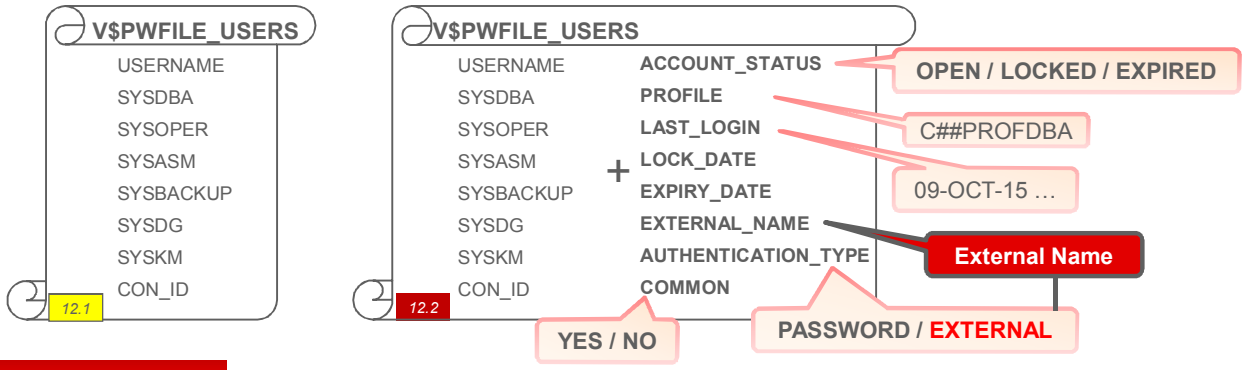
Oracle Database 12.2 enhances the security of administrative users by enforcing the associated profile's password limits:

- Account locked after specified number of failed logins: `failed_login_attempts` and `password_lock_time`
- Password expiration: `password_life_time` and `password_grace_time`
- Password complexity check: `password_verify_function`. The password complexity verification function (PVF) is enforced for the `SYS` user.

In Oracle Database 12.2, if a database user account does not log on for a predefined number of days, the associated user account is automatically locked, which needs to be unlocked by an administrator before it is available for use again. You can define the number of days with the new `INACTIVE_ACCOUNT_TIME` user resource profile limit.

Password File Enhancements

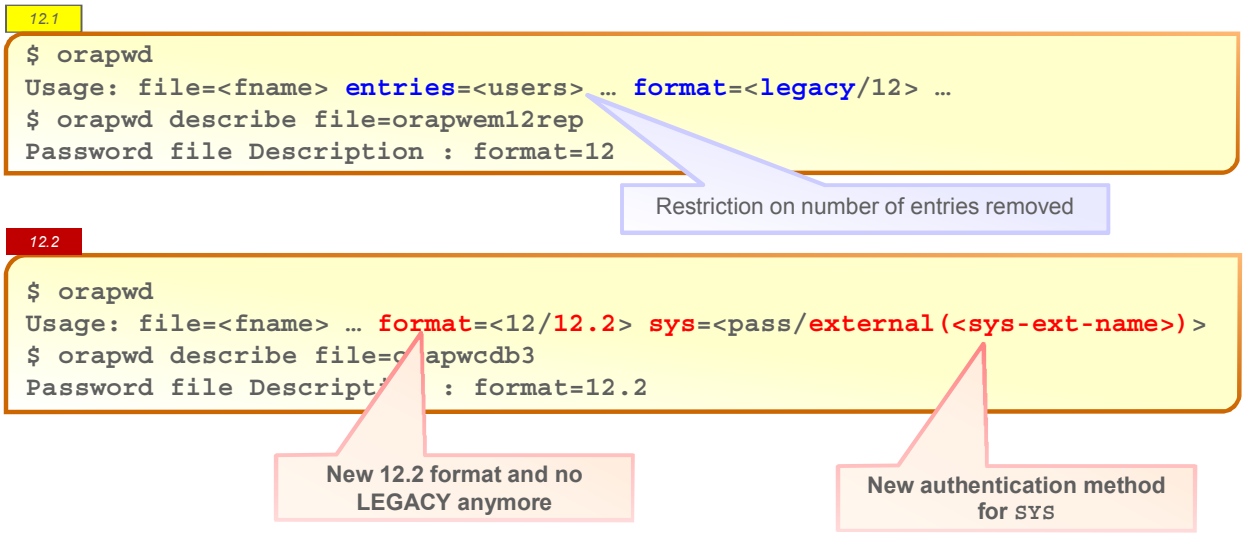
- New information in the new 12.2 format
  - Account status
  - Last successful login time
  - Authentication method used
- Enhancements enforced on application common and local administrative users in the CDB and PDBs



The password file shows new information as soon as the file is migrated to the new 12.2 format.

- The account status shows if the administrative user is `OPEN`, `LOCKED` (the user cannot connect anymore), or `EXPIRED` (the user is mandated to change the password at the connection).
- The Last Successful Login Time for the administrative user connections that use password file authentication is captured and stored in the password file.
- Apart from the password file, there are other mechanisms for administrative user authentication.
  - Operating System: The OS user account must belong to one of the appropriate OS defined groups. Then the OS user can connect to the database without requiring a password such as "`connect / as SYSDBA`."
  - External Directory server: For example, you can use Oracle Internet Directory (OID).
- In CDBs, there are two types of administrative users:
  - Common administrative users who can perform administrative tasks pertaining to the entire CDB or to an application container (`COMMON =YES`)
  - Local administrative users who can perform only container-specific administrative tasks (`COMMON =NO`)

Password File Migration



- Password file migrated from 12.1 to the 12.2 format

```
$ mv orapwcdb1 orapwcdb1.12
$ orapwd file=orapwcdb1 input_file=orapwcdb1.12 format=12.2
$ orapwd describe file=orapwcdb1
Password file Description : format=12.2
```


The `ORAPWD` utility provides the migration option.

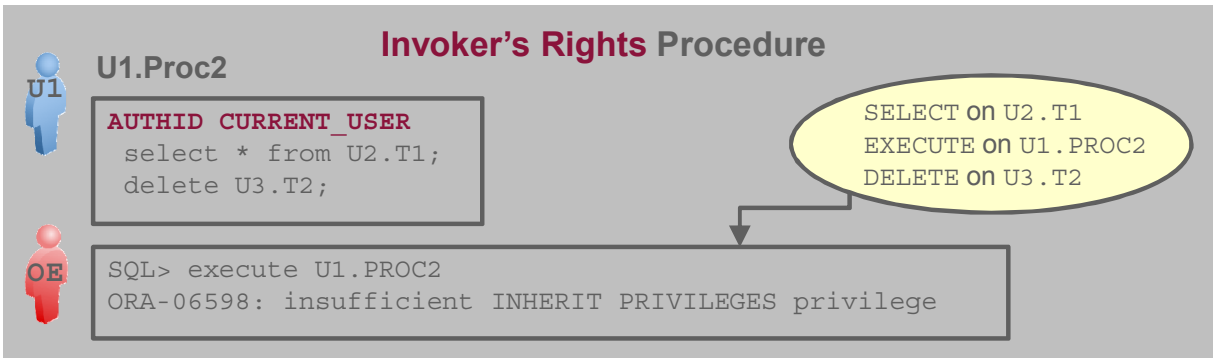
- `INPUT_FILE`: The command-line argument specifies the name of the 12.1 password file from where the user entries will be migrated to the new 12.2 password file. The user entries from the input password file are copied directly to the new output password file.
- `FORMAT`: If the format is not specified, the default is `12` and the password file created supports only the 12.1 features. If `12.2` is specified, the password file created has the new 12.2 information.

After migration, you can check the format of the password file by using the `DESCRIBE` parameter as shown in the slide.

Remark: The 12.2 password is auto-extensible, which allows more entries to be added.

Privilege Checking: Invoker's Rights Procedure

12.1



Additional required privilege checking at run time:

- Of a database user passing into an `AUTHID CURRENT_USER` PL/SQL routine or "callspec" over a C or Java routine
- Invoking users grant the `INHERIT PRIVILEGES` object privilege only to trusted users.

```
SQL> CONNECT oe
SQL> grant INHERIT PRIVILEGES ON USER oe TO u1 ;
```


Risks

A low privilege user could own an invoker's rights procedure that could potentially perform unintended or malicious actions if it is executed by a high-privileged user.

An invoker's right procedure can perform inappropriate actions if it is invoked by another procedure that does not expect an invoker's rights procedure.

In Oracle Database 11*g*, the caller to a procedure had no control over who accessed the caller's privileges. Only the owner of the procedure controlled the right's inheritance.

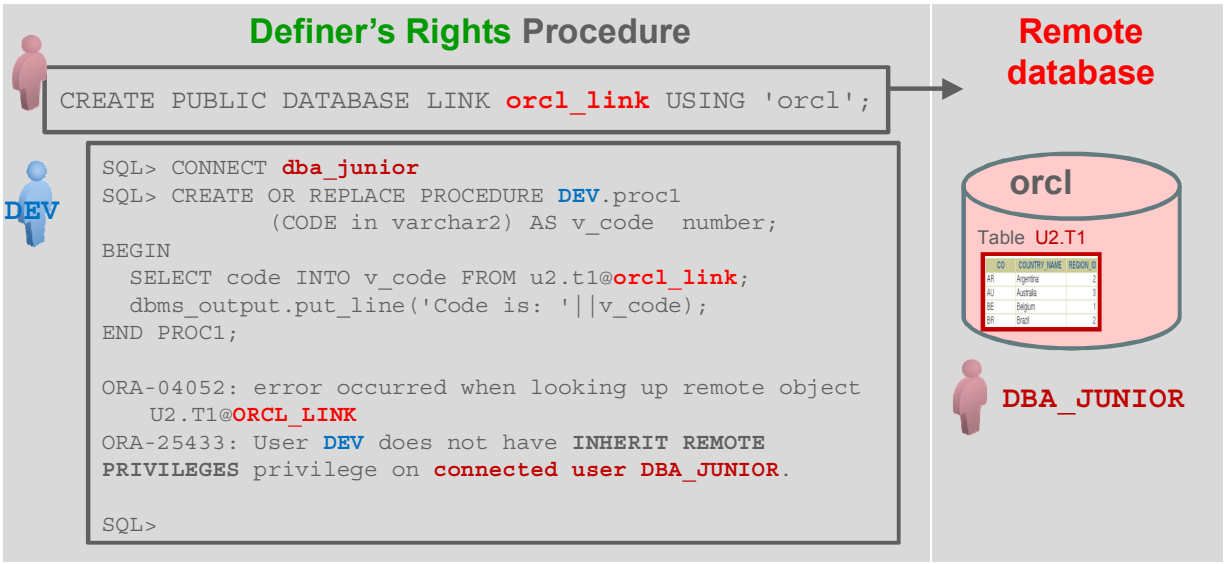
New Privilege Checking

Privilege checking in Oracle Database 12*c* implements a new restriction, not a new power. Existing cases that did not require a privilege check now require one. When a user runs an invoker's rights procedure, Oracle Database checks the procedure owner's privileges before initiating or running the code. The owner must have the `INHERIT PRIVILEGES` object privilege on the invoking user or the `INHERIT ANY PRIVILEGES` privilege. If this is not the case, the runtime system raises an error.

The session is temporarily switched into an environment that treats the entered routine as the definer's rights. It then checks that it has the `INHERIT PRIVILEGES` object privilege on the caller's active current user or that it has the `INHERIT ANY PRIVILEGES` system privilege. The session then reverts to its prior environment. This treatment of the routine as definer's rights mirrors the treatment of the routine during compilation. The benefit of these privileges is that they give invoking users control over who can access their privileges when they run an invokers' rights procedure.

Privilege Checking: Definer's Rights Procedure

12.2



- Allow a connected user to use a current user database link from within a definer's rights procedure to allow operations.

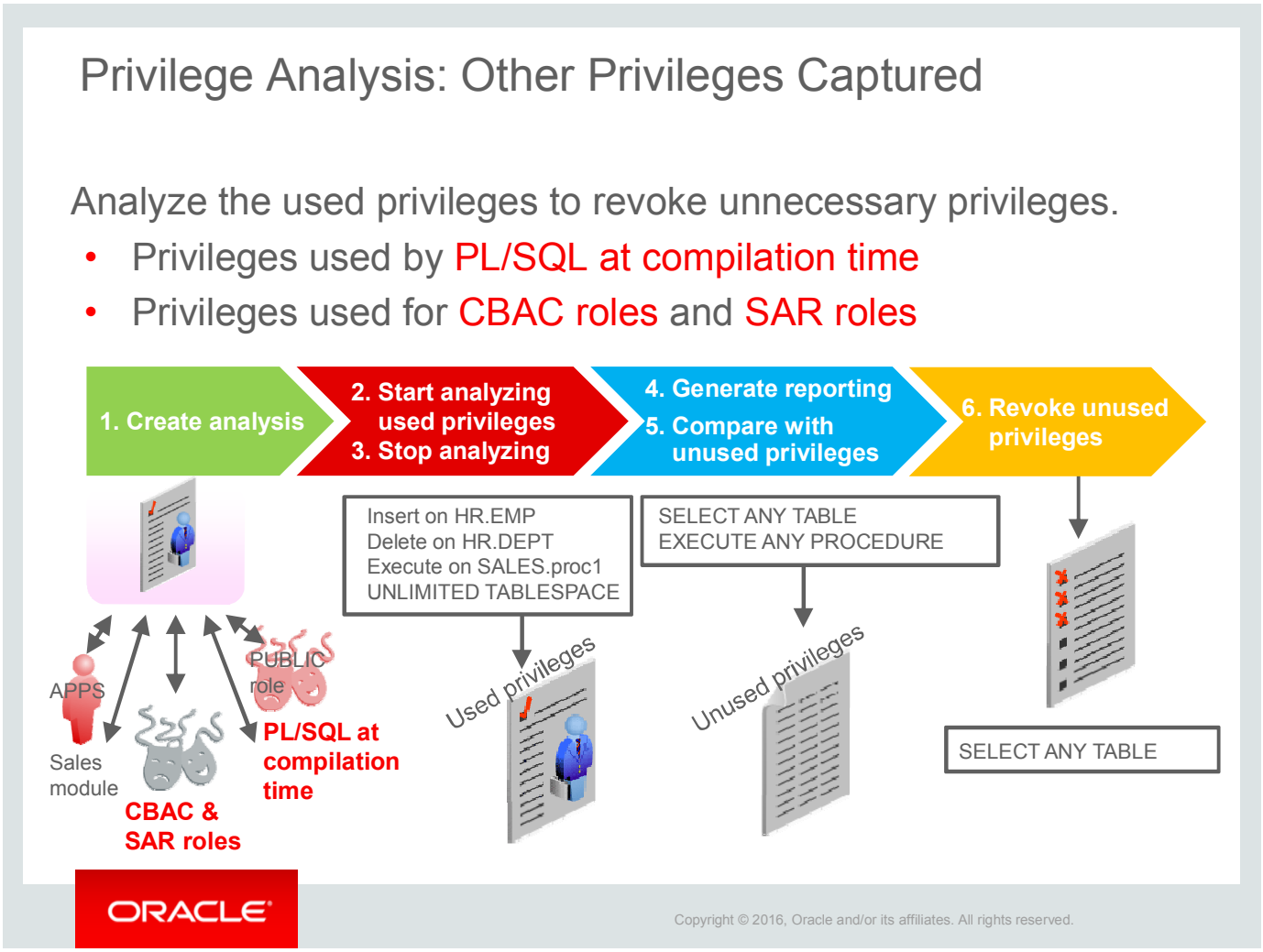
```
SQL> CONNECT system
SQL> GRANT INHERIT REMOTE PRIVILEGES ON USER dba_junior TO dev;
```


A user connected on a database instance should not allow a definer's rights procedure owner to perform database operations on a remote database instance, logging in as that connected user through a current user database link, even unknowingly. A privilege should be explicitly granted to allow such operations.

The new `INHERIT (ANY) REMOTE PRIVILEGES` privilege allows a connected user to use a current user database link from within a definer's rights procedure. Without this privilege, the definer's rights procedure is not able to connect via the current user database link.

The DBA can either grant the object the `INHERIT REMOTE PRIVILEGES` privilege or the system `INHERIT ANY REMOTE PRIVILEGES` privilege on the connected user to the definer's rights procedure.

```
SQL> GRANT INHERIT ANY REMOTE PRIVILEGES to dev;
```



Oracle Database Vault in Oracle Database Release 12.2 still includes the Privilege Analysis feature to help you increase security for your applications and database operations. Privilege analysis is available from Oracle Enterprise Manager Cloud Control 12*c* Release 3 Plug-in Update 1 (12.1.0.3).

PL/SQL Compilation Privileges Capture

In Oracle Database 12.1, if a privilege capture is created and enabled after the PL/SQL procedure is compiled, only the *runtime* privileges used are captured by Privilege Analysis. However, the *compilation* time privileges used are not captured. Oracle Database 12.2 is able to capture the privileges required for procedure compilation and report them in the relevant views. The privileges that are required for PL/SQL compilation are captured during a database-wide capture. The privileges required are stored in the default `ORA$DEPENDENCY` capture that contains all the privileges used for compiling dependency objects.

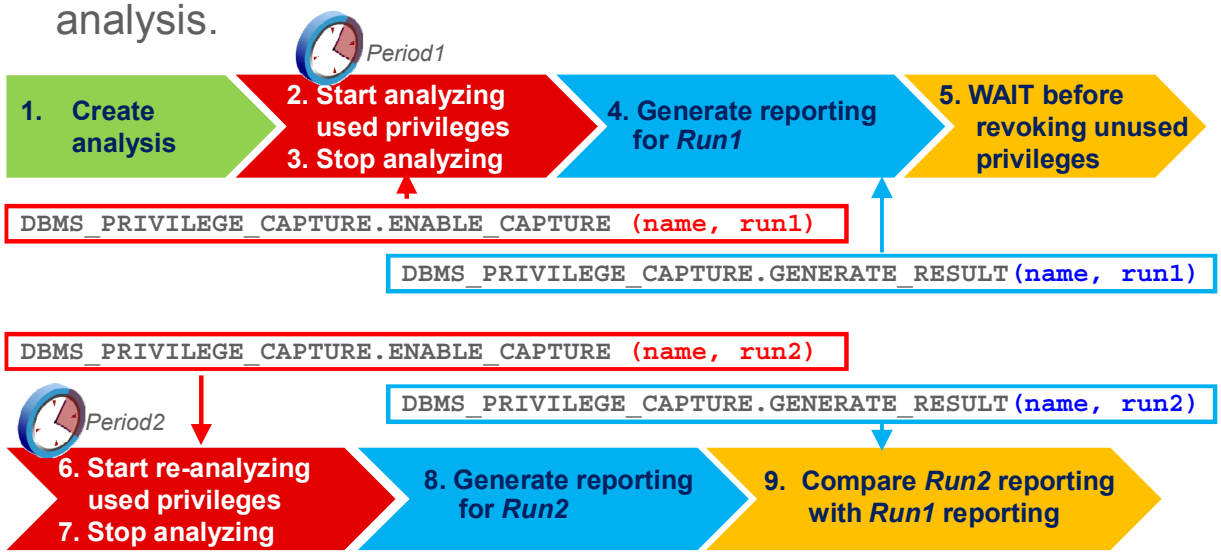
CBAC Roles and SAR Capture

- A role can be granted to a procedure or function so that the invoker can exercise the Code Based Access Control (CBAC) roles only during the execution of the procedure. Oracle Database 12.2 is able to capture the privileges used from the CBAC roles and report the grant path information, including the procedure information in the relevant views.
- If a Secure Application Role (SAR) is enabled during a user session, the privileges used from the SAR are reported along with the grant path from the privileges to the application package.

The privileges required for CBAC and SAR roles are captured during role-wide captures.

Privilege Analysis: Runs

- Generate two run reports at different periods for the same analysis.



- Compare the privilege results between the two runs.
- Revoke unnecessary privileges.

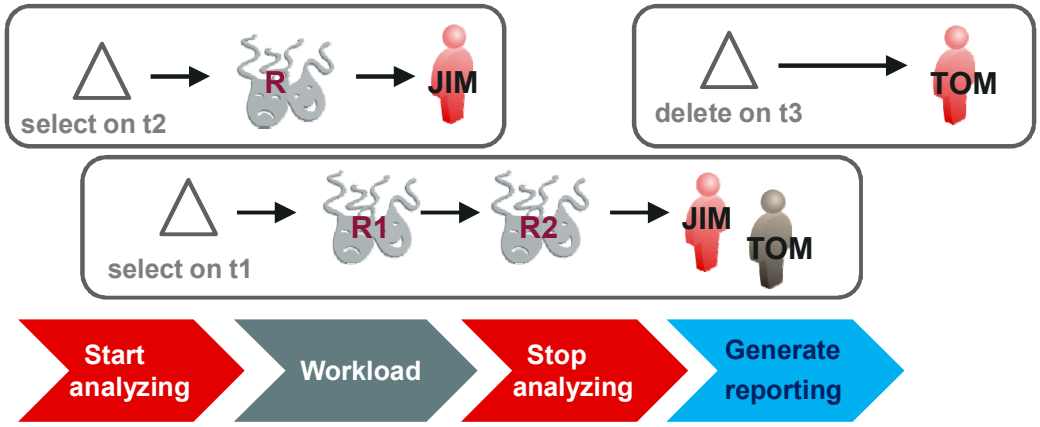
ORACLE

In Oracle Database 12.2, a run of a capture defines a period of time from enabling a capture to its subsequent disabling. A capture can have multiple runs if it has been enabled and disabled multiple times. You can then compare the results from the two runs to see the difference on privilege usage.

1. When creating a capture, you first define the targeted objects to be analyzed in the used privileges. You do that by setting the type of analysis.
2. The next step is to enable the capture to start analyzing the used privileges. In Oracle Database 12.2, you can define a run name that is used to represent this enabled run.
3. After a certain time, you stop analyzing.
4. When generating results, this run name can be used to specify which run will be used to generate the privilege results.
5. You can then repeat the operation: Enable another analysis run for the same capture (6), stop analyzing after a certain time (7), and generate results for this second run name (8).
9. You can finally compare the results between the two runs to be sure to revoke unnecessary privileges.

Be aware that if the captures run over a longer period or too many runs are kept, the `SYSAUX` tablespace increases. Delete the analysis or at least the runs after the analysis is over.

Privilege Analysis: Unused Grants




```
SQL> SELECT capture, run_name, grantee, rolename,
            obj_priv, object_name FROM dba_unused_grants
     WHERE rolename in ('R','R1','R2')
     OR    grantee in ('JIM','TOM');

CAPTURE    RUN_NAME GRANTEE ROLENAME OBJ_PRIV OBJECT_NAME
---------- -------- ------- -------- -------- -----------
All_privs RUN1      JIM      R
All_privs RUN1      TOM               DELETE   T3
All_privs RUN1      TOM      R2
```

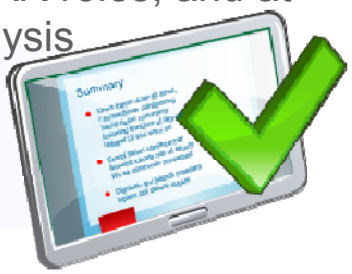

With Oracle Database 12.1, the privilege report shows results based on user and privilege pairs. When the privilege is not a direct grant to the user, it is a non-trivial task for the user to continue reducing the unnecessary privilege grants.

Oracle Database 12.2 introduces a new view, `DBA_UNUSED_GRANTS`, which shows information about unused grants to help reducing privilege grants without the risk of breaking applications' functionalities. If an unused path is `SELECT ANY TABLE -> R1 -> R2 -> JIM`, and there is no other usage, including the path `SELECT ANY TABLE -> R1`, the `SELECT ANY TABLE -> R1` grant is an unused grant. It is safe to revoke such grants.

Summary

In this lesson, you should have learned how to:

- Describe the new `SYSRAC` administrative system privilege
- Enhance security for administrative users by using the password file
- Describe the new format of the password file
- Lock inactive database accounts
- Explain the `INHERIT (ANY) REMOTE PRIVILEGES` privileges
- Capture privileges used by the CBAC and SAR roles, and at PL/SQL compilation time, with Privilege Analysis
- Compare privileges usage through Privilege Analysis runs
- Detect unused grants with Privilege Analysis

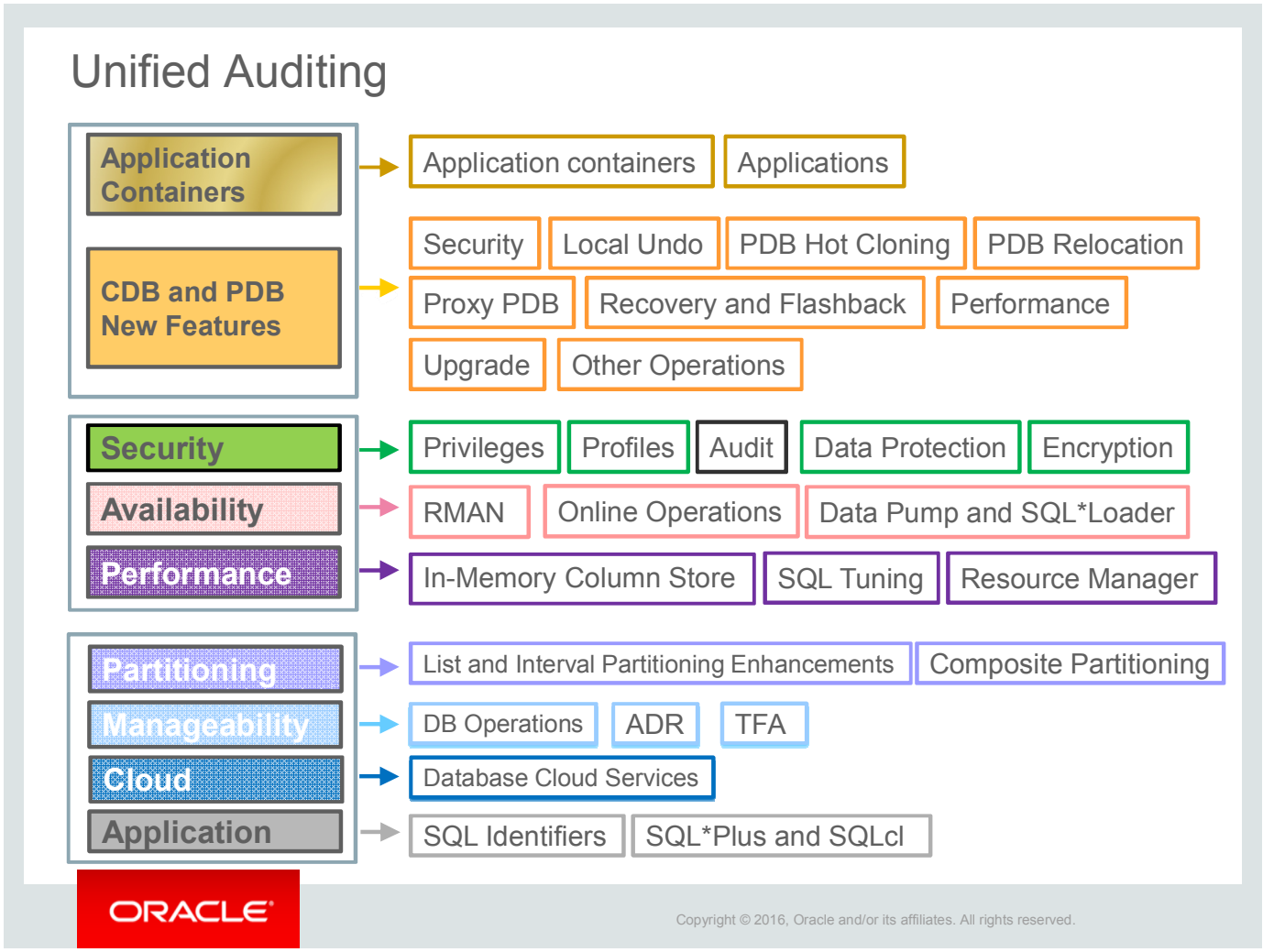


Practice 9: Overview

- 9-1: Migrating the password file to the 12.2 format
- 9-2: Managing administrative privileged users' connections
- 9-3: Using the INHERIT REMOTE PRIVILEGES privilege (*Optional*)
- 9-4: Using Privilege Analysis runs and CBAC roles

10

Auditing



This lesson explains enhancements in Unified Auditing.

Objectives

After completing this lesson, you should be able to:

- Audit all users to whom roles are granted directly
- Capture VPD-generated predicates in Unified Audit Trail

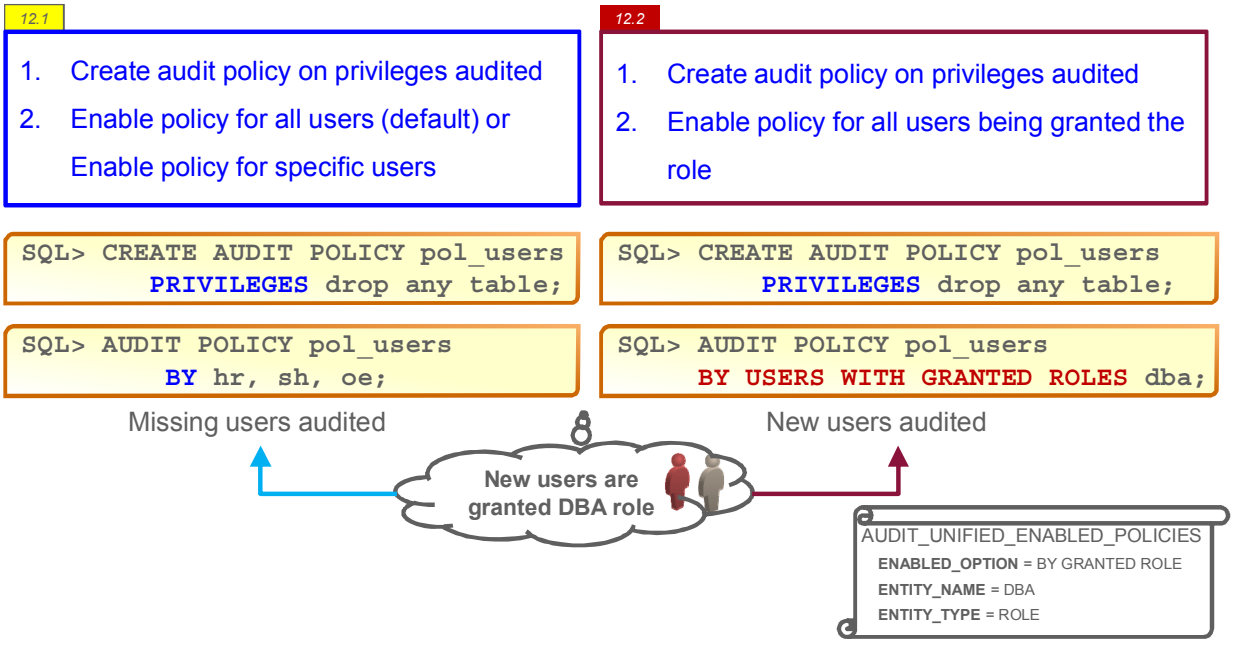
To get detailed information about how to perform any of the operations explained in this lesson, refer to the following guides in Oracle documentation:

- *Oracle Database Security Guide 12c Release 2 (12.2)*
- *Oracle Database Administrator's Guide 12c Release 2 (12.2)*

Auditing Users Being Granted Roles

Find users who use privileges conveyed by specific roles.

Example: Who uses the `DROP ANY TABLE` system privilege conveyed by the **DBA** role?



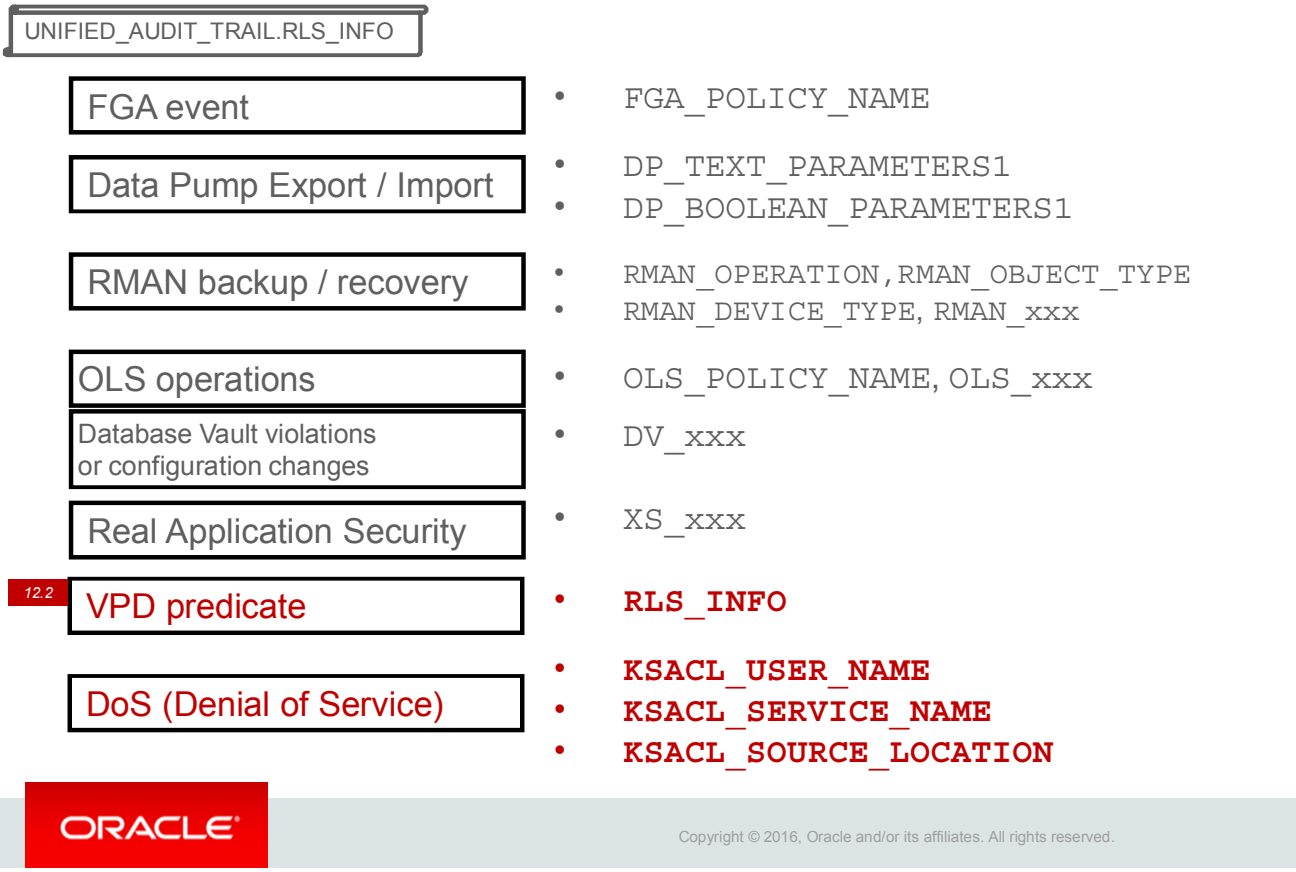
In Oracle Database 12.1, the privilege audit option audits the use of system or object privileges as shown in the example on the left in the slide. All users are audited by default, except if you define a list of those audited. The audit-administrator has to enable the audit policies on all individual users explicitly.

Oracle Database 12.2 introduces a new enablement option for unified audit policies by using database roles. An audit policy can be enabled for users to whom the role is granted directly.

The new clause is `BY USERS WITH GRANTED ROLES <role_list>`. A good example of using the new feature is the predefined role called `DBA`, which contains most of the system privileges, granted to special privileged users, and which might be considered for auditing.

Over a period of time, there could be new users with the `DBA` role granted. Some of the earlier DBA users might no longer have the `DBA` role. The audit-administrator has to keep track of such changing auditing requirements and enable the audit policies appropriately to new sets of DBA users. Similarly users who no longer have `DBA` role granted should be excluded from auditing to avoid generating unnecessary audit records. This is a tedious and repetitive admin task. The new enhancement allows you to enable an audit policy on the `DBA` role so that it is effective for all users to whom the `DBA` role is granted.

Extended Audit Information



In Oracle Database 12.1, database components such as the following can be audited:

- FGA component, which is registered with the framework. The EAI is stored in the `FGA_POLICY_NAME` column.
- Data Pump when export or import operations are executed. The EAI is stored in two new columns: `DP_TEXT_PARAMETERS1` and `DP_BOOLEAN_PARAMETERS1`.
- Oracle RMAN when backup or restore or recovery operations are executed. The EAI is stored in several new columns: `RMAN_OPERATION`, which would contain `'Backup'` or `'Restore'` or `'Recover'`.

Oracle Database 12.2 adds new columns to store information such as the following:

- Virtual Private Database (VPD) policy names and predicates separated by a delimiter
- The connecting username, the target database service name, and the source location of the initiating connection, which are meaningful only when `UNIFIED_AUDIT_TRAIL.RETURN_CODE` is `46981`, which is the denial-of-service (DoS) error code

Summary

In this lesson, you should have learned how to:

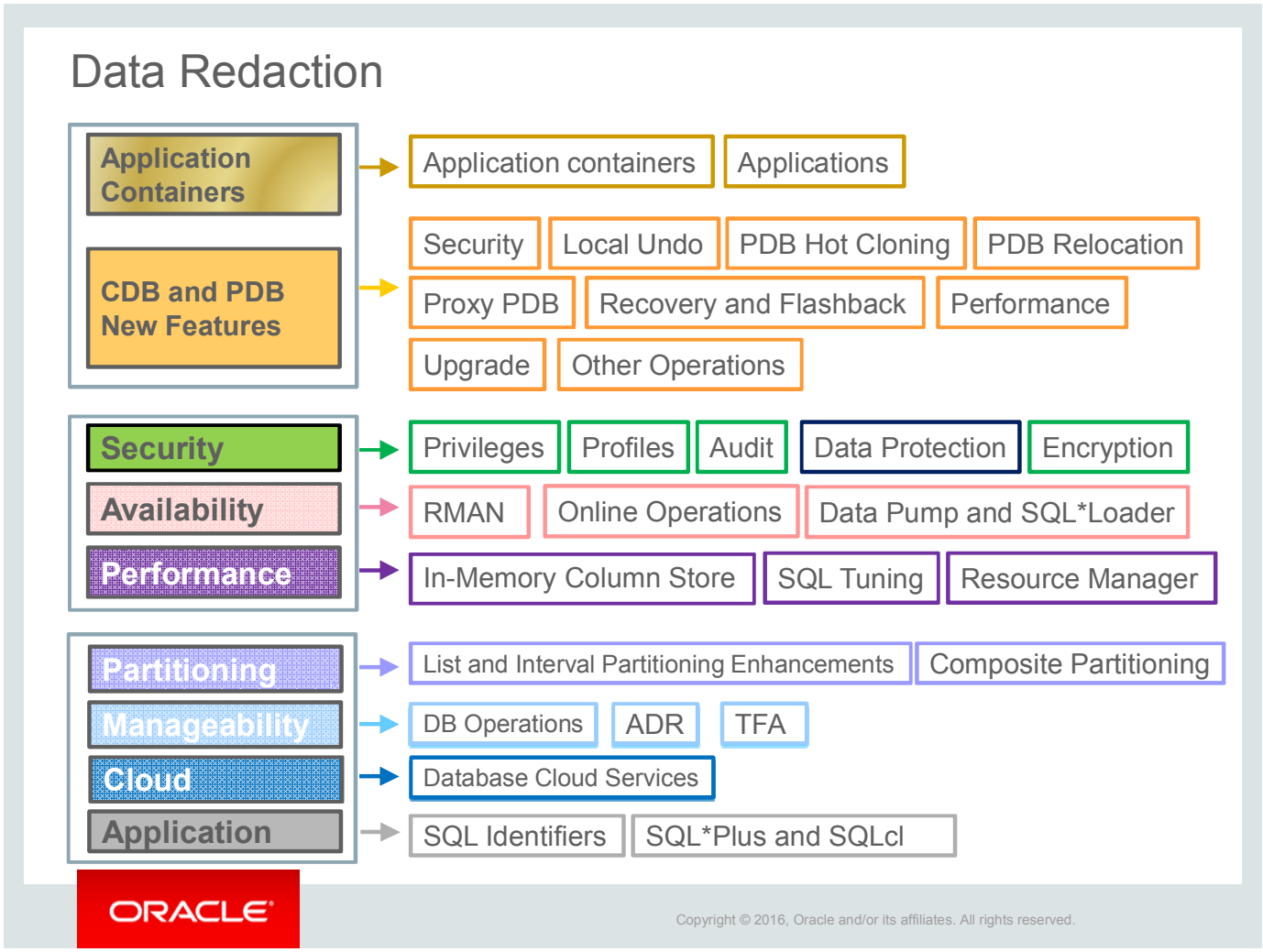
- Audit all users to whom roles are granted directly
- Capture VPD-generated predicates in Unified Audit Trail

Practice 10: Overview

- 10-1: Auditing users using roles
- 10-2: Auditing tables with VPD policies (*Optional*)

11

Data Redaction



This lesson explains enhancements in Oracle Data Redaction.

Objectives

After completing this lesson, you should be able to:

- Create and reuse data redaction formats from the format library
- Create policy expressions
- Apply policy expressions on columns to define different situations of redaction

To get detailed information about how to perform any of the operations explained in this lesson, refer to the following guide in Oracle documentation:

- *Oracle Database Advanced Security Guide 12c Release 2 (12.2)*

What Is a Redaction Policy?

The redaction policy dictates: 12.1

- What to redact, as specified by: `HR.EMP.SALARY`
- 12.2 **Data Redaction Format** 
- How to redact, as specified by:
  - *Function type* (`FUNCTION_TYPE`)
  - *Function parameters* (`FUNCTION_PARAMETERS`) or *regular expression parameters* (`REGEXP_*`)
- When to redact: *policy expression* (`EXPRESSION`)

12.1 When you create the policy, you can provide only one "how to redact" specification.

12.2 Each column can have its own *policy expression*.

ORACLE

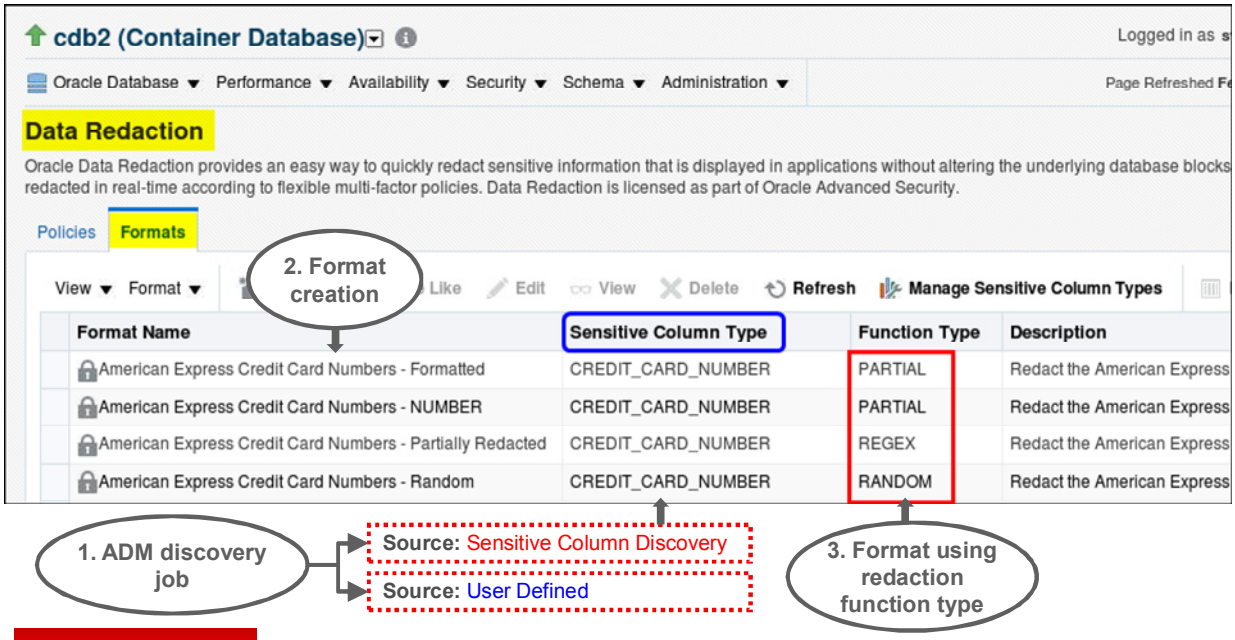
In Oracle Database 12.1, to implement Oracle Data Redaction, you define a redaction policy that specifies what should be redacted, how the redaction should be applied, and when the redaction should take place.

- First, you identify a schema, table, or view, and the column to be redacted.
- Second, you specify the redaction method to use for the column, along with any required parameters. Only one column can be set when you first create the policy. However, you can later modify the policy to add more columns. If you do not provide a value for `FUNCTION_TYPE`, full redaction is used for the specified column.
  Oracle Database 12.2 allows you to create redaction formats, store them in a library, and use any of them on the table columns identified for the new policy.
- Lastly, you specify the exact conditions that must be met for the redaction to be enforced. The "when" condition is set once for the table or view, so it applies to all the columns that are added to the policy.
  Oracle Database 12.2 allows you to define a distinct policy expression for each column of the policy.

Data Redaction Formats Library

12.2

- Use Oracle-supplied formats to create redaction policies.
- Create new formats by using specific redaction expressions.



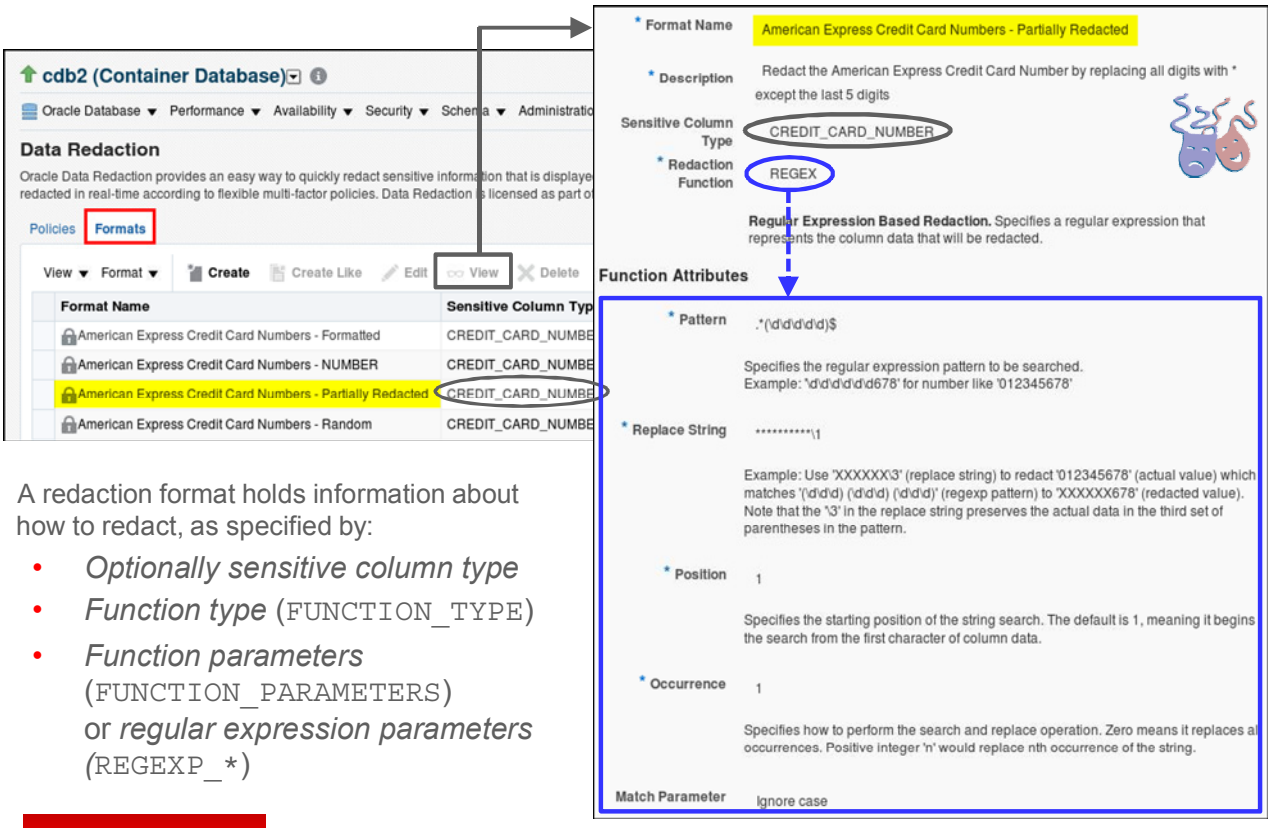
Oracle Database 12.2 and the Oracle Advanced Security option provide a Data Redaction format library that is stored in the Enterprise Manager Cloud Control repository. It contains:

- Predefined data redaction formats that are useful for various sensitive column types
- User-defined data redaction formats that are useful for any sensitive column types defined in the Application Data Model (ADM)

The formats are ready and always available for creation of Data Redaction policies. The example in the slide shows the first four predefined data redaction formats provided by Oracle Advanced Security in the Enterprise Manager Cloud Control repository. The Data Redaction format library is similar to the Data Masking format library because they both reside in the Enterprise Manager repository.

The four predefined data redaction formats are applicable on the `CREDIT_CARD_NUMBER` sensitive column type, which is discovered by the Application Data Model job discovery and is stored in the Application Data Model (ADM). They use one of the different predefined redaction function types, namely `PARTIAL`, `REGEX`, and `RANDOM`.

Viewing Data Redaction Formats



A redaction format holds information about how to redact, as specified by:

- *Optionally sensitive column type*
- *Function type* (`FUNCTION_TYPE`)
- *Function parameters* (`FUNCTION_PARAMETERS`) or *regular expression parameters* (`REGEXP_*`)

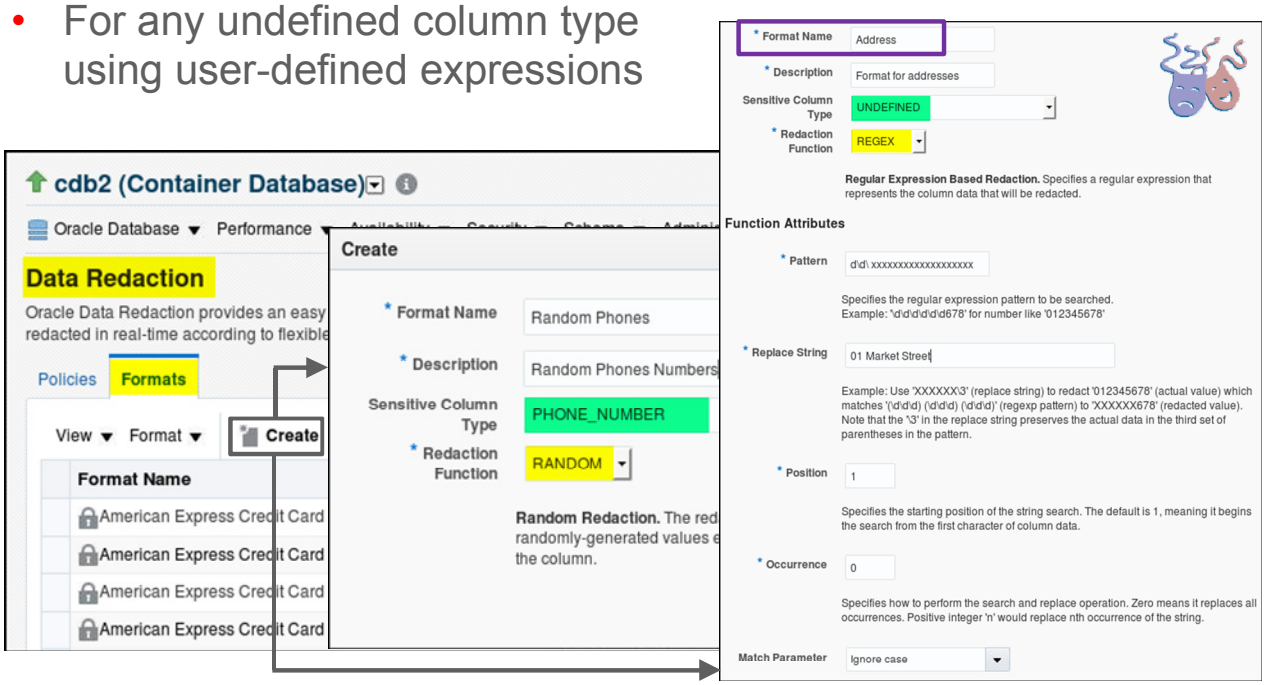
To view the attributes of predefined data redaction formats, select the format and click View.

A format holds all the attributes that you need to define to create a policy, such as:

- The function type: `FULL`, `PARTIAL`, `RANDOM`, `REGEXP`, `NONE`
- The function parameters or regular expression parameters

Creating Data Redaction Formats

- For new sensitive column types discovered in ADM
- For any undefined column type using user-defined expressions



To create a new format, connect to Enterprise Manager Cloud Control on any database.

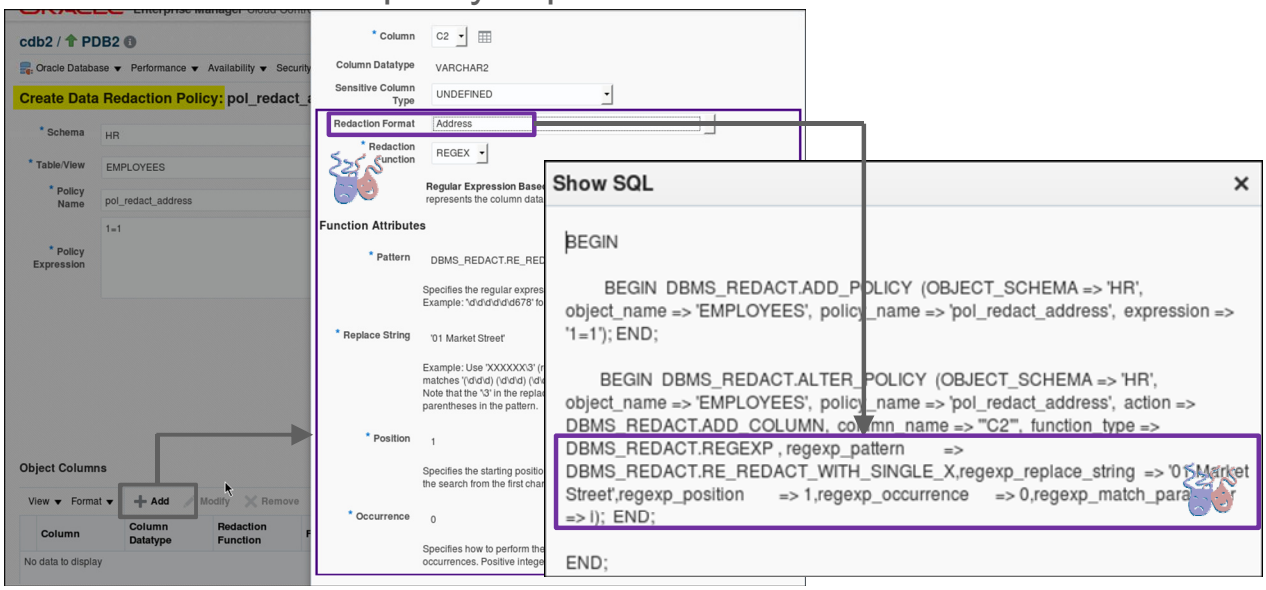
Provide the following information:

- A name for the new format
- Optionally a sensitive column type, one from the list of sensitive column types stored in ADM. If none is available or useful for the new format, specify `UNDEFINED`. The data type of the column on which the format is applied will be used at policy creation.
- The redaction function type: `FULL`, `PARTIAL`, `RANDOM`, `REGEXP`, `NONE`
- The function parameters or regular expression parameters as you would do when creating a Data Redaction policy in Oracle Database 12.1

After a format is created and stored in the EM repository, it is available for Data Redaction policy creation in any database.

Creating Data Redaction Policies Using Formats

- What to redact: schema, table name, column name
- How to redact: Data Redaction format
- When to redact: policy expression

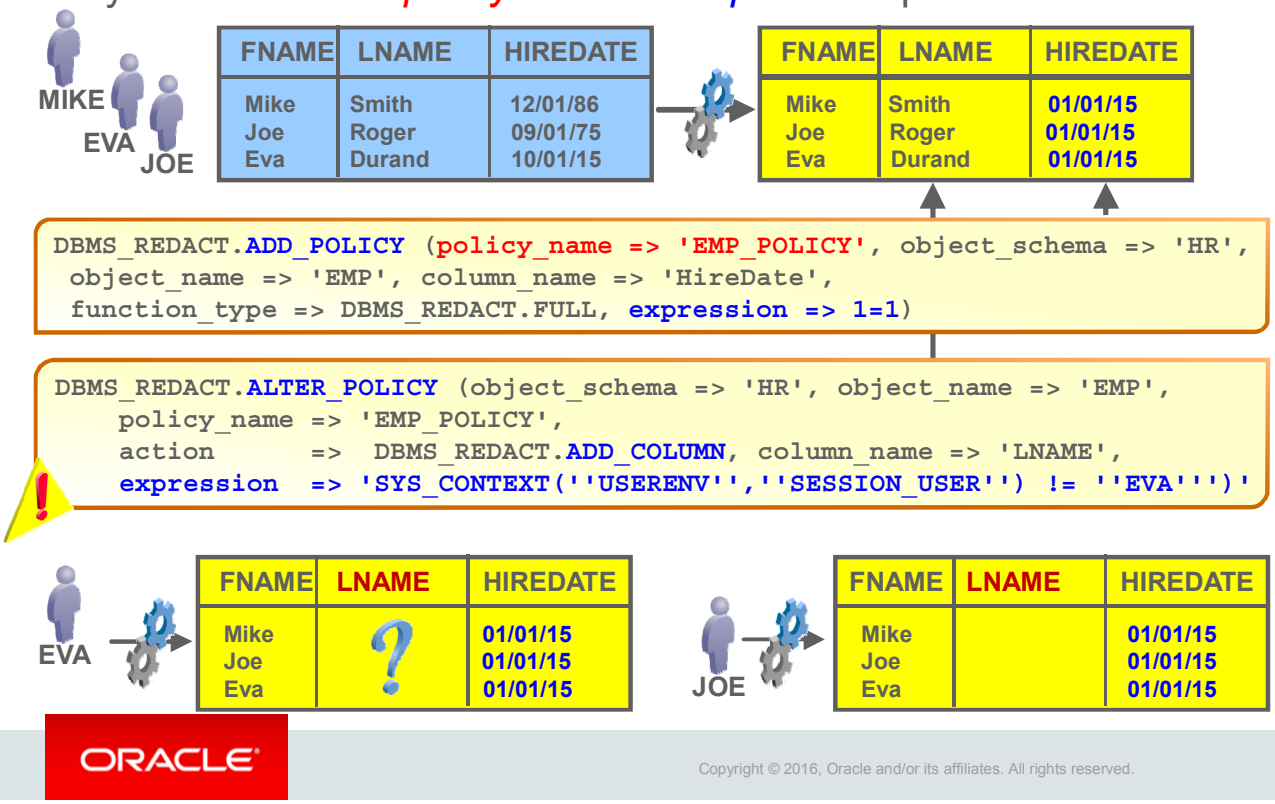


The slide shows how to create a redaction policy using Data Redaction formats from the Data Redaction library.

Policy Expression

12.1

Only one *redaction policy* and *one expression* per table:



In Oracle Database 12.1, Data Redaction allows only one redaction policy to be defined on a table or view, and that redaction policy allows only one policy expression. A policy expression is a Boolean expression for the table or view, which defines when redaction should occur, by using either the `SYS_CONTEXT` function or `1=1`. Redaction takes place only if this policy expression evaluates to `TRUE`.

In the example in the slide, an initial expression, `1=1`, is defined for the `EMP_POLICY` redaction policy. The expression applies whichever situation the connected user is in. This means that all the columns being redacted (in this case, `HIREDATE` and then `LNAME`) are to obey to the same initial expression defined at the policy creation time. Be aware that even if you add redacted columns to the policy with a different expression, the information that is taken into account is the column name and its redaction method but not the expression. If you alter the policy and modify the expression with `DBMS_REDACT.MODIFY_EXPRESSION`, the updated expression replaces the initial one. In the example in the slide, the action adds a redacted column to the policy by using `DBMS_REDACT.ADD_COLUMN`. Thus the `'SYS_CONTEXT(''USERENV'',''SESSION_USER'')!=''EVA'''` expression does not replace the initial `'1=1'`. The added `LNAME` column displays the default, FULL redacted value for the character string values, specifically NULL.

Policy Expression

12.2

- Each column can have its own *policy expression:*

```
DBMS_REDACT.CREATE_POLICY_EXPRESSION (policy_expression_name => 'EXPR_LN',
    expression => ''SYS_CONTEXT(''USERENV'',''SESSION_USER'') != ''EVA'''')
```

```
DBMS_REDACT.APPLY_POLICY_EXPR_TO_COL (object_schema => 'HR', object_name =>
    'EMP', column_name => 'LNAME', policy_expression_name => 'EXPR_LN')
```

FNAME	LNAME	HIREDATE
Mike	Smith	01/01/15
Joe	Roger	01/01/15
Eva	Durand	01/01/15

FNAME	LNAME	HIREDATE
Mike		01/01/15
Joe		01/01/15
Eva		01/01/15

- A policy expression modification automatically updates all the policies that are using that named expression.

To circumvent the Oracle Database 12.1 restriction, Oracle Database 12.2 allows the creation of policy expressions and the association of policy expressions with any individual column of a redacted table or view. Policy expressions form a kind of policy expression library. A policy expression that is applied to a column in a redacted table on which a default expression is applied by the policy does not override the default expression of the policy.

In the example in the previous slide, an initial expression, `1=1`, is defined for the `EMP_POLICY` redaction policy, and the expression applies whichever situation the connected user is in and when any column is redacted. This means that all the columns that are being redacted (in this case, `HIREDATE` and `LNAME`), are to obey the same initial expression that is defined at policy creation time. But you need to specify a different expression for the `LNAME` redacted column:

1. You create a policy expression with a new expression and no column or table defined. This policy expression can be applied to any column of any redacted table.
2. You apply the policy expression to the `LNAME` column. The FULL redaction on the column is redacted differently when `EVA` or any other user connects to query the `HR.EMPLOYEES` table and the `LNAME` column. When `EVA` connects, the redaction on the column does not take place. When a user other than `EVA` connects, the FULL redaction on the `LNAME` column applies and therefore, displays NULL.

If you change the definition of a policy expression, it automatically updates all the policies that are using that named expression.

Summary

In this lesson, you should have learned how to:

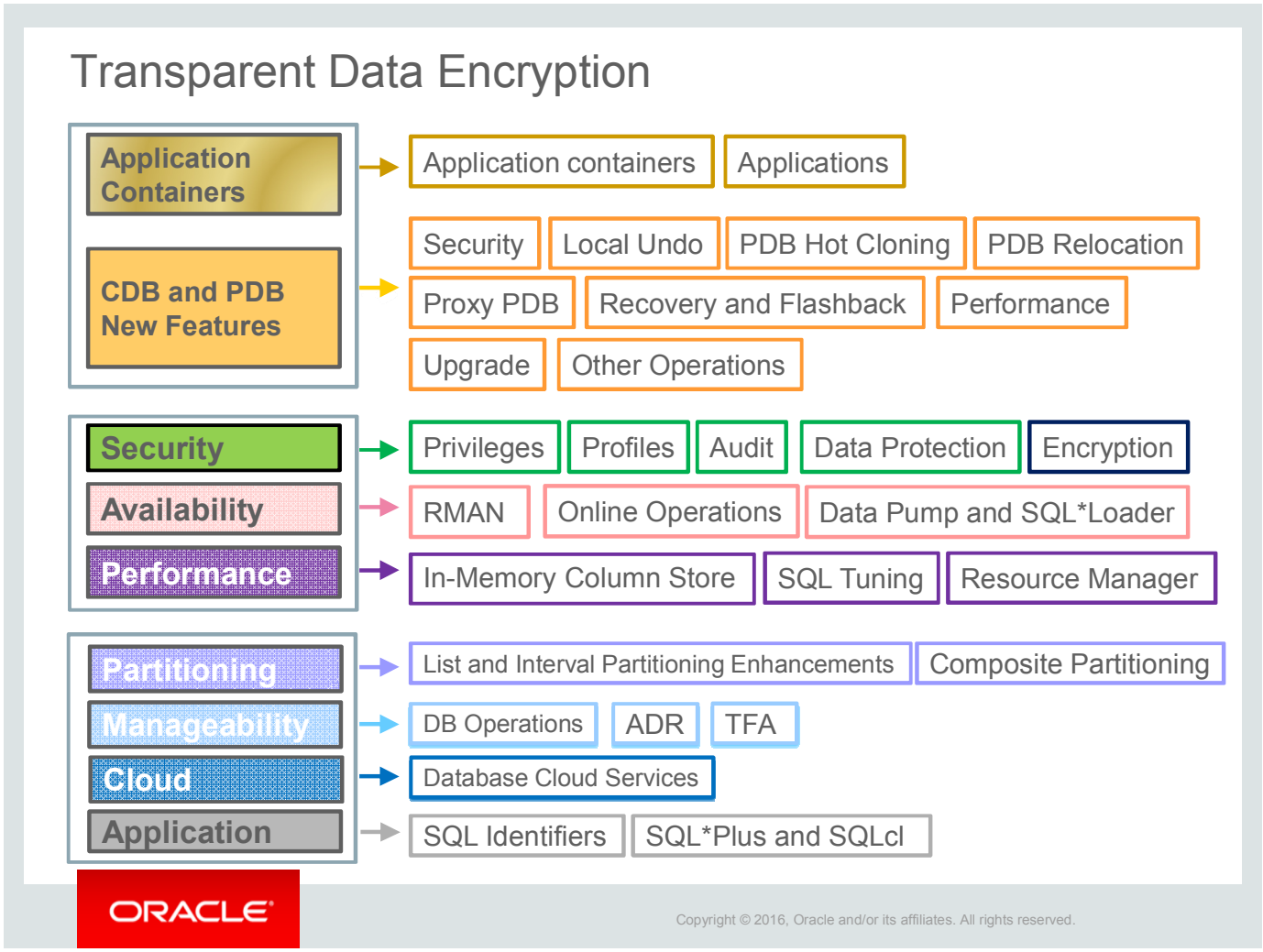
- Create and reuse data redaction formats from the format library
- Create policy expressions
- Apply policy expressions on columns to define different situations of redaction

Practice 11: Overview

- 11-1: Using data redaction formats in Data Redaction policies
- 11-2: Managing policy expressions

12

Data Encryption



This lesson explains enhancements in Transparent Data Encryption (TDE).

Objectives

After completing this lesson, you should be able to:

- Encrypt, decrypt, and re-key existing datafiles
- Encrypt, decrypt, and re-key existing tablespaces
- Accelerate tablespace encryption, decryption, and re-keying
- Configure automatic tablespace encryption for Cloud

To get detailed information about performing any of the operations explained in this lesson, refer to the following guide in Oracle documentation:

- *Oracle Database Advanced Security Guide 12c Release 2 (12.2)*

Encryption of Existing Datafiles

- SYSTEM, SYSAUX, and UNDO datafiles are encryptable only in mount mode.
- Offline is the default option and there is no online option.
- The tablespace can be made online if all datafiles are encrypted.
- When encryption is issued on the primary database, a warning is printed in the alert log to encrypt on the standby.
- Parallelism can be achieved by encrypting at the file level in multiple SQL sessions.

```
Session1 SQL> ALTER DATABASE DATAFILE 'tbsfile1.dbf' ENCRYPT;
```

```
Session2 SQL> ALTER DATABASE DATAFILE 'tbsfile2.dbf' ENCRYPT;
```

- Temporary tablespaces cannot be converted as encrypted.
- Temporary tablespaces can be created as encrypted.

The Oracle Database 12.2 capability offers the possibility to encrypt existing tablespaces.

- User-defined tablespace datafiles need to be offline during the encrypt operation.
- Encryption of non-user tablespace datafiles, such as SYSTEM, SYSAUX, and UNDO, can be performed only when the database is in mount mode. Oracle recommends against encrypting the data files of an undo tablespace. Doing so prevents the keystore from being closed, which prevents the database from functioning. Furthermore, this practice is unnecessary because all undo records that are associated with an encrypted tablespace are already automatically encrypted in the undo tablespace.

This offline encrypt solution, despite the impact of availability, has the advantage of encrypting existing files in-place, with no extra disk storage required.

Moreover, you can parallelize the operation at the datafile level. Then you must encrypt all the files for a given tablespace to put the tablespace back online.

In a Data Guard configuration, the DBA can mitigate down time by encrypting the standby first, switch over, and then encrypt the original primary.

You can create encrypted temporary tablespaces but existing temporary tablespaces cannot be converted as encrypted.

Conversion of Tablespaces: ONLINE Encryption

- Encryption of an existing tablespace: ONLINE by default

```
SQL> ALTER TABLESPACE tbsnoenc ENCRYPTION ONLINE USING 'AES256' ENCRYPT
          FILE_NAME_CONVERT = ('tbsnoenc01.dbf', 'tbsenc01.dbf') ;
```

- OFFLINE keyword:
  - No encryption attribute after ENCRYPTION (*implicitly uses the default database key with AES128*)
  - No extra disk space required

```
SQL> ALTER TABLESPACE tbsnoenc ENCRYPTION OFFLINE ENCRYPT;
```

- Decryption and re-keying an existing tablespace: ONLINE only

```
SQL> ALTER TABLESPACE tbsnoenc ENCRYPTION DECRYPT;
```

```
SQL> ALTER TABLESPACE tbsnoenc ENCRYPTION ONLINE REKEY;
```

- Parallelism achieved by encrypting at the tablespace level in multiple SQL sessions

- In Oracle Database 12.2, ALTER TABLESPACE can convert user-defined tablespaces as encrypted tablespaces, either online or offline. The best use case for OFFLINE encryption is enabling encryption within a maintenance window as fast as possible and later using the ONLINE re-key to switch algorithm.
- The SYSTEM and SYSAUX tablespaces cannot be created as encrypted tablespaces because of bootstrapping issues. You need to create the wallet and a master key first to enable encryption, but the master key can be created only with the database open. In Oracle Database 12.2, ALTER TABLESPACE can convert SYSTEM and SYSAUX tablespaces into encrypted tablespaces. If any of the non-user tablespaces, such as SYSTEM, SYSAUX, or UNDO, are encrypted, the wallet or key store cannot be closed anymore, or the database will not function.
- Encrypting an existing temporary tablespace is not possible; however, you can create a temporary tablespace as encrypted.
- Decrypting UNDO tablespaces is not recommended.
- Re-keying is available for all encrypted tablespaces.
- Parallelism for online conversion can be achieved when using multiple foreground sessions to encrypt different tablespaces, although within each tablespace, the datafiles are converted sequentially.

If an ONLINE encryption, decryption, or rekeying command fails to complete, execute the FINISH command.

Encryption Algorithms

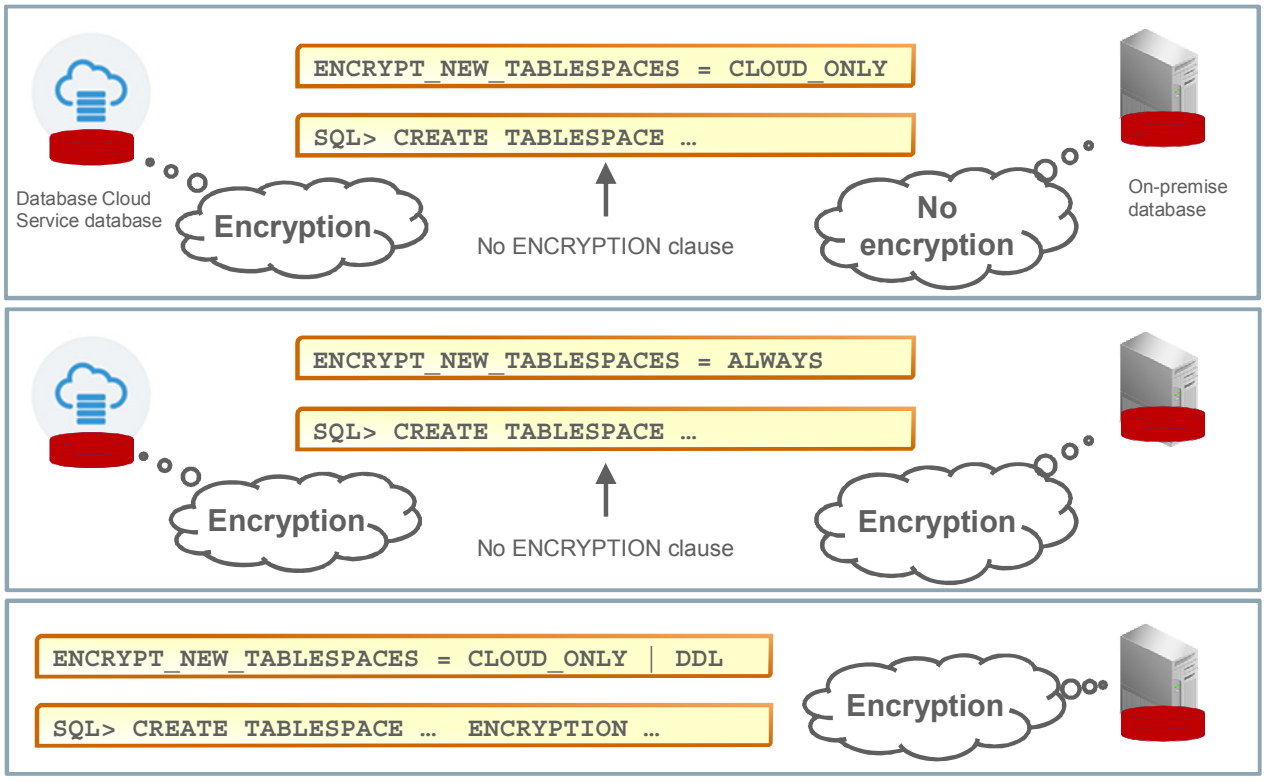
Pre-12c

- 3DES168
- AES128 (default)
- AES192
- AES256

12.2

- ARIA128
- ARIA192
- ARIA256
- SEED128
- GOST256

Automatic Tablespace Encryption



Oracle Database 12.2 allows encryption of new user-defined tablespaces via a new `ENCRYPT_NEW_TABLESPACES` instance parameter.

- A user-defined tablespace that is created in the Cloud is transparently encrypted with Advanced Encryption Standard 128 (AES 128) if the `ENCRYPTION` clause for the SQL `CREATE TABLESPACE` statement is not specified.
- A user-defined tablespace that is created in an on-premise database is not transparently encrypted. Only the `ENCRYPTION` clause of the `CREATE TABLESPACE` statement determines if the tablespace is encrypted.

The `ENCRYPT_NEW_TABLESPACES` instance parameter can be set to three values:

- `CLOUD_ONLY` (the default value): The behavior explained in the preceding paragraphs is enabled.
- `ALWAYS`: Whether the user-defined tablespace is created in the Cloud or in an on-premise database, the tablespace is transparently encrypted with AES128 if the `ENCRYPTION` clause is not specified in the `CREATE TABLESPACE` statement.
- `DDL`: Whether the user-defined tablespace is created in the Cloud or in an on-premise database, the `ENCRYPTION` clause of the `CREATE TABLESPACE` statement determines if the tablespace is encrypted with AES 128.

Summary

In this lesson, you should have learned how to:

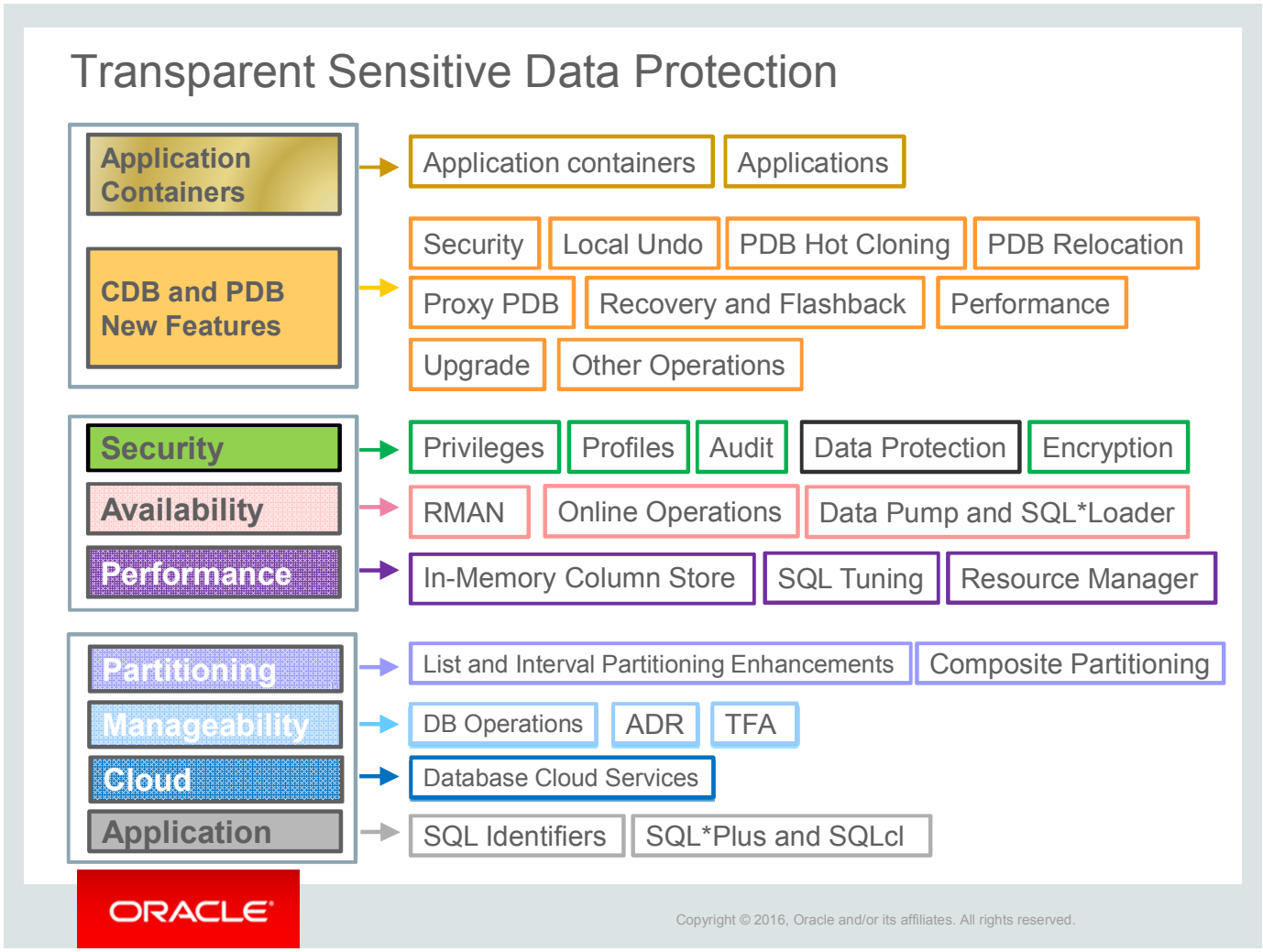
- Encrypt, decrypt, and re-key existing datafiles
- Encrypt, decrypt, and re-key existing tablespaces
- Accelerate tablespace encryption, decryption, and re-keying
- Configure automatic tablespace encryption

Practice 12: Overview

- 12-1: Encrypting essential and temporary tablespaces
- 12-2: Decrypting existing tablespaces online in a PDB

13

Transparent Sensitive Data Protection



This lesson explains enhancements in Transparent Sensitive Data Protection (TSDP).

Objectives

After completing this lesson, you should be able to use TSDP policies with:

- Unified Auditing
- Fine-grained Auditing
- Column encryption

To get detailed information about performing any of the operations explained in this lesson, refer to the following guides in Oracle documentation:

- *Oracle Database Security Guide 12c Release 2 (12.2)*

Benefits of TSDP

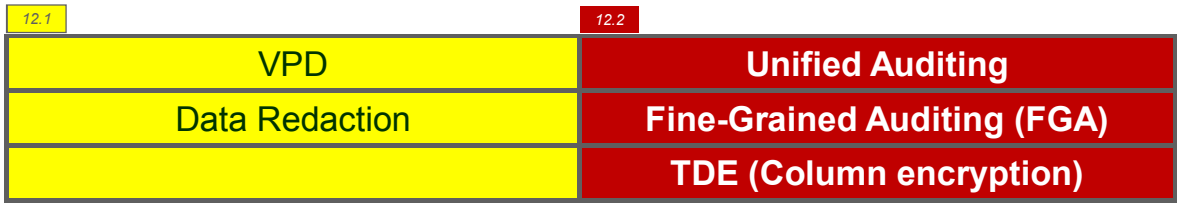
- Configure sensitive data protection once, and then deploy this protection.
  - Specify the target data when necessary.
  - Add new sensitive columns that match the sensitive type.
  - Export or import the TSDP policies into another database.
- Manage protection of multiple sensitive columns based on:
  - An ADM source
  - A sensitive type
  - A specific schema, table, or column

TSDP automatically protects sensitive information as per the data security compliance guidelines in databases across an enterprise.

- You can configure TSDP policies to designate how a class of data (for example, credit card numbers) must be protected without actually having to specify the target data. You do not need to include references to the actual target columns that you want to protect when you create the TSDP policy. The TSDP policy finds these target columns based on a list of sensitive columns in the database and the policy's associations with the specified sensitive types. This can be useful when you add more sensitive data to your databases after you have created the TSDP policies. Deploy the TSDP policy by exporting the policy to or importing the policy from another database.
- You can enable or disable protection for multiple sensitive columns based on a suitable attribute (such as the source database of the identification, the sensitive type itself, or a specific schema, table, or column).
- Data Discovery and Modeling discovers sensitive columns in databases, and stores these columns, categorized by type, in Application Data Models (ADMs). Using EM Cloud Control, use export to TSDP catalog to copy sensitive columns and sensitive column types to TSDP-enabled databases and in your databases, import the sensitive types and columns from the catalog.

Protection with Auditing and TDE

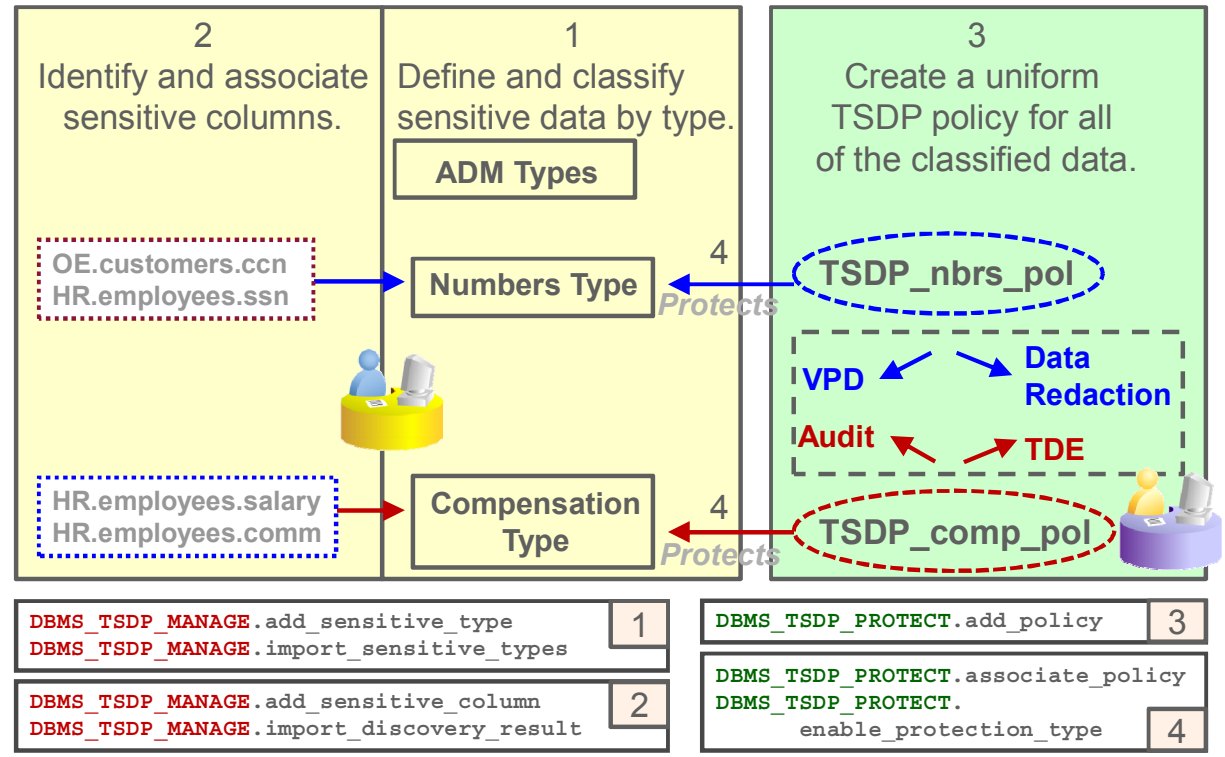
- Protect the sensitive columns discovered by the database or ADM with TSDP policies.
- TSDP supports Oracle Database security features:



  - Data Redaction and VPD, which are access control features that prevent access to sensitive data for low-privileged users
  - Unified Auditing

- Create TSDP policies that protect sensitive data in table columns by using either Oracle Data Redaction or Oracle Virtual Private Database settings, and then enable the TSDP policies.
- With Oracle Database 12.2, TSDP policies can protect sensitive data in table columns by using Unified Auditing, Fine-Grained Auditing (FGA), or Transparent Data Encryption (TDE) settings.

TSDP Overview



TSDP allows the application administrator to use the DBMS_TSDP_MANAGE procedures to:

1. Define and classify sensitive data by creating sensitive types or by retrieving the list of discovered sensitive column types in Enterprise Manager Cloud Control Application Data Modeling (ADM). The SOURCE_TYPE column in the DBA_SENSITIVE_COLUMN_TYPES view displays either DB or ADM. If you manually create the types, use the DBMS_TSDP_MANAGE.ADD_SENSITIVE_TYPE procedure. If you use an ADM to retrieve the existing types, use the DBMS_TSDP_MANAGE.IMPORT_SENSITIVE_TYPES procedure to import the list of types.
2. Find all the table columns in a database that hold sensitive data (such as credit card or Social Security Numbers) and associate the sensitive columns to the sensitive types. If you manually identify the columns, use the DBMS_TSDP_MANAGE.ADD_SENSITIVE_COLUMN procedure. If you use an ADM, use the DBMS_TSDP_MANAGE.IMPORT_DISCOVERY_RESULT procedure to import the list of sensitive columns. The association of table columns to a TSDP sensitive type can be completed before or after policy creation.

TSDP allows the security administrator to use the DBMS_TSDP_PROTECT procedures to:

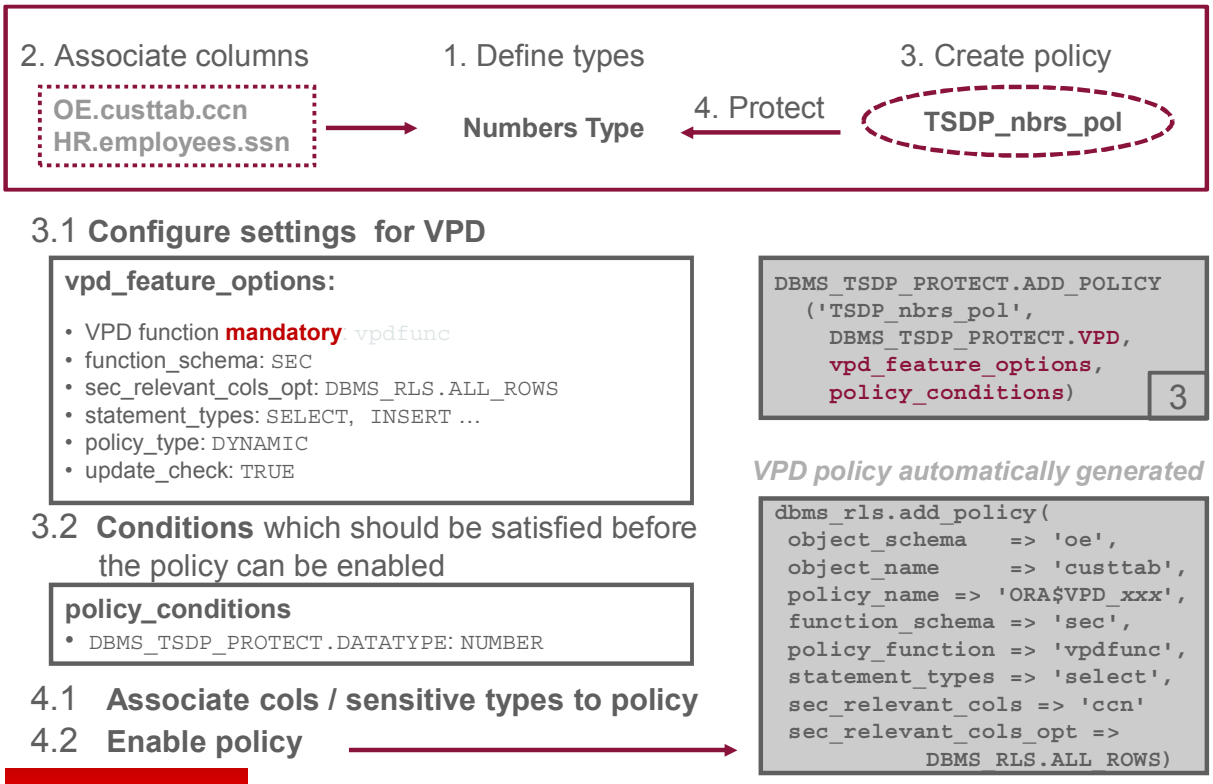
3. Create a policy that protects data as a whole for a given class. Use the DBMS_TSDP_PROTECT.ADD_POLICY procedure. The TSDP policy protects sensitive data in table columns by using either Oracle Data Redaction or Oracle Virtual Private Database settings. These settings are allowed with Oracle Database 12.1. You can create a uniform TSDP policy for all of the data that you classify, and then modify this policy as necessary, as compliance regulations change. More settings are allowed with Oracle Database 12.2 such as Unified Auditing, Fine-Grained Auditing (FGA), or Column Encryption (TDE).
4. Associate sensitive table types to TSDP policies. Use the ASSOCIATE_POLICY procedure.

   Finally you can enable the TSDP policy protections at different levels by using the following procedures:

   - ENABLE_PROTECTION_SOURCE to protect all sensitive columns of an ADM
   - ENABLE_PROTECTION_TYPE to protect all sensitive columns of a sensitive type
   - ENABLE_PROTECTION_COLUMN to protect all sensitive columns of a specific schema and/or tables and/or columns

Enabling the TSDP policy either creates the VPD policies, Data Redaction policies, Unified Audit policies, or FGA policies or adds encryption on the protected objects and columns that are associated with sensitive types.

Using a TSDP Policy with VPD



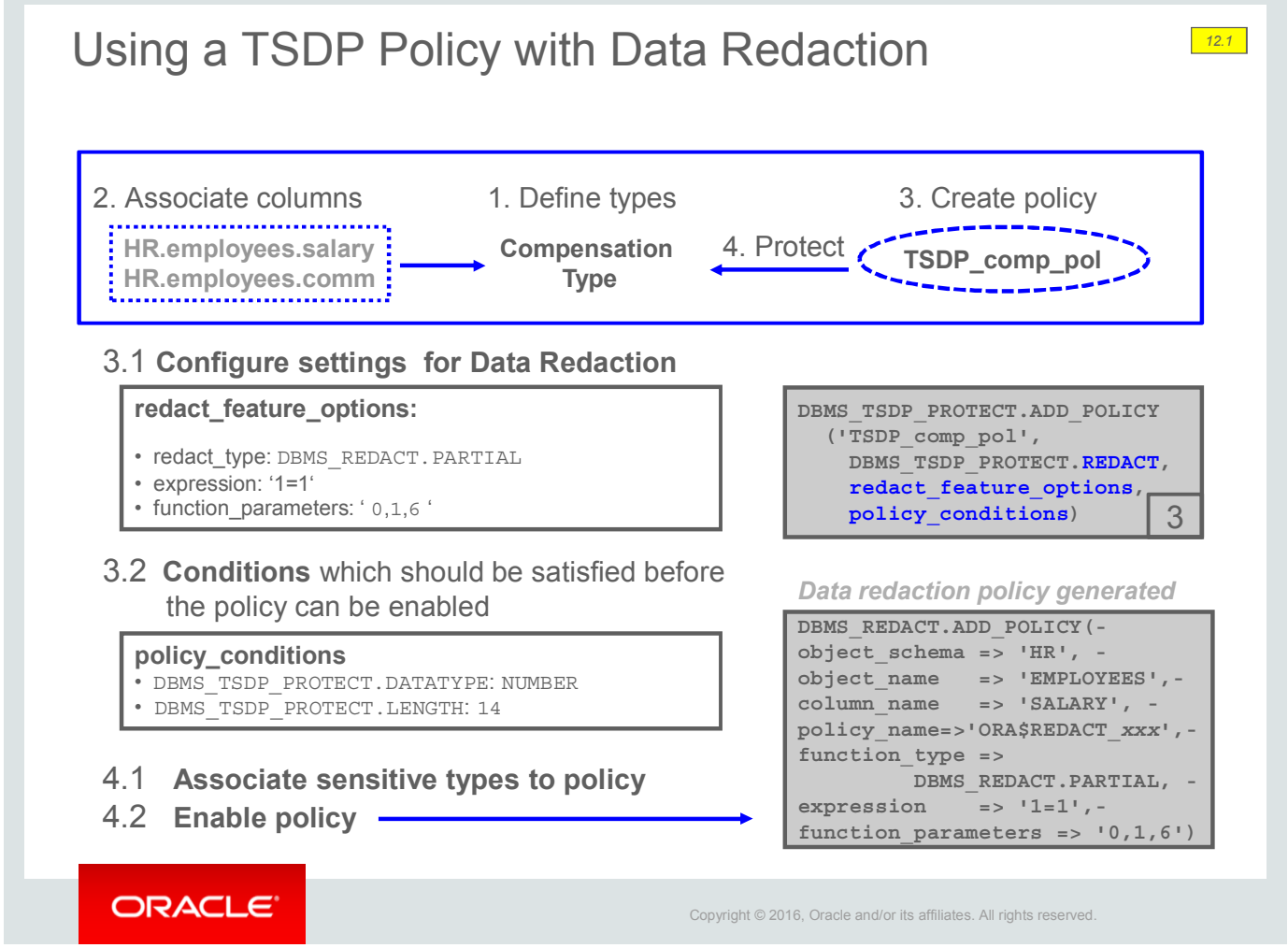
You can incorporate Oracle VPD protection with TSDP policies. This requires that a VPD function is created before you configure the settings for the VPD in the TSDP creation.

- You create a TSDP policy with the necessary VPD settings similar to the VPD policy function. The TSDP policy uses `vpd_feature_options` settings from the `DBMS_RLS.ADD_POLICY` procedure to provide VPD protection. For example, you can set the `sec_relevant_cols_opt`, `statement_types` to values that will be used in the VPD policy creation when later the TSDP policy is associated and enabled with sensitive types. You can also set policy conditions to test when the policy is enabled. For example, in the example of the slide, the data type of the protected columns should be `NUMBER`.
- You associate the TSDP policy with the necessary sensitive types by using the `DBMS_TSDP_PROTECT.ASSOCIATE_POLICY` procedure.
- You enable the TSDP policy by using any of the `DBMS_TSDP_PROTECT.ENABLE_PROTECTION_SOURCE`, `ENABLE_PROTECTION_TYPE`, `ENABLE_PROTECTION_COLUMN` procedures. This automatically generates a VPD policy using the settings defined in the `vpd_feature_options` argument. Notice that the `sec_relevant_cols` parameter (of `DBMS_RLS.ADD_POLICY`) is not set in the TSDP policy: it is set to the name of the sensitive columns on which TSDP enables the VPD policy. Enabling protection enforces the VPD policies on all columns identified as `NUMBERS` type.

When users query the tables, the output for the columns is based on both the VPD protections **and** the TSDP policy that are in place. Disabling the TSDP policy automatically drops the VPD policy for the type or column. Reenabling the TSDP policy recreates the VPD policy.

In the example on this slide, the TSDP policy is enabled on the `OE.CUSTTAB.CCN` and `HR.EMPLOYEES.SSN` columns if these columns are of the `NUMBER` data type and have a length of `14`. Even though the `OE.CUSTTAB.CCN` and `HR.EMPLOYEES.SSN` columns are associated with the TSDP policy, the policy is not enabled if any of these columns fail to satisfy the conditions (data type and length).

Using a TSDP Policy with Data Redaction

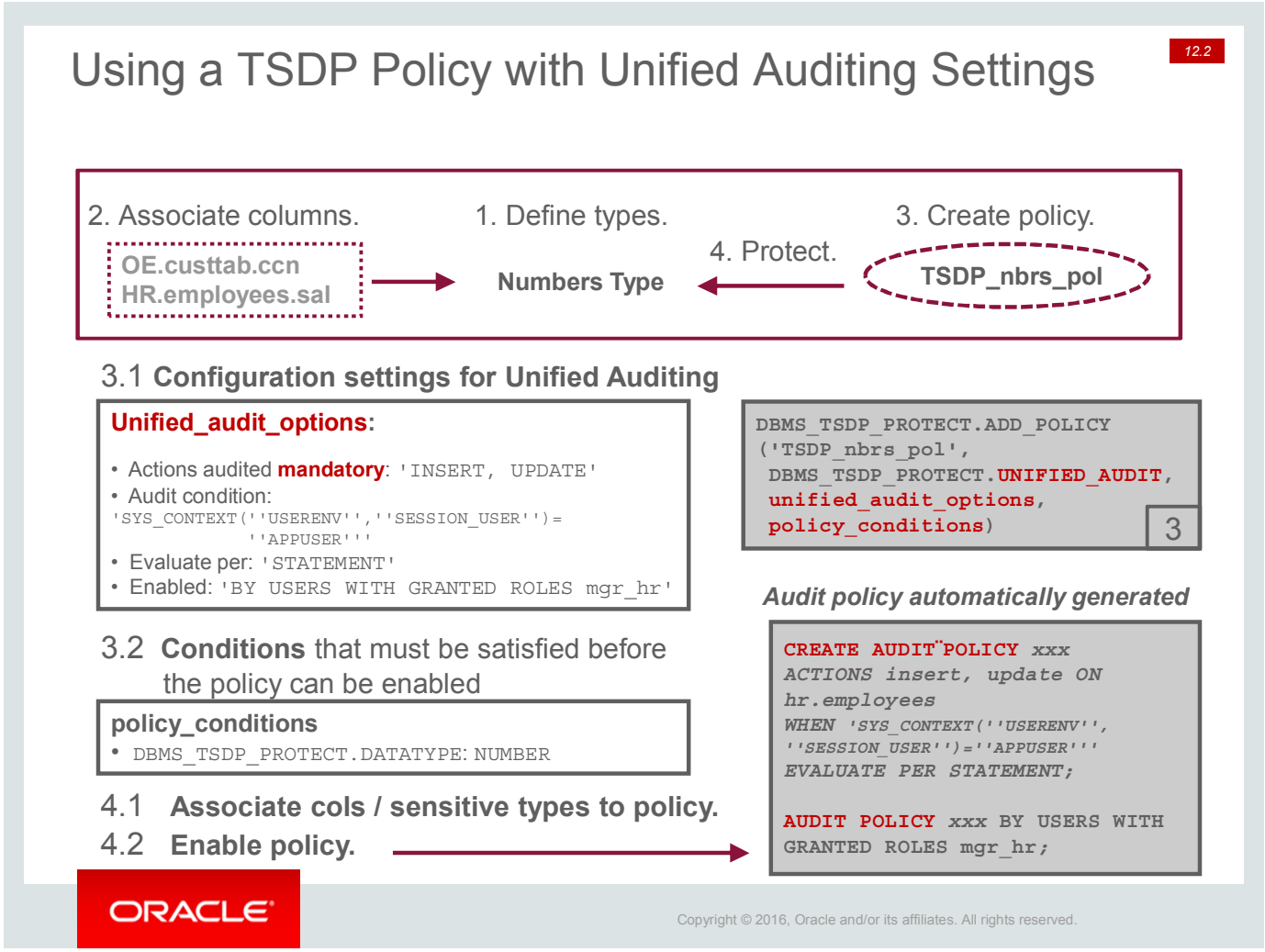


You can incorporate Oracle Data Redaction protection with TSDP policies.

- You create a TSDP policy with the necessary redaction settings similar to the data redaction policy function. The TSDP policy uses `redact_feature_options` settings from the `DBMS_REDACT.ADD_POLICY` procedure to provide data redaction protection. For example, you can set the `function_type`, `function_parameters` to values that will be used in the redaction policy creation when later the TSDP policy is associated and enabled with sensitive types. You can also set policy conditions to test when the policy is enabled. For example, in the example on this slide, the data type of the protected columns should be `NUMBER` with a length of `14`.
- You associate the TSDP policy with the necessary sensitive types by using the `DBMS_TSDP_PROTECT.ASSOCIATE_POLICY` procedure.
- You enable the TSDP policy by using any of the `DBMS_TSDP_PROTECT.ENABLE_PROTECTION_SOURCE`, `ENABLE_PROTECTION_TYPE`, `ENABLE_PROTECTION_COLUMN` procedures. This automatically generates a redaction policy using the settings defined in the `redact_feature_options` argument. Notice that the `column_name` parameter (of `DBMS_REDACT.ADD_POLICY`) is not set in the TSDP policy: it is set to the name of the sensitive columns on which TSDP enables the data redaction policy. Enabling protection enforces the data redaction policies on all columns identified as `COMPENSATION` type.

When users query the tables, the output for the columns is based on both the data redaction protections **and** the TSDP policy that are in place. Disabling the TSDP policy automatically drops the data redaction policy for the type or column. Reenabling the TSDP policy recreates the data redaction policy.

In the example of the slide, the TSDP policy is enabled on the `HR.EMPLOYEES.SALARY` and `HR.EMPLOYEES.COMM` columns if these columns are of the `NUMBER` data type and have a length of `14`. Even though the `HR.EMPLOYEES.SALARY` and `HR.EMPLOYEES.COMM` columns are associated with the TSDP policy, the policy is not enabled if any of these columns fail to satisfy the conditions (data type and length).

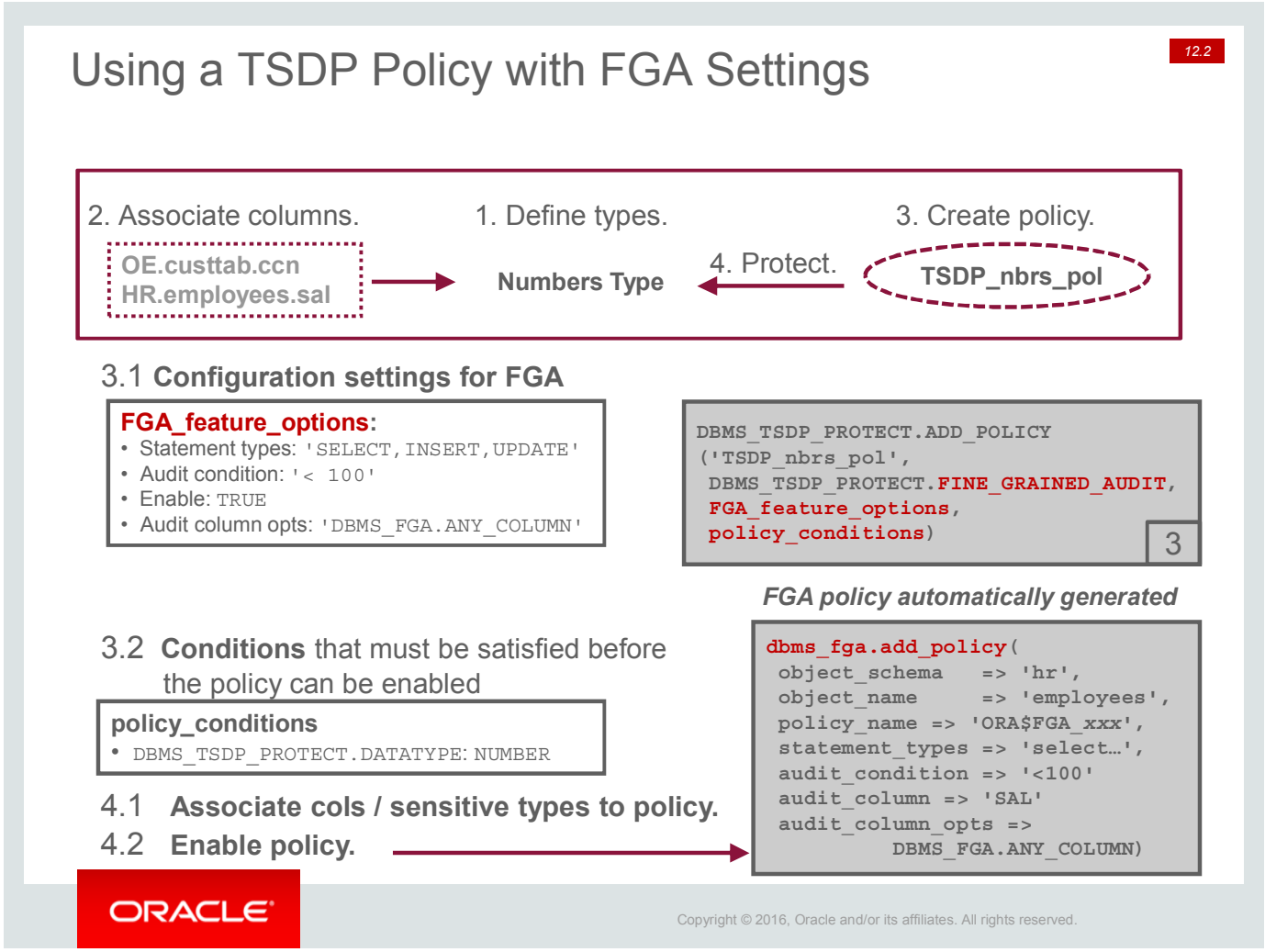


You can incorporate Oracle Unified Auditing with TSDP policies.

- You create a TSDP policy with the necessary Unified Audit settings similar to the audit policy. The TSDP policy uses the `unified_audit_options` settings to create and enable a Unified Audit policy. For example, you can set the action types and audit condition to values that will be used in audit policy creation when later the TSDP policy is associated and enabled with sensitive types. You can also set policy conditions to test when the policy is enabled. For example, in the slide, the data type of the protected columns should be `NUMBER`. A TSDP policy that is enabled on a group of objects (based on sensitive column type or the source of identification) translates into a single Unified Audit policy because they have the same auditing requirements.
- You associate the TSDP policy with the necessary sensitive types by using the `DBMS_TSDP_PROTECT.ASSOCIATE_POLICY` procedure.
- You enable the TSDP policy by using any of the `DBMS_TSDP_PROTECT.ENABLE_PROTECTION_SOURCE`, `ENABLE_PROTECTION_TYPE`, or `ENABLE_PROTECTION_COLUMN` procedures. This automatically generates and enables an audit policy by using the settings defined in the `unified_audit_options` argument. Enabling protection enforces the audit policies on all columns that are identified as the `NUMBERS` type.

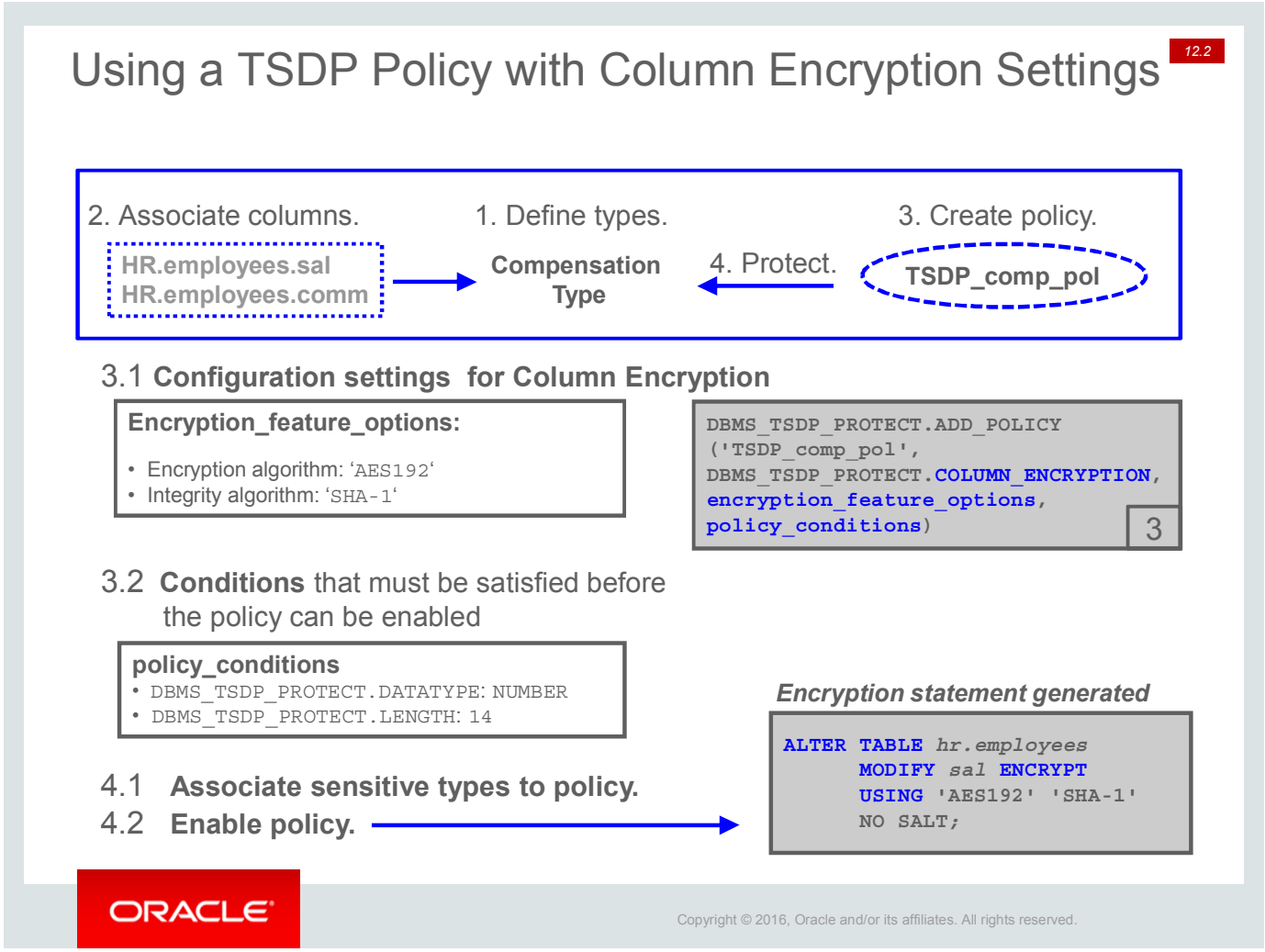
When users insert or update rows in the tables, the statements are automatically audited. Disabling the TSDP policy automatically disables and drops the audit policy for the type or column. Re-enabling the TSDP policy re-creates and enables the audit policy.

In the example in the slide, the TSDP policy is enabled on the `OE.CUSTTAB.CCN` and `HR.EMPLOYEES.SAL` columns if these columns are of the `NUMBER` data type. Even though the `OE.CUSTTAB.CCN` and `HR.EMPLOYEES.SAL` columns are associated with the TSDP policy, the policy is not enabled if any of these columns fail to satisfy the conditions (data type).



You can incorporate FGA with TSDP policies.

- You create a TSDP policy with the necessary FGA settings similar to the FGA policy. The TSDP policy uses the `FGA_feature_options` settings from the `DBMS_FGA.ADD_POLICY` procedure to create an FGA policy. For example, you can set the audit column and audit condition to values that will be used in FGA policy creation when later the TSDP policy is associated and enabled with sensitive types. You can also set policy conditions to test when the policy is enabled. For example, in the example in the slide, the data type of the protected columns should be `NUMBER`.
- You associate the TSDP policy with the necessary sensitive types by using the `DBMS_TSDP_PROTECT.ASSOCIATE_POLICY` procedure.
- You enable the TSDP policy by using any of the `DBMS_TSDP_PROTECT.ENABLE_PROTECTION_SOURCE`, `ENABLE_PROTECTION_TYPE`, or `ENABLE_PROTECTION_COLUMN` procedures. This automatically generates an FGA policy by using the settings defined in the `FGA_feature_options` argument. Enabling protection enforces the FGA policies on all columns that are identified as the `NUMBERS` type.



You can incorporate column TDE encryption with TSDP policies.

- You create a TSDP policy that is configured for column encryption to tell TSDP that when the policy is enabled on an object, it needs to enable column encryption on the sensitive columns. The TSDP policy uses the `encryption_feature_options` settings to enable column encryption on the protected columns. For example, you can set the encryption algorithm to a value that will be used in column encryption when later the TSDP policy is associated and enabled with sensitive types.
- You associate the TSDP policy with the necessary sensitive types by using the `DBMS_TSDP_PROTECT.ASSOCIATE_POLICY` procedure.
- You enable the TSDP policy by using any of the `DBMS_TSDP_PROTECT.ENABLE_PROTECTION_SOURCE`, `ENABLE_PROTECTION_TYPE`, or `ENABLE_PROTECTION_COLUMN` procedures. This automatically alters the columns by using the settings defined in the `encryption_feature_options` argument. Enabling protection enforces encryption on all columns that are identified as the `COMPENSATION` type.

When users access the tables, both encryption protection and the TSDP policy are in place. Disabling the TSDP policy automatically removes the `ENCRYPT` attribute from columns of the `COMPENSATION` type. Re-enabling the TSDP policy resets the `ENCRYPT` attribute on columns of the `COMPENSATION` type.

In the example in the slide, the TSDP policy is enabled on the `HR.EMPLOYEES.SAL` and `HR.EMPLOYEES.COMM` columns if these columns are of the `NUMBER` data type and have a length of `14`. Even though the `HR.EMPLOYEES.SAL` and `HR.EMPLOYEES.COMM` columns are associated with the TSDP policy, the policy is not enabled if any of these columns fail to satisfy the conditions (data type and length).

Using the Predefined REDACT_AUDIT TSDP Policy

Use the predefined REDACT_AUDIT TSDP policy to mask bind values with an '*':

- Bind variables and sensitive columns in the expressions of conditions

```
SELECT count(*) FROM hr.emp WHERE sal IN (:VAR1, :VAR2);
```

```
SELECT emp_id FROM hr.emp
WHERE  sal > (SELECT sal FROM hr.emp WHERE mgr_id = :VAR1);
```

- A bind variable and a sensitive column appearing in the same SELECT item

```
SELECT (sal*:VAR1), (comm*:VAR2), (emp_id+:VAR3)
FROM    payroll.account;
```

- Bind variables in expressions assigned to sensitive columns in INSERT or UPDATE operations

```
UPDATE payroll.account SET acct_num = :VAR1, sal= :VAR2;
```

ORACLE

The bind values of the bind variables that are used in SQL statements can appear in the following places:

- The audit records when auditing is configured
- The trace files when the appropriate event is set
- The V$SQL_BIND_DATA dynamic view

The predefined REDACT_AUDIT TSDP policy masks bind values of bind variables that are considered to be sensitive or "associated" with sensitive columns with an '*' value.

By default the REDACT_AUDIT policy is associated with every sensitive type in the database.

Bind variables are considered to be sensitive or "associated" with sensitive columns in the cases listed in the slide.

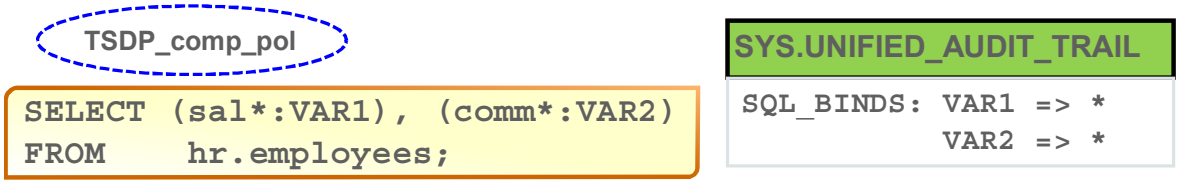
- In comparison conditions, a sensitive column and a bind variable appear in the expressions that are being compared. In the first example, the sensitive column and the bind variable are the arguments of the IN condition. The bind values in VAR1 and VAR2 are masked because VAR1, VAR2, and SAL appear as arguments to the IN condition. The second example shows a condition with a nested subquery as an argument. The bind variables and sensitive columns that appear in the nested subquery are not considered to be associated with the condition.

- If a column in a SELECT item is sensitive, then all the binds in the SELECT item are considered sensitive. The third example shows a bind variable VAR1 considered sensitive because it appears in the same SELECT item as SAL. VAR2 is considered sensitive because it appears in the same SELECT item as the sensitive column COMM.
- You can assign multiple bind variables to different columns in one INSERT or UPDATE statement.

Disabling the REDACT_AUDIT TSDP Policy

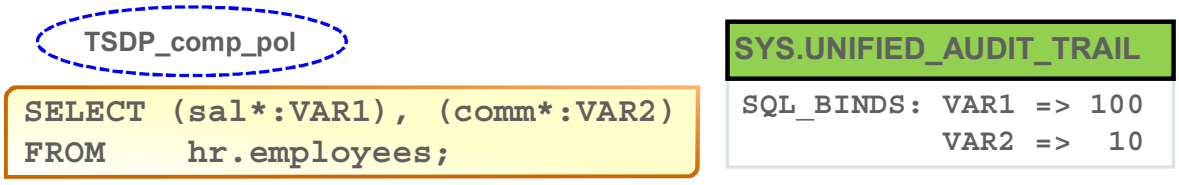
Change the behavior of the predefined REDACT_AUDIT TSDP policy.

- Masking bind variables is the default behavior.



- Disabling the policy displays the bind values.

```
DBMS_TSDP_PROTECT.DISABLE_PROTECTION_COLUMN ( policy => ' REDACT_AUDIT' )
```



The predefined REDACT_AUDIT TSDP policy masks bind values of bind variables that are considered to be sensitive or "associated" with sensitive columns with an '*' because each REDACT_AUDIT is enabled.

You can disable it for a specific sensitive column or all sensitive columns using the DBMS_TSDP_PROTECT.DISABLE_PROTECTION_COLUMN procedure as follows:

- Disable the policy for all sensitive columns:

```
SQL> exec DBMS_TSDP_PROTECT.DISABLE_PROTECTION_COLUMN -
          ( policy => 'REDACT_AUDIT')
```

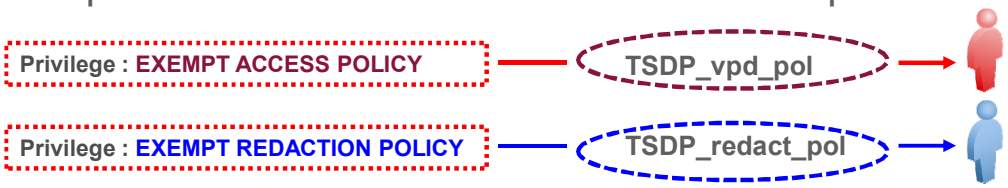
- Disable the policy for a sensitive column:

```
SQL> exec DBMS_TSDP_PROTECT.DISABLE_PROTECTION_COLUMN -
          ( schema_name => 'HR', -
            table_name  => 'EMPLOYEES', -
            column_name => 'SAL', -
            policy      => 'REDACT_AUDIT')
```

As soon as you re-enable the REDACT_AUDIT policy, the values for the bind variables are masked with an '*'.

Exempting Users from TSDP Policies

- Conditions included in policy expressions may allow users to see actual data.
- SYS is exempt from all TSDP policies.
- Grant a system privilege to exempt other users from all TSDP policies associated to VPD or redaction policies.



- Best Practices:
  - Use default deny (white list) conditions in policy expressions.
  - Grant the EXEMPT ACCESS/REDACTION POLICY privilege judiciously to ensure that the VPD or redaction policies are enforced appropriately.

Because the SYS user is exempt from VPD and redaction policies, he is able to view actual values for data protected by TSDP policies.

If other users need access to the actual values, you must grant them either the EXEMPT ACCESS POLICY system privilege or the EXEMPT REDACTION POLICY system privilege. These privileges exempt the users from all VPD and redaction policies.

Users granted the DBA role are exempt from redaction policies but not from VPD policies because the DBA role contains the EXP_FULL_DATABASE role, which is granted the EXEMPT REDACTION POLICY system privilege.

Because applications may need to perform CREATE TABLE AS SELECT operations that involve redacted source columns, you can grant the application the EXEMPT DDL REDACTION POLICY system privilege.

Summary

In this lesson, you should have learned how to use TSDP policies with:

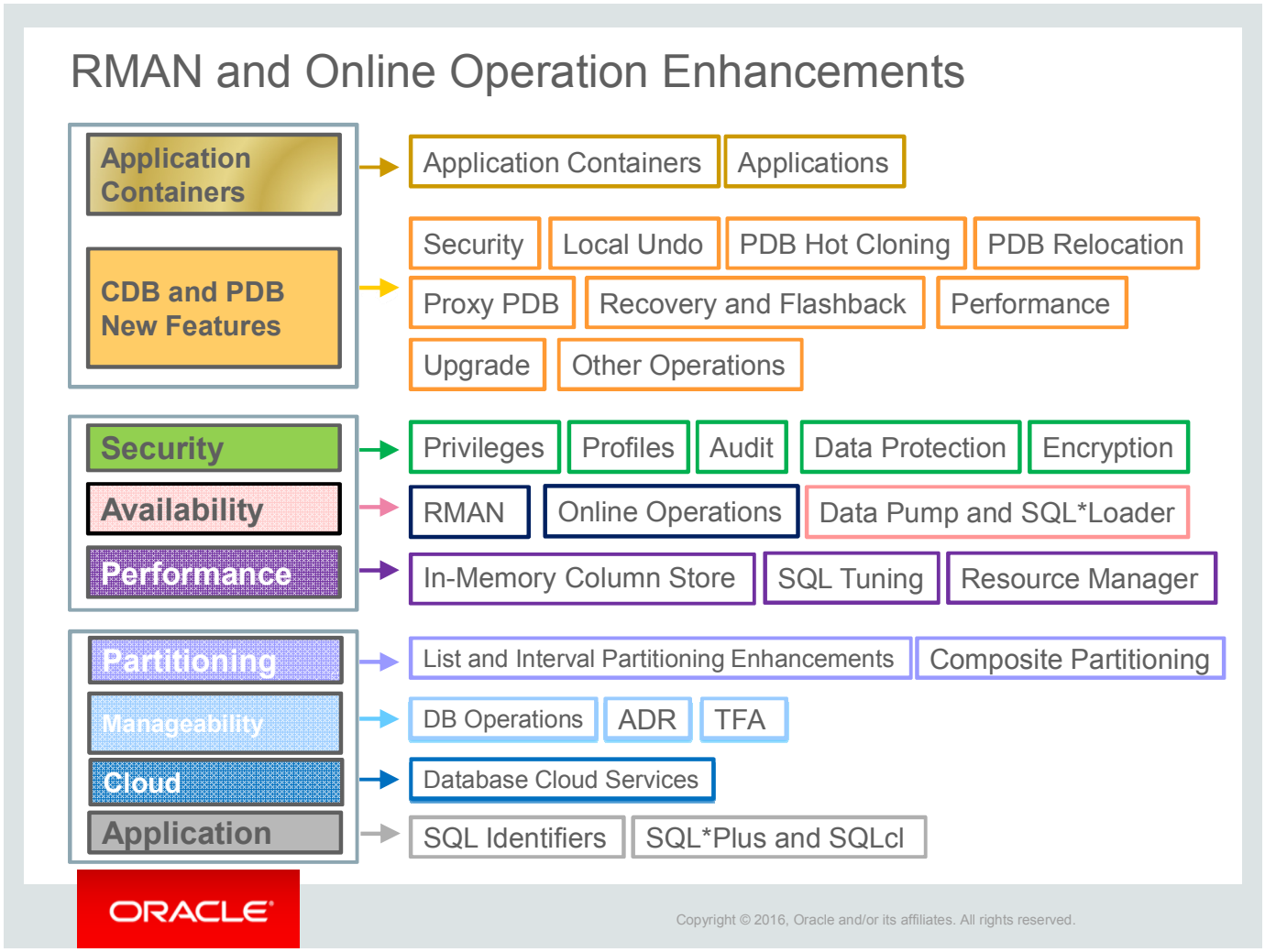
- Unified Auditing
- Fine-grained Auditing
- Column encryption

Practice 13: Overview

- 13-1: Creating a TSDP policy with Unified Auditing settings
- 13-2: Creating a TSDP policy with FGA settings *(Optional)*
- 13-3: Creating a TSDP policy with TDE settings *(Optional)*

14

Data Availability



This lesson explains enhancements with Recovery Manager (RMAN) and online operations.

Objectives

After completing this lesson, you should be able to:

- Describe RMAN enhancements
- Describe database and table recovery enhancements
- Use encrypted backups whose decryption keys are in a non-auto-login wallet
- Explain online operation enhancements

To get detailed information about how to perform any of the operations explained in this lesson, refer to the following guides in the Oracle documentation:

- *Oracle Database Backup and Recovery Guide 12c Release 2 (12.2)*
- *Oracle Database Administrator's Guide 12c Release 2 (12.2)*
- *Oracle Database VLDB and Partitioning Guide 12c Release 2 (12.2)*

RMAN Enhanced Commands

- Upgrade and drop the recovery catalog in one command:

```
RMAN> UPGRADE CATALOG NOPROMPT;
```

```
RMAN> DROP CATALOG NOPROMPT;
```

- Take files offline, restore files, recover files, and bring files back online in a single command:

```
RMAN> REPAIR DATABASE;
```

```
RMAN> REPAIR PLUGGABLE DATABASE pdb1;
```

```
RMAN> REPAIR TABLESPACE example;
```

```
RMAN> REPAIR DATAFILE 12;
```

- RMAN in Oracle Database 12.1 provides the capability of updating and dropping the catalog by entering the `UPGRADE CATALOG` and `DROP CATALOG` commands twice. The `UPGRADE CATALOG` option can be followed by a specific tablespace. `DROP CATALOG`, on the other hand, accepts no other options. These options, however, require a user to input the commands twice. RMAN in Oracle Database 12.2 provides the ability to add a `NOPROMPT` option to both the `UPGRADE` and `DROP CATALOG` commands to bypass the multiple entry process.
- RMAN `RESTORE` and `RECOVER` commands are frequently run consecutively. RMAN in Oracle Database 12.2 extends the `REPAIR` command to accept `DATAFILE`, `TABLESPACE`, `PLUGGABLE DATABASE` and `DATABASE` options, and perform both `RESTORE` and `RECOVER` commands. Where possible, `REPAIR` automatically takes a file offline, restores and recovers it, and then brings it back online again:
  - Automatically offlines files (if applicable)
  - Restores designated files
  - Recovers designated files
  - Automatically onlines any files offlined in step 1

Recover Database Until Available Redo

12.1 Recover the database while the redo log file is missing:

```
RMAN> RECOVER DATABASE;
```

1. You have to find the last available archive redo log file:

```
RMAN> RECOVER DATABASE UNTIL CANCEL;
```

2. Then you can recover the database till the last available redo file:

```
CANCEL
```

```
RMAN> ALTER DATABASE OPEN RESETLOGS;
```

12.2 Automatic retrieval of the last available redo log file during recovery:

```
RMAN> RECOVER DATABASE UNTIL AVAILABLE REDO;
```

```
RMAN> ALTER DATABASE OPEN RESETLOGS;
```

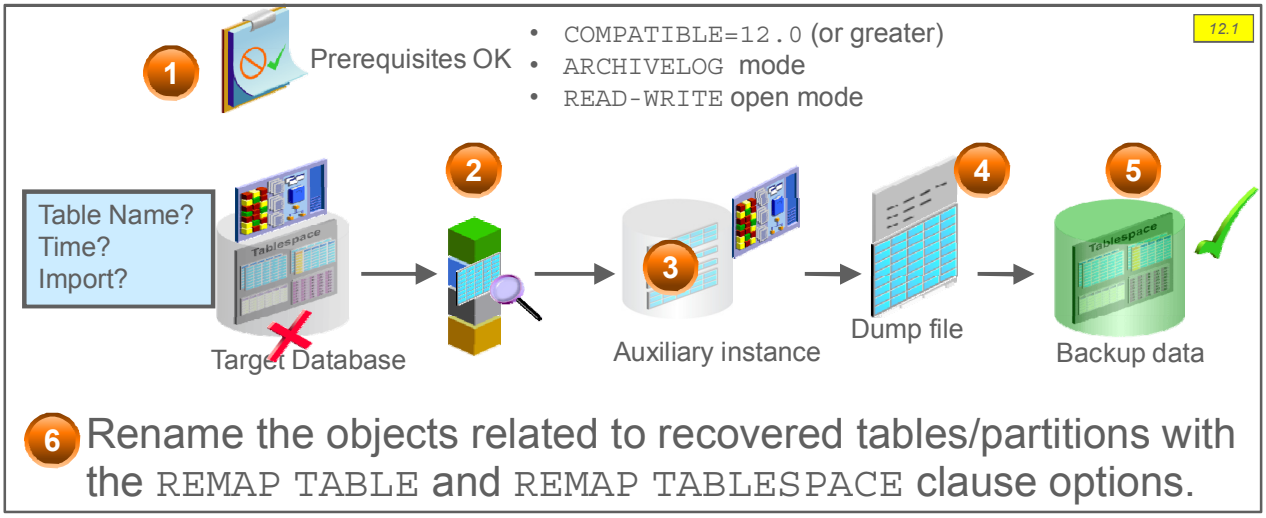

In Oracle Database 12.1, when a user executes the `RECOVER DATABASE` command while some online redo log files or archived redo log files are missing, RMAN signals an error asking for more redo log files after applying all known log files. It is not easy to figure out the missing log files and the DBA has to know which was the last archived redo log file. When the DBA figures out the last archived redo log file to apply, it has to issue the `RECOVER DATABASE UNTIL CANCEL` command again. The DBA continues applying redo log files until the last log has been applied to the restored data files, and then cancels recovery by executing the `CANCEL` command.

RMAN in Oracle Database 12.2 adds a new option, `UNTIL AVAILABLE REDO`, to the `RECOVER DATABASE` command to automate the two-step manual process into one recovery operation:

1. Automatically finds the last available archive redo log file.
2. Recovers the database until the missing archive redo log file is found and then applies the last available redo log file.

Note: The feature is available for CDB root only. If an incomplete PDB recovery is required, specify an SCN to recover the PDB.

Table Recovery: Customization



- Recover the table/partition:
  - Within the same schema 12.1
  - Across different schemas 12.2

```
RMAN> RECOVER TABLE hr.employees UNTIL SCN 2146235
              AUXILIARY DESTINATION '/u01/app/oracle/backup_test'
              REMAP TABLE hr.employees:hr_new.employees;
```

ORACLE

RMAN in Oracle Database 12.1 enables you to recover one or more tables or table partitions to a specified point in time without affecting the remaining database objects.

1. RMAN uses backups previously created to recover tables/partitions to a specified point in time.
2. RMAN determines the backup based on your specification.
3. RMAN creates an auxiliary instance and recovers tables/partitions up to the specified point in time in the auxiliary instance.
4. RMAN creates a Data Pump export dump file that contains the recovered objects.
5. RMAN imports the recovered objects into the target database.
6. You can customize by optionally renaming the recovered tables or table partitions in the target database. When you recover tables or table partitions, you can either retain the existing names or rename the objects after they are imported into the target database. These options, however, do not offer the flexibility of cross-schema table recovery.

Oracle Database 12.2 provides table recovery across different schemas, presenting a better handle of table-related indexes, triggers, constraints, and other objects if these object names already exist under the current schema.

Table Recovery: Automatic Space Check

12.1

Disk space is not checked before the auxiliary instance is created:

1. Starts restoring the auxiliary set
2. When no space is left, it errors out

12.2

Disk space is checked before the auxiliary instance is created:

➔ If there is not enough space, there is an immediate error and table recovery is not executed.

ORACLE

Oracle Database 12.2 also provides an upfront check on the available disk space for the auxiliary instance, before RMAN executes a table recovery. If the available disk space is not sufficient to restore `SYS`, `SYSAUX`, and `UNDO` tablespaces for the auxiliary instance, the recovery attempt immediately errors out on the OS level.

Encrypted Tablespaces Transportability

- Transport encrypted tablespaces across platforms.
- Use encrypted backups whose decryption keys are in an auto-login wallet, a non-auto-login wallet, or user-specified passwords.

Cross-Platform Migration Support for Encrypted Tablespaces

This feature addresses support for encrypted tablespaces which are also to be migrated using cross-platform transport.

DUPLICATE Support for Non-Auto-Login Wallet-Based Encrypted Backups

Oracle Database 12.1 does not allow you to run the `DUPLICATE` command using encrypted backups where the backup keys are stored in a password-based Oracle Wallet or a key store. Even if a user opens the wallet by starting an auxiliary instance, the `DUPLICATE` command restarts the auxiliary instance many times during the command execution. Hence, the restores from the encrypted backups after instance bounce fail because of the inability to decrypt the backups (as Oracle Wallet or the key store is not opened).

Oracle Database 12.2 removes the restriction.

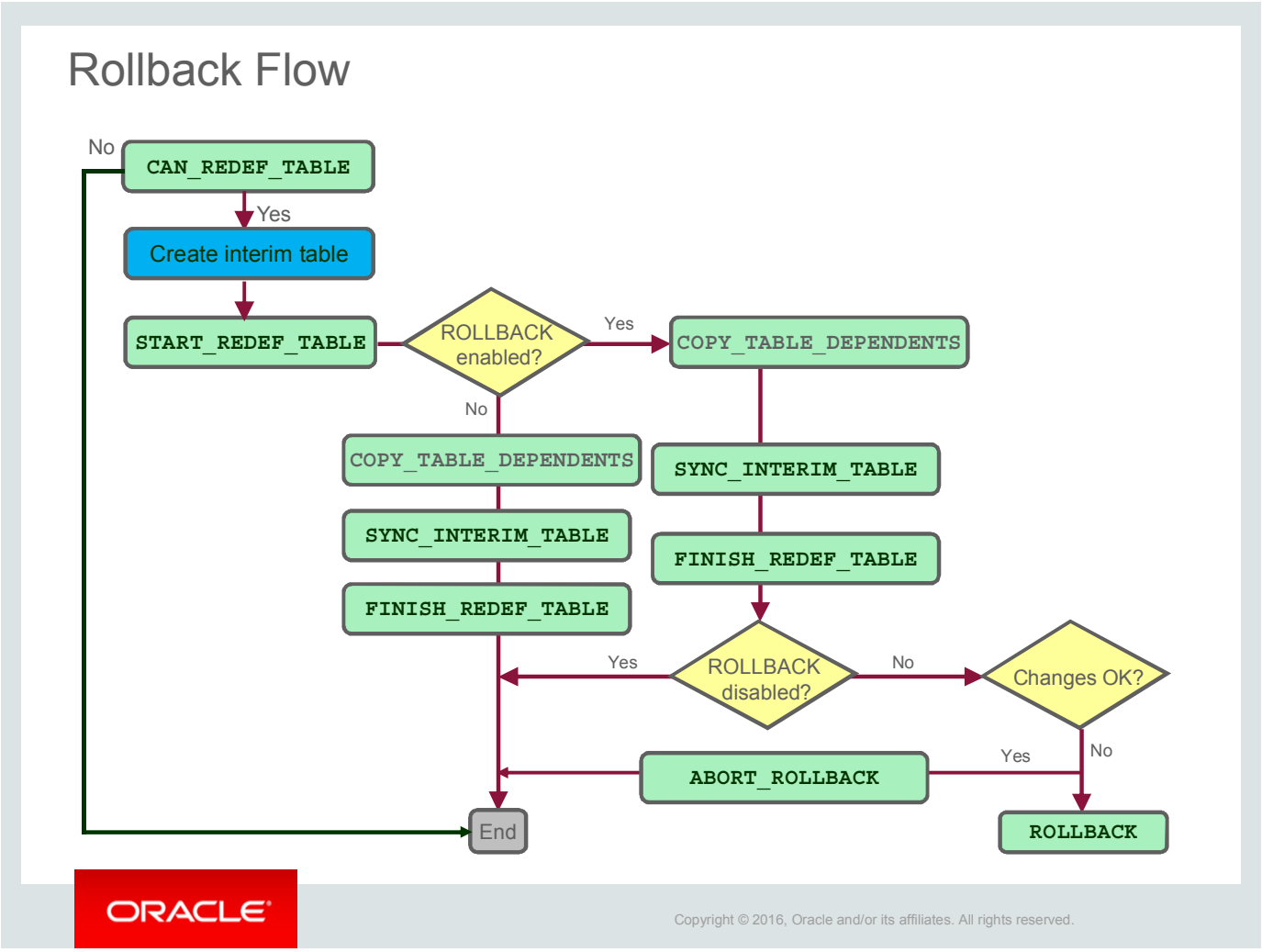
The new feature enables complete support for encrypted backups with encrypted keys storage being auto-login wallet, non-auto-login wallet, or user-specified passwords.

Online Redefinition Enhancements

- Increase robustness and usability:
  - Roll back option after `FINISH_REDEF_TABLE`/`REDEF_TABLE`
  - Redo log overhead removed during table mass update
  - Complete lock-free redefinition in `FINISH_REDEF_TABLE`
  - Full restartability for online table redefinition
- Completeness:
  - Redefinition of tables with BFILE columns supported
  - New option to refresh dependent materialized views (MVs)
  - Invisible columns for online redefinition supported
  - Migrate XML type tables between different storage
- Performance enhancement:
  - Reduce metadata overhead in sync interim table during `FINISH_REDEF_TABLE`

The enhancements for online redefinition allow the following actions:

- Quickly roll back the table to its state before online table redefinition.
- Redefine a table without redo usage using direct load insert and by disabling logging on index during direct load insert.
- Allow DML to proceed as normal during all phases of redefinition without any DML locks held.
- Resume the redefinition process at the point of failure whenever possible.
- Redefine tables with BFILE columns.
- Incrementally refresh dependent materialized views (MVs).
- Show invisible column values after online table redefinition.
- Use online redefinition to migrate XML type tables from CLOB to binary XML storage, from binary XML to CLOB storage, from binary XML to binary XML, from object relational to binary XML, or from binary XML to object relational.



Online table redefinition (OTR) is very frequently used to change the storage characteristics of the table. But sometimes, the changes are not satisfactory. For example, the compression method opted could be slowing down the application more than what was expected. In that case, there is no quick solution to roll back the changes made to the table. When you start the online redefinition process, you can use the `ENABLE_ROLLBACK` parameter. Setting it to `TRUE` enables the `ROLLBACK` option. Later, one can use the `ROLLBACK` procedure to roll back the changes made in `FINISH_REDEF_TABLE`.

Rollback

The **enable_rollback** parameter in the START_REDEF_TABLE procedure allows the rollback of all changes before FINISH_REDEF_TABLE.

1. Verify that the table can be redefined online.
2. Create the interim table.
3. Start the redefinition:

```
SQL> EXEC DBMS_REDEFINITION.START_REDEF_TABLE (… ENABLE_ROLLBACK => TRUE)
```

4. Complete the redefinition process:

```
SQL> EXEC DBMS_REDEFINITION.FINISH_REDEF_TABLE (… DISABLE_ROLLBACK => TRUE)
```

- Decide to roll back all changes if not disabled in step 4:

```
SQL> EXEC DBMS_REDEFINITION.ROLLBACK(…);
```

- Or, if you decide not to roll back the changes, release the resources:

```
SQL> EXEC DBMS_REDEFINITION.ABORT_ROLLBACK(…);
```


The ENABLE_ROLLBACK option can be used only when there are no operators/functions in column mapping (that is, col_mapping of start_redef_table procedure).

Later, during redefinition completion, you can use the DISABLE_ROLLBACK parameter. Setting it to TRUE disables the ROLLBACK option enabled in START_REDEF_TABLE.

An ABORT_ROLLBACK procedure can be used to abort rollback. For example, in FINISH_REDEF_TABLE, the rollback parameter was still enabled. This means one has an intention to roll back the changes. But after evaluating the changes, if the DBA is convinced that the changes to the table are good, the DBA can use this procedure to terminate the intention of rollback and release the rollback resources.

Mass Update Using Online Redefinition

Mass update on a table avoiding redo log overhead:

```
SQL> EXEC DBMS_REDEFINITION.EXECUTE_UPDATE ('UPDATE t.emp SET sal=sal*0.1')
```

1. Parses the `UPDATE` statement and finds the table
2. Creates an interim table identical to the online table
3. Takes an exclusive lock on the online table
4. Creates a materialized view log on the online table
5. Creates a prebuilt materialized view on the interim table
6. Records current scn
7. Releases locks
8. Performs a direct load insert into the interim table
9. Completes the redefinition process:
    - 9.1 Copy of all the table dependents of online table
    - 9.2 Final fast refresh of materialized view

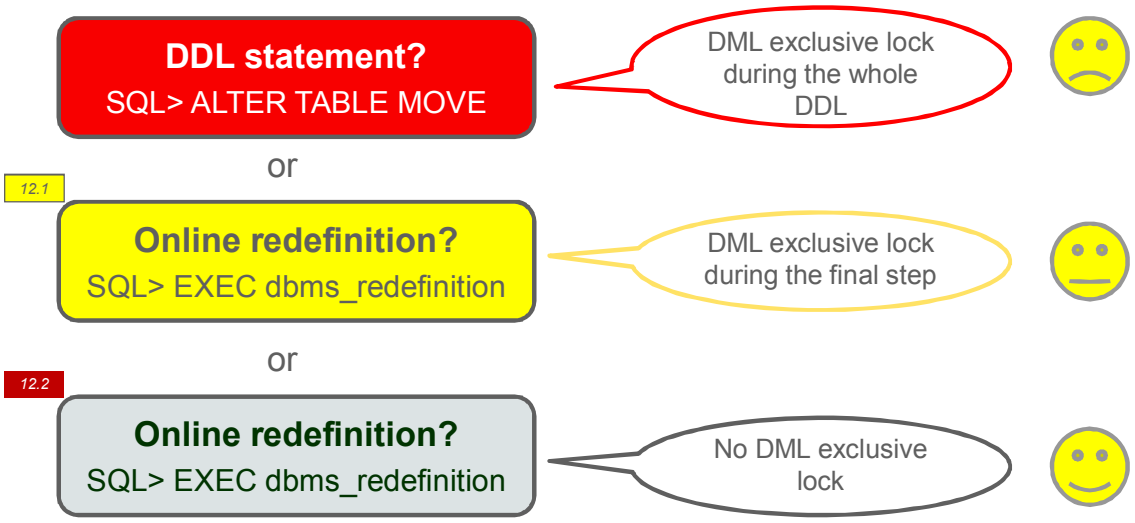
The new `EXECUTE_UPDATE` procedure uses direct load insert to complete a mass update on a table without redo usage.

The benefits of this method include:

1. No redo usage as:
    - Direct load insert is used.
    - Logging on index is disabled during direct load insert.
2. No fragmentation in the table as the rows are created afresh in the interim table
3. Atomic update
4. Ease of use as the `EXECUTE_UPDATE` procedure is provided to do the job

No Exclusive DML Locks

Modify a table without significantly affecting the availability of the table:



When the shape of a table needs to be changed (for example, to add a new column, change the storage properties, or move a partition to a different tablespace), online redefinition is preferred to the ALTER TABLE command as the DDL is too costly.

Online redefinition achieves the availability of the table through the use of materialized views.

In Oracle Database 12.1, during the redefinition process, the only time a DML lock is taken is for a final MV refresh and the table swap. This leaves a much smaller window in which DMLs cannot be run. In comparison, ALTER TABLE DDLs hold the DML lock for the entire duration of the statement.

In Oracle Database 12.2, the need for a DML exclusive lock during the redefinition process is removed, allowing DML to proceed as normal during all phases of redefinition.

Restartability and Monitoring

- Actions to resume a suspended operation:

```
DBA_REDEFINITION_STATUS
      ERR_TXT = "ORA-42009: error occurred
                 while synchronizing the redefinition "
      ACTION = Rerun operation
```

- Monitor the progression of the online redefinition progress:

```
V$ONLINE_REDEF
REDEFINITION_ID   = 43521
…
TABLE_NAME        = SALE
INTERIM_TABLE     = INT_TABLE1
OPERATION         = copy_table_dependents
SUB_OPERATION     = Copy the constraints
DETAILED_MESSAGE  = 1) Start Time "05/01/2013 10:58:12";
PROGRESS          = step 4 out of 7
```


One common pain point reported is the lack of restartability when a failure occurs in the redefinition procedure execution. When SYNC_INTERIM_TABLE or FINISH_REDEF_TABLE fails, ABORT_REDEF_TABLE is the only suggested method to call and it forces the redefinition process to start over (that is, call START_REDEF_TABLE). All the previous redefinition work is, therefore, abandoned and wasted. It is particularly a serious problem when the redefinition process is near the end. It is observed that in some cases, when failures happen, the redefinition process is resumable. For example, if SYNC_INTERIM_TABLE reports a "not enough space" error, the problem can be fixed by increasing the tablespace size. Then synchronization can be retried. The existing DBA_REDEFINITION_STATUS view uses new ERR_TXT and ACTION columns to provide information about how to handle the resumable situation.

In the online redefinition process, some redefinition procedures take quite a long time to execute. The new V$ONLINE_REDEF view allows you to monitor the progression of each of the steps. In the example in the slide, the output message shows that the redefinition is operating on the original table named SALE. The current OPERATION being executed is "copy_table_dependents." The SUB_OPERATION is "Copy the constraints." The DETAILED_MESSAGE shows the start time of the SUB_OPERATION. The corresponding PROGRESS message is "Step 4 out of 7," which indicates that "Copy the constraints" being executed is the fourth suboperation out of the seven steps. The user can query and use the information to monitor the progress and anticipate the completion.

Refresh Dependent MVs

Enable incremental refresh of dependent MVs of an online table:

```
SQL> EXEC DBMS_REDEFINITION.START_REDEF_TABLE (… refresh_dep_mviews => 'Y')
```

With Oracle Database 12.1, the dependent materialized views (MVs) on the original table are not maintained or refreshed during the online redefinition process. The DBA needs to manually execute complete refresh on all dependent MVs after executing `FINISH_REDEF_TABLE`, regardless of whether the MVs can be incrementally maintained or not.

With Oracle Database 12.2, you can enable or disable the incremental refresh of dependent MVs during the execution of relevant redefinition procedures, including `START_REDEF_TABLE`, `SYNC_INTERIM_TABLE` and `FINISH_REDEF_TABLE`, by setting the new `refresh_dep_mviews` parameter in the `START_REDEF_TABLE` procedure.

Enhanced Online DDL Capabilities

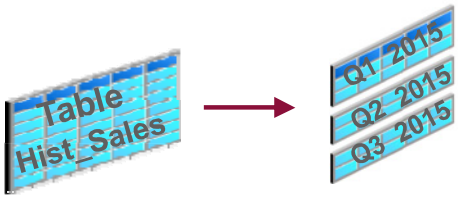
12.2

The **ONLINE** keyword allows the execution of DML statements during DDL operations:

- ALTER TABLE MOVE

```
SQL> ALTER TABLE emp MOVE TABLESPACE low_cost ONLINE UPDATE INDEXES;
```

- ALTER TABLE MODIFY




```
SQL> ALTER TABLE sales MODIFY PARTITION BY RANGE (c1) INTERVAL (100)
            (PARTITION p1 …, PARTITION p2 …) ONLINE UPDATE INDEXES;
```

- ALTER TABLE SPLIT/MERGE PARTITION

```
SQL> ALTER TABLE orders SPLIT PARTITION ord_p1 AT ('01-JUN-2015')
          INTO (PARTITION ord_p1_1 COMPRESS, PARTITION ord_p1_2)
          ONLINE UPDATE INDEXES;
```


Beginning with the Oracle Database 12*c* Release 1 (12.1), you can use the new ONLINE keyword to allow the execution of DML statements during the following DDL operations:

- DROP INDEX
- DROP CONSTRAINT
- ALTER INDEX UNUSABLE
- SET COLUMN UNUSED
- ALTER TABLE MOVE PARTITION

There are no application disruptions for schema maintenance operations.

Oracle Database 12.2 allows an online version for other DDL statements:

- MOVE TABLE for heap tables: This DDL is frequently used in moving nonpartitioned tables from one tablespace to another or in changing the compression property for Information Lifecycle Management purposes.

Enhanced Online DDL Capabilities

- Conversion of nonpartitioned tables to partitioned tables without application interruption. This type of operation is not supported for IOTs or object type tables, and in the presence of domain indexes. The `UPDATE INDEXES` clause can be used to change the partitioning state of indexes and the storage properties of the indexes on the table being modified to a partitioned state, but not the uniqueness of the columns on which the list of indexes was originally defined.
  If the `UPDATE INDEXES` clause is not specified or does not specify all the indexes in the original nonpartitioned table, the following defaulting behavior applies for all unspecified indexes:
  - Nonunique indexes are converted to local partitioned indexes.
  - All unique indexes that are prefixed by the partitioning keys are converted to local partitioned indexes.
  - All unique indexes that are nonprefixed by the partitioning keys are converted to global unique indexes.
- Partition maintenance operations such as `SPLIT`, `MERGE` are often used in managing partitions. These operations acquire an X DML lock on the relevant partition(s) for the entire duration of the operation. Meanwhile, this prevents DMLs on the specific partitions. Oracle Database 12.2 allows an online version where the `SPLIT`, `MERGE` operation finishes after all DMLs that started before the DDL statement complete.

Create Table for Exchange

12.1

```
SQL> CREATE TABLE tab_nonpart (id number(2), name varchar2(15));

SQL> ALTER TABLE tab_part
           EXCHANGE PARTITION p1 WITH TABLE tab_nonpart;
```

Define the whole table description in columns.

12.2

- Facilitate the creation of a table that matches an existing table in terms of column properties and column orderings:

```
SQL> CREATE TABLE tab_nonpart TABLESPACE tbsexch
                  FOR EXCHANGE WITH TABLE tab_part;
```

- Use this table in an exchange partition DDL to move data into or out of a partition.

The new `CREATE TABLE FOR EXCHANGE WITH TABLE` DDL statement creates a clone of a table in terms of shape and column properties. The goal is to use the new table in an exchange partition DDL to move data into or out of a partition.

Summary

In this lesson, you should have learned how to:

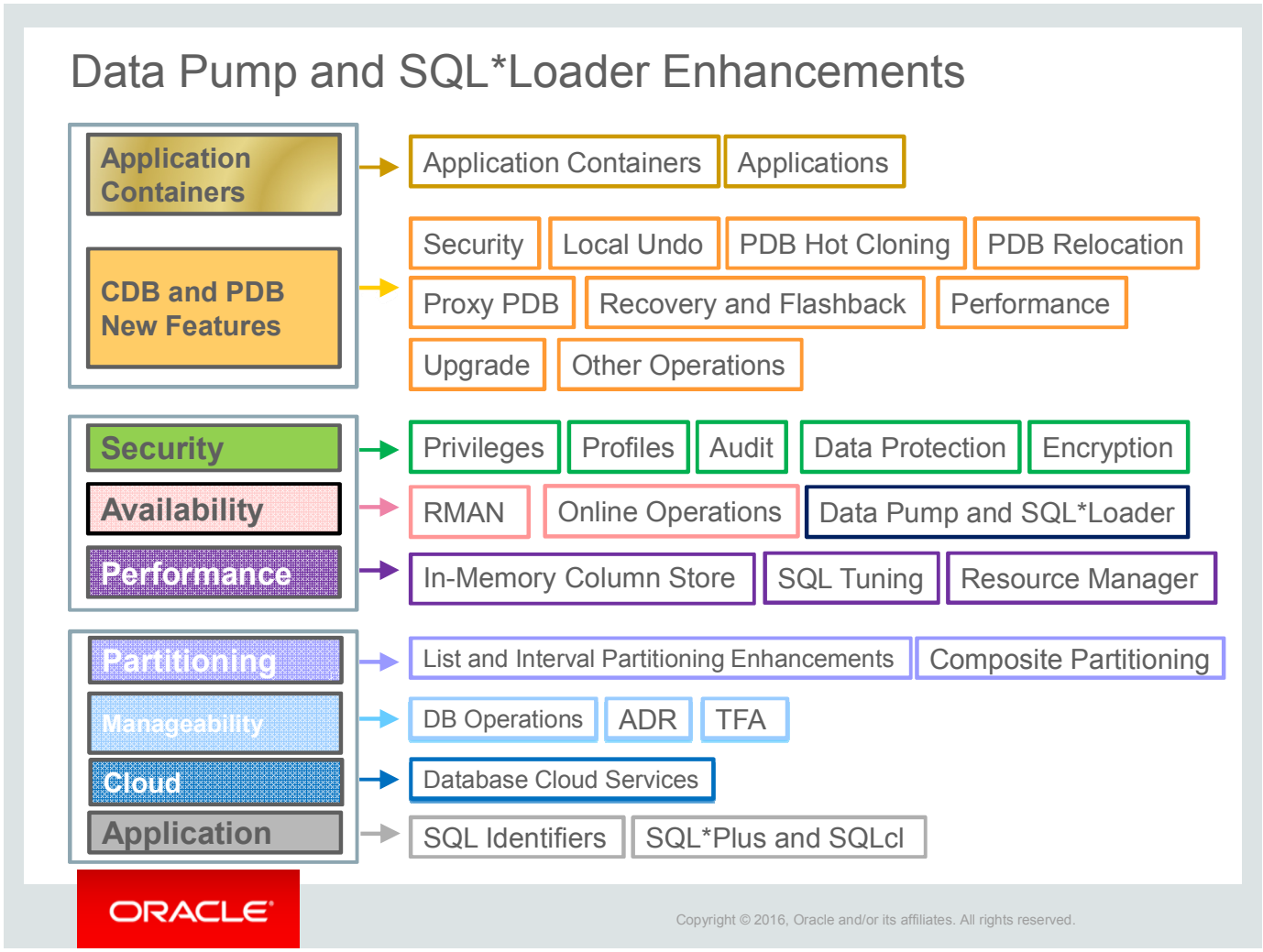
- Describe RMAN enhancements
- Describe database and table recovery enhancements
- Use encrypted backups whose decryption keys are in a non-auto-login wallet
- Explain online operation enhancements

Practice 14: Overview

- 14-1: Recovering database until available redo
- 14-2: Recovering using new commands
- 14-3: Recovering tables across schemas
- 14-4: Moving and compressing tables online

15

Oracle Data Pump, SQL*Loader, and External Tables



This lesson explains enhancements with Oracle Data Pump, SQL*Loader, and external tables.

Objectives

After completing this lesson, you should be able to:

- Describe Oracle Database 12.2 new features for Data Pump, SQL*Loader, and External Tables
- Explain applicable use cases and configuration details for the new features of Oracle Data Pump, SQL*Loader, and External Tables

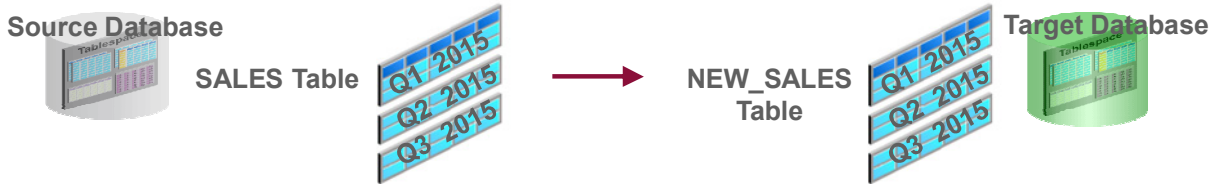
To get detailed information about how to perform any of the operations explained in this lesson, refer to the O*racle Database Utilities 12c Release 2 (12.2)* guide in the Oracle documentation.

Parallel Export and Import

- Enables multiple workers to perform metadata and data export and import operations at the same time

```
$ expdp … PARALLEL = n
```

- Enables data from multiple partitions to be loaded in parallel:
  - Same partitioning method and partition names



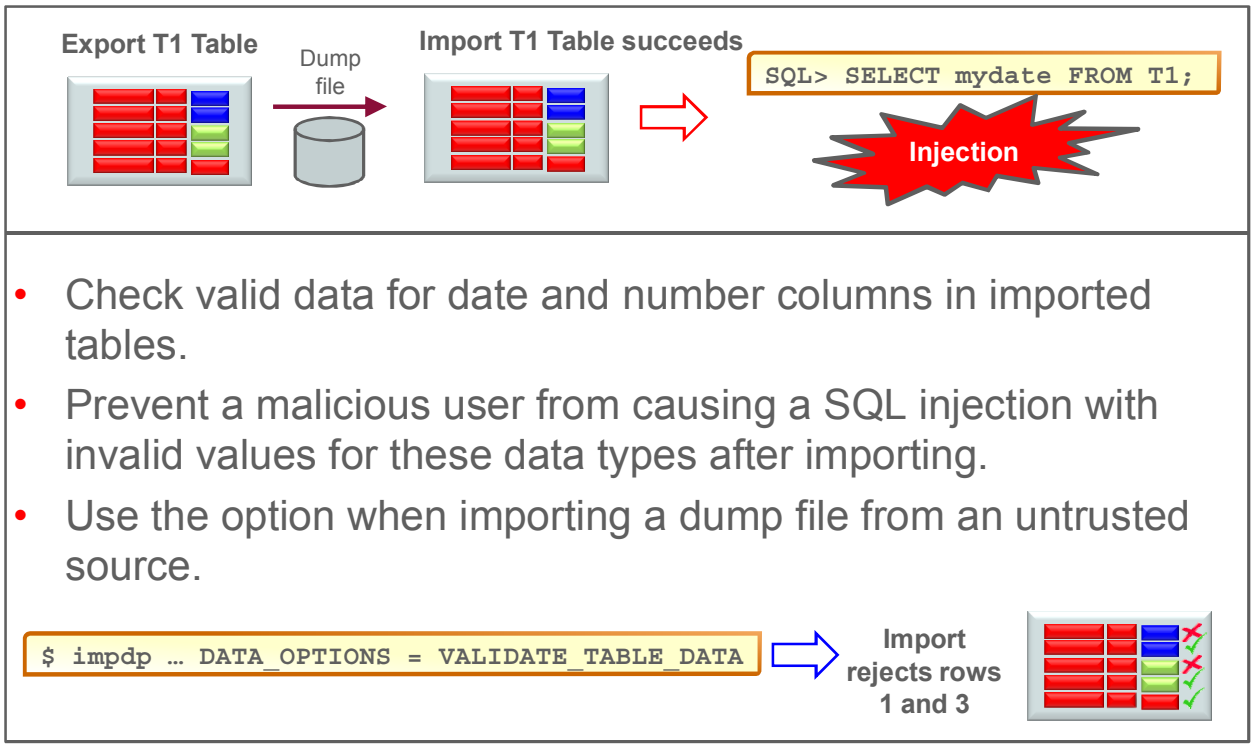

```
$ impdp … DATA_OPTIONS = TRUST_EXISTING_TABLE_PARTITIONS
```

  - Unloads the data for all table partitions in one operation, instead of unloading each partition separately

```
$ expdp … DATA_OPTIONS = GROUP_PARTITION_TABLE_DATA
```


- Oracle Data Pump in Oracle Database 12.1 supports unloading and loading table data in parallel, but metadata can only be performed by one process. Oracle Data Pump in Oracle Database 12.2 shortens the elapsed time for unloading and loading metadata in parallel through multiple background processes. This new enhancement allows faster database migration. Parallel export and import of metadata happens when the `PARALLEL` parameter is specified.
- When migrating a table, the DBA sometimes pre-creates the table before moving the table data because the new table might have new attributes (such as compression) or different partitioning. Oracle Data Pump imports the table data from a partitioned table into the newly created table one partition at a time because the definition of the new table could cause rows from one partition in the exported table to be loaded into multiple partitions in the new table. For a table with many partitions, this can be very slow. Oracle Data Pump in Oracle Database 12.2 allows data from multiple partitions to be loaded at the same time in two cases:
  - If the exported and imported table have the same partitioning method and partition names
  - If the exported table data in all partitions is treated as one unit, then data for all partitions are imported at the same time with a SQL `INSERT AS SELECT` statement executing in parallel.

Dump File Data Validation



Oracle Data Pump in Oracle Database 12.1. designates how certain types of data should be handled during import operations with the `DATA_OPTIONS` parameter.

- `DISABLE_APPEND_HINT`: Specifies that you do not want the import operation to use the `APPEND` hint while loading the data object
- `SKIP_CONSTRAINT_ERRORS`: Affects how nondeferred constraint violations are handled while a data object (table, partition, or subpartition) is being loaded
- `REJECT_ROWS_WITH_REPL_CHAR`: Specifies that you want the import operation to reject any rows that experience data loss because the default replacement character was used during character set conversion

Oracle Data Pump in Oracle Database 12.1. does not perform any validation of the imported data. It is therefore possible for a malicious user to cause a SQL injection with invalid values for number and date data types.

Oracle Data Pump in Oracle Database 12.2 allows you to verify the format number and date values contained in number and date data types in table columns. It rejects rows that contain column data that is not valid and writes them to the Data Pump `import.trc` file.

The default does not perform any validation. It is highly recommended that you use this option if the source of the Data Pump dump file is not trusted.

New Substitution Variables in Names for Dump Files

- '%L' versus '%U':

```
$ expdp … DUMPFILE = exp_%L_dj
```

  - 2-digit to 10-digit
  - Variable width
  - Incrementing integers starting at 01, ending at 2147483646

- New substitution variables for dates:
  - %D
  - %M
  - %T (YYYYMMDD)
  - %Y (YYYY)

```
$ expdp … DUMPFILE = exp_%t_%L
```

When generating file names, a data pump job may create many dump files. Oracle Data Pump in Oracle Database 12.1 allows the '%U' substitution variable to be included in the dump file names that substitutes %u with a two-digit number.

Oracle Data Pump in Oracle Database 12.2 expands the '%U' substitution variable to more than two digits with the '%L' substitution variable. The resulting file names expand into 3-digit to 10-digit, variable-width, incrementing integers starting at 100 and ending at 2147483646.

Oracle Data Pump in Oracle Database 12.2 allows new substitution variables for dates based on the job start time:

- %D: DD
- %M: MM
- %T: YYYYMMDD
- %Y: YYYY

The only wildcard names that can be given to import are file names containing only '%U' or '%L' substitution variables.

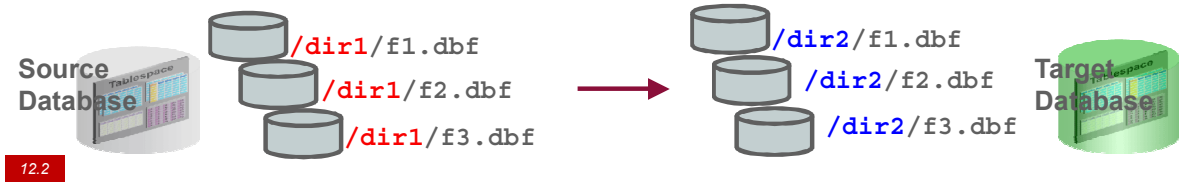
Data File Renaming with Transportable Tablespace

New mechanism for importing data files with TTS:

- New `REMAP_DIRECTORY` import parameter
- Changes the source directory string to the target directory string in all SQL statements such as:
  - `CREATE TABLESPACE`
  - `CREATE LIBRARY`
  - `CREATE DIRECTORY`

12.1

```
$ impdp … TRANSPORT_DATAFILES='/dir1/f1.dbf','/dir1/f2.dbf','/dir1/f3.dbf'
```



12.2

```
$ impdp … REMAP_DIRECTORY='''/dir1/':'/dir2/'''
```

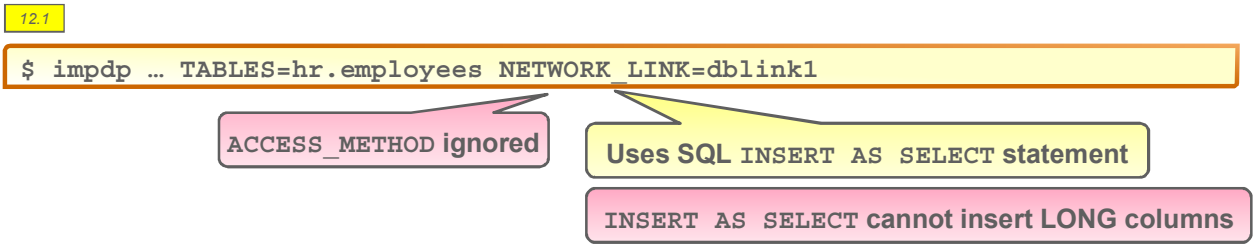

When using transportable tablespaces, the data files are copied from the export platform to the import platform in directories different from the export platform. Oracle Data Pump import in Oracle Database 12.1 requires that every data file be explicitly specified with the old and new file specification for every data file of the `TRANSPORT_DATAFILES` parameter. This can be tedious for large databases that can have hundreds or even thousands of data files.

Oracle Data Pump import in Oracle Database 12.2 allows wildcard syntax for the leaf, file name portion of a data file specification so that only each unique directory on the import platform needs to be specified. The wildcard syntax allows the DBA to move all data files from one directory in the export database to the same directory in the import database. The new `REMAP_DIRECTORY` parameter replaces the directory portion of a data file with a different path. This is done by changing the source directory string to the target directory string in all SQL statements where the source directory is the portion at the extreme left of a full file or directory specification. When the data files matching the wildcard specification have been selected, they are verified and processed in the same way that Oracle Data Pump does for transportable-mode import.

The `REMAP_DIRECTORY` and `REMAP_DATAFILE` parameters are mutually exclusive.

It is recommended that the directory be properly terminated with the directory file terminator for the respective source and target platforms.

LONG Columns Loaded with Network Import



12.2 Network import supports the ACCESS_METHOD parameter.

- DIRECT_PATH: Can move tables with LONG columns, but has other restrictions
- INSERT_AS_SELECT: Uses the SQL INSERT AS SELECT statement
- AUTOMATIC: Chooses between DIRECT_PATH and INSERT_AS_SELECT for each table

```
$ impdp … TABLES=hr.employees
                  NETWORK_LINK=dblink1 ACCESS_METHOD=DIRECT_PATH
                  DATA_OPTIONS=ENABLE_NETWORK_COMPRESSION
```


In Oracle Database 12.1, the ACCESS_METHOD parameter is ignored when the NETWORK_LINK parameter is used. Therefore, network import does not support direct path mode. Network import uses the SQL INSERT AS SELECT command to import table data.

In Oracle Database 12.2, network import allows LONG columns to be loaded. The ACCESS_METHOD parameter is compatible with the NETWORK_LINK parameter, allowing different values:

- DIRECT_PATH: Can move tables with LONG columns, but has other restrictions. For example, it cannot move tables with BFILE columns.
- INSERT_AS_SELECT: Is a new option, only valid for network mode imports. It uses the SQL INSERT AS SELECT statement to move data, which is the only way network mode import moves data before Oracle Database 12.2.
- AUTOMATIC: Chooses between DIRECT_PATH and INSERT_AS_SELECT for each table. It prefers DIRECT_PATH since it moves data faster.

You can also specify the new ENABLE_NETWORK_COMPRESSION value in DATA_OPTIONS to send compressed data over the network link.

Multicharacter Delimiters for SQL*Loader Express Mode

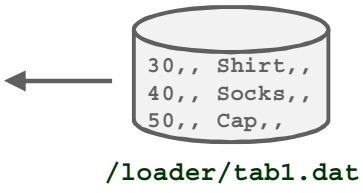
12.1

- Specify a table name to initiate an Express Mode load.
- There is no need to create a control file.
  - The data file contains only single character delimiters.

12.2

  - The data file can contain multicharacter delimiters:
    - TERMINATED_BY
    - ENCLOSED_BY
    - OPTIONALLY_ENCLOSED_BY

```
$ sqlldr hr TABLES=tab1 TERMINATED_BY=\",,\"
```



If you activate SQL*Loader Express Mode, specifying only the username and the TABLE parameter, it uses default settings for a number of other parameters. You can override most of the defaults by specifying additional parameters on the command line.

In Oracle Database 12.1, SQL*Loader Express mode can use the three delimiter parameters that can be only single character delimiters:

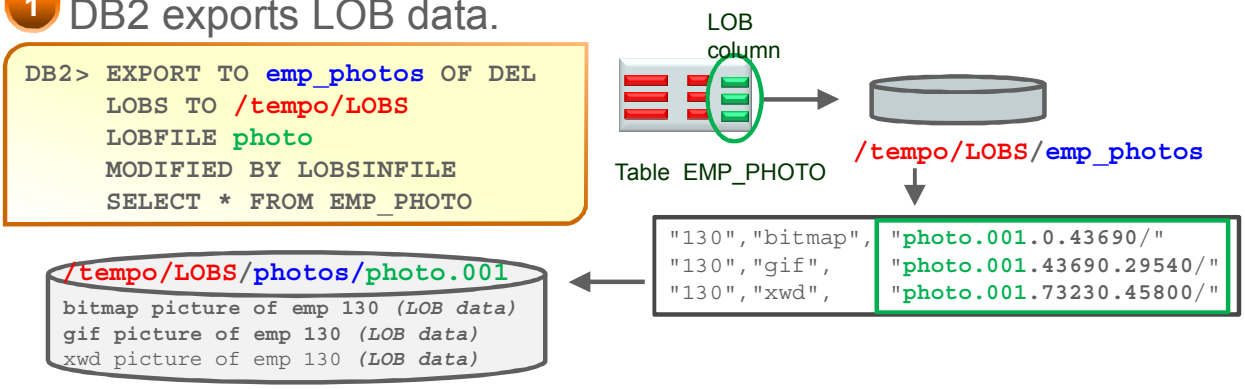
- The TERMINATED_BY parameter, which specifies a field terminator
- The ENCLOSED_BY parameter, which specifies a field enclosure character
- The OPTIONALLY_ENCLOSED_BY parameter, which specifies an optional field enclosure character

In Oracle Database 12.2, the three delimiters can be a multicharacter string.

Loading DB2 Data

1 DB2 exports LOB data.

```
DB2> EXPORT TO emp_photos OF DEL
     LOBS TO /tempo/LOBS
     LOBFILE photo
     MODIFIED BY LOBSINFILE
     SELECT * FROM EMP_PHOTO
```



2 SQL*Loader reads and loads DB2 LOB data.

1. Retrieves the file name containing the LOB data from the text file
2. Uses a new LLS clause to define where to read the LOB data:
   - <file-name>.<ext> is the name of the LOB file (`photo.001`)
   - <nnn> is the offset of the LOB within the file, in bytes (`0` for the first LOB data)
   - <mmm> is the length of the LOB, in bytes

ORACLE

Migrating data from DB2 to Oracle becomes easy with SQL*Loader and external tables, which can read LOB data unloaded by the DB2 export utility.

In the first step in the slide, the DB2 export utility unloads `EMP_PHOTO` table data into the `/tempo/LOBS/emp_photos` text file, using the '`LOBFILE`' option to unload the LOB data into a separate file. DB2 writes a Lob Locator Specifier (LLS) into the text file and the LOB data in another file. The LLS is a string that points to the location of the LOB data. Its format is <filename>.<ext>.<nnn>.<mmm>/, where:

- <file-name>.<ext> is the name of the LOB file name (`PHOTO.001`)
- <nnn> is the offset of the LOB within the file, in bytes (`0` for the first LOB data)
- <mmm> is the length of the LOB, in bytes

SQL*Loader uses a new clause in the control file to indicate that a field in the data file is an LLS field. SQL*Loader uses the information in the field to read the data for the LOB column.

This is a new way for SQL*Loader and external tables to read LOB data from data files.

Usage notes:

- The names of the file with the LOB data is a field in the text file being loaded.

Loading DB2 Data

- The user running SQL*Loader must have READ access to those files. For external tables, the files must live in a specified directory object or the default directory object for the external table. The user must have READ access to the directory object.
- If an LLS field is referenced by a clause for any other field (for example, a `NULLIF` clause) in the control file, then the value used for evaluating the clause is the string in the data file, not the data in the file pointed to by that string.
- The character set for the data in the file pointed to by the LLS clause is assumed to be the same character set as the data file.

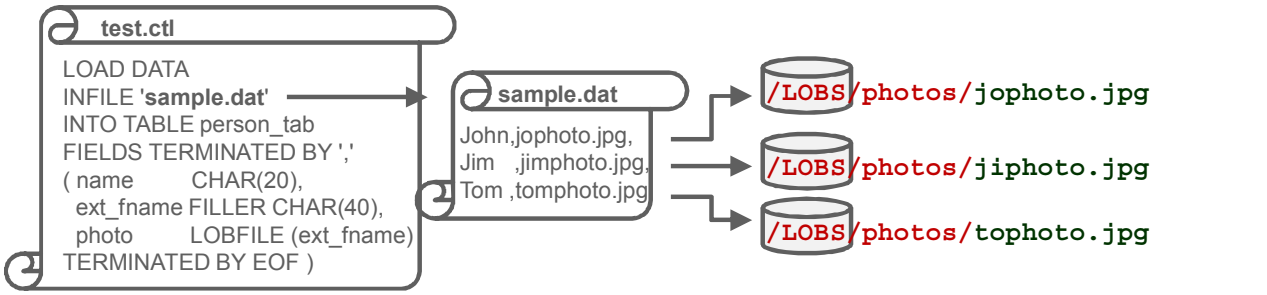
Example:

```
LOAD DATA
INFILE *
TRUNCATE
INTO TABLE tklglls
FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"' TRAILING
NULLCOLS
  (col1 , col2 NULLIF col1 = '1' LLS)
BEGINDATA
1,"photo.001.1.11/"
```

Secondary Data Files Location

- Specify a directory prefix to add to the file names of LOBFILES and secondary data files to be loaded.
  - Using `SDF_PREFIX` allows the file names to be relative to a different directory
  - Using `SDF_PREFIX` provides more flexibility to file placement

Note: `SDF_PREFIX` should not be used if the file specifications for the LOBFILES or SDFs contain full file names.




```
$ sqlldr … CONTROL=test.ctl SDF_PREFIX=/LOBS/photos
```

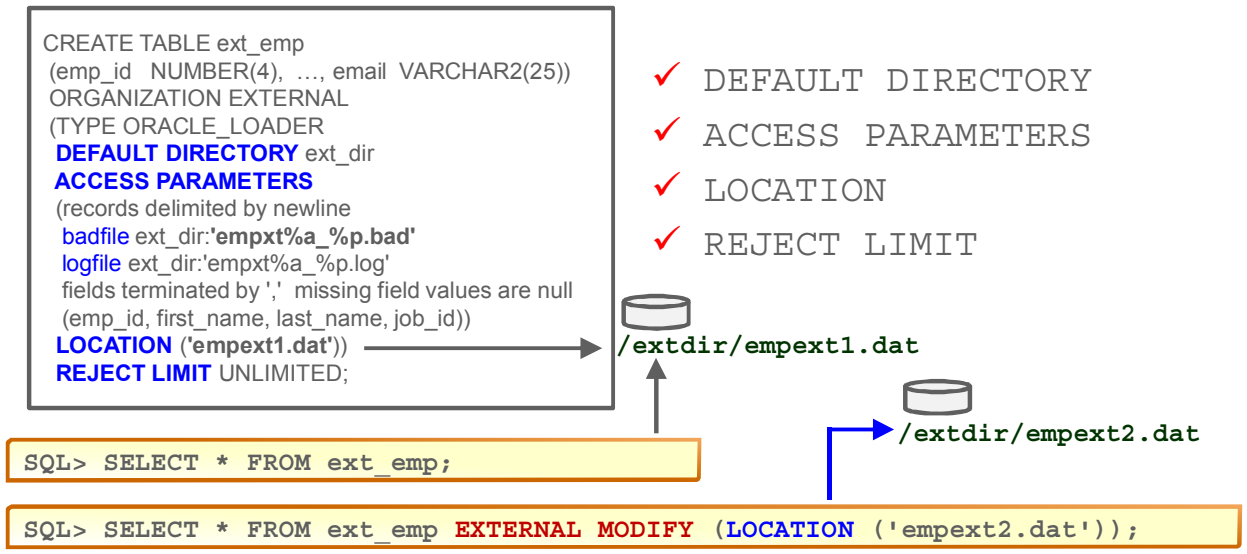

With SQL*Loader in Oracle Database 12.1, DBAs who need to distribute data like LOBFILEs and secondary data files (SDFs) to multiple people have to create scripts that preset directories so that they can be run from anywhere.

With SQL*Loader in Oracle Database 12.2, the new `SDF_PREFIX` parameter value appended to the front of the name for all LOBFILEs and secondary data files (SDFs) opened by SQL*Loader allows DBAs to create a data file that can be loaded from different directories on different systems without hard coding the complete file specification in the data file. Instead, the file names in the data file can all be relative to the path specified in `SDF_PREFIX`. This parameter can be specified in the `OPTIONS` clause inside the SQL*Loader control file.

If `SDF_PREFIX` is not specified, file names for the files to load are assumed to be relative to the current working directory. Using `SDF_PREFIX` allows those files names to be relative to a different directory, giving the user more flexibility in creating the data file and in running the SQL*Loader command.

Querying External Tables

- Override external table clauses when querying against external tables.



- No need to alter the external table definition

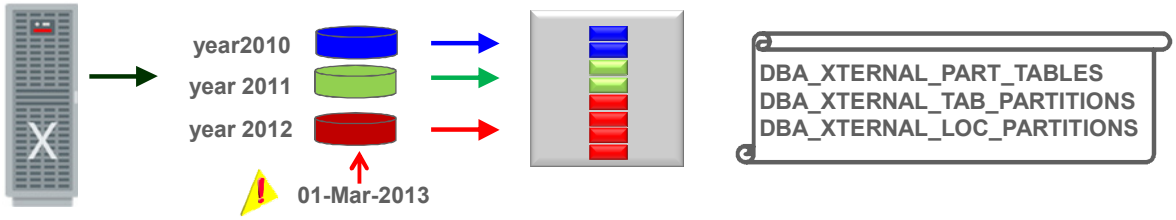
In Oracle Database 12.1, creating an external table allows `DEFAULT DIRECTORY`, `ACCESS PARAMETERS`, `LOCATION`, and `REJECT LIMIT` to be defined during `CREATE TABLE` or `ALTER TABLE`. When querying the external table, none of these parameters can be modified.

In Oracle Database 12.2, queries against an external table can override the `DEFAULT DIRECTORY`, `ACCESS PARAMETERS`, `LOCATION`, and `REJECT LIMIT` clauses. Modifications in `ACCESS PARAMETERS` are limited to `DISCARDFILE`, `BADFILE`, and `LOGFILE`.

If the second query in the slide is running concurrently with the first one that reads the data from '`empext1.dat`,' there is no need to create a separate external table for the second query. The second query overrides the default `LOCATION` to fetch external data from another location. The queries share the external metadata which was created for a single `EXT_EMP` table.

External Tables and Partitions

Files for external tables can be organized on partitioning criteria.



- Declaring RELY, PK-FK constraints improves optimizer plans.
  - Static partition pruning
  - Dynamic pruning, including bloom pruning
  - Partitionwise join for queries
- Collecting statistics per external table partition gets better plans when partition pruning occurs.
- Defining virtual columns is possible, but partitioning on virtual columns is not supported yet.

External tables in Oracle 12.1.0.2 Exadoop (Exadata and Hadoop clusters) and Oracle 12.2 BigSQL provide a prominent way of analyzing very big external data.

Declaring partitioning and RELY PK, PK-FK constraints on external tables significantly improve query optimizer plans. Firstly, partitioning external tables allows existing optimizations like static partition pruning, dynamic pruning, including bloom pruning, and partitionwise join for queries over external tables. Secondly, collecting statistics per external table partition results in better plans when partition pruning occurs.

- External table partitioning can use single level `RANGE`, `LIST`, `INTERVAL`, multi-column `LIST` partitioning, and all combinations of `RANGE`, `LIST`, and `INTERVAL` as composite partitioning. Row validation is not supported yet. Therefore, you must ensure that the files satisfy partitioning conditions.
- Subpartitioning can be defined on external tables.
- `ACCESS PARAMETERS` and `REJECT LIMIT` parameters are not supported at the partition level. Only `DEFAULT DIRECTORY` and `LOCATION` parameters are supported at the partition level.
- The `EXTERNAL MODIFY` clause in a query works at the table level in a partitioned external table, but not for modifying partition-level parameters.
- Incremental partition-based statistics are not supported yet.

Summary

In this lesson, you should have learned how to:

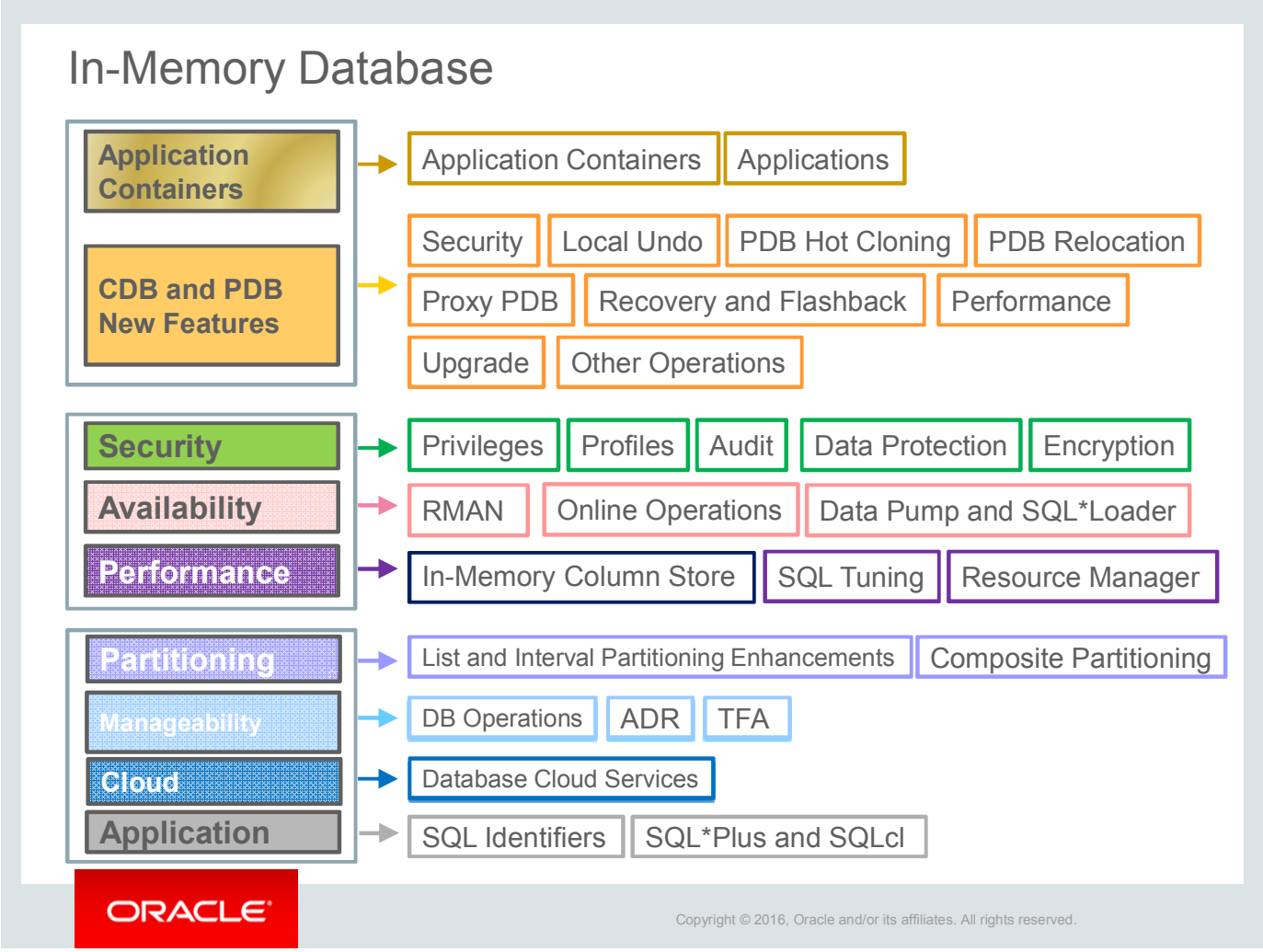
- Describe Oracle Database 12.2 new features for Data Pump, SQL*Loader, and External Tables
- Explain applicable use cases and configuration details for the new features of Oracle Data Pump, SQL*Loader, and External Tables

Practice 15: Overview

- 15-1: Querying a partitioned external table

16

In-Memory Column Store



Oracle Database 12.2 introduces In-Memory expressions and In-Memory virtual columns as well as In-Memory FastStart.

In addition, Automatic Data Optimization (ADO) support for In-Memory Column Store automatically ensures optimal use of the In-Memory Column Store.

Objectives

After completing this lesson, you should be able to:

- Explain how to deploy the In-Memory Column Store in a database
- Benefit from the Oracle Database In-Memory Advisor
- Configure IM FastStart to accelerate IM segment population at startup
- Optimize join queries by creating join groups
- Store expressions and virtual column values in the IM Column Store
- Use heat map statistics and ADO policy to add, evict or compress objects in the IM Column Store

To get detailed information about how to perform any of the operations explained in this lesson, refer to the following guides in the Oracle documentation:

- *Oracle Database In-Memory Guide 12c Release 2 (12.2)*
- *Oracle Database Administrator's Guide 12c Release 2 (12.2)*
- *Oracle Database Concepts 12c Release 2 (12.2)*
- *Oracle Database VLDB and Partitioning Guide 12c Release 2 (12.2)*

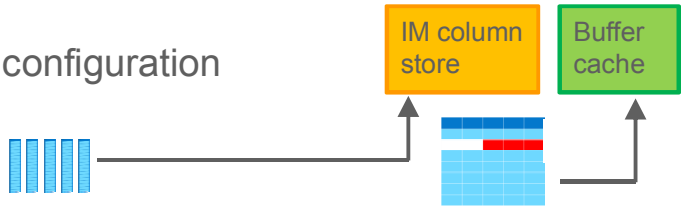
Goals of In-Memory Column Store

12.1.0.2

- Instant query response:
  - Faster queries on very large tables on any columns (**100x**)
  - Uses scans, joins, and aggregates
  - Does not use indexes
  - Best suited for analytics: few columns/many rows
- Faster DML: Removal of most analytics indexes **(3 to 4x)**
- Full application transparency

ORACLE E-BUSINESS SUITE
ORACLE FUSION APPLICATIONS
ORACLE SIEBEL
ORACLE JD EDWARDS
ORACLE PEOPLESOFT

- Easy setup:
  - In-memory column store configuration
  - Segment attributes



ORACLE

The In-Memory Column Store enables objects (tables, partitions, and other types) to be stored in memory in a new format known as the columnar format. This format enables scans, joins, and aggregates to perform much faster than the traditional on-disk format, providing fast reporting and DML performance for both OLTP and DW environments. This is particularly useful for analytic applications that operate on few columns returning many rows rather than for OLTP operating on few rows returning many columns. The DBA has to define which segments need to be populated into the in-memory column store (IM column store), such as hot tables, partitions and more precisely more frequently accessed columns.

The in-memory columnar format does not replace the on-disk or buffer cache format. It is a consistent copy of the table or of some columns of a table converted to the new columnar format that is independent of the disk format and only available in memory. Because of this independence, applications are able to transparently use this option without any changes. For the data to be converted into the new columnar format, a new pool is requested in the SGA. The pool is the IM column store.

If sufficient space is allocated for the IM column store, a query accessing objects that are candidates to be populated into the IM column store performs much faster. The improved performance allows ad hoc analytic queries to be executed directly on the real-time transaction data without impacting the existing workload.

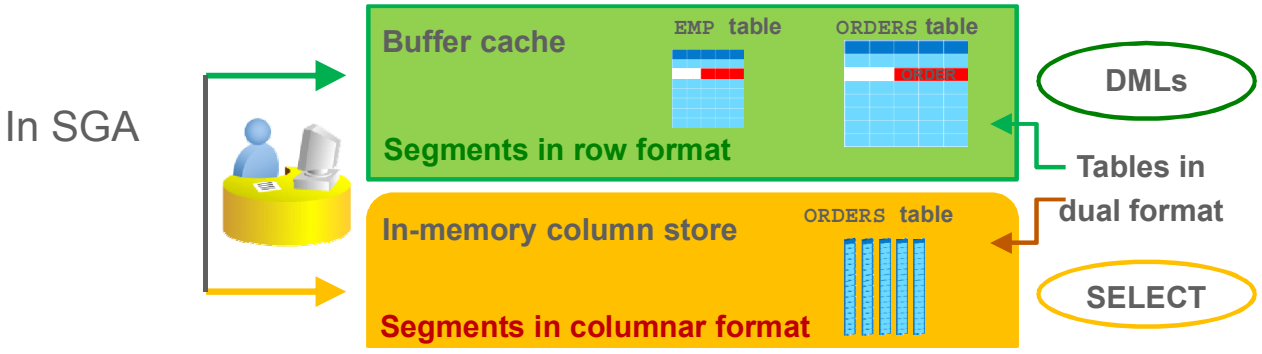
There are three main advantages:

- Queries run a lot faster: All data can be populated in memory in a compressed columnar format. No index is required and used. Queries run at least 100 times faster than when fetching data from buffer cache thanks to the columnar compressed format.
- DMLs are faster: Analytics indexes can be eliminated being replaced by scans of the IM column store representation of the table.
- Arbitrary ad hoc queries run with good performance, since the table behaves as if all columns are indexed.

In-Memory Column Store: Overview

12.1.0.2

- A new pool in SGA: In-Memory column store
  - Segments populated into the IM column store are converted into a columnar format.
  - In-Memory segments are transactionally consistent with the buffer cache.
- Only one segment on disk and in row format



The in-memory columnar format does not replace the on-disk or buffer cache format. This means that when a segment, such as a table or a partition, is populated into the IM column store, the on-disk format segment is automatically converted into a columnar format and optionally compressed. The columnar format is a pure in-memory format. There is no columnar format storage on disk. It never causes additional writes to disk and therefore does not require any logging or undo space.

All data is stored on disk in the traditional row format.

Moreover, the columnar format of a segment is a transaction-consistent copy of the segment either on disk or in the buffer cache. Transaction consistency between the two pools is maintained.

If sufficient space is allocated to the IM column store in SGA, a query accessing objects that are populated into the IM column store performs much faster. The improved performance allows more ad hoc analytic queries to be executed directly on the real-time transaction data without impacting the existing workload. A lack of IM column store space does not prevent statements from executing against tables that could have been populated into the IM column store.

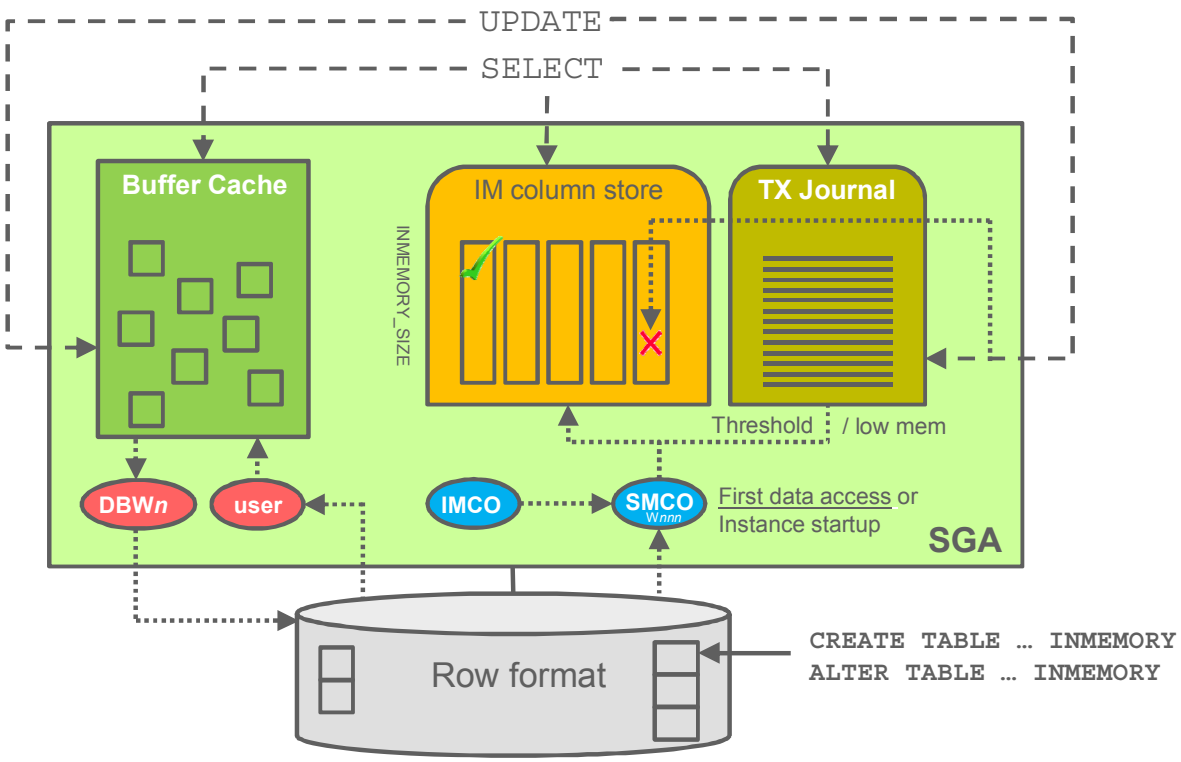
The DBA must decide, according to the types of queries and DMLs executed against the segments, which segments should be defined as non in-memory segments and those as in-memory segments. The DBA can also define more precisely which columns are good candidates for the IM column store:

- In **row format exclusively:** The segments being frequently accessed by OLTP style queries, operating on few rows returning many columns, are good candidates for the buffer cache. These segments should not necessarily be defined as in-memory segments and will be sent to the buffer cache only.
- In **dual format simultaneously:** The segments being frequently accessed by analytical style queries, operating on many rows returning few columns, are good candidates for the IM column store. If a segment is defined as an in-memory segment but has some columns defined as non in-memory columns, the queries that select any non in-memory columns are sent to the buffer cache and those selecting in-memory columns only are sent to the IM column store. Any fetch-by-rowid is performed on the segment through the buffer cache.

Any DML performed on these objects is executed via the buffer cache.

Dual Format In Memory

12.1.0.2



After allocating space to the IM column store in SGA, the DBA turns on the `INMEMORY` attribute at the object creation or when altering it to convert an existing object to an in-memory candidate. An in-memory table gets IMCUs allocated in the IM column store at first table data access or at database startup. An in-memory copy of the table is made by doing a conversion from the on-disk format to the new in-memory columnar format. This conversion is done each time the instance restarts as the IM column store copy resides only in memory. When this conversion is done, the in-memory version of the table gradually becomes available for queries. If a table is partially converted, queries are able to use the partial in-memory version and go to disk for the rest, rather than waiting for the entire table to be converted.

There is a new background process, `IMCO`, that creates and refreshes IMCUs to populate and repopulate the IM column store. `IMCO` is the background coordinator process which schedules the objects to be populated/repopulated in the IM column store. `SMCO` and `Wnnn` are background processes which actually populate the objects in memory.

When rows in the table are updated, the corresponding entries in the IMCUs are marked stale. The row version recorded in the journal is constructed from the buffer cache, and is unaffected by what subsets of columns are in the IMCU. IMCU synchronization is performed by `IMCO`/`SMCO`/`Wnnn` background processes with the updated rows populated in the transaction journal based on events:

- An internal threshold including a number of invalidations to the rows per IMCU
- Transaction journal running low on memory
- RAC invalidations

Deploying IM Column Store

1. Verify the database compatibility value.

```
COMPATIBLE = 12.2.0.0.0
```

2. Configure the IM column store size.

12.1.0.2

```
INMEMORY_SIZE = 100G
```

- You can dynamically **increase** the IM column store size.

12.2

```
SQL> ALTER SYSTEM SET inmemory_size = 110g scope=both;
```

No special installation is required to set up the feature. It is shipped in Oracle Database 12*c* Release 1 PS1.

1. The database compatibility must be set to 12.1.0.2, 12.2.0.0, or higher.
2. Configure the IM column store size by setting the `INMEMORY_SIZE` instance parameter. Use the K, M, or G letters to define the amount unit. Since Oracle Database 12.2, the size of the in-memory area can be dynamically increased after instance startup but not decreased. The memory allocated to the area is deducted from the total available memory for `SGA_TARGET`. There is no LRU algorithm to manage the in-memory objects. In-memory objects may be partially populated into the IM column store if there is not enough space to accommodate the entire object. When this object is queried, as much of the data from the column store is retrieved and the rest is retrieved either from the buffer cache, flash cache, or disk. The DBA can set priorities on objects to define which in-memory objects should be populated in the IM column store.
   Set the `INMEMORY_SIZE` parameter to a minimum of 128M and logically to at least the sum of the estimated size of in-memory tables.

The parameter can be set per-PDB to limit the maximum size used by each PDB. Note that the sum of the per-PDB values does not necessarily have to be equal to the CDB value. It may even be greater.

Deploying IM Column Store: Objects Setting

12.1.0.2

3. Enable/disable objects to be populated into IM column store.
   - Enable/disable a whole segment:
     - IMCUs are initialized and populated at query access only.

```
SQL> CREATE TABLE large_tab (c1 …) INMEMORY;          → Dual format
SQL> ALTER TABLE t1 INMEMORY ;                        → Dual format
SQL> ALTER TABLE sales NO INMEMORY;                   → Row format only
```

     - IMCUs can be initialized at database open.

```
SQL> CREATE TABLE test (…) INMEMORY PRIORITY CRITICAL;
```

     - Use MEMCOMPRESS to define the compression level.

```
SQL> ALTER TABLE t1 INMEMORY MEMCOMPRESS FOR CAPACITY HIGH;
```


3. Turn on the INMEMORY attribute at the object creation or when you alter it, to convert the object into a columnar representation in the IM column store. All columns of the in-memory table are populated into memory unless some columns are disabled by using the NO INMEMORY clause. It is recommended to specify all columns at once rather than having an ALTER TABLE for each column, as it is more efficient.
   Two INMEMORY subattributes define the following behaviors:
   - The loading priority of the object data in the IM column store: The INMEMORY clause can be decorated with the PRIORITY subclause. An in-memory table is populated into memory at the first data access by default. This default behavior is the "on demand" behavior. Using different priority levels, table data can be populated into the IM column store soon after the database starts up.
   - The degree of compression of the columns of the object in the IM column store: The INMEMORY clause can be decorated with the MEMCOMPRESS subclause.

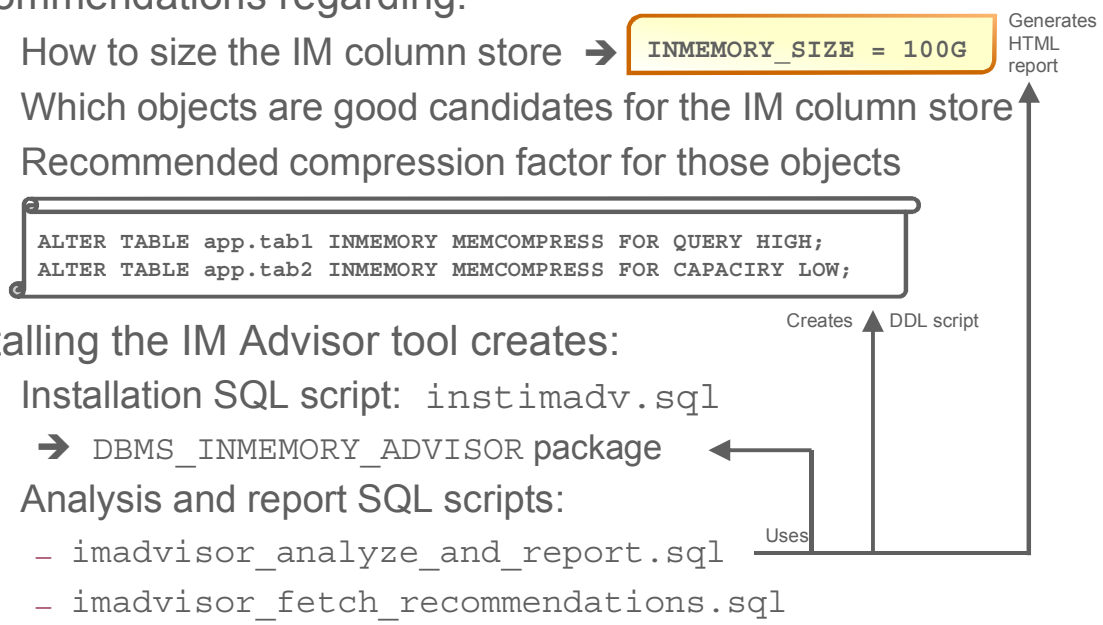
- The segments that are compatible with the INMEMORY attribute are the tables, partitions, subpartitions, inline LOBs, materialized views, materialized join views, and materialized view logs.
- Clustered tables and IOTs are not supported with the INMEMORY clause.

In-Memory Advisor

- The IM Advisor analyzes your workload and makes specific recommendations regarding:
  - How to size the IM column store ➔ `INMEMORY_SIZE = 100G`
  - Which objects are good candidates for the IM column store
  - Recommended compression factor for those objects

```
ALTER TABLE app.tab1 INMEMORY MEMCOMPRESS FOR QUERY HIGH;
ALTER TABLE app.tab2 INMEMORY MEMCOMPRESS FOR CAPACIRY LOW;
```

- Installing the IM Advisor tool creates:
  - Installation SQL script: `instimadv.sql`
    - ➔ `DBMS_INMEMORY_ADVISOR` package
  - Analysis and report SQL scripts:
    - `imadvisor_analyze_and_report.sql`
    - `imadvisor_fetch_recommendations.sql`



The purpose of `DBMS_INMEMORY_ADVISOR` is to provide estimates of the performance gains with different IM column store sizes. According to the IM column store size chosen, it provides recommendations for which tables/partitions/subpartitions to place in-memory, along with which in-memory compression methods provide the best estimated performance gain. Although it works with IM column store size estimates, it also employs a lot of database activity statistics like SQL plan cardinality, Active Session History (ASH), Automatic Workload Repository (AWR), and the use of parallel query to estimate performance gains with in-memory placement.

Therefore, the In-Memory Advisor is an interesting tool for two types of recommendations:

- Sizing the IM column store
- Finding the best candidates for in-memory population

Note: You can install the IM Advisor in a non-CDB and in a CDB, in the CDB root or in a PDB (The IM Advisor is licensed as part of the Oracle Tuning Pack). The installation is completed until integration in Oracle Database 12.2 through a downloadable `imadvisor.zip` file (My Oracle Support *Document ID 1965343.1*).

12.1.0.2

IM Advisor or Compression Advisor?

The compression advisor analyzes how much space in the IM column store is required for a specific object to use different in-memory compression levels.

```
BEGIN
   DBMS_COMPRESSION.GET_COMPRESSION_RATIO (
      'TS_DATA', 'SSB', 'LINEORDER', NULL,
       DBMS_COMPRESSION.COMP_INMEMORY_QUERY_LOW,
      blkcnt_cmp, blkcnt_uncmp, row_cmp, row_uncmp, cmp_ratio,
      comptype_str,10000,1);
 DBMS_OUTPUT.PUT_LINE('Block count uncompressed = ' || blkcnt_uncmp);
 DBMS_OUTPUT.PUT_LINE('Row count per block uncompressed=' || row_uncmp);
 DBMS_OUTPUT.PUT_LINE('Compression type = ' || comptype_str);
 DBMS_OUTPUT.PUT_LINE('Comp ratio= '|| cmp_ratio );
 DBMS_OUTPUT.PUT_LINE('   ');
 DBMS_OUTPUT.PUT_LINE('The IM column store space needed is about 1 byte
in IM column store for '|| cmp_ratio  ||' bytes on disk.');
END;
/
```

The purpose of DBMS_COMPRESSION is to give estimates of compression, both on disk and in memory, of a specific table, partition, subpartition, LOB, or other compatible type of object. Unlike the IM Advisor, it does not give any estimates for performance benefit.

Dividing the expected on-disk size by the reported ratio gives a proper estimate of the IM column store space required for the segment. Depending on whether DISTRIBUTE or DUPLICATE is specified, the size is either the total size or the size per RAC node.

Use new constants for the COMPTYPE parameter:

- COMP_INMEMORY_NOCOMPRESS: To test IM column store basic compression
- COMP_INMEMORY_QUERY_LOW/HIGH: To test IM column store compression for query
- COMP_INMEMORY_CAPACITY_LOW/COMP_INMEMORY_CAPACITY_HIGH: To test columnar format for capacity low/high compression
- COMP_INMEMORY_DML: In-Memory compression level for DML

Note: The DBMS_COMPRESSION does not require any license.

The result of the procedure execution could be the following:

```
 Block count uncompressed = 146
 Row count per block uncompressed = 68
 Compression type = "In-memory Memcompress Query Low"
 Comp ratio= 18
 The IM column store space needed is about 1 byte in IM column store
for 18 bytes on disk.
```

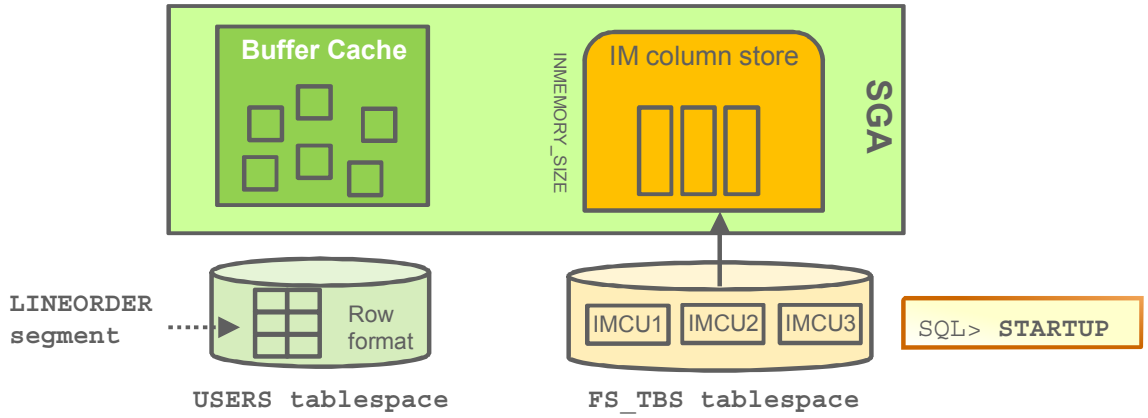
12.2

IM FastStart

Optimize the population of the IM column store when the database restarts.

1. Create the tablespace designated as the IM FastStart area.
2. Enable IM FastStart.

```
SQL> exec DBMS_INMEMORY_ADMIN.ENABLE_FASTSTART ('fs_tbs')
```



In Oracle Database 12.1.0.2, the columnar format has only been available in-memory. That meant that after a database restart, the In-Memory Column Store would have to be populated from scratch using a multiple step process that converts traditional row formatted data into the compressed columnar format and placed it in-memory.

In Oracle Database 12.2, In-Memory FastStart enables data to be repopulated into the In-Memory Column Store at a much faster rate than previously possible. This is done by saving a copy of the data currently populated in the In-Memory Column Store on disk in its compressed columnar format. The feature is supported only on Oracle-engineered systems.

- When you enable In-Memory FastStart (IM FastStart) for the IM column store, you specify a tablespace for the IM FastStart area. The IM FastStart area stores data that optimizes the population of the IM column store when the database restarts.
- When you disable In-Memory FastStart (IM FastStart) for the IM column store, the IM FastStart area is no longer maintained when the database is running, and IM FastStart is not used to populate the IM column store when the database restarts.

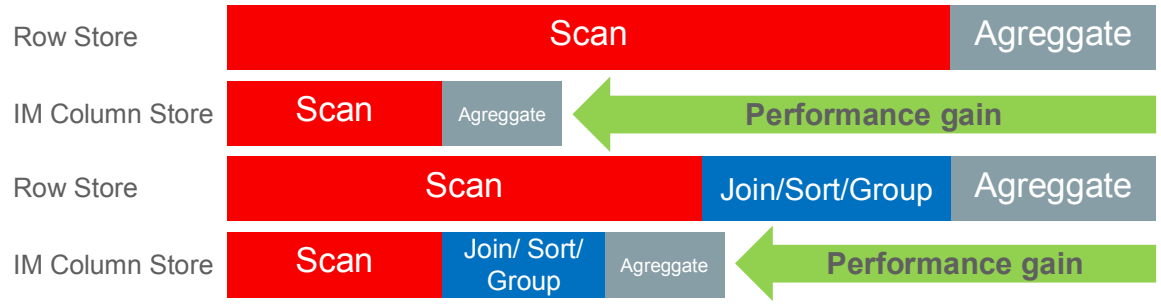
The `DBMS_INMEMORY_ADMIN` package provides procedures to enable and disable IM FastStart and migrate the IM FastStart to another tablespace, and a function to retrieve the name of the tablespace that is currently designated as the IM FastStart area.

Query Benefits

12.1.0.2

Ideal for:

- Scanning large numbers of rows and applying filters
- Querying a subset of columns in a table
- Joining small tables (dimensions) to larger tables (facts)
  - Leverage repeated dimension values in fact table
- Aggregating data



In-memory column store capability is ideal for queries that perform operations such as:

- Scanning large numbers of rows and applying filters like: =, <, >, in-lists
- Querying a subset of columns in a table (for example, select 5 of 100 columns in a table)
- Joining small tables (dimensions) to larger tables (facts)
  - Leverage repeated dimension values in fact table
- Aggregating data

Queries on In-Memory Tables: Join Groups

12.2

Optimize queries using frequently joined columns:

```
SQL> SELECT SUM(lo_extendedprice * lo_discount) revenue
     FROM   oe.lorder l, app.date_dim d
     WHERE  l.orderdate = d.datekey;
```

- Define join groups ➔ Primary column values coded

Columns Joined	Domain #
app.date_dim.datekey (master table)	1
oe.lineorder.orderdate	1
oe.shipments.ship_date	1

DBA_JOINGROUPS

Join Group Name	Domain #
j1	1

IM_DOMAIN$

23-DEC-14	4
13-JAN-15	2
10-APR-15	6
28-FEB-16	7

```
SQL> CREATE INMEMORY JOIN GROUP j1
       ( app.date_dim(datekey),
         oe.lorder(orderdate));
```

```
SQL> ALTER INMEMORY JOIN GROUP j1
       ADD (oe.shipments(ship_date));
```

ORACLE

Creating join groups on columns from the same table or from different columns frequently used in join optimizes the performance of queries on these joined columns.

This operation creates a segment storing dictionary codes from the values of the column defined as the primary column. The primary column is the column defined in the master table, which is the first table declared in the join group.

The segment or global dictionary holds a set of master dictionary IDs corresponding to a single column for a set of IMCUs. The columns in a join group share the same dictionary codes, which saves expensive memory copies between one row source to another.

When creating a join group, the columns participating in the join group must belong to the same domain. For example, consider the column `name` in the `scott.emp` and `scott.address` tables. The column `name` belongs to the same domain. Their values fall in similar ranges. Thus, these two columns belonging to the same domain tables can be meaningfully joined on those columns. Creating a join group creates a new domain.

When creating a join group, a new SQL object is created that lists columns belonging to the same domain. Join groups can be created on concatenated columns.

Join Groups Example

12.2

Departments Table

Row_id	AV	Comp Data
	1	
	2	
	3	
	4	

Metadata stored for Join Group

HR	1
IT	2
MG	3
VP	4

EmployeesTable

Row_id	AV	Comp Data
	3	
	4	
	3	
	1	
	1	
	2	
	4	
	3	
	2	

In the slide above a Join Group created on `Department_ID` column of the `Departments` and `Employees` tables as the two tables join on this column. The metadata for this join group store an array showing the assigned value for each `Department_ID` and a column is added to each IM table in memory that stored the array value associated to each row of the tables. Then the records are joined based on the array value (AV) assigned to each row of the tables without requiring the data to be decompressed for a hash join operation that would had been used if the join groups were not created.

Population of Expressions and Virtual Columns Results

12.2

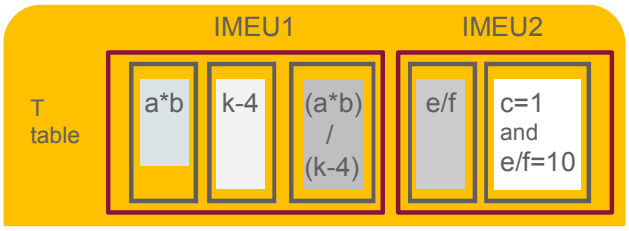
- Query performance improved by caching:
  - Results of frequently evaluated query expressions

```
INMEMORY_EXPRESSIONS_USAGE = ENABLE
```

  - Results of user-defined virtual columns

```
INMEMORY_VIRTUAL_COLUMNS = ENABLE
```

  - Results of useful internal computations
- Results stored in IM expression units (IMEUs) for subsequent reuse



- Candidates detected by `DBMS_INMEMORY_ADMIN.IME_CAPTURE_EXPRESSIONS` and eligible according to expression statistics stored in the Expression Statistics Store (ESS)

Automatically identifying frequently used complex expressions or calculations and then storing their results in the IM column store can improve query performance. Storing precomputed virtual column results can also significantly improve query performance by avoiding repeated evaluations.

The cached results can range from function evaluations on columns used in application, scan, or join expressions, to bit-vectors derived during predicate evaluation for in-memory scans. Caching can also address other internal computations that are not explicitly recited in a database query, such as hash value computations for join operations.

Where are the In-Memory expressions and virtual column results (IMEs) stored? An IMCU is a basic unit of the in-memory copy of the table data. Each IMCU has its own in-memory expression unit (IMEU), which contains expression results corresponding to the rows stored in that IMCU.

Why are expressions and virtual columns considered good IME candidates? Statistics such as frequency of execution and cost of evaluation on a per-segment basis are regularly maintained by the optimizer and stored in the Expression Statistics Store (ESS). ESS uses an LRU algorithm to automatically track which expressions are most frequently used.

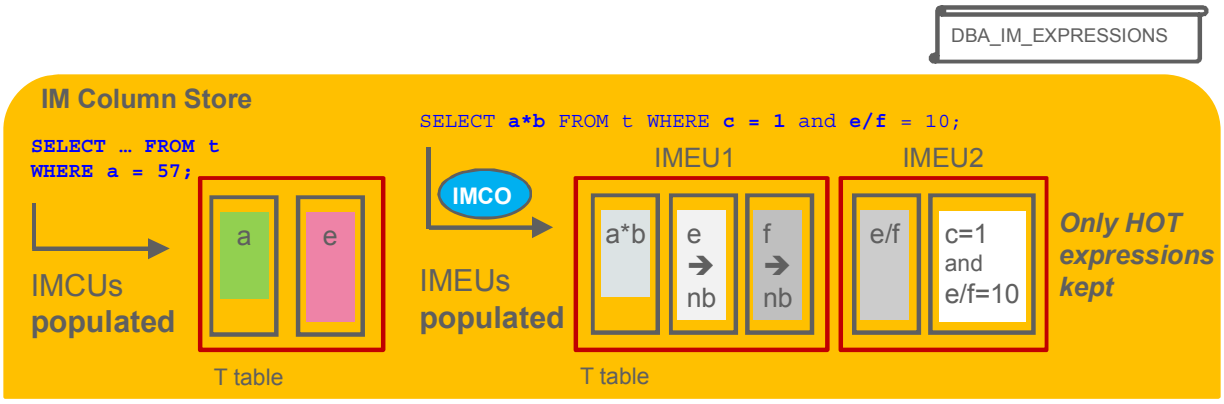
Population of Expressions and Virtual Columns Results

Instance Parameters

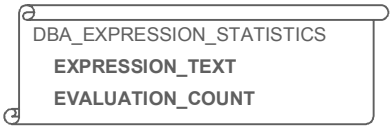
- DBMS_INMEMORY_ADMIN.IME_CAPTURE_EXPRESSIONS procedure identifies the most frequently accessed ("hottest") expressions in the database in the specified time range, materializes them as hidden virtual columns, and adds them to their respective tables during the next repopulation.
- IME_POPULATE_EXPRESSIONS procedure forces the population of expressions captured in the latest invocation of DBMS_INMEMORY_ADMIN.IME_CAPTURE_EXPRESSIONS.
- INMEMORY_EXPRESSIONS_USAGE: The parameter controls which IMEs are eligible to be populated in the IM column store and available for queries.
  - STATIC_ONLY: Tables enabled for in-memory and containing certain data types such as Oracle numbers or JSON will have these columns populated in the IM column store using a more efficient representation.
  - DYNAMIC_ONLY: Frequently used or "hot" expressions detected within the capture window will be automatically created and populated into the IM column store.
  - ENABLE: Both static and dynamic IM expressions will be populated into the IM column store and available to be used by queries. This is the default value.
  - DISABLE: No IM expressions of any kind will be populated into the IM column store.
- INMEMORY_VIRTUAL_COLUMNS: This parameter controls the default IM population behavior of user-defined virtual columns.
  - ENABLE: For a table or partition that is enabled for in-memory storage, all virtual columns are stored in-memory at the default table or partition MEMCOMPRESS level unless:
    - They have been explicitly excluded using the NO INMEMORY attribute.
    - They have been altered to have a different MEMCOMPRESS level than the base table or partition, in which case they are stored at the specified MEMCOMPRESS level.
  - MANUAL: For a table or partition that is enabled for in-memory storage, no virtual columns are stored in-memory unless:
    - They have been explicitly marked for INMEMORY, in which case they are stored in-memory at the table or partition MEMCOMPRESS level.
    - They have been marked for INMEMORY with a different MEMCOMPRESS level than the base table or partition, in which case they are stored at the specified MEMCOMPRESS level.
  - DISABLE: For a table that is enabled for in-memory storage, no virtual columns will ever be stored in-memory. Any changes to the INMEMORY attributes for a virtual column, including changes in MEMCOMPRESS level, are recorded but not acted upon with regards to population of virtual columns.

In-Memory Expression Unit (IMEU)

12.2



- Only hot expressions are kept in the IM column store.
- IMEUs are kept consistent with the data in parent IMCUs (especially with DML activity).
- Space unused by IMEUs can be reclaimed by IMCUs.
- The ESS are persisted to disk and kept consistent with DDLs.



In the second example in the slide, there are several potential good candidates for in-memory expressions (IMEs).

- '`a*b`': If queries on T table consistently evaluate the costly expression `a*b`, caching the expression result in an IMEU for future reuse can provide significant speedup of subsequent query executions.
- Data conversion of number columns '`e`' and '`f`' into their respective native representations for faster computation of the fraction '`e/f`
- '`e/f`': If queries on T table consistently evaluate the costly expression `e/f`, caching the expression result in an IMEU for future reuse can provide significant speedup of subsequent query executions.
- Internal computation of the predicates 'c=1' and 'e/f = 10,' resulting in a bit-vector

None of these expression results are stored directly on-disk, and hence must be recomputed every time the previous query is executed. Storing them in an IMEU can provide significant speedup of subsequent query executions.

Selection of IMEs relies on the statistics maintained in the ESS. Thus, the ESS must be kept consistent with DDLs and other changes to the underlying table.

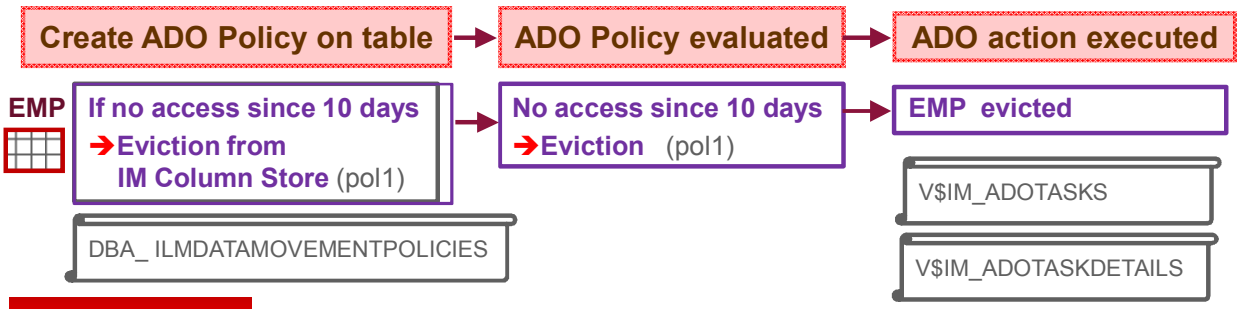
The DBA can decide to drop all expressions using the `DBMS_INMEMORY_ADMIN.IME_DROP_ALL_EXPRESSIONS` procedure or one particular expression using the `DBMS_INMEMORY.IME_DROP_EXPRESSIONS` procedure.

Automatic Data Optimization Interaction

12.2

- Types of ADO In-Memory policies based on heat map statistics: `HEAT_MAP = ON`
  - Policy to set IM attribute
  - Policy to modify IM compression
  - Policy to unset IM attribute ➔ Eviction from IMCS

```
SQL> CREATE TABLE app.emp (c number) INMEMORY;
SQL> ALTER TABLE  app.emp ILM ADD POLICY
          NO INMEMORY SEGMENT AFTER 10 DAYS OF NO ACCESS;
```



In Oracle Database 12.1.0.2, a segment that is populated in the IM column store is removed only if the segment is dropped, moved, or the `inmemory` option on the segment is removed. This behavior can result in memory pressure if the size of the data to be loaded into the memory is more than the free space available in the IM column store. The performance of the user workload would be optimal if the IM column store contains the most frequently queried segments.

In Oracle Database 12.2, there are three types of ADO (Automatic Data Optimization) In-Memory policies.

- You can define an ADO policy for the anticipated length of inactivity (`NO ACCESS` or `MODIFICATION`) that would indicate an eviction of the object from the IM column store. The ADO policy considers heat map statistics. The object is kept in the IM column store as long as the activity does not subside. Eviction unsets the `inmemory` attribute on the object.
- You can define an ADO policy to set the `inmemory` attribute on an object. This type of policy allows specification of an IM clause as part of the ADO policy clause and annotates the table/partition with this IM clause when the policy condition is satisfied. It does not populate the segment to the IM store; the segment gets populated based on the priority in the IM clause.

```
SQL> ALTER TABLE t1 ILM ADD POLICY SET INMEMORY AFTER 5 days OF creation;
```

- Policy to modify IM compression: Change the compression level of an object from a lower level of compression to a higher level.

```
SQL> ALTER TABLE t1 ILM ADD POLICY MODIFY INMEMORY MEMCOMPRESS FOR QUERY HIGH
   AFTER 10 days OF no access;
```

Summary

In this lesson, you should have learned how to:

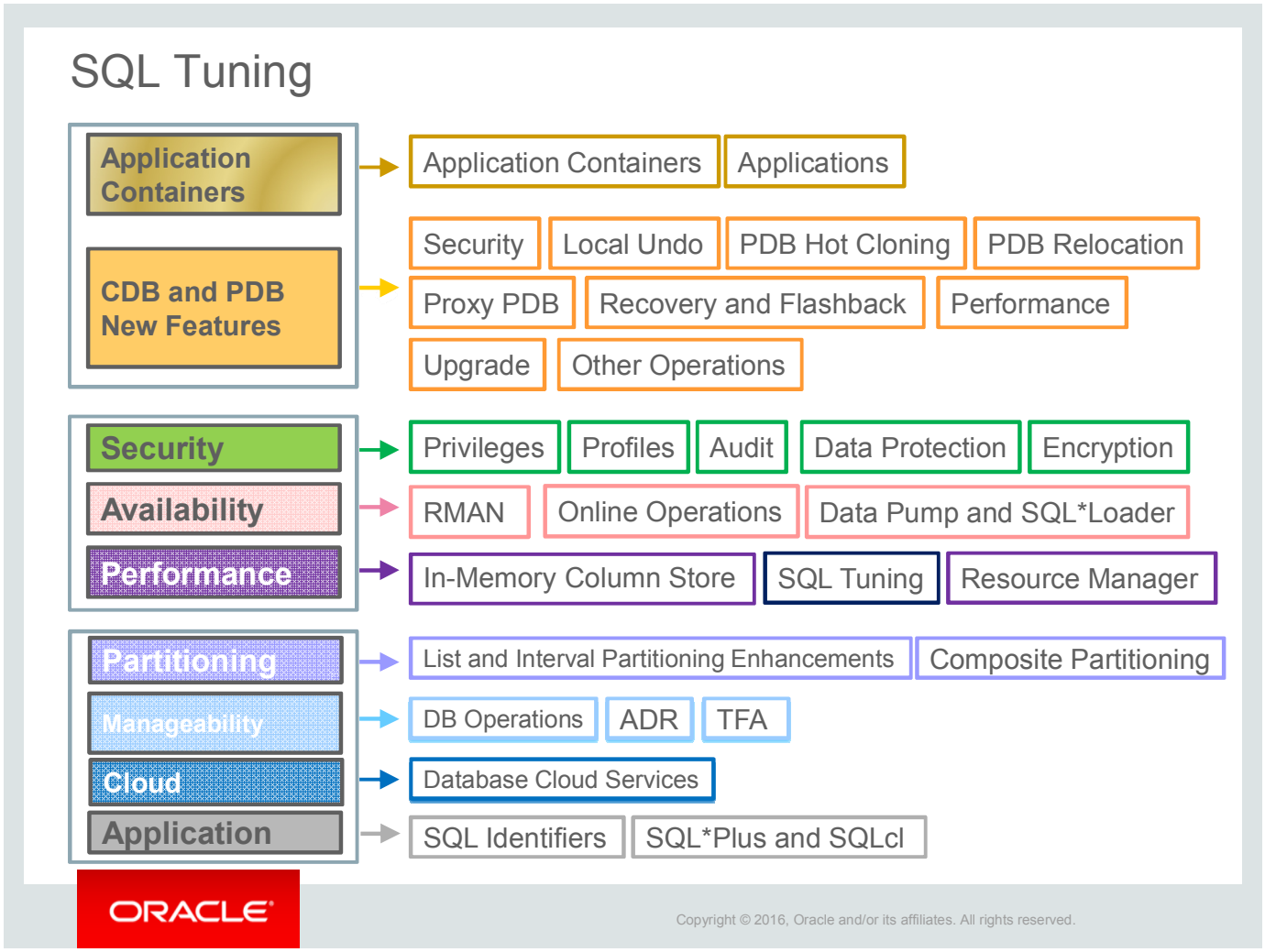
- Explain how to deploy the In-Memory Column Store in a database
- Benefit from the Oracle Database In-Memory Advisor
- Configure IM FastStart to accelerate IM segment population at startup
- Optimize join queries by creating join groups
- Store expressions and virtual column values in the IM Column Store
- Use heat map statistics and ADO policy to add, evict or compress objects in the IM Column Store

Practice 16: Overview

- 16-1: Resizing the IM Column Store using the IM Advisor
- 16-2: Increasing query performance by using join groups
- 16-3: Optimizing queries by implementing expressions capture in the IM Column Store
- 16-4: Creating an ADO policy for table eviction of the IM Column Store

17

SQL Tuning Enhancements



There are several optimizer advisor enhancements, SQL Plan Management (SPM) and SQL Performance Analyzer (SPA) enhancements, and new JOIN methods.

Objectives

After completing this lesson, you should be able to:

- Describe SQL Plan Management (SPM) enhancements
- Minimize DBA intervention to get the right statistics
- Configure and use the Optimizer Statistics Advisor
- Explain SQL Performance Analyzer (SPA) enhancements
- Describe Database Replay enhancements
- Use new SQL JOIN processing
- Explain Continuous Adaptive Query Plans

To get detailed information about how to perform any of the operations explained in this lesson, refer to the following guides in the Oracle documentation:

- *Oracle Database Administrator's Guide 12c Release 2 (12.2)*
- *Oracle Database SQL Tuning Guide 12c Release 2 (12.2)*
- *Oracle Database Testing Guide 12c Release 2 (12.2)*

Performance Enhancements

- Lead to:
  - Accurate diagnosis of a performance problem
  - Improved Oracle Quality of Service Management
  - Better quality testing with the lowest risk and effort
- Enhance system performance and reliability
- Lower your overall management costs

SQL Performance Analyzer and SQL Plan Management enhancements, and those in the Optimizer Statistics collection, lead to accurate diagnosis of a performance problem, improved Oracle Quality of Service Management, and better quality testing with the lowest risk and effort. It also enhances system performance and reliability, and lowers your overall management costs.

SQL Plan Management Enhancements

- Include alternate plans in the SPM Evolve Advisor list:

The evolve function is an automatic advisory task: SYS_AUTO_SPM_EVOLVE_TASK (12.1)

DBAs to create a task that evolves plans:

```
DBMS_SPM.CREATE_EVOLVE_TASK
DBMS_SPM.EXECUTE_EVOLVE_TASK
DBMS_SPM.REPORT_EVOLVE_TASK
DBMS_SPM.IMPLEMENT_EVOLVE_TASK
```

Searches other repositories for good plans: (12.2)

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

- Capture plan baselines using DBMS_SPM.**CONFIGURE** procedure:
  - AUTO_CAPTURE_SQL_TEXT
  - AUTO_CAPTURE_PARSING_SCHEMA_NAME
  - AUTO_CAPTURE_MODULE/AUTO_CAPTURE_ACTION

DBA_SQL_MANAGEMENT_CONFIG

- Load plan baselines from AWR:
  DBMS_SPM.**LOAD_PLANS_FROM_AWR**

In Oracle Database 12.1, SQL Plan Management (SPM) only works on nonaccepted plans that the optimizer has found and already added to its plan history. In other words, the evolve process (both manual and auto) operates only on SPM-managed statements. SPM Evolve Advisor is a SQL advisor that evolves plans that have recently been added to the SQL plan baseline. The advisor simplifies plan evolution by eliminating the manual chore of evolution. Functions introduced in the DBMS_SPM package in Oracle Database 12.1 work on the advisory task infrastructure.

In Oracle Database 12.2, the SPM Evolve Advisor can search for additional plans that are not yet in the SQL Management Base (SMB) into other sources like AWR, STS, and cursor cache. If such plans are found, the advisor tries to evolve them 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. The ability to look into SQL plans in the AWR is subject to license requirements for the Diagnostic pack.

Three new task parameters are added to the existing SET_EVOLVE_TASK_PARAMETER() function.

SQL Plan Management Enhancements

- ALTERNATE_PLAN_SOURCE: Determines which sources to search for alternate plans
  - CURSOR_CACHE, AUTOMATIC_WORKLOAD_REPOSITORY, or SQL_TUNING_SETS
  - Multiple values can be combined by delimiting with '+'. The default value is CURSOR_CACHE+AUTOMATIC_WORKLOAD_REPOSITORY.
- ALTERNATE_PLAN_BASELINE: Value EXISTING indicates that alternate plans should be loaded for statements with existing SQL plan baselines. Value NEW indicates that alternate plans should be loaded for statements which do not have a SQL plan baseline, in which case a new baseline for the statement will be created. Value EXISTING+NEW loads alternate plans in both cases.
- ALTERNATE_PLAN_LIMIT: The maximum number of alternate plans to load. Default value is 0.

Four new parameters are added to the existing procedure CONFIGURE() for the auto capture filters.

- AUTO_CAPTURE_SQL_TEXT: The SQL text 'select a%' in the following example will be used as a search pattern of LIKE predicate.

```
SQL> exec DBMS_SPM.CONFIGURE('AUTO_CAPTURE_SQL_TEXT', 'select a%', TRUE)
```

- AUTO_CAPTURE_PARSING_SCHEMA_NAME: If the third value is set to FALSE, the schema name is used as a search pattern of NOT LIKE predicate.

```
SQL> exec DBMS_SPM.CONFIGURE('AUTO_CAPTURE_PARSING_SCHEMA_NAME', 'OE',
   FALSE)
```

- AUTO_CAPTURE_MODULE
- AUTO_CAPTURE_ACTION

A new function LOAD_PLANS_FROM_AWR() is added in the DBMS_SPM package. This function can be used to load the SQL Management Base (SMB) with SQL plan baselines for a set of SQL statements using the plans from the AWR.

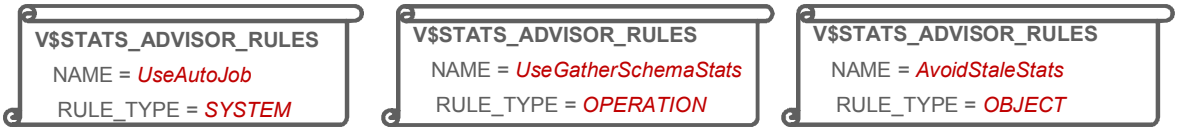
The function uses the following parameters:

- BEGIN_SNAP: Begin snapshot
- END_SNAP: End snapshot
- BASIC_FILTER: SQL predicate to filter the SQL from AWR. NULL means all plans in AWR are selected.
- FIXED: Default NO means the loaded plans will not change the current FIXED property of the SQL plan baseline into which they are loaded.
- ENABLED: Default YES means the loaded plans will be considered by the optimizer.
- COMMIT_ROWS: Number of SQL plans to load before doing a periodic commit. This helps to shorten the undo log.

Optimizer Statistics Advisor

If best practices change in a new release, Optimizer Statistics Advisor encodes these practices in its rules.

- ➔ The advisor always provides the most up-to-date recommendations.
  - Track and analyze how statistics are collected.
    - Class of findings: System, Operations, Objects



  - Scope of findings
    - Problems with gathering of statistics
    - Status of automatic statistic gathering jobs
    - Quality of current statistics
- ➔ Suggestion for changes to the statistics collection

Since the introduction of the Cost-Based Optimizer, Optimizer Statistics plays a significant part in determining the execution plan for queries. Therefore, it is critical for the Optimizer to have accurate and up-to-date statistics. The Optimizer provides a comprehensive package—dbms_stats—dedicated for this purpose and it is improved in every release by adding new features. However, under many circumstances, these new features have not been fully utilized by customers or are being used in incorrect ways. Customers often use scripts and settings from one release to the next, based on earlier experience. These settings and methods may have been superseded, or produce statistics that no longer give the most effective Optimizer results.

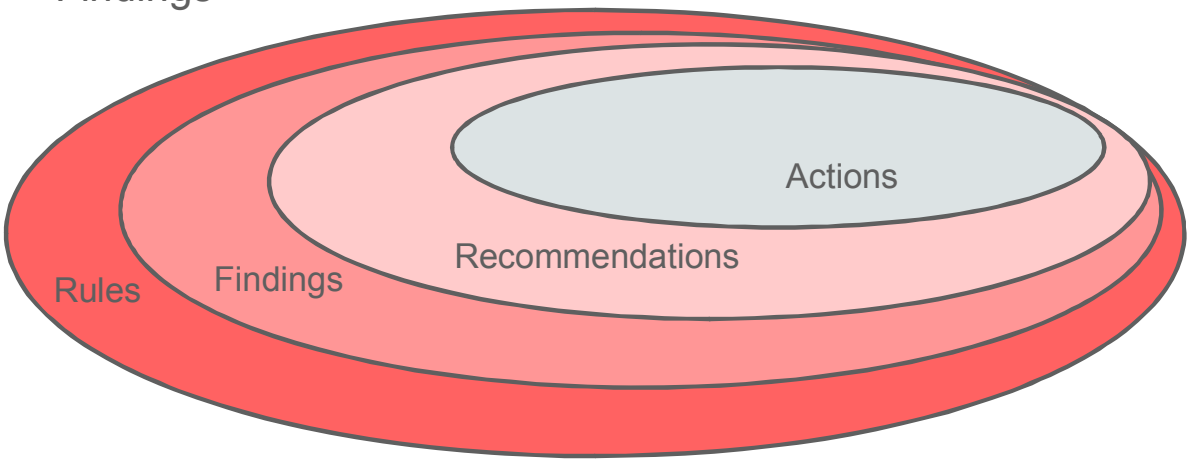
The Optimizer Statistics Advisor in Oracle Database 12.2 uses rules consistent with the current release, to recommend changes to the way statistics are being gathered.

The advisor has a set of rules or recommended practices that are compared against the current statistics to generate findings. The rules are applied at the system, operation, or object level, such as whether the Automatic Gather Statistics jobs are scheduled, the Statistics Gathering procedures are using default parameters, and statistics are consistent across related objects. These rules check on issues related to the gathering of statistics— the schedules, parameters, and errors related to the automatic statistics gathering jobs. The rules include a variety of object-related issues, including whether incremental mode setting is efficient.

Optimizer Statistics Advisor Report

Report sections:

- Header
- Summary
- Errors
- Findings

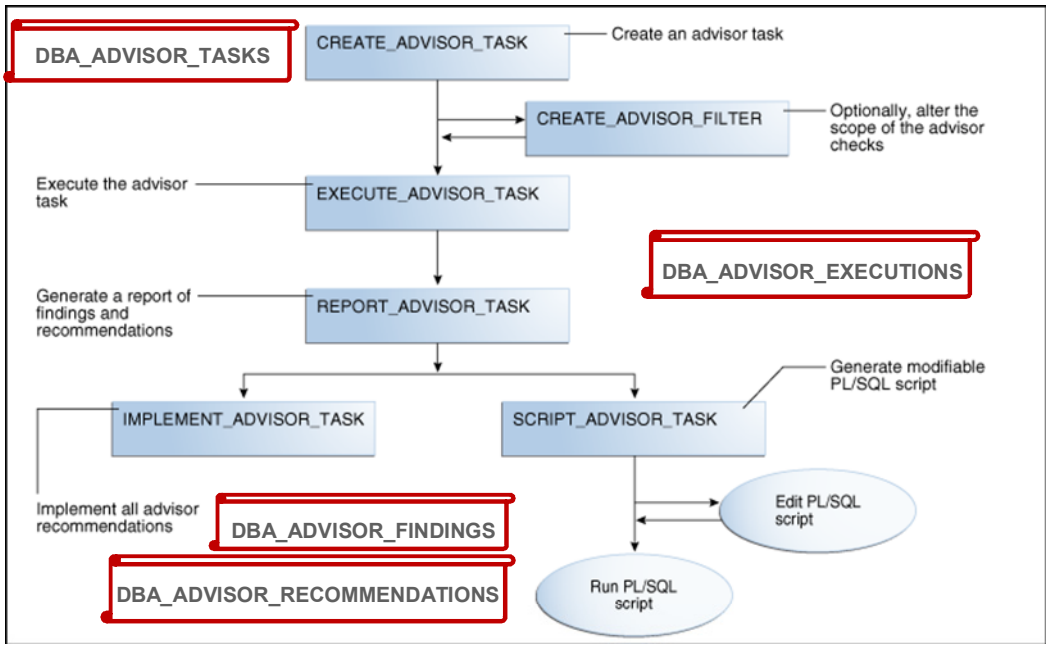


The Optimizer Statistics Advisor report has four basic concepts:

- Rules: Check the current configuration, history, and current statistics. Rules are added and changed by release to reflect best practices.
- Findings: They are generated when rules are not followed. An individual rule may generate many findings, but each finding is generated by only one rule. Some findings may be informational only, such as object staleness.
- Recommendations: They are responses the customer could make to resolve the finding. It is possible that several recommendations could be generated, and further investigation would be required by the customer. One or more rationales are given for each recommendation. Not all findings generate recommendations.
- Actions: They are PL/SQL statements or commands that the user can simply run in the command line to solve problems. They are provided in the form of scripts. Not all recommendations generate actions. For some recommendations, it is not possible to generate an action.

The report has sections for header, summary, errors, and findings. The header includes the advisor task parameters. The summary lists findings, and the errors section lists any errors the task encountered. The findings section includes the rule, findings, recommendations, and actions for each rule that produces a finding.

Executing Optimizer Statistics Advisor Tasks



An Optimizer Statistics Advisor task can be executed with PL/SQL calls. Each task must be provided a unique task name. The definition of the `CREATE_ADVISOR_TASK()` function parameters are:

- `TASK_NAME`: Name of the Statistics Advisor task
- `TIME_LIMIT`: The maximum duration the task can run

A filter list can be applied to the task to limit the scope of an advisor task using inclusion or exclusion lists for a user-specified set of rules, schemas, or operations. System rules are always checked. For example, you can configure an advisor task to include only recommendations for the `SH` schema. Also, you could exclude all violations of the rule for stale statistics.

You can create filters with the following `DBMS_STATS` procedures either individually or in combination, `CONFIGURE_ADVISOR_OBJ_FILTER`, `CONFIGURE_ADVISOR_RULE_FILTER` and `CONFIGURE_ADVISOR_OPR_FILTER`. An Optimizer Statistics Advisor Report can be generated with PL/SQL calls. The `report_advisor_task` function produces a report in text, HTML, or XML format. The `implement_advisor_task` implements the recommendations of the task based on the filters in place. An additional parameter, `LEVEL`, can be set to either `TYPICAL` or `ALL`. `TYPICAL` is the default and `ALL` ignores the filters.

SQL Performance Analyzer Enhancements

Three new task parameters:

- EXECUTE_TRIGGERS: Use this parameter to enable or disable database triggers that are fired recursively when a SPA trial is run in FULLDML mode.
- REPLACE_SYSDATE_WITH: Use this parameter in SPA trials to set a fixed date for SQL that references the SYSDATE function.
- NUM_ROWS_TO_FETCH: This parameter enables you to restrict the number of rows that a SQL statement fetches based on the optimizer mode setting.

In Oracle Database 12.1, database triggers are run by default irrespective of whether test execute is set in FULLDML mode or not during the SQL Performance Analysis task. (EXECUTE_FULLDML set to TRUE executes DML statements fully and rolls back DML execution to prevent persistent changes from being made by the DML).

Oracle Database 12.2 introduces a new task parameter, EXECUTE_TRIGGERS, which is set to FALSE by default to disable database triggers during test SPA task execution.

```
SQL> EXEC dbms_sqlpa.set_analysis_task_parameter
                    ('sqlpa_task1', 'EXECUTE_TRIGGERS', 'FALSE')
```

This parameter only affects running triggers within the SPA task execution. The trigger can potentially be run outside the SPA task execution in a different session.

In Oracle Database 12.1, whenever a SQL or PL/SQL is test executed and contains a call to SYSDATE, that call would always return the current SYSDATE. This could potentially lead to divergence in stats and plans.

Oracle Database 12.2 introduces a new task parameter, REPLACE_SYSDATE_WITH, which can be set to SQLSET_SYSDATE to return the SYSDATE captured in STS. After this is set, all calls to SYSDATE within the SPA task execution returns this date. The two possible values for this parameter are CURRENT_SYSDATE and SQLSET_SYSDATE.

```
SQL> EXEC dbms_sqlpa.set_analysis_task_parameter
          ('sqlpa_task2', 'REPLACE_SYSDATE_WITH', 'SQLSET_SYSDATE')
```

SQL Performance Analyzer Enhancements

In Oracle Database 12.1, whenever a task is test executed and if a SQL within it happens to return a lot of rows, there is no option to restrict the fetch to a specified number of rows. This is especially problematic for applications like Siebel, which do not fetch all the rows but restrict them to a specified number of rows.

Oracle Database 12.2 introduces a new task parameter, `NUM_ROWS_TO_FETCH`, which indicates to SPA that only a certain number of rows needs to be fetched for that SQL. The parameter can have the following values:

- `ALL_ROWS`: Fetch all the rows for this SQL. This is the default value.
- `AVERAGE`: Fetch the average number of rows from the past SPA executions.
- `AUTO`: Fetch the same number of rows as specified in the optimizer mode.
- A valid number: Fetch the exact number of rows as specified by the user.

```
SQL> EXEC dbms_sqlpa.set_analysis_task_parameter
              ('sqlpa_task3', 'NUM_ROWS_TO_FETCH', 'ALL_ROWS')
```

If the parameter is set to `ALL_ROWS`, `AUTO`, or `AVERAGE`, then the number of rows fetched can vary from one SQL to another.

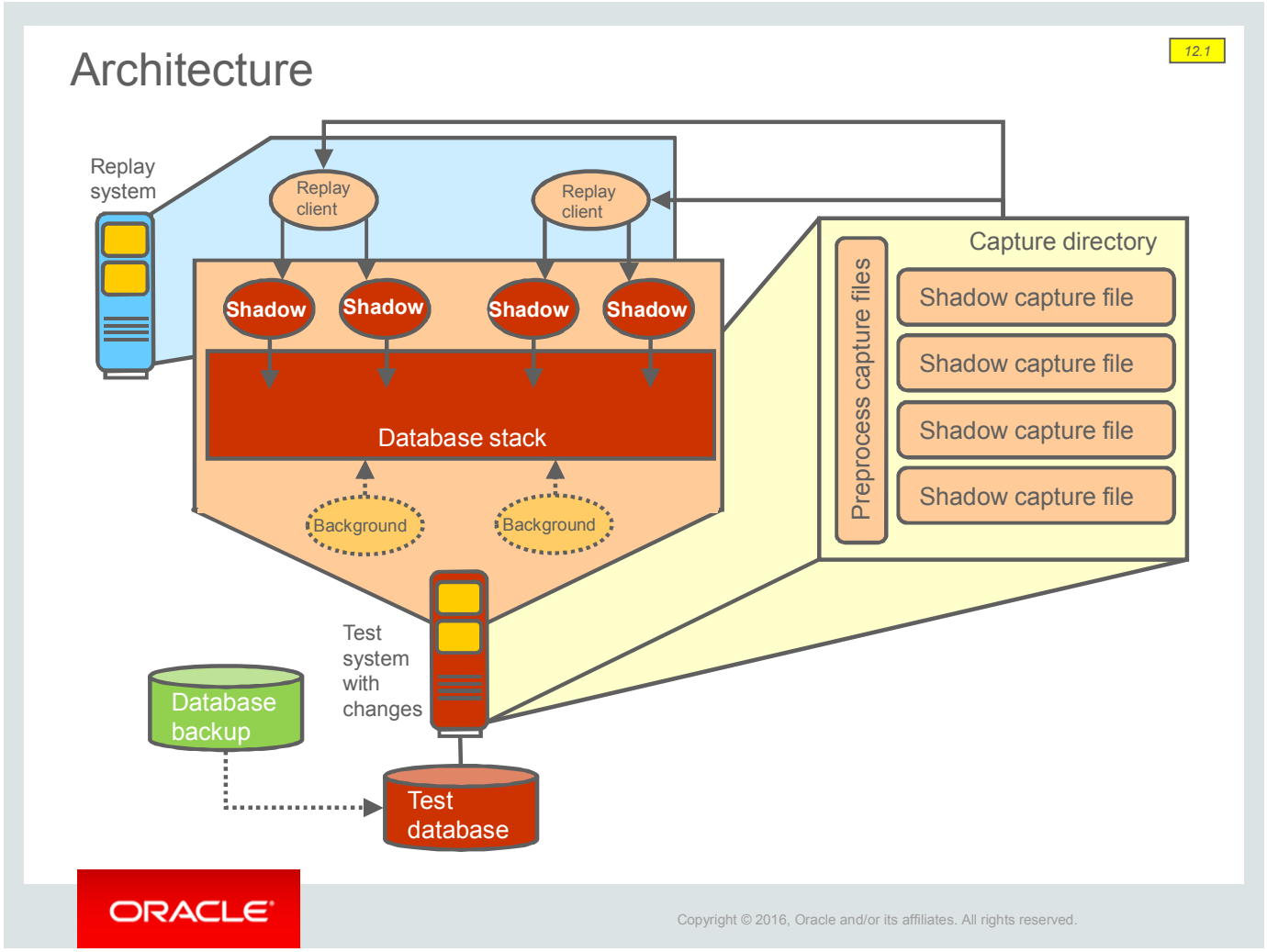
DB Replay Enhancements

Specify how user PL/SQL calls are handled during workload capture and workload replay with DB Replay.

- Starting a workload capture
- Preprocessing a database workload
- Initializing replay data

Database Replay has always had the ability to capture and replay PL/SQL. It has done so by capturing and replaying the top-level PL/SQL calls. As of Oracle Database 12*c* Release 2 (12.2), Database Replay gives you the choice of replaying the top-level PL/SQL calls or the recursive SQL called within the PL/SQL procedures.

Depending on the workload, you can specify how user PL/SQL calls are handled during workload capture and workload replay with DB Replay. The new replay mode can perform replays with more accuracy and less divergence, allowing faster, easier, and more complete testing of workloads having a lot of PL/SQL.



With Oracle Database 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 of Database Replay offers confidence in the ease of upgrade during a database server upgrade. Other useful application of Database Replay is the performance testing of a new server configuration like RAC, and debugging.

Capture the workload on the production system and then, before replaying the workload on the replay system, be sure to do the following:

1. Restore the replay database on a test system to match the capture database at the start of the workload capture.
2. Make changes (such as performing an upgrade) to the test system as needed.
3. Copy the workload to the test system.

The workload recording is consumed by a special application called the *replay driver*, which sends requests to the RDBMS on which the workload is replayed. The RDBMS on which the workload is replayed is usually a test system. The replay driver consists of one or more clients that connect to the replay system, and the replay driver sends requests based on the workload capture. The replay driver equally distributes the workload capture streams among all the replay clients based on the network bandwidth, CPU, and memory capability.

Database Replay Workflow

DBA_WORKLOAD_CAPTURES
PLSQL_MODE = *EXTENDED*
TOP_LEVEL

1. Capture the workload on the PROD database: start/ workload/stop:

```
SQL> exec DBMS_WORKLOAD_CAPTURE.START_CAPTURE ( name => 'june_peak',-
                          dir => 'jun16', plsql_mode => 'extended')
```

12.2

```
SQL> exec DBMS_WORKLOAD_CAPTURE.FINISH_CAPTURE()
```

2. Restore the replay database on a test system.
3. Make changes to the test system as required.
4. Copy the workload to the test system.
5. Preprocess the captured workload:

```
SQL> exec DBMS_WORKLOAD_REPLAY.PROCESS_CAPTURE ( capture_dir => 'jun16',-
                                        plsql_mode => 'extended')
```

12.2

6. Configure the test system for the replay:

```
SQL> exec DBMS_WORKLOAD_REPLAY.INITIALIZE_REPLAY(replay_name => 'j_r' ,-
                    replay_dir => 'jun16', plsql_mode => 'extended')
SQL> exec DBMS_WORKLOAD_REPLAY.REMAP_CONNECTION(…) / PREPARE_REPLAY(…)
SQL> exec DBMS_WORKLOAD_REPLAY.START_REPLAY()
```

12.2

7. Replay the workload on the restored database:

```
$ wrc userid=system password=oracle replaydir=/home/oracle/dbreplay
```

ORACLE

The optional `plsql_mode` parameter determines how PL/SQL is handled by DB Replay during capture, preprocessing, and replays.

Two values can be set for the parameter during capturing, preprocessing, and replays:

- `top_level`: With this setting, the entire PL/SQL block is captured, preprocessed, and replayed, which is how DB Replay handled PL/SQL prior to Oracle Database 12*c* Release 2 (12.2.0.1). This is the default value.
- `extended`: With this setting, user PL/SQL calls as well as their internal SQL statements are captured.

The `extended` value can be set only for workloads that were captured with the `plsql_mode` parameter set to `extended`. If `extended` is specified, but the capture was not executed in `extended` mode, then you will receive an error message.

SQL Processing: N-Way Hash Join and Band Join

- N-Way Hash Join
  - Join multiple tables at a time
  - Explain plan operation: `N-WAY HASH JOIN`

```
---------------------------------------
| Id  | Operation            | Name   |
---------------------------------------
|   0 | SELECT STATEMENT     |        |
|   1 |  SORT AGGREGATE      |        |
|   2 |   N-WAY HASH JOIN    |        |
|   3 |    TABLE ACCESS FULL | TEDGES |
|   4 |    TABLE ACCESS FULL | TEDGES |
|   5 |    TABLE ACCESS FULL | TEDGES |
---------------------------------------
```

- Band Join, new non-equijoin
  - Reduce the number of candidate rows
  - Enable hash join
  - Explain plan operations:
    - `HASH JOIN BAND`
    - `MERGE JOIN BAND`

```
---------------------------------------
| Id  | Operation            | Name   |
---------------------------------------
|   0 | SELECT STATEMENT     |        |
|*  1 |  HASH JOIN BAND      |        |
|   2 |   TABLE ACCESS FULL  | EMP    |
|   3 |   TABLE ACCESS FUL   | EMP    |
---------------------------------------
```


Traditional joins are only able to join two tables at a time, so multitable joins are performed a pair at a time and the result is joined with another table. This can often cause an explosion of intermediate result rows, causing large spills to disk, I/O, and temporary memory usage. The traditional hash join creates a hash table from the smallest input row source, creating hash buckets (partitions) that hold the rows that contain the join column value that hash to a particular hash value. The rows from the second source are hashed one at a time on the join column value. The hash bucket is probed for the matching column value. If there is a match, the row is returned. The N-way hash join works in a similar manner. All but one of the row sources are hashed into hash buckets on the join keys. The last row source is used one row at a time to probe the hash buckets for a match. If there are matches in all the hash buckets the row is returned. If matching fails for any of the buckets, the row is rejected. The explain plan shows a new operation: `N-WAY HASH JOIN`.

A band join uses a new algorithm for a class of non-equijoins using a range of values, such as the BETWEEN condition. Currently this class of non-equijoins cannot use a hash join. This new algorithm breaks the table into bands (rows partitioned by a range of values) by internally adding equality conditions. These bands, being much smaller than the full table, can then be joined using either hash or merge join based on the cost. By reducing the number of qualified rows, performance is improved by reducing I/O and the number of throwaway rows. The explain plan shows a new operation: `HASH JOIN.BAND` or `MERGE JOIN BAND`.

Continuous Adaptive Query Plans

- Adapt for every execution of the same cursor instead of only once
- Adapt more effectively for a wider range of query types
  - Recursive queries in which the underlying data does not change, but the data input to the query changes
  - Queries that use the GROUP BY placement transformation
  - Parallel queries with skewed joins
- Adapt a plan based on statistics obtained during execution

```
OPTIMIZER_FEATURES_ENABLE >= 12.1.0.1
```

```
OPTIMIZER_ADAPTIVE_REPORTING_ONLY = FALSE
```

In Oracle Database 12*c* Release 1 (12.1.0.2), the optimizer resolves the plan and it is resolved once during the lifetime of the cursor. The optimizer considers plan decisions as final and stores them within the cursor. On a subsequent execution, the database honors the earlier decisions and executes the plan. Freezing the plan decision is not ideal when part of the plan is iterative (as in recursive WITH queries), the underlying data changes frequently, or the bind values change.

Starting in this release, the optimizer can pick the best-performing plan during any execution of the statement, not just the first execution. If the underlying data changes, or if queries re-execute with different input data, then the optimizer can adapt its plan to match the statistics gathered in the current execution. The continuous adaptive query plan adapts for every execution of the same cursor instead of only once.

- Recursive queries in which the underlying data does not change, but the data input to the query changes
- Queries that use the `GROUP BY` placement transformation. If the optimizer makes a mistake in estimating the number of rows, then the placement of `GROUP BY` may not decrease the cost of the plan as expected. Therefore, making the GROUP BY placement adaptive at run time can disable the additional GROUP BY when the reduction in the number of rows is not same as the optimizer estimate.

Continuous Adaptive Query Plans

- Parallel queries with skewed joins. If the input values are popular and if the optimizer chooses hash distribution, then the PQ processes handling the popular values might do disproportionately more work. In this case, the optimizer may decide that a more optimal distribution method is to hash the popular values but broadcast the nonpopular values.

Summary

In this lesson, you should have learned how to:

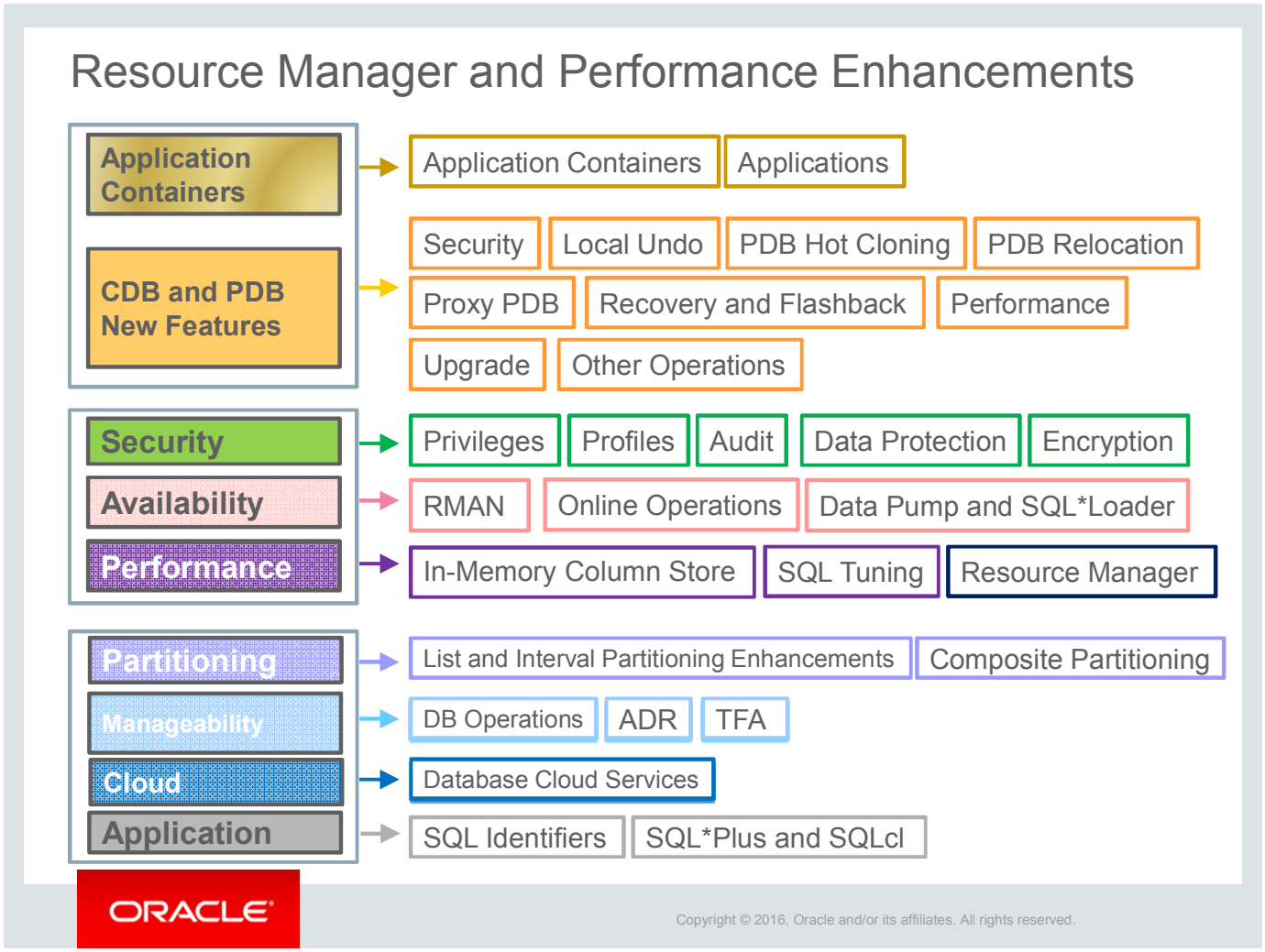
- Describe SQL Plan Management (SPM) enhancements
- Minimize DBA intervention to get the right statistics
- Configure and use the Optimizer Statistics Advisor
- Explain SQL Performance Analyzer (SPA) enhancements
- Describe Database Replay enhancements
- Use new SQL JOIN processing
- Explain Continuous Adaptive Query Plans

Practice 17: Overview

- 17-1: Configuring SPM to capture specific statement plans
- 17-2: Using the Optimizer Statistics Advisor to improve the quality of statistics gathering
- 17-3: Disabling database triggers during SPA task execution

18

Resource Manager and Other Performance Enhancements



This lesson describes how to manage the PGA limit with Resource Manager and also other performance enhancements.

Objectives

After completing this lesson, you should be able to:

- Manage session PGA limit
- Reduce cursor invalidations for DDLs
- Describe the Advanced Index Compression HIGH level
- Explain ADO enhancements
- Manage Real-Time materialized views with On Query Computation
- Use On-Statement refresh materialized views

For a complete understanding of Resource Manager and Materialized Views usage, refer to the following guides in the Oracle documentation:

- *Oracle Database Administrator's Guide 12c Release 2 (12.2)*
- *Oracle Database VLDB and Partitioning Guide 12c Release 2 (12.2)*
- *Oracle Data Warehousing Guide 12c Release 2 (12.2)*

Session PGA Limit

For security purposes:

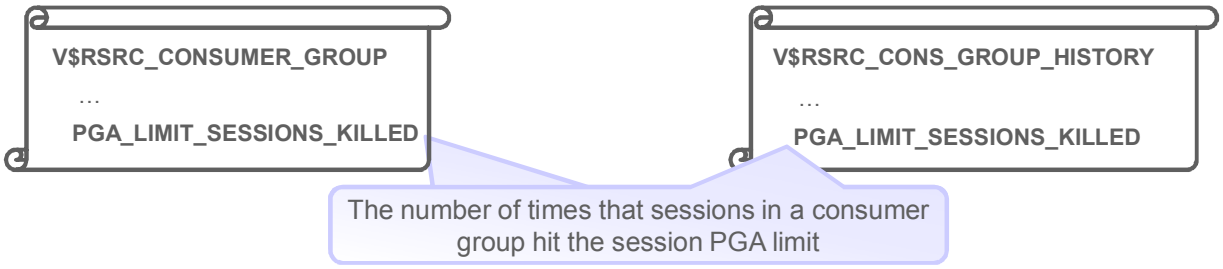
- Avoid excessive usage of PGA memory
- Set the PGA limit that a session can use before it hits an error

DAY plan

Consumer Group	Session_PGA_limit
OLTP	350
REPORTS	100
OTHER	50

NIGHT plan

Consumer Group	Session_PGA_limit
OLTP	50
REPORTS	400
OTHER	50



This functionality is important from a security perspective. There are numerous ways to write PL/SQL to use large amounts of PGA memory:

- A user can create a temporary LOB and append to it.
- A user can create a large PL/SQL varray.
- A user can write a PL/SQL statement that does a large recursion using memory at each level.

This is of particular concern in consolidated environments like DB Cloud.

In Oracle Database 12.1, the only way to get a PGA limit on a session level is to enable the 10261 event.

Oracle Database 12.2 allows the DBA to define a per-session PGA limit at the consumer-group level by using the new `SESSION_PGA_LIMIT` argument of the `DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE` procedure. Set the value to the number of Mb of PGA that a session can use before it hits the PGA limit and gets an error. If a session exceeds this limit, it receives an error that aborts the current call. If aborting the current call does not free up enough PGA, the system may kill the session. Resource Manager tracks how many times a consumer group hits the PGA limit and presents this information in `V$RSRC_CONSUMER_GROUP` and `V$RSRC_CONS_GROUP_HISTORY` views.

Cursor Invalidations for DDLs

12.1

Any DDL on an object:

- Invalidates all cursors that depend on the object
- Does not truly require all existing cursors to be recompiled
- Makes an application unavailable until all cursors are recompiled
- Degrades application performance

12.2

Reducing cursor invalidations for DDLs:

- Revalidates cursors over time
- Improves application availability

Oracle Database 12.2 reduces the number of cursor hard parses caused by DDLs.

- This capability improves application performance when DDLs are mixed with the query/DML workload by reducing the number of hard parses and spreading the hard parse workload over time so that users do not see a spike in hard parse activity.
- When a large number of hard parses happen immediately after a DDL, an application is effectively unavailable until the hard parses complete. This capability improves availability by reducing the number of hard parses and spreading the hard parse workload over time.

Deferred Invalidation

- Declared at the DDL statement:

```
SQL> ALTER TABLE … IMMEDIATE INVALIDATION;
```

12.1 behavior

```
SQL> CREATE INDEX … DEFERRED INVALIDATION;
```

12.2 behavior

  - CREATE … INDEX …
  - ALTER … INDEX … REBUILD|UNUSABLE …
  - ALTER TABLE … MODIFY PARTITION/SUBPARTITION …
  - ALTER TABLE … *partitioning* …
  - TRUNCATE TABLE …

- Declared at the session or system level:

```
SQL> ALTER SYSTEM SET CURSOR_INVALIDATION = DEFERRED SCOPE=BOTH;
```

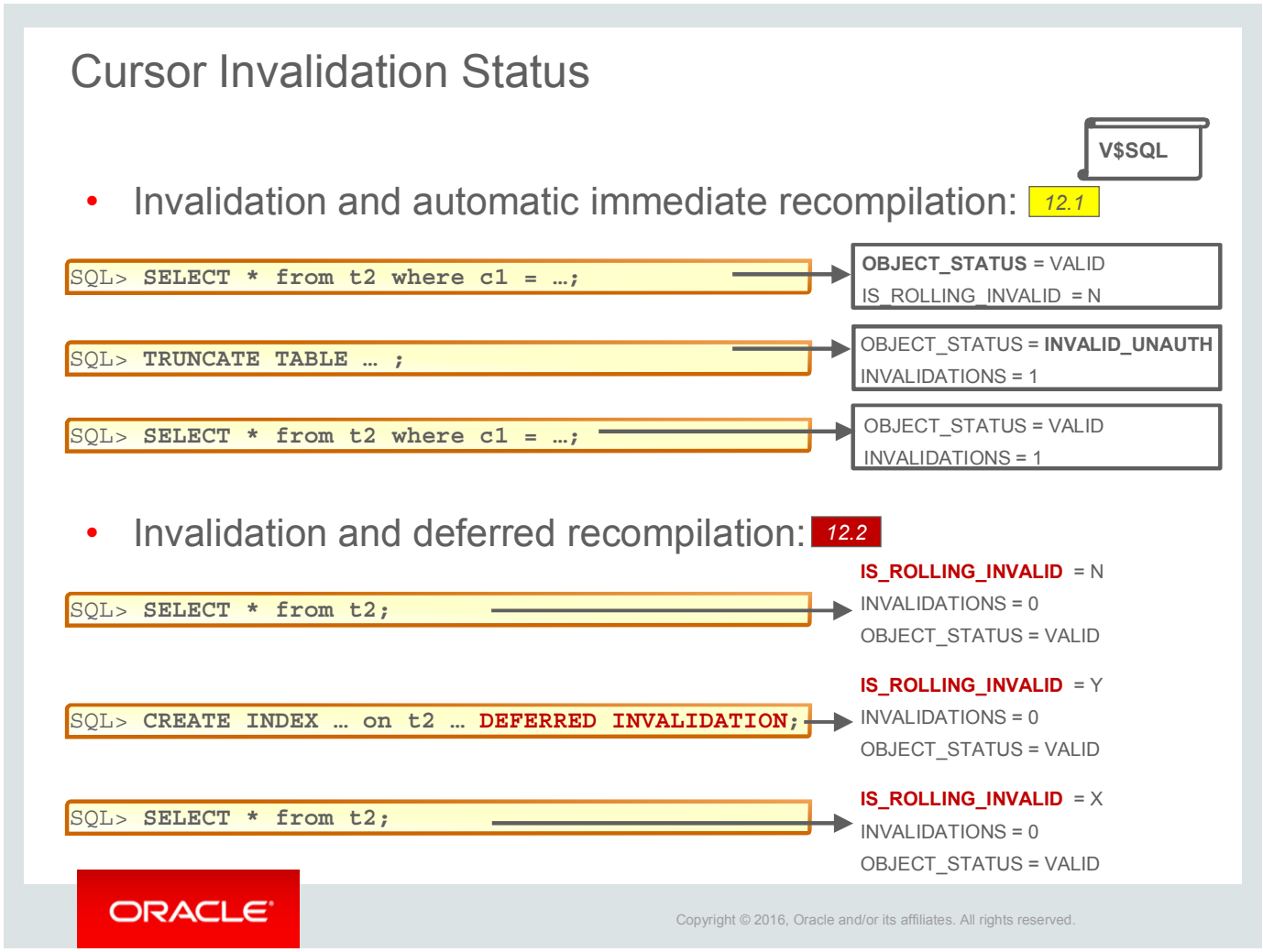
A new clause in DDLs defines the behavior of dependent cursors invalidation.

- `DEFERRED INVALIDATION`: If the clause is specified in a DDL, Oracle avoids or defers invalidating dependent cursors when possible.
- `IMMEDIATE INVALIDATION`: If the clause is specified in a DDL, Oracle immediately invalidates dependent cursors, as it did in Oracle 12.1 and prior releases.

If neither option is specified, the `CURSOR_INVALIDATION` parameter controls the default behavior for your session or for the instance and is set to `IMMEDIATE` by default.

Restrictions:

- Deferred invalidation does not happen during the `CREATE INDEX REBUILD` statement if creating an index on an IOT.
- `TRUNCATE TABLE DEFERRED INVALIDATION` only happens if the table is partitioned.

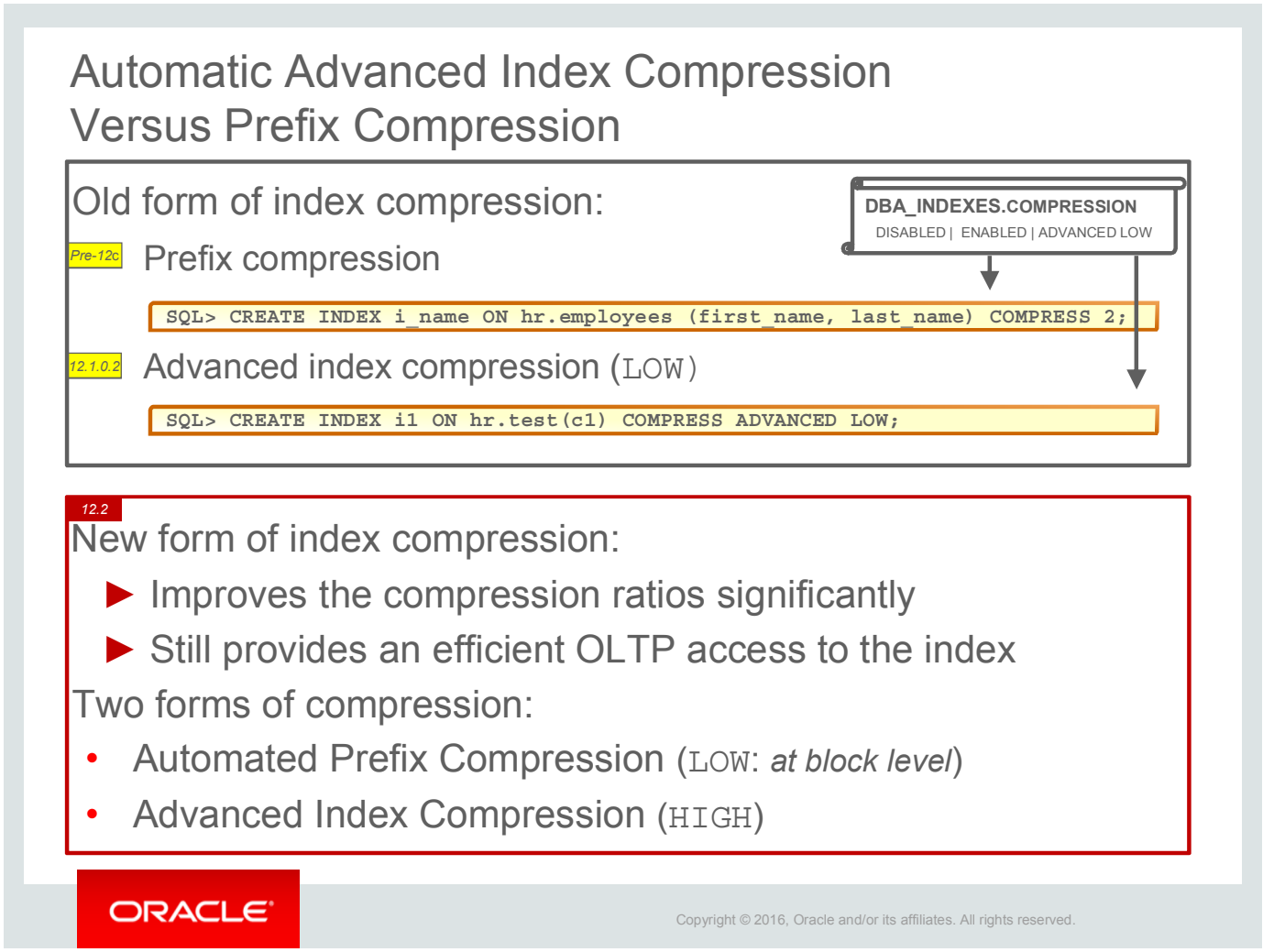


There are four actions that can be taken for cursors in the fine-grained approach. The state of the cursor after a DDL using deferred invalidation is visible in the `V$SQL` view in three new columns.

- `DDL_NO_INVALIDATE`: Indicates if a DDL updated a dependent object and did not invalidate a cursor. The cursor is valid and still has an optimal plan. The values are:
  - `N`: There has not been a DDL that updated a dependent object without invalidating this cursor.
  - `Y`: A DDL updated a dependent object and did not invalidate this cursor, but the cursor has not executed since this happened.
  - `X`: A DDL updated a dependent object and did not invalidate this cursor, and the cursor has executed since this happened.
- `IS_ROLLING_INVALID`: Indicates if a cursor is rolling invalidated. The cursor is still valid, but may have a suboptimal plan. The recompilation can be deferred.
  - `N`: This cursor is not rolling invalidated.
  - `Y`: This cursor is rolling invalidated, but the cursor has not executed in this state.
  - `X`: This cursor is rolling invalidated, and the cursor has executed in this state.

Cursor Invalidation Status

- `IS_ROLLING_REFRESH_INVALID`: Indicates if a cursor is rolling invalidated and requires execution-time refresh. Some metadata is stale, so the metadata is refreshed at cursor start time, and the cursor can continue to be used with possibly a suboptimal plan. The recompilation is deferred.
- There are some cursors that depend on the object and the DDL forces the cursor to be recompiled immediately.

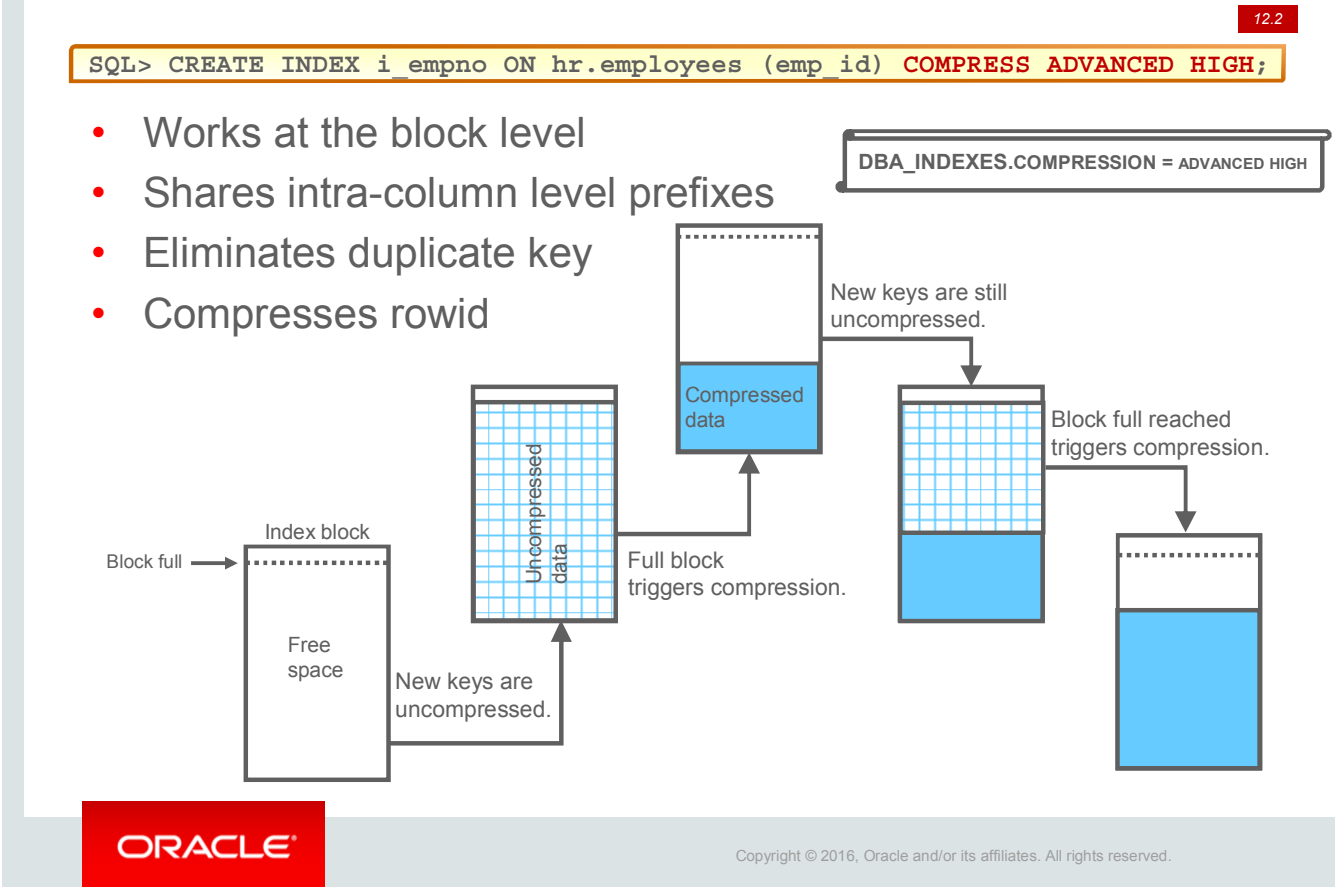


Indexes are extensively used inside OLTP databases since they are capable of efficiently supporting a wide variety of access paths to the data stored in relational tables, either via index key lookups or via range predicates implemented as range scans. Therefore, a large number of indexes are created on a single table to support the multitude of access paths for OLTP applications. Indexes contribute to a greater share of the overall storage of the relational database when compared to the size of the base tables alone.

Advanced Index Compression applies a similar set of compression algorithms to indexes that Advanced Row Compression applies to tables. Oracle has had a prefix index compression prior to Oracle Database 12.1.0.2, but it required the DBA to know the data and select the number of columns to be compressed. It was consistently applied to all the blocks in an index. Oracle Database 12.1.0.2 introduced the Advanced Index Compression with which you do not have to specify the prefix keys since the decision is made on a block basis. Oracle Database 12.2 provides a new option which significantly increases the compression ratio of indexes.

- The Advanced Index Compression uses a high compression ratio.
- The Advanced Index Compression in Oracle Database 12.1.0.2 uses a low compression ratio. In Oracle Database 12.2, the low index compression is called the Automated Prefix Compression.

Advanced Index Compression



Advanced Index Compression works at the block level. When a block becomes full and is closed to being split, an attempt to compress the block is performed and avoids the split if the space savings generated are sufficient to insert the incoming key.

You can set the compression attribute at the creation of the index (by default, index compression is `DISABLED` and by default, enabled index compression is `HIGH`) or after the index is created.

```
SQL> ALTER INDEX i_empno REBUILD COMPRESS ADVANCED HIGH;
```

You can also disable the compression attribute.

```
SQL> ALTER INDEX i_empno REBUILD NOCOMPRESS;
```

Dynamically enabling or disabling advanced compression on indexes without rebuild can only be supported if the index was created under Oracle Database 12*c*.

The compression attribute for an index partition is inherited from the base index. However, the DBA is free to over-ride it at the partition level.

Restrictions

IOTs (Index Organized Tables) and bitmap indexes do not support index compression.

Using the Compression Advisor for Indexes

- The compression advisor helps to determine optimal index compression ratios.

```
BEGIN
 DBMS_COMPRESSION.GET_COMPRESSION_RATIO ('USERS', 'HR', 'I_COL1', NULL,
      DBMS_COMPRESSION.COMP_INDEX_ADVANCED_HIGH,
      blkcnt_cmp, blkcnt_uncmp, row_cmp, row_uncmp, cmp_ratio,
      comptype_str, dbms_compression.COMP_RATIO_MINROWS,
      dbms_compression.OBJTYPE_INDEX);
 DBMS_OUTPUT.PUT_LINE('Block count compressed = ' || blkcnt_cmp);
 DBMS_OUTPUT.PUT_LINE('Block count uncompressed = ' || blkcnt_uncmp);
 DBMS_OUTPUT.PUT_LINE('Row count per block compressed = ' || row_cmp);
 DBMS_OUTPUT.PUT_LINE('Row count per block uncompressed = ' || row_uncmp);
 DBMS_OUTPUT.PUT_LINE('Compression type = ' || comptype_str);
 DBMS_OUTPUT.PUT_LINE('Compression ratio = '||cmp_ratio);
END;
```

- Two advanced index compression levels:
  - DBMS_COMPRESSION.COMP_INDEX_ADVANCED_**HIGH**
  - DBMS_COMPRESSION.COMP_INDEX_ADVANCED_LOW

The compression advisor helps in identifying indexes which are suitable or unsuitable candidates for advanced compression. With Oracle Database 12.2, the advisor is enhanced to use the new advanced index compression level. In the example in the slide, the new constant used to compute the ratio for compressing an index with the Advanced Index Compression is DBMS_COMPRESSION.COMP_INDEX_ADVANCED_HIGH.

```
Block count compressed = 133
Block count uncompressed = 1044
Row count per block compressed = 4004
Row count per block uncompressed = 510
Compression type = "Compress Advanced High"
Compression ratio = 8
```

This procedure analyzes the compression ratio of the index and gives information about compressibility of the index with the Advanced Index Compression.

ADO Enhancements

- Automatic Data Optimization (ADO) supports In-Memory Column Store.
- `DBMS_ILM.STOP_ILM` stops ADO jobs.

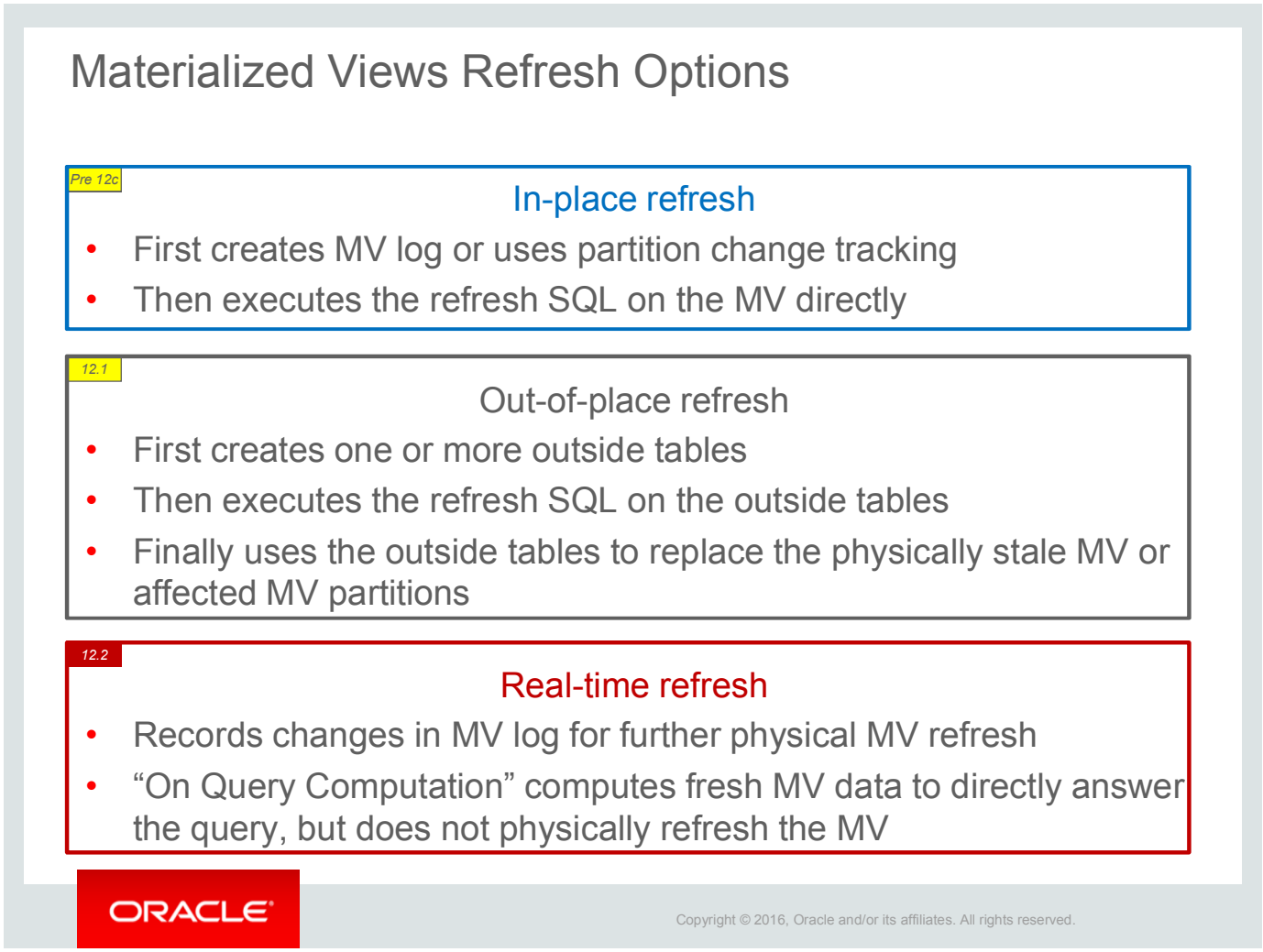
ADO manages the In-Memory Column Store by automatically moving objects (tables, partitions, or subpartitions) out of the memory based on Heat Map statistics. Refer to the previous lesson titled, "*In-Memory Column Store.*"

Automatic Data Optimization (ADO) supports Hybrid Columnar Compression (HCC) for row-level policies. Rows from cold blocks can be HCC compressed even if there is DML activity on other parts of the table or partition. ADO policies for HCC are now effective even if changes are constantly being made to a small subset of the table or partition. This means that HCC space saving and performance benefits can be transparently extended to more data.

```
SQL> ALTER TABLE employees ILM ADD POLICY
     COLUMN STORE COMPRESS FOR QUERY ROW AFTER 30 DAYS OF NO MODIFICATION;
```

An HCC row-level policy can be defined on any table regardless of the compression type of the table. Rows that are HCC compressed by an ILM row-level policy are moved into a new HCC compression and will have a new ROWID.

The `STOP_ILM` procedure enables termination of ILM jobs for a particular execution task ID. Additionally, this procedure enables termination of a particular ILM job by job name. You can optionally choose to stop running jobs.



Materialized views (MVs) are used for two purposes:

- To store the summary data for direct access
- In query rewrite to replace the joins and/or aggregates. In Oracle Database 12.1, physically stale MVs cannot be used for query rewrite.

For both cases, users may need up-to-date data from MVs.

Oracle Database 12.1 uses regular MVs which can be physically stale if data is not in sync with the base tables.

Oracle Database 12.2 can use real-time MVs. An "On Query Computation" (OQC) MV can always provide up-to-date results to the user, even if its current content is not in sync with the base tables.

When a query comes and the MV that is needed either by query rewrite or by direct access is physically stale, OQC is executed to compute the fresh MV data to answer the query. Unlike the out-of-place log-based refresh option, which inserts the computed fresh MV data into an outside table and uses the outside table to replace the stale MV, OQC does not store the computed data in any table. Instead, it directly feeds the data to the query execution pipeline to answer the query. The query does not refresh the MV physically.

OQC MVs must be refreshable using out-of-place log-based refresh and do not rely on refresh on commit.

Real-Time Materialized Views

Enable OQC:

DBA_MVIEWS.ON_QUERY_COMPUTATION = Y | N

```
SQL> CREATE MATERIALIZED VIEW mv_employees
          REFRESH FAST ON DEMAND
          ENABLE ON QUERY COMPUTATION
       AS SELECT * FROM employees;
```

- For direct MV access, use the FRESH_MV hint:

```
SQL> select /*+ FRESH_MV */ from mv_employees;
```

  - Refresh base tables, MV log, and MV frequently.
- For query rewrite:
  - Set the QUERY_REWRITE_INTEGRITY parameter to ENFORCED or TRUSTED
  - Collect statistics on base tables, MV, and MV log

In Oracle Database 12.1, an MV can be refreshed on demand, on commit, or as scheduled.

In Oracle Database 12.2, a real-time MV can still be refreshed with its regular paces.

- However, when a user executes a query that directly accesses the MV, if the MV is physically stale, the OQC is executed to provide the fresh MV data.
- If a query does not directly reference a real-time MV but can be rewritten using this MV, the MV is considered as eligible for rewrite:
  - Even if it is physically stale
  - And if the SQL session does not operate in the stale-tolerance mode (QUERY_REWRITE_INTEGRITY = ENFORCED or TRUSTED)

  If the MV ends up being selected to rewrite the query and is physically stale, the OQC is executed to compute the fresh MV data. The query does not refresh the MV.

Recommendations

Because the query used in the OQC to compute the fresh MV data combines the MV and the logged base table changes, the query performance can be assured only when the delta is small. Therefore, the real-time MV, once created, should be refreshed frequently.

For query rewrite, because the cost-based rewrite mechanism uses the optimizer cost to decide whether to use the rewritten query and the cost relies on statistics, statistics on base tables, real-time MV, and MV logs should be collected to enable a more accurate optimizer cost.

On Statement Refresh

Depending on the method specified, MVs can be refreshed at different times:

Pre 12c

ON DEMAND refresh

- When a user manually executes one of the procedures of the DBMS_MVIEW package (REFRESH, REFRESH_ALL_MVIEWS, REFRESH_DEPENDENT)
- Scheduled to be refreshed with the same procedures

ON COMMIT refresh

- Every time any change commits to a base table

12.2

ON STATEMENT refresh

- Every time a DML operation takes place on a base table
- No need to commit the transaction

Note: For materialized join views (MJV) only

DBA_MVIEWS.REFRESH_MODE = STATEMENT

Depending on the refresh method, an MV can be refreshed at different times. It can be defined to be refreshed ON COMMIT or ON DEMAND.

- ON COMMIT MVs are refreshed every time any change commits to a base table. This method may not be suitable if many users are concurrently changing the tables upon which the MV is based. The only disadvantage is the time required to complete the commit, slightly longer because of the extra processing involved.
- ON DEMAND MVs are manually refreshed by using the REFRESH or other procedures of the DBMS_MVIEW package. The ON DEMAND refresh execution can be scheduled.

Despite the timing differences, the refresh methods require each base table to have a materialized view log (MV log) if the MV needs to do fast refresh. This requirement can cause overhead in terms of refresh performance, as during each refresh, validation, setup, cleanup need to be done related to MV log.

Oracle Database 12.2 introduces the new ON STATEMENT refresh capability which automatically refreshes at DML time. Therefore, whenever a DML happens on any table on which an MV is defined, the change is automatically reflected in the MV. The advantage of using this approach is that the user no long needs to create an MV log on each of the base tables in order to do fast refresh. The refresh can then avoid the overhead introduced by MV logging, but still keep the MV refreshed at all times.

On Statement Refresh

Restrictions:

- For materialized join views (MJVs) only
- Base tables referenced in the MV defining query must be connected in a join graph of star/snowflake shape. It contains exactly one centralized fact table and multiple dimension tables which are further normalized into subdimension tables. All pairs of joined tables are related with PK/FK referential constraints.
- Cannot create an on-statement MV under SYS
- Need to be FAST refreshable
- The defining query needs to include the ROWID column of the fact table in the SELECT list.
- UPDATE operation on any dimension table is not supported. It makes on-statement MV unusable.
- PMOP and `TRUNCATE` base table are not supported. They makes on-statement MV unusable.
- The defining query should NOT include:
  - Invisible column
  - Ansi join syntax
  - Complex defining query
  - (Inline) view as base table
  - Composite primary key
  - LONG/LOB column

Summary

In this lesson, you should have learned how to:

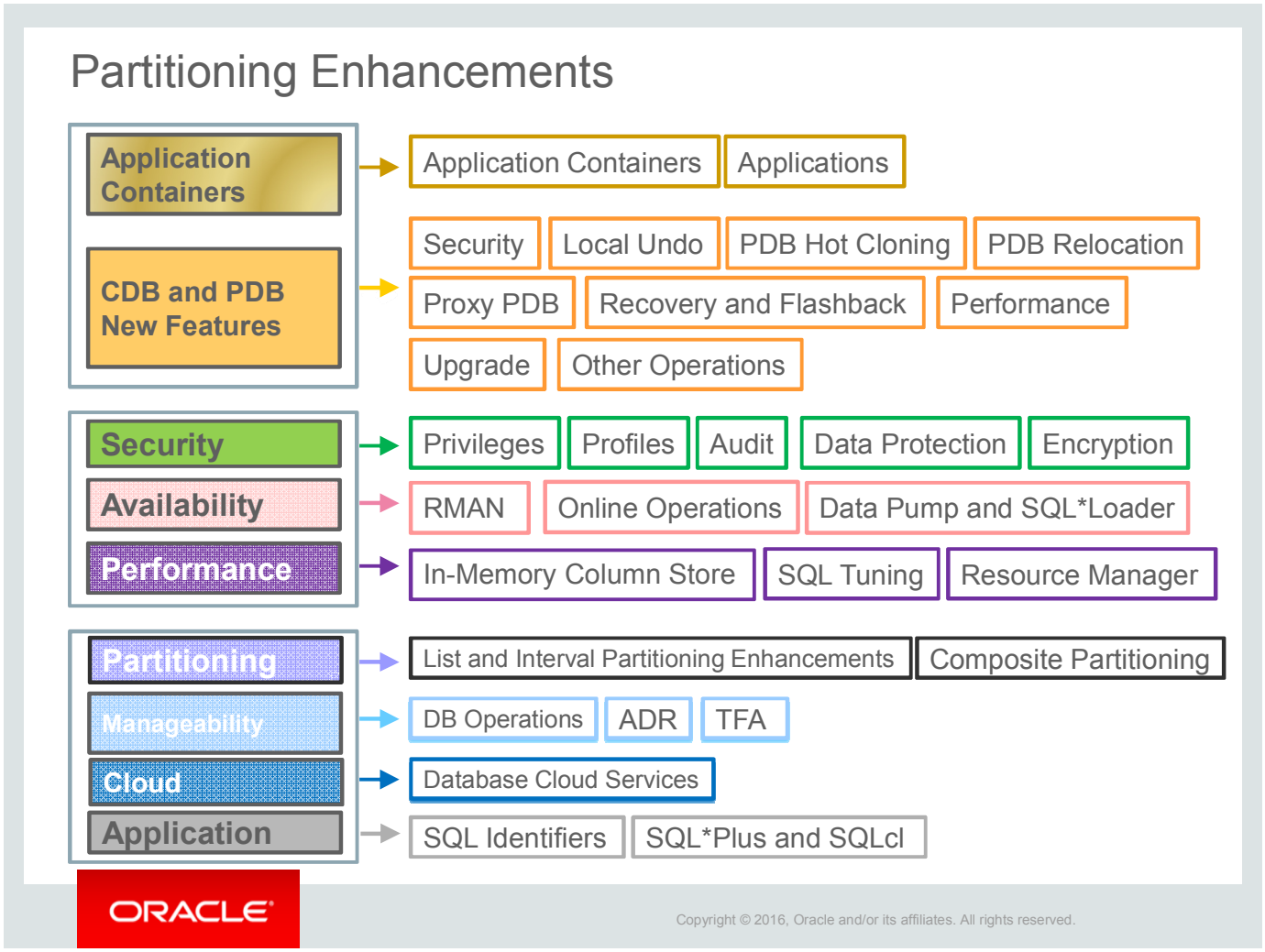
- Manage session PGA limit
- Reduce cursor invalidations for DDLs
- Describe the Advanced Index Compression HIGH level
- Explain ADO enhancements
- Manage Real-Time materialized views with On Query Computation
- Use On-Statement refresh materialized views

Practice 18: Overview

- 18-1: Avoiding excessive usage of PGA memory
- 18-2: Using cursor invalidations with DDL statements
- 18-3: Using Advanced Index Compression
- 18-4: Creating real-time materialized views with on query computation for direct MV access and query rewrite
- 18-5: Creating on-statement materialized views on MJVs (*optional*)

19

Partitioning Enhancements



There are several Partitioning enhancements like interval subpartitioning, auto-list partitioning and subpartitioning, and multicolumn list partitioning.

Objectives

After completing this lesson, you should be able to:

- Explain auto-list partitioning
- Describe composite partitioning
- Explain multicolumn list partitioning
- Disallow DML operations on read-only partitions
- Enable deferred segment creation for subpartitions of composite interval and auto-list partitioned tables
- Filter data during partition maintenance operations

For a complete understanding of partitioning enhancements and usage, refer to the following guides in the Oracle documentation:

- *Oracle Database VLDB and Partitioning Guide 12c Release 2 (12.2)*
- *Oracle Database Data Warehousing Guide 12c Release 2 (12.2)*

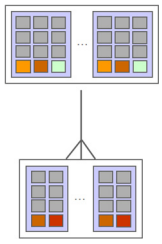
Partitioning Methods

Pre-12c

- Interval partitioning
- System partitioning
- Virtual column–based partitioning
- Reference partitioning
- Composite partitioning enhancements

12.1.0.2

- Interval reference partitioning



12.2

- Auto-list partitioning
- Composite partitioning enhancements
- Multicolumn list partitioning

ORACLE

Oracle Database 11*g* introduced a useful assortment of partitioning enhancements. These enhancements included:

- Addition of interval partitioning
- Addition of system partitioning
- Composite partitioning enhancements
- Addition of virtual column-based partitioning
- Addition of reference partitioning

Oracle Database 12.1 introduced a new partitioning method, the interval reference partitioning, where a reference-partitioned table leverages interval partitioning as the top partitioning strategy.

Oracle Database 12.2 introduces a new partitioning method, the auto-list partitioning.

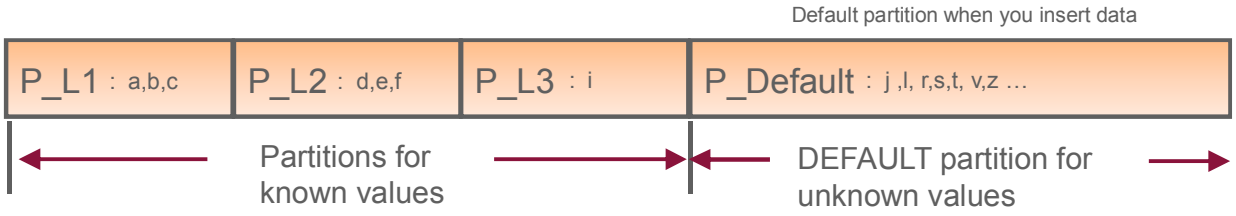
Oracle Database 12.2 also introduces the multicolumn list partitioning for both internal partitioned tables and external partitioned tables.

Auto-List Partitioning

DBA_PART_TABLES. AUTOLIST = YES | NO

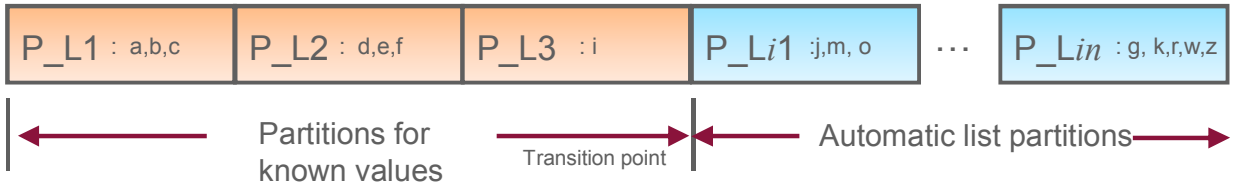
Pre-12c

- Create in advance all partitions for the known values.
- Plan a `DEFAULT` partition for all unknown values.



12.2

- Create partitions for known values only.
- All others are created automatically *(or manually).*



Using list partitioning in Oracle Database 12.1 requires that the DBA explicitly specifies the list of values which define each partition during table/partition creation. The `DEFAULT` keyword enables you to avoid specifying all possible values for a list-partitioned table by using a default partition, so that all rows that do not map to any other partition do not generate an error. The DBA either has to know all list values in advance or has to plan for unknown values with the `DEFAULT` on-catches-all partition.

Oracle Database 12.2 allows list partition creation on demand for every new distinct list value for the table. Partition creation on demand for auto-list partitioning is similar to what interval partitioning provides for range partitioning. It automatically creates a new partition for every distinct partition key value.

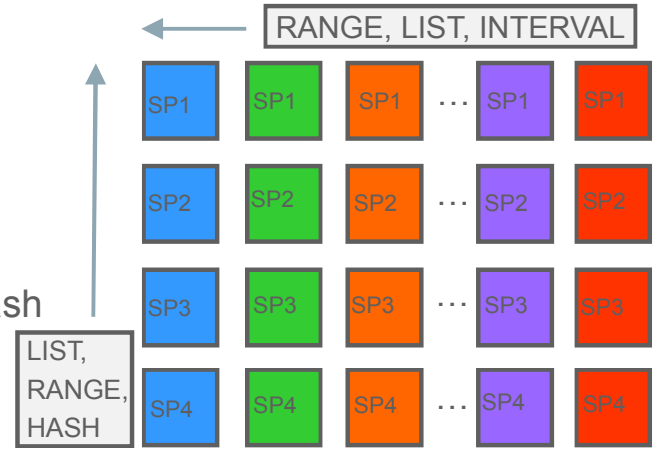
Creating an auto-list partitioned table requires that the partitions are defined for the known partitioning key values only. As data is loaded into the table and the database creates a new partition if the loaded partitioning key value does not correspond to any of the existing partitions.

Restrictions: Auto-list partitioned IOTs and domain indexes are not supported on auto-list partitioned tables.

Composite Partitioning

Pre-12c

- Range top level
  - Range – Range
- List top level
  - List – Range/List/Hash
- Interval top level
  - Interval – Range/List/Hash



12.2

- Partitioning: Range/List/Hash/Interval/Auto-list
  - Subpartitioning: Range/List /Hash

Oracle Database 12.1 provides composite partitioning methods for composite partitioned tables like list-list, list-hash, list-range, range-range, interval-range, interval-list, and interval-hash.

- Range-Range Partitioning: Enables logical range partitioning along two dimensions (for example, range partitioning by `order_date` and range subpartitioning by `shipping_date`)
- List-Range Partitioning: Enables logical range subpartitioning within a given list partitioning strategy (for example, list partitioning by `country_id` and range subpartitioning by `order_date`)
- List-Hash Partitioning: Enables hash subpartitioning of a list-partitioned object (for example, to enable partitionwise joins)
- List-List Partitioning: Enables logical list partitioning along two dimensions (for example, list partitioning by `country_id` and list subpartitioning by `sales_channel`)

Oracle Database 12.2 provides new composite partitioning methods for composite partitioned tables:

- Auto-list partitioning as a new top-level partitioning method combined with all existing methods at the subpartition level

Multicolumn List Partitioning

New partitioning method: Multicolumn List

	Partitioning	Single or multicolumn partitioning
Pre-12c	Range/Hash	Both
	List	Single
12.2	List	Both

- Allows more than one column as the list partitioning key:
  - For internal partitioned tables
  - For external partitioned tables: Intuitive mapping of partitioned external tables for data stored externally
- Optimizes requests by using pruning and partitionwise joins
- Supports all partition maintenance operations

As of Oracle Database 12.2, the existing multicolumn partitioning strategies allowing the specification of more than one column as the partitioning key, each with one or more values, are the range and hash partitioning.

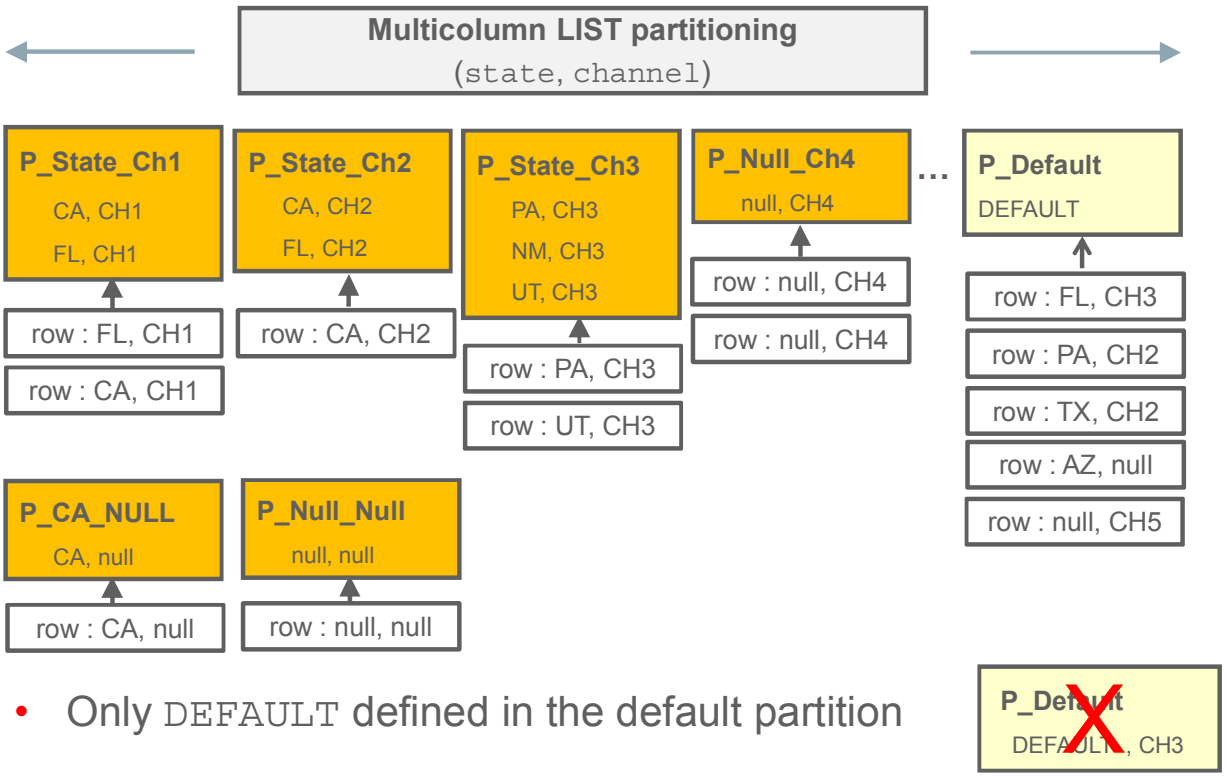
Oracle Database 12.2 introduces multicolumn list partitioning for both internal partitioned tables and external partitioned tables.

One of the main target data stores for partitioned external tables is data stored in Hive. Hive allows for the manipulation of data in HDFS (a distributed file system providing fault-tolerant storage) using a variant of SQL. Hive offers support for one partitioning strategy only, which is equivalent to a multi-column list partitioned table. Oracle Database 12.2 multicolumn list partitioning provides the foundation for the most intuitive mapping of partitioned external tables for data stored in Hive tables.

Multicolumn list partitioning supports:

- Multicolumn list for auto-list partitioning and subpartitioning
- All relevant partition maintenance operations for both single and multiple partitions in offline mode like ADD, DROP, SPLIT, MOVE, TRUNCATE, and EXCHANGE
- Existing online partition maintenance operations, namely MOVE

Multicolumn List Partitioning



- Only `DEFAULT` defined in the default partition

Individual values can be specified for multiple partitions as long as the full qualification of all partition key columns differ and allow a unique and deterministic row placement.

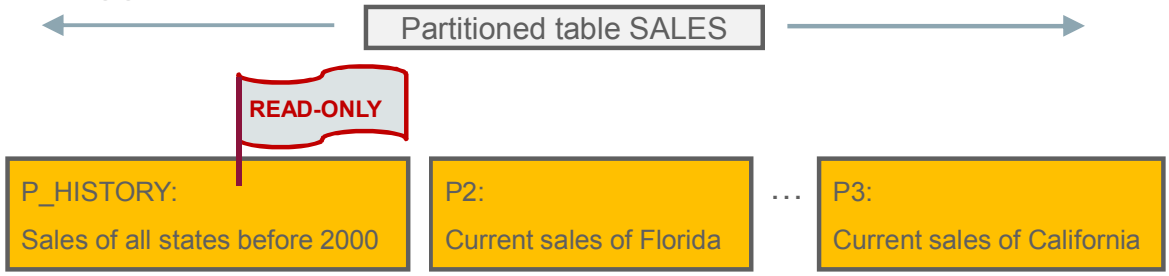
If a default partition is specified, no other value than `DEFAULT` can be specified.

Multicolumn list partitioning supports:

- Up to 16 columns as partition key columns
- One or multiple values as a list of discrete values for each partition key column
- NULL as value for a partition key column

Read-Only Partition

- Protect data in partitions from unintentional DML by any user or trigger.



- Insert, delete, or update results in an error.

```
SQL> insert into sales values (1,'A',1000, 'CA', '23-DEC-1999')
ORA-14466: Data in a read-only partition or subpartition cannot be modified
```

- Modify partition state to READ WRITE to enable DML.

```
SQL> ALTER TABLE sales
        MODIFY PARTITION p_history READ WRITE;
```

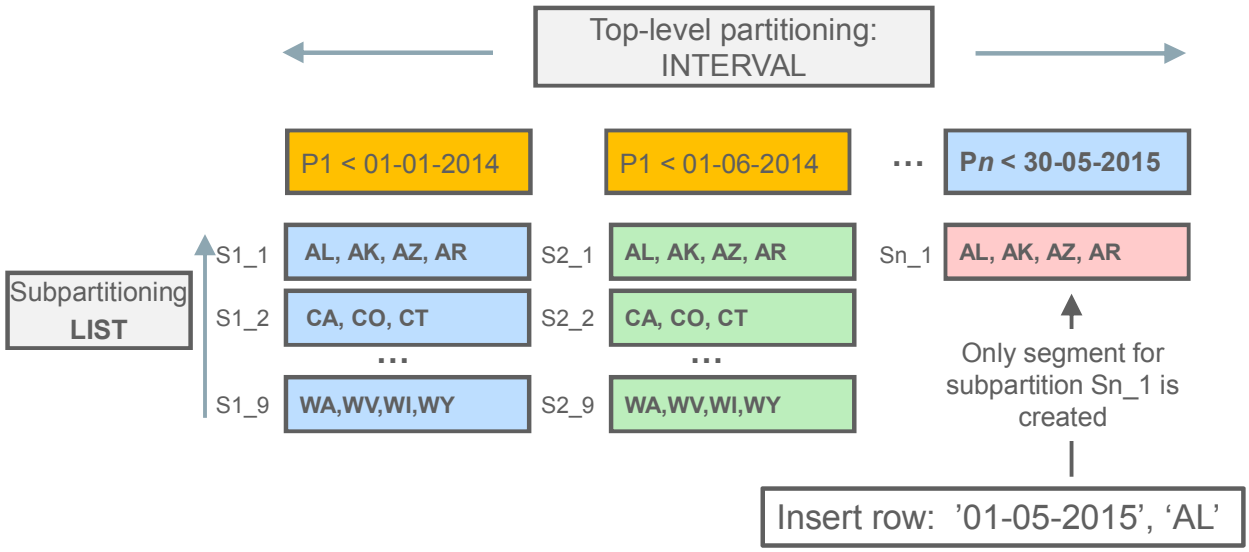
READ-WRITE

CREATE and ALTER TABLE SQL statements support a READ-ONLY clause for partitions and subpartitions. This feature protects data from unintentional DML by any user or trigger. Any attempt to insert, delete, or update the data from a table while it is in read-only status results in an error.

```
CREATE TABLE sales
 (deptno NUMBER, deptname VARCHAR2(20), quarterly_sales NUMBER(10,2),
  state VARCHAR2(2), channel VARCHAR2(1))
 PARTITION BY LIST (state, channel)
 (PARTITION q1_northwest_direct VALUES (('OR', 'D'), ('WA','D')) read only,
  PARTITION q1_northwest_indirect VALUES (('OR', 'I'), ('WA','I')),
  PARTITION rest VALUES (DEFAULT));
SQL> insert into sales_by_region_and_channel2 values (1,'A',1000, 'OR','D');
insert into sales_by_region_and_channel2 values (1,'A',1000, 'OR','D')
            *
ERROR at line 1:
ORA-14466: Data in a read-only partition or subpartition cannot be modified.
```

Deferred Segment Creation for Subpartitions

- Pre-12c Deferred segment creation exists for partitions.
- 12.2 Deferred segment creation for subpartitions of composite interval and auto-list partitioned tables



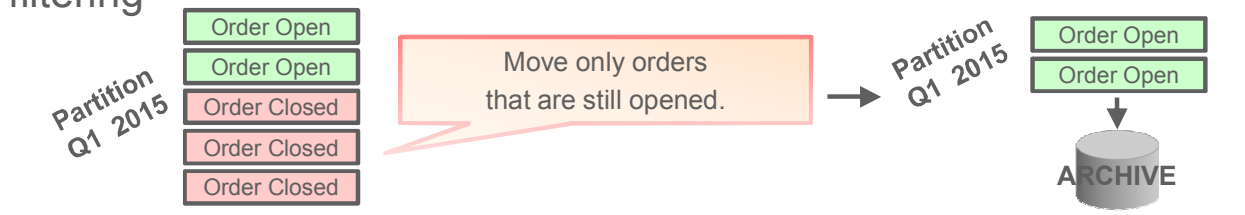
In Oracle Database 12.1, creating a partitioned table defers the creation of the segments for each partition until the first row is inserted into a partition. The behavior exists since the `DEFERRED_SEGMENT_CREATION` instance parameter was introduced with Oracle Database 11.2. This saves disk space and minimizes install time.

Inserting a new row in an interval composite partitioned table with a hash, list, or range subpartitioning strategy creates a new partition if there exists no matching partition and materializes all its subpartition segments. It is potentially an efficient use of resources to allocate space for all the subpartitions when all that is required is to allocate space only for the subpartition linked to the row being inserted.

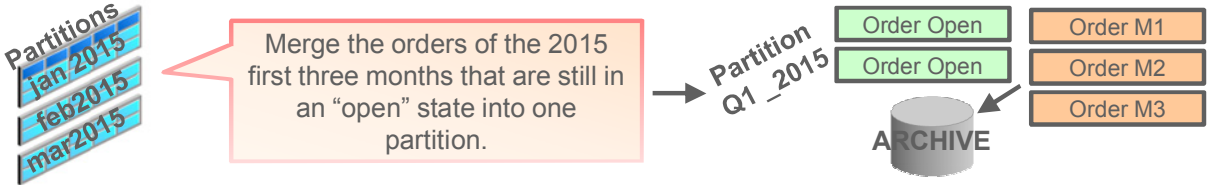
Oracle Database 12.2 allows deferred subpartition segment creation. Therefore, a subpartition segment is materialized only when the first matching row gets inserted. The same behavior can be observed for auto-list composite partitioned tables with a hash, list, or range subpartitioning strategy as well.

Filtered Partition Maintenance Operations

MOVE, MERGE, and SPLIT partition operations: Include data filtering




```
SQL> ALTER TABLE sales MOVE PARTITION Q1_2015 TABLESPACE archive COMPRESS
          INCLUDING ROWS WHERE order_state != 'CLOSED'
          ONLINE UPDATE INDEXES;
```




```
SQL> ALTER TABLE orders MERGE PARTITIONS jan2015, feb2015, mar2015
     INTO PARTITION q1_2015 COMPRESS FOR QUERY HIGH TABLESPACE archive
     INCLUDING ROWS WHERE order_state = 'OPEN';
```


Very often `MOVE`, `SPLIT`, and `MERGE` data maintenance operations are not 100% applicable to the data contained in the partitions but have to deal with some outlier records. The three clauses are enhanced to take a filter predicate as additional input.

- `ALTER TABLE ... MOVE PARTITION` statement: The first example in the slide shows how to move a partition to another tablespace keeping only orders that are opened.
- `ALTER TABLE ... MERGE PARTITIONS` statement: The second example in the slide show how to merge the orders for the 2015 first three months keeping only orders where the status is still open.
- `ALTER TABLE ... SPLIT PARTITION` statement: For example, if you want to split the data of the last month into four separate partitions for each week and only preserve active transactions starting with the first calendar week, you could issue the following statement:

```
SQL> ALTER TABLE orders SPLIT PARTITION q1_2013 INTO
    PARTITION w1_2013 VALUES LESS THAN to_date('09-JAN-2013','dd-mon-yyyy'),
    PARTITION w2_2013 VALUES LESS THAN to_date('01-JAN-2013','dd-mon-yyyy'),
    PARTITION w3_2013 VALUES LESS THAN to_date('01-JAN-2013','dd-mon-yyyy'),
    PARTITION w4_2013 TABLESPACE hot)
    INCLUDING ROWS WHERE order_date < '02-JAN-2013' AND status = 'active'
    UPDATE INDEXES;
```

Summary

In this lesson, you should have learned how to:

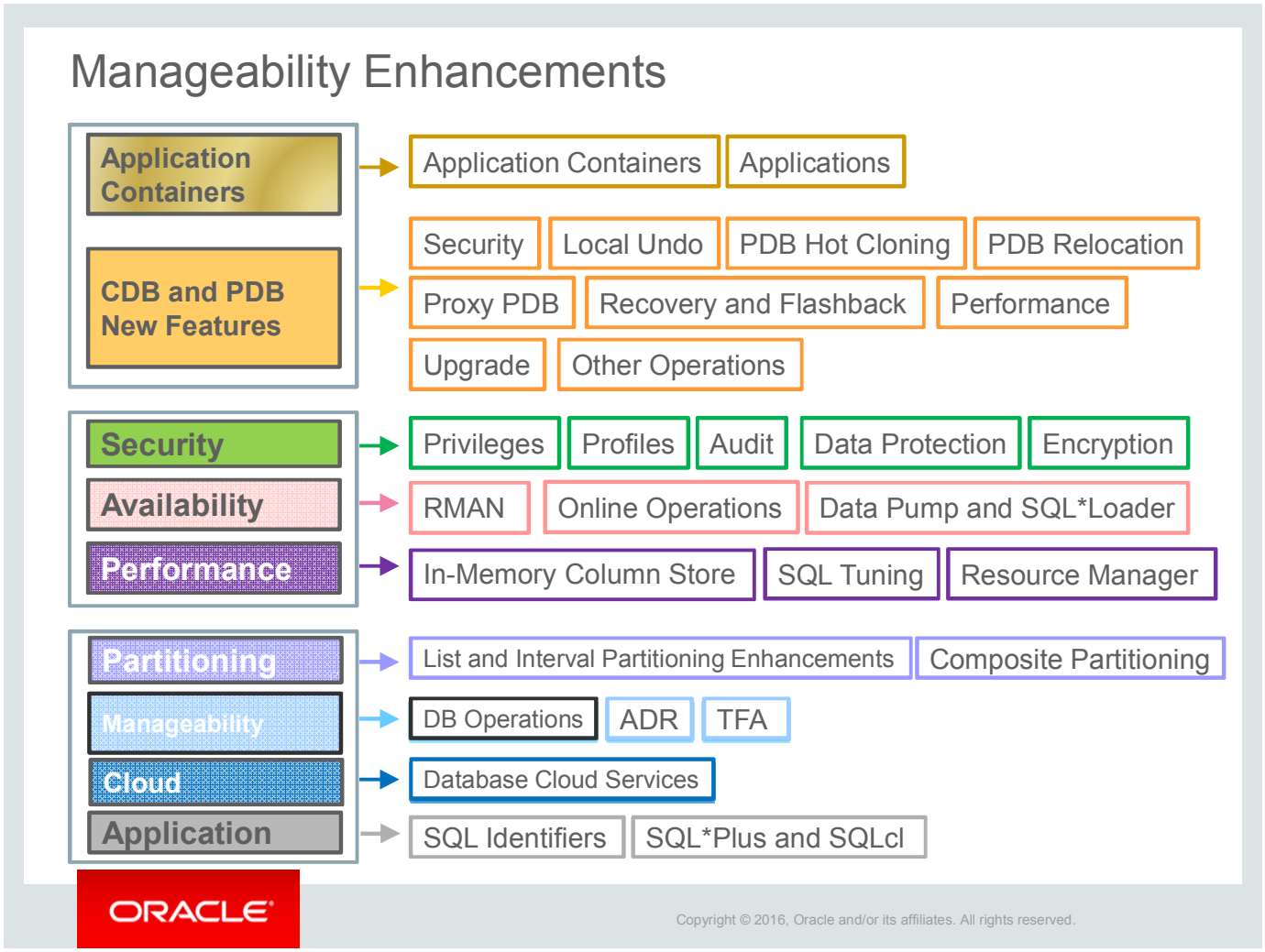
- Explain auto-list partitioning
- Describe composite partitioning
- Explain multicolumn list partitioning
- Disallow DML operations on read-only partitions
- Enable deferred segment creation for subpartitions of composite interval and auto-list partitioned tables
- Filter data during partition maintenance operations

Practice 19: Overview

- 19-1: Creating auto-list partitioned tables
- 19-2: Converting partitioned tables to auto-list partitioned tables
- 19-3: Creating a multicolumn auto-list partitioned table
- 19-4: Protecting partitions data from DML
- 19-5: Applying filter conditions to a partition operation

20

Manageability Enhancements



This lesson describes how to manage new enhancements with Database Operations.

Objectives

After completing this lesson, you should be able to:

- Start and stop DB operation in a session

To get detailed information about how to perform any of the operations explained in this lesson, refer to the following guides in the Oracle documentation:

- *Oracle Database Administrator's Guide 12c Release 2 (12.2)*
- *Oracle Database SQL Tuning Guide 12c Release 2 (12.2)*

Real-Time Database Operation Monitoring: Overview

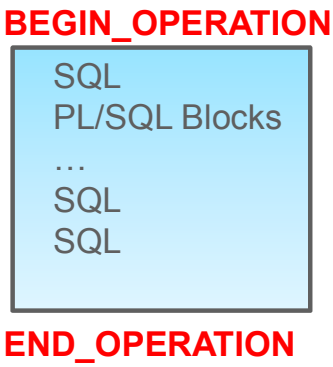
12.1

A database operation is:

- One or more SQL or PL/SQL statements
- A series of statements started from one or more sessions

A database operation can:

- Be monitored
- Produce active reports



ORACLE

Real-time SQL Monitoring helps in monitoring a single SQL statement or PL/SQL procedure/function. Oracle Database 12.1 introduces DB Operation to fill the gap of monitoring operations that contain multiple PL/SQL or SQL statements. Real-Time Database Operation Monitoring extends and generalizes Real-Time SQL Monitoring.

Real-Time SQL Monitoring helps determine where a currently executing SQL statement is in its execution plan and where the statement is spending its time.

You can use Real-Time Database Operation Monitoring for performance monitoring of active SQL statements, PL/SQL procedures, and functions. You can also see the breakdown of time and resources used for recently completed statements.

In databases where batch jobs run regularly, it can be very difficult to determine to which batch job an offending SQL statement belongs. In Decision Support Systems (DSS), it is not uncommon for many batch jobs to be running simultaneously. In some systems, hundreds of concurrent jobs could be running. Real-Time Database Operations Monitoring allows these jobs to be monitored while they are running.

Real-Time Database Operation Monitoring identifies an operation with a tag, determines what to monitor, and can generate reports. Real-Time Database Operation Monitoring monitors the progress of the operation, packages raw data for offline analysis, and creates active reports that do not require access to production systems.

Monitoring DB Operations in Sessions

12.2

1. Identify a session to monitor DB operations.
   - ▶ Operation Name
   - ▶ Execution ID
   - ▶ Session ID
   - ▶ Serial #
2. Start monitoring DB operations in the identified session with `DBMS_SQL_MONITOR.BEGIN_OPERATION`.
3. Stop monitoring DB operations in the identified session with `DBMS_SQL_MONITOR.END_OPERATION`.

```
SQL_DBA> VAR dbop_eid NUMBER;
SQL_DBA> EXEC :dbop_eid := DBMS_SQL_MONITOR.BEGIN_OPERATION
         ('ORA.MV.refresh',session_id => 12,session_serial => 34)
SQL12_34> SELECT ...
SQL_DBA> EXEC DBMS_SQL_MONITOR.END_OPERATION
          ('ORA.MV.refresh', :dbop_eid)
SQL12_34> COMMIT;
```

In Oracle Database 12.1, DB operation for a particular session could only be started/stopped from that same session by using the `DBMS_SQL_MONITOR.BEGIN_OPERATION` function and `DBMS_SQL_MONITOR.END_OPERATION` procedure.

The `DBMS_SQL_MONITOR.BEGIN_OPERATION` function is enhanced in Oracle Database 12.2 to allow the DBA to start/stop a DB operation from any session by providing the `session_id` and `session_serial` of any other session.

The `DBMS_SQL_MONITOR.END_OPERATION` procedure existed in Oracle Database 12.1 and is remains the same in Oracle Database 12.2. It disassociates a session from the specified DB operation/execution pair.

Remark: If you monitor a DB operation that runs queries only, to complete the DB operation monitoring of a session, ask the user in the session to commit after querying.

Summary

In this lesson, you should have learned how to:

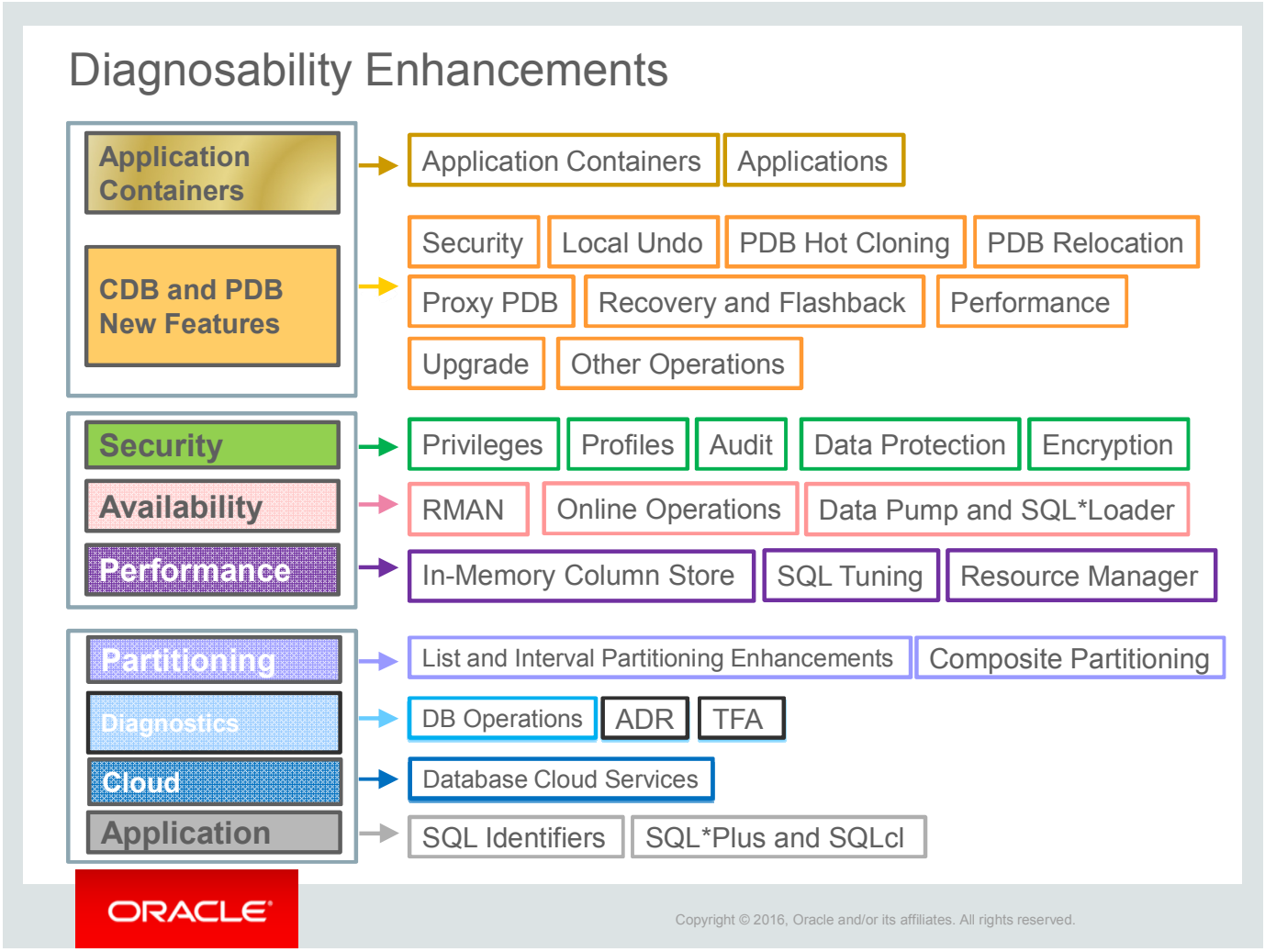
- Start and stop DB operations in a session

Practice 20: Overview

- 20-1: Monitoring DB operations in several sessions

21

Diagnosability Enhancements



This lesson describes how to manage diagnostics data with ADR space enhancements, TFA Collector and TFA Web, and IPS.

Objectives

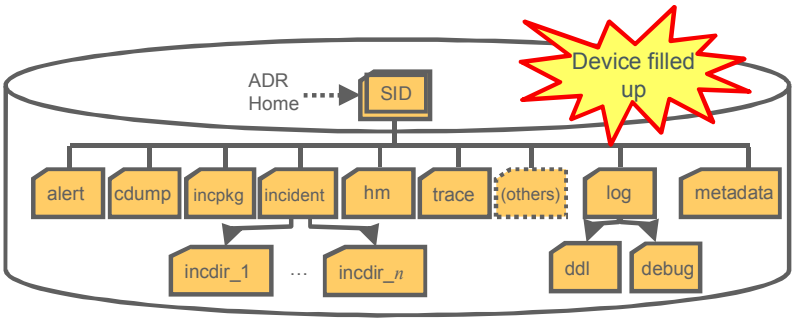
After completing this lesson, you should be able to:

- Describe ADR file automatic space management
- Collect diagnostic data with Trace File Analyzer Collector
- Analyze TFA collected data
- Trace Data Pump operations while a job is running
- Diagnose refresh performance issues related to MVs refreshed statistics

To get detailed information about how to perform any of the operations explained in this lesson, refer to the following guides in the Oracle documentation:

- *Oracle Database Utilities 12c Release 2 (12.2)*
- *Oracle Database Data Warehousing Guide 12c Release 2 (12.2)*
- *Oracle Real Application Clusters Administration and Deployment Guide 12c Release 2 (12.2)*

ADR Retention



12.1

- Limitations:
  - Removes ADR files that have expired
  - Does not prevent ADR home from filling the entire device

12.2

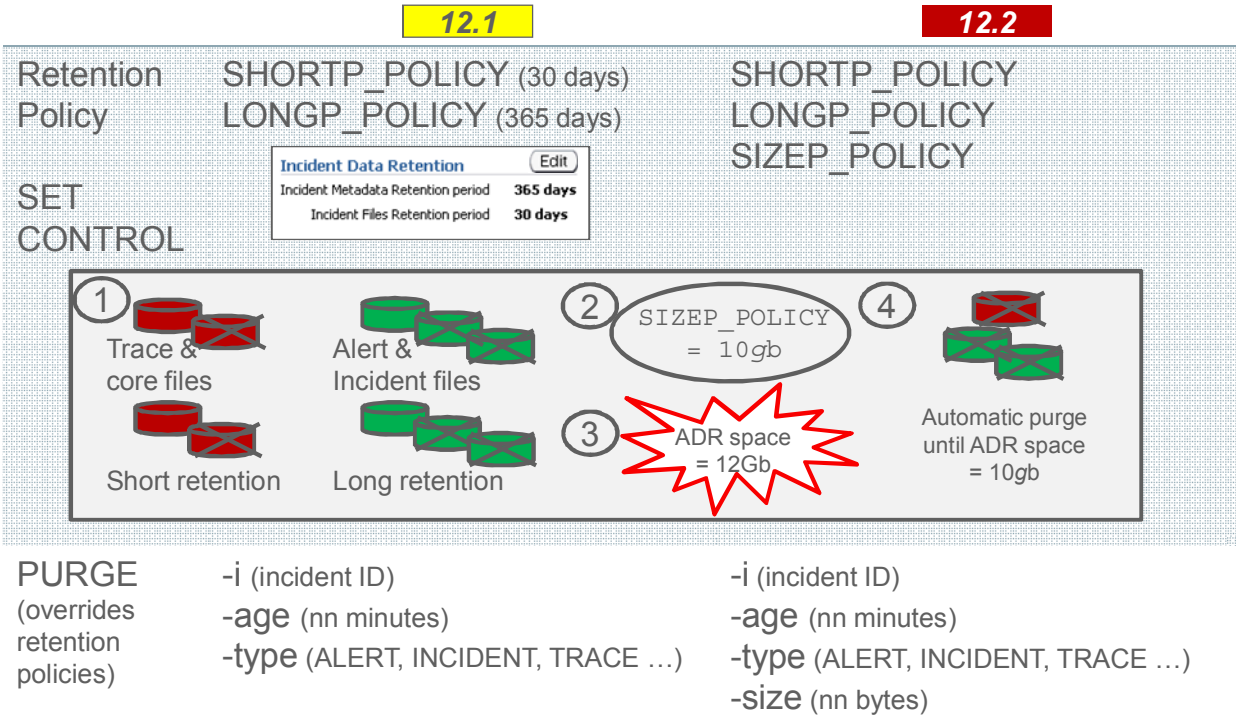
- Specify ADR size-based retention.
- Select the less relevant data to be deleted first.
  - Automatically deletes diagnostic data that reaches the target size
  - Uses the retention advisor

With Oracle Database 12.1, the Automatic Diagnostic Repository (ADR) is self-purging and removes diagnostics that have expired, including logs, traces, and dumps, as well as metadata about errors and other diagnostic information.

The repository has various features for managing space. One such feature is automatic purging of data after some expiration period has been exceeded. However, it is difficult for DBAs to predict and control how much exact disk space will be used up by further diagnostics.
Oracle Database 12.2 introduces features for automatically handling ADR's space usage:

- Automatically limiting the amount of space used by ADR to some user-specified limit
- Providing a feedback mechanism for estimating how much disk space is required for a given retention period
- Alternatively, providing a feedback mechanism for estimating how much retention you get for some particular disk space limit
- Selectively deleting less relevant data (package staging directories and old core dumps) before more relevant data (incident dumps and metadata) when ADR is under space pressure

Automatic ADR Files Purge



In Oracle Database 12.1, the purging mechanism allows you to specify a retention policy stating how old ADR contents should be before they are automatically deleted.

- The long retention period is used for the relatively higher-value diagnostic data, such as incidents and alert log. (Default value is 365 days).
- The short retention period is used for traces and core dumps. (Default value is 30 days).

Some components use these periods in slightly different ways. For instance, IPS, the packaging facility, uses the short retention period to determine when to purge packaging metadata and the staging directory contents. However, the age of the data is based on when the package was completed, not when it was originally created.

In Oracle Database 12.2, size-based retention allows administrators to specify a target size for an ADR home. When purging, the old data, determined by the time-based retention periods, is deleted first. If the size of the ADR home is still greater than the target size, diagnostics are automatically deleted until the target size is no longer exceeded.

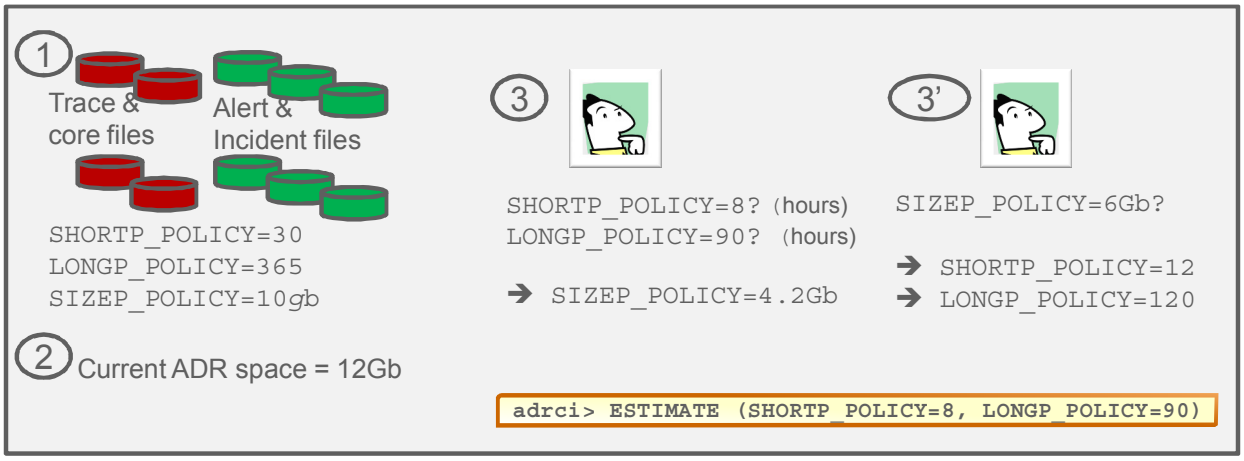
Older items are deleted first. The long retention period items are typically older than any of the items in the short retention period. So a mechanism is used in which the time periods are “scaled,” so that roughly the same percentage of each gets deleted.

You can manually purge ADR files down to some target size, without permanently changing the retention policy settings. You can reduce the amount of disk space used for a particular type of diagnostics.

ADR Retention Advisor

Choose the correct retention settings:

- SHORTP_POLICY
- LONGP_POLICY
- SIZEP_POLICY



ORACLE

The ADR advisor predicts how much space is going to be required to store the diagnostics for a particular retention period configuration. Some interesting cases could be:

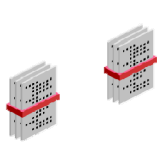
- The ADR has not reached the retention period yet and you want to know how much space ADR is going to use when it reaches the retention period
- You want to know how much data ADR is going to end up using after the next purge if you change the retention period

Note: Because the space prediction logic depends on extrapolating from existing diagnostics, it does not work in an empty or near-empty ADR.

Trace File Analyzer (TFA) Collector

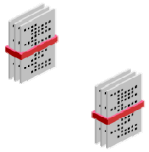
12.2

- 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 the incident time and/or components.



- Automatic trace collection on configured events with flood control
- Manual trace collection with 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

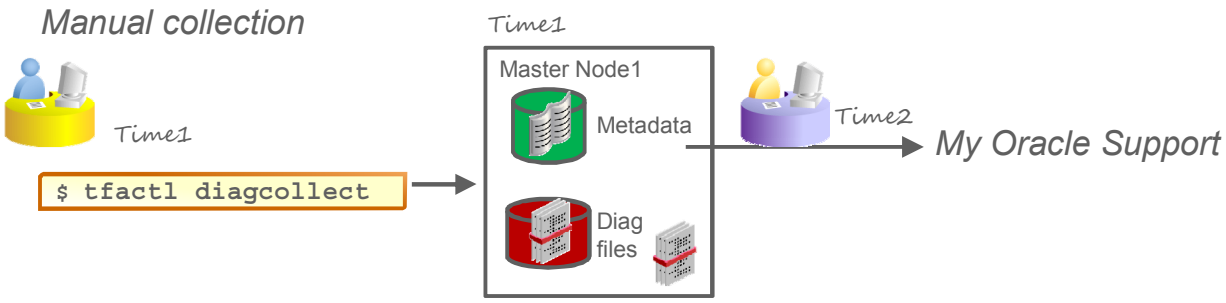
To get more information about TFA Collector, refer to the My Oracle Support note: *TFA Collector - Tool for Enhanced Diagnostic Gathering (Doc ID 1513912.1)*

Trace File Analyzer (TFA) Collector

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



Automatic collection detected:

- ORA-00600
- ORA-07445
- ORA-04031

Instance termination

TFA collection is an automated process by default. This means that periodic collections occur on single-instance databases (and on the all nodes of a cluster of RAC databases) to detect predefined events like `ORA-600`, `ORA-7445`, `ORA-4031`, node evictions, and instance termination. You can configure other events for which you want data to be automatically collected.

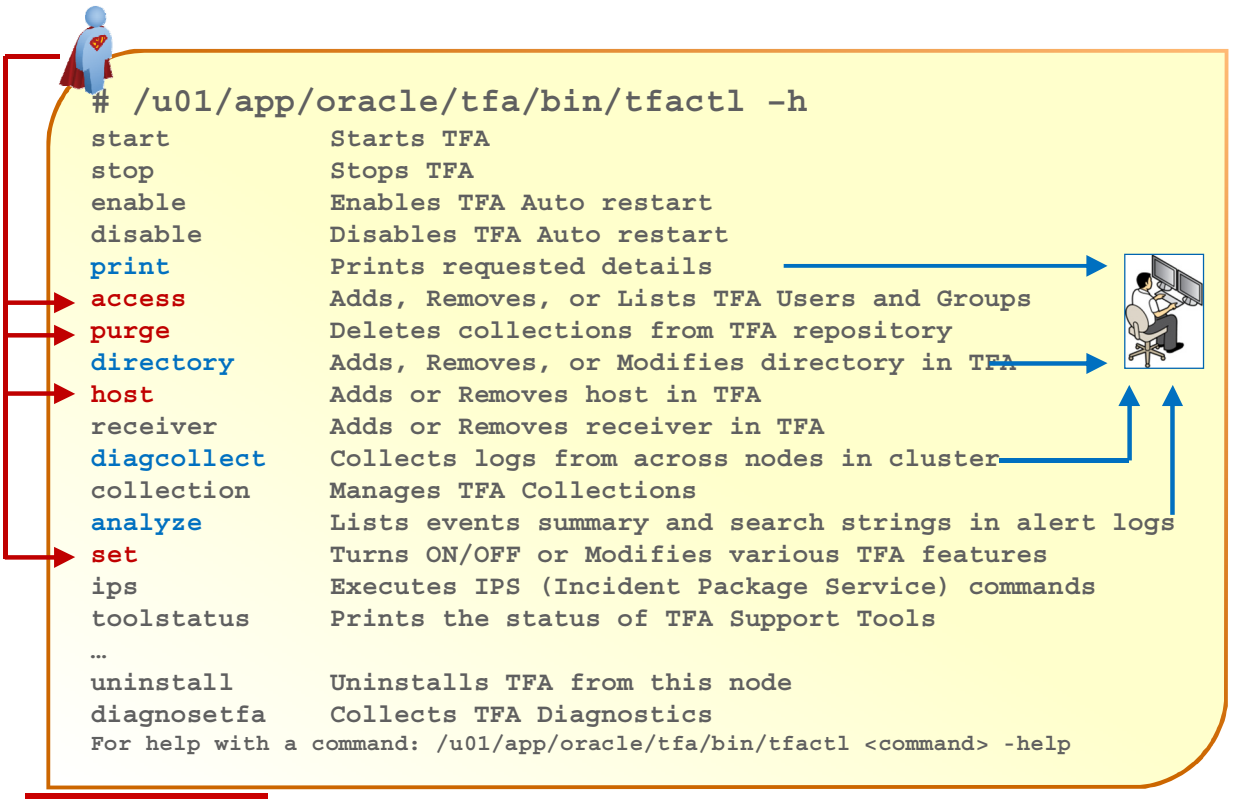
You can also start a manual collection when an unexpected issue occurs.

The manual collection is completed on the local node. Then you send all diagnostics files collected to Oracle Support for analysis.

The utility to complete collection operations is `$TFA_HOME/tfactl`.

The `diagcollect` command of the `tfactl` utility collects by default all diagnostics files (for all nodes of the cluster) for all components that were modified within the last four hours.

TFA Collector Utility



The `/01/app/oracle/tfa/bin/tfactl` utility is used to complete collection operations and other TFA Collector configuration operations. Some of these operations can be completed by `root` only and others by DBAs.

The operations completed by `root` only are `start`, `stop`, `enable`, `disable`, `access`, `purge`, `host`, and `set`. The operations that DBAs can complete are `diagcollect`, `analyze`, `directory` and `print`.

There are other commands listed in the slide. You can observe that the `IPS` command is fully integrated into TFA Collector as it is with ADR Command Interpreter `adrci`.

To get detailed information about the usage of each command, use `help <command>`.

```
# /u01/app/oracle/tfa/bin/tfactl

tfactl> help diagcollect

Usage: /u01/app/oracle/tfa/bin/tfactl diagcollect [-all | -database
<all|d1,d2..> | -asm | -dbwlm | -tns | -crs | -wls | -emagent | -oms | -ocm | -
emplugins | -em | -acfs | -install | -cfgtools | -os | -ips [-oraclehome
<oracle_home>] [-adrbasepath <adr_basepath>] [-adrhomepath <adr_homepath>] [-
level <corr_level>] [-incident <adr_incident> | -problem <prob_id> | -
problemkey <prob_key>] …
```

TFA Collector Utility

```
Options:
  -all       Collect all logs (If no time is given for collection, then files
             for the last 24 hours will be collected)
  -asm       Collect ASM logs
…
  -emagent   Collect EMAGENT logs
  -oms       Collect OMS logs
…
  -em        Collect EM logs
…
  -ips       Collect Incident Packaging Service
  -oraclehome ORACLE_HOME to be used for the ADRCI binary
  -adrbasepath ADR basepath(s) to be used for the IPS command
  -adrhomepath ADR home(s) to be used for the IPS command
  -level       ADR correlation level (basic,typical,all)
  -incident    ADR incident number
  -problem     ADR problem number
  -problemkey  ADR problemkey
…
  -database  Collect database logs from databases specified
…
  -since <n><h|d>    Files from past 'n' [d]ays or 'n' [h]ours
  -last <n><h|d>    Files from past 'n' [d]ays or 'n' [h]ours
  -from "Mon/dd/yyyy hh:mm:ss"    From <time>
  -to   "Mon/dd/yyyy hh:mm:ss"    To <time>
  -for  "Mon/dd/yyyy"             For <date>
…
  -z <zipname> The files will be collected in the tagname directory with the
               specified zipname
Examples:
 Trim and Zip all files updated in the last 8 hours as well as chmos/osw data
from across the cluster, and collect at the initiating node

/u01/app/oracle/tfa/bin/tfactl diagcollect -database hrdb,fdb -since 1d -z foo
…
```

TFA Collector Configuration

```
# /u01/app/oracle/tfa/bin/tfactl
tfactl> help set

Turn ON/OFF or Modify various TFA features
autodiagcollect     Allow for automatic diagnostic collection when
                    an event is observed (default ON)
autopurge           Allow automatic purging of collections when
                    less space is observed in repository (default
                    OFF)
minagetopurge       Set the age in hours for collections to be
                    skipped by AutoPurge (default 12 Hours)
trimfiles           Allow trimming of files during diagcollection
                    (default ON)
tracelevel          Control the trace level of log files in
                    /u01/app/oracle/tfa/edcdr19p1/tfa_home/log
                    (default 1 for all facilities)
repositorydir=<dir> Set the diagcollection repository to <dir>
reposizeMB=<n>      Set the maximum size of diagcollection
                    repository to <n>MB
logsize=<n>         Set the maximum size of each TFA log to <n>MB
                    (default 50 MB)
logcount=<n>        Set the maximum number of TFA logs to <n>
                    (default 10)
…
```

To configure the collection processing, the `root` user uses the `SET` command of the `tfactl` utility.

Examples:

```
# /u01/app/oracle/tfa/bin/tfactl set autodiagcollect=ON
# /u01/app/oracle/tfa/bin/tfactl set autopurge=ON
# /u01/app/oracle/tfa/bin/tfactl set tracelevel=INVENTORY:3
# /u01/app/oracle/tfa/bin/tfactl set reposizeMB=20480
# /u01/app/oracle/tfa/bin/tfactl set logsize=100
```

TFA Collector Analysis

Which Files
CRS, ASM, and DB alert logs
System logs

How
By component
By time
By specific search patterns
By type (error, warning)

- Show summary of events from alert logs, system messages in last 5 hours.
```
# tfactl analyze -since 5h
```
- Show summary of events from system messages in last one day.
```
# tfactl analyze -comp os -since 1d
```
- Search string ORA- in alert and system logs in the past two days.
```
# tfactl analyze -search "ORA-" -since 2d
```
- Run oratop in batch mode for database cdb1.
```
# tfactl analyze -comp oratop -database cdb1 -bn1
```

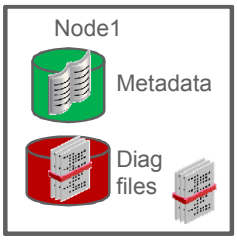
ORACLE

The `ANALYZE` command of the `tfactl` utility allows you to quickly get diagnostic information about events and also search strings in diagnostic files.

TFA Collector Repository

Check storage used by TFA Collector on nodes repository.

10*g*b
by default




```
# tfactl print config
```

- 30Gb in the repository

No more collections when maximum configured size of repository reached

```
# tfactl set reposizeMB=100000
# tfactl purge -older 30d
# tfactl set repositorydir=newreposdir
```


It is important to check when the repository for collections is filled up. The `print` command of the `tfactl` utility displays the current space used by the repository for all collections. TFA Collector does not generate any new collection when the maximum size of the repository is reached, avoiding the filling of file systems by new collections.

You can either increase the maximum size for the repository or delete existing collections (older than 30 days in the example in the slide) or even change the location of the repository to another directory on another file system.

Tracing Data Pump

- 12.1 Start tracing by setting the TRACE parameter when the Export Data Pump or Import Data Pump is started.
- 12.2 Change DP tracing without stopping and restarting the job.
  - Use the Data Pump interactive command line.
  - Set the appropriate trace value.
- Useful for changing tracing level for long DP jobs

```
$ expdp system@pdb1 job_name=j_pdb1
 ...
Starting "SYSTEM"."J_PDB1":
system/********@pdb_orcl job_name=j_pdb1
directory=dp_trace
Processing object type
…
;;; Ext Tbl Query Coord.: worker id 1 for
"SYS"."IMPDP_STATS"
;;; Ext Tbl Query Coord.: worker id 1 for
"SYS"."IMPDP_STATS"
;;; Ext Tbl Shadow: worker id 1 for
"SYS"."IMPDP_STATS"
…
```

```
$ expdp system@pdb1 attach=j_pdb1
 ...
CLIENT_COMMAND system/********@pdb1
job_name=j_pdb1 directory=dp_trace
      TRACE                  0
  State: EXECUTING
Export> TRACE = 1ff0b00
Export>
```

- Useful for Oracle Support to diagnose

With Oracle Database 12.1, tracing Data Pump export and import is possible by specifying the TRACE parameter and a trace level when the Export Data Pump or Import Data Pump is started. This is the standard way to activate Data Pump tracing.

If an export or import Data Pump job is started without the TRACE parameter, it can be temporarily stopped, and restarted with the TRACE parameter.

Oracle Database 12.2 allows Data Pump tracing to be enabled or disabled, and the tracing level changed while a Data Pump job is running without stopping and then restarting the job.

MVs Refreshed Statistics History

Diagnose MV refresh performance issues with refreshed statistics.

- Collect MVs refresh statistics at the level of individual MVs.
- Store historical MVs refresh statistics.
  - ➔ Provides insight into how the MV ecosystem or a specific MV has evolved
  - ➔ Provides unique insight, both for historical analysis and for diagnosis purposes

DBA_MVREF_RUN_STATS
DBA_MVREF_STATS
DBA_MVREF_CHANGE_STATS
DBA_MVREF_STMT_STATS
DBA_MVREF_STATS_PARAMS
DBA_MVREF_STATS_SYSTEM_DEFAULTS

```
dbms_mview_stats.set_system_default('COLLECTION_LEVEL','TYPICAL')
dbms_mview_stats.set_system_default('RETENTION_PERIOD', 31)

dbms_mview_stats.set_mvref_stats_params('MV1,MV2','ADVANCED',90)
dbms_mview_stats.set_mvref_stats_params(NULL,'NONE',NULL)

dbms_mview_stats.purge_refresh_stats(null,3)
```

Materialized Views refresh statistics history helps users view the refresh history in their system and track refresh performance over time.

MVs refreshed statistics history complements the `EXPLAIN_REFRESH` information by providing statistics on actual refresh execution times. It serves as a reporting and diagnostic tool useful when customers face refresh performance issues. Typically, this happens at times when users want to resolve the issue quickly. With detailed refresh history statistics, the DBA is able to quickly diagnose the reason for the slowdown, such as whether it is due to increased load or a bug.

MVs refreshed statistics history provides a new package, `DBMS_MVIEW_STATS`, to collect refresh statistics at different levels of granularity, for all MVs or specific MVs. This is useful since the refresh patterns of MVs can vary widely and the DBA may be interested in monitoring only some MVs.

The collection-level property specifies the level of detail at which refresh statistics are collected for the MV.

- `NONE`: No statistics are collected.
- `TYPICAL`: Basic statistics are collected. This is the default value.
- `ADVANCED`: Detailed statistics are collected.

The retention-period property specifies the retention period in days for which the refresh statistics are retained. The initial systemwide default value is 31 days.

MVs Refreshed Statistics History

Displaying basic statistics for a MV refresh operation includes the refresh method, refresh time, number of rows in the MV at the start of the refresh operation, and the number of rows at the end of the refresh operation.

```
SQL> SELECT refresh_id, refresh_method, elapsed_time,
            initial_num_rows, final_num_rows
     FROM   dba_mvref_stats
     WHERE  mv_name = 'NEW_SALES_RTMV' and mv_owner = 'SH';

REFRESH_ID    REFRESH_METHOD    ELAPSED_TIME    INITIAL_NUM_ROWS
FINAL_NUM_ROWS
----------    --------------    -------------   ----------------    ----
------------
49            FAST              0               766                 788
61            FAST              1               788                 788
81            FAST              1               788                 798

SQL>
```

Summary

In this lesson, you should have learned how to:

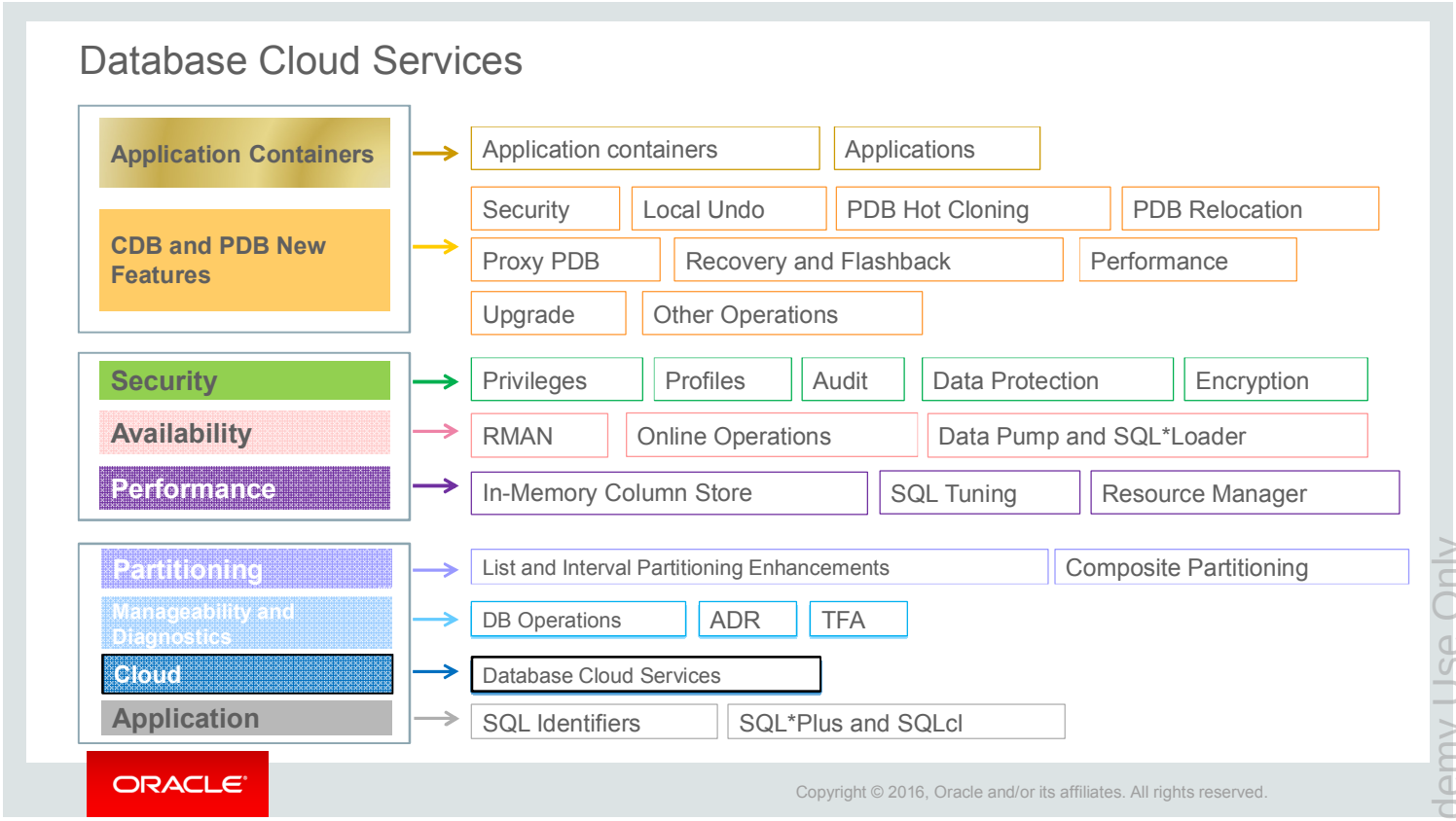
- Describe ADR file automatic space management
- Collect diagnostic data with Trace File Analyzer Collector
- Analyze TFA collected data
- Trace Data Pump operations while a job is running
- Diagnose refresh performance issues related to MVs refreshed statistics

Practice 21: Overview

- 21-1: Using ADR advisor to set ADR size retention policy
- 21-2: Configuring TFA Collector and analyzing collections

22

Oracle Database Cloud Services



This lesson gives a good overview of the Oracle Database Public Cloud Services.

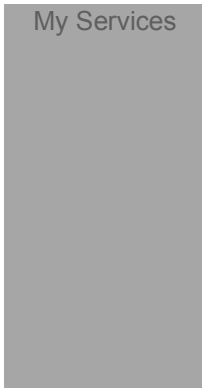
Objectives

After completing this lesson, you should be able to:

- Describe the main services of Oracle Database Cloud Services (DBCS)
- Describe the components of DBCS and their interactions
- Define migration methods for on-premises databases to Cloud

For a complete understanding of Oracle Cloud Public Services and components, refer to the following documentation in the Oracle Cloud Support documentation:

- *Cloud documentation* (*http://www.oracle.com/pls/topic/lookup?ctx=cloud*)



This video shows you at a high level the various Oracle Database Cloud single-instance service components and their interactions.

From now one we will refer to this service as Database Cloud Service or DBCS.

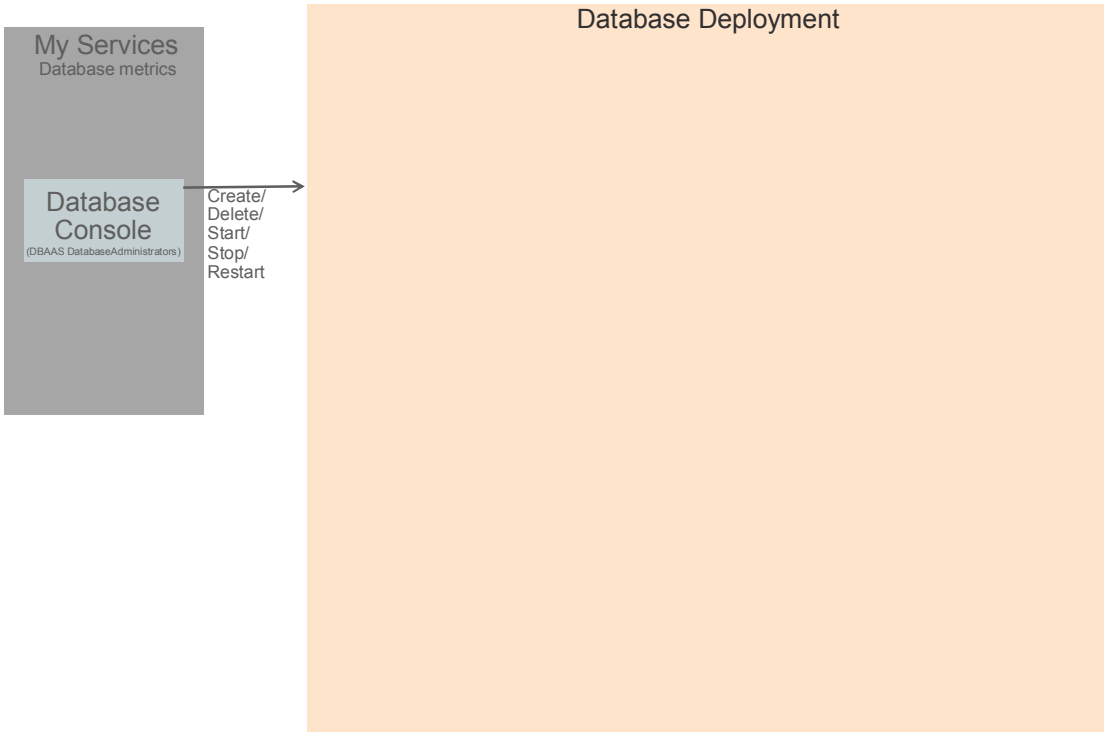
We do not get into the details of how you can get an Oracle Cloud account and we suppose you are already done with this phase.

As an Identity Domain Administrator or a Service Administrator, your main entry point to manage your services is what we call the My Services web console or dashboard.

From that web page you can get an overall information about all services running in your identity domain.

From the My Services page, you can open what is called the Oracle Database Cloud Service Console: another web page allowing you to manage your DBCS environment.

Note the My Services page computes database metrics periodically.

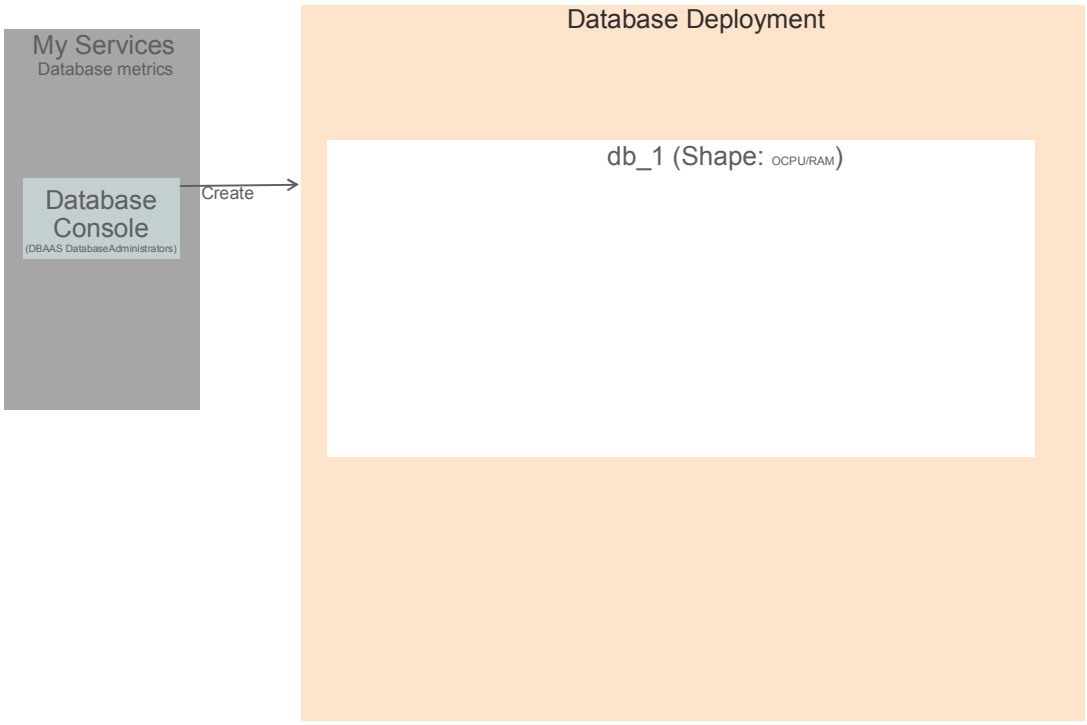


From the Database Console, you can create, delete, start, stop, restart, patch, backup, recover, add new ssh public keys, or scale up and down what are called DBCS instances or Database Deployments: a combination of resources used to give you access to an Oracle Database.

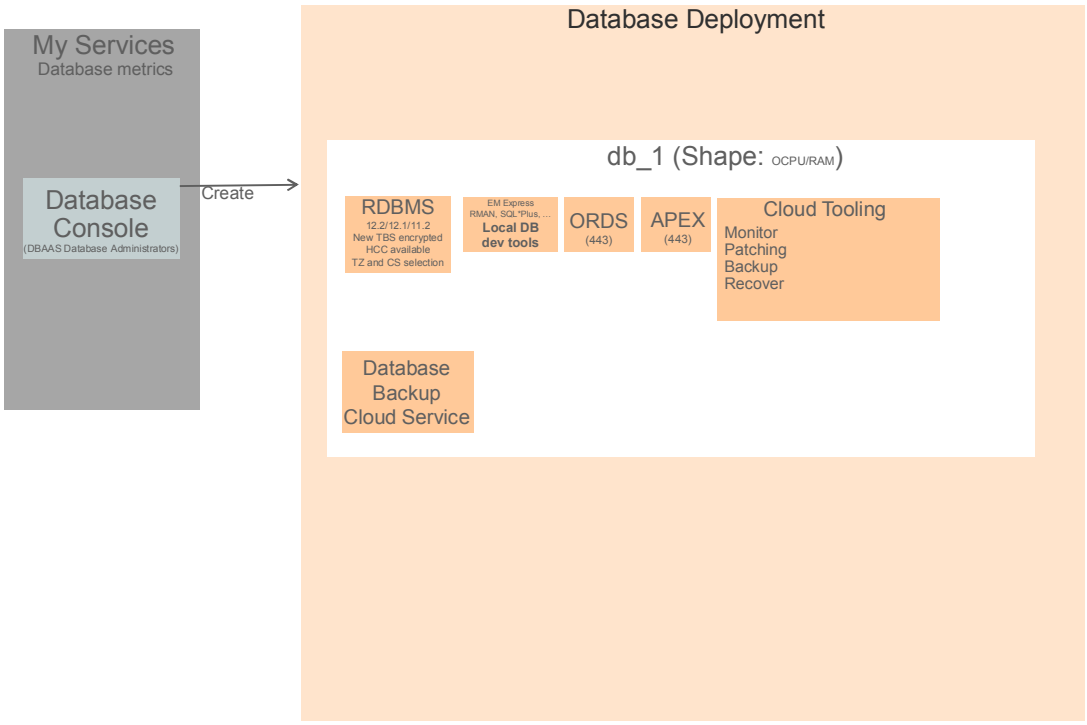
To be able to get to both the My Services and Database consoles, you need certain privileges granted to your Oracle Database Cloud Service account through roles.

For Database as a Service, one role exists and is automatically granted to the Identity Domain administrator: DBAAS DatabaseAdministrators.

With this role you can manage all instances in your domain.



One of the main components of a DBCS Instance is a Virtual Machine called db1 and created with a certain compute power that we call a shape: a combination of a number of CPUs (OCPU) and RAM.



Within this VM, you find an Oracle Database already installed and created. This could be an 11gR2 (11.2.0.1), 12cR1 (12.1.0.2), or 12cR2 (12.2.0.1) database depending on how you defined the creation of your service instance from the Database Console.

By default this database is created using AL32UTF8 character set but you can select a different one like you can select a time zone different from default UTC.

Also by default, every non-system tablespaces are encrypted using TDE technology, and Hybrid Columnar compression is also available.

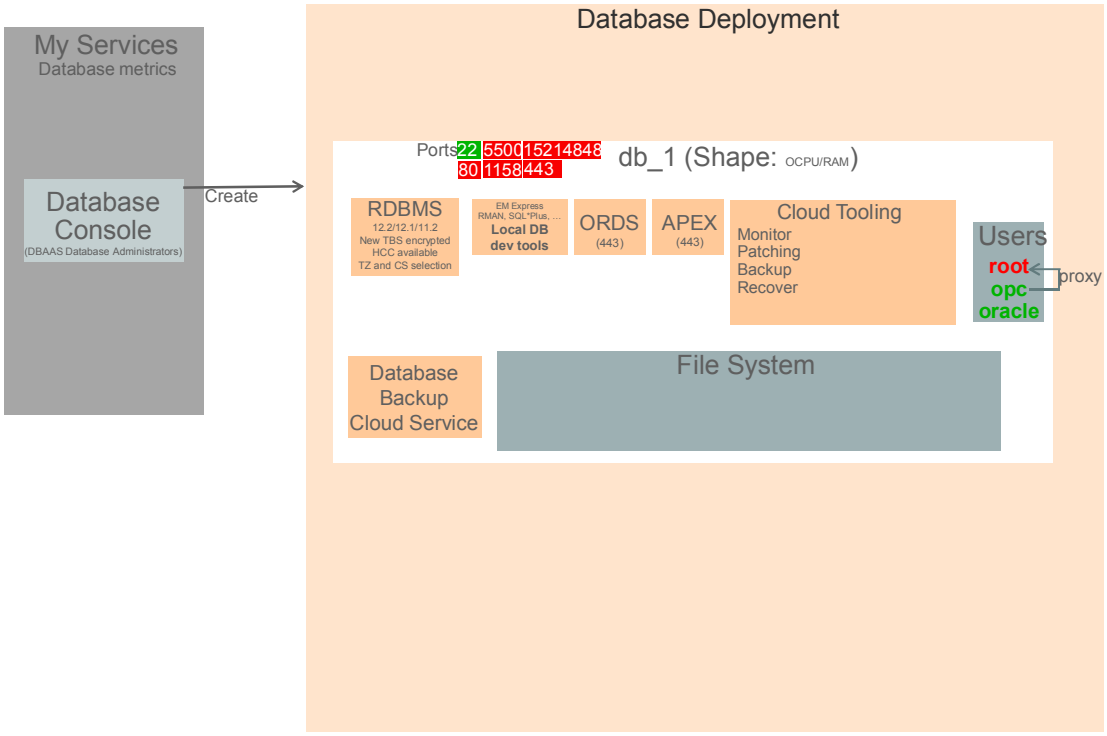
Of course, you also find within the VM all local tools you are familiar with to administer your database.

They are the same as the ones you find on your on-premises installations like SQL*Plus, RMAN, EM Express, ...

In addition, you find Oracle REST Data Services (ORDS) which makes it easy to develop modern REST interfaces for relational data in the Oracle Database as well as the Oracle Application Express.

Something you will not find on your on-premises installations are the Cloud tooling for easier patching, backup, recover, and monitoring of your Oracle Database.

If you want you can backup your database to Cloud using the Database Backup Cloud Service.



In terms of pre-created users, you will find oracle and opc for which you have access.

OPC or Oracle Public Cloud user is a user with sudo privileges to run root commands.

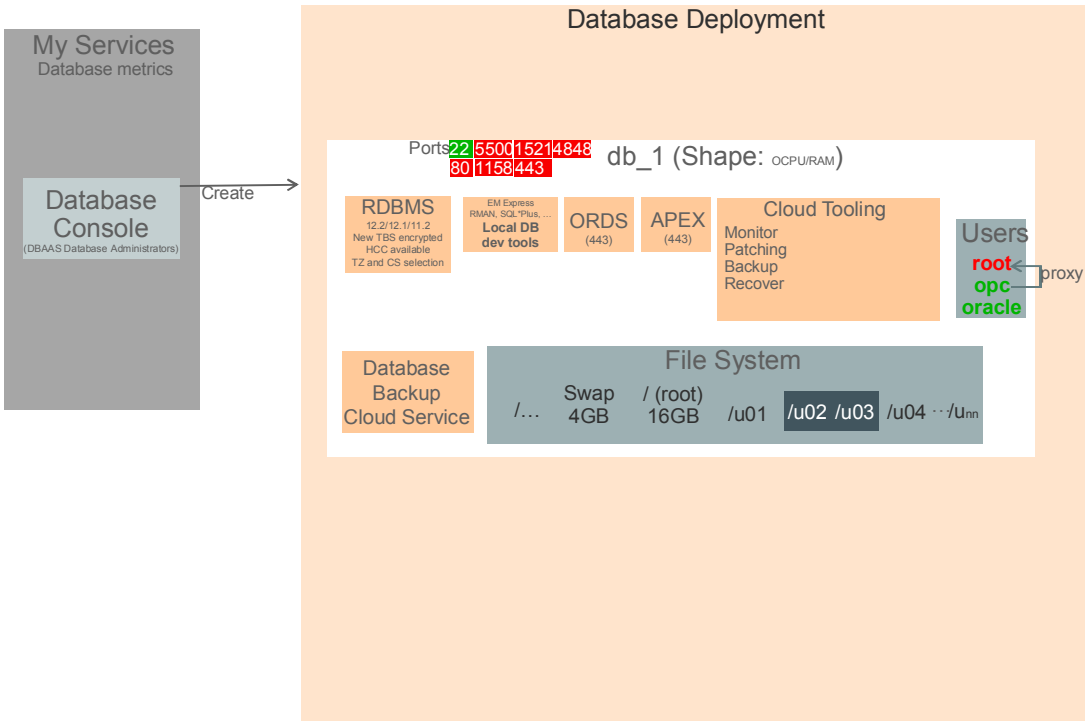
The oracle user is the one used to install the Oracle software.

Very important are the accessible network ports by which you can access your VM through the network.

By default, only SSH port 22 is open for communication, which allows you to create tunnels for the other ports while they are blocked.

As we will see, you have the possibility to unblock them if you want.

Last but not least, you will get a file system where software and data are staged.



You have 4GB of swap space, 16GB for the root, where /u01 stores software, /u02 your database data files, /u03 your database backup files, and /u04 your database redo log files.

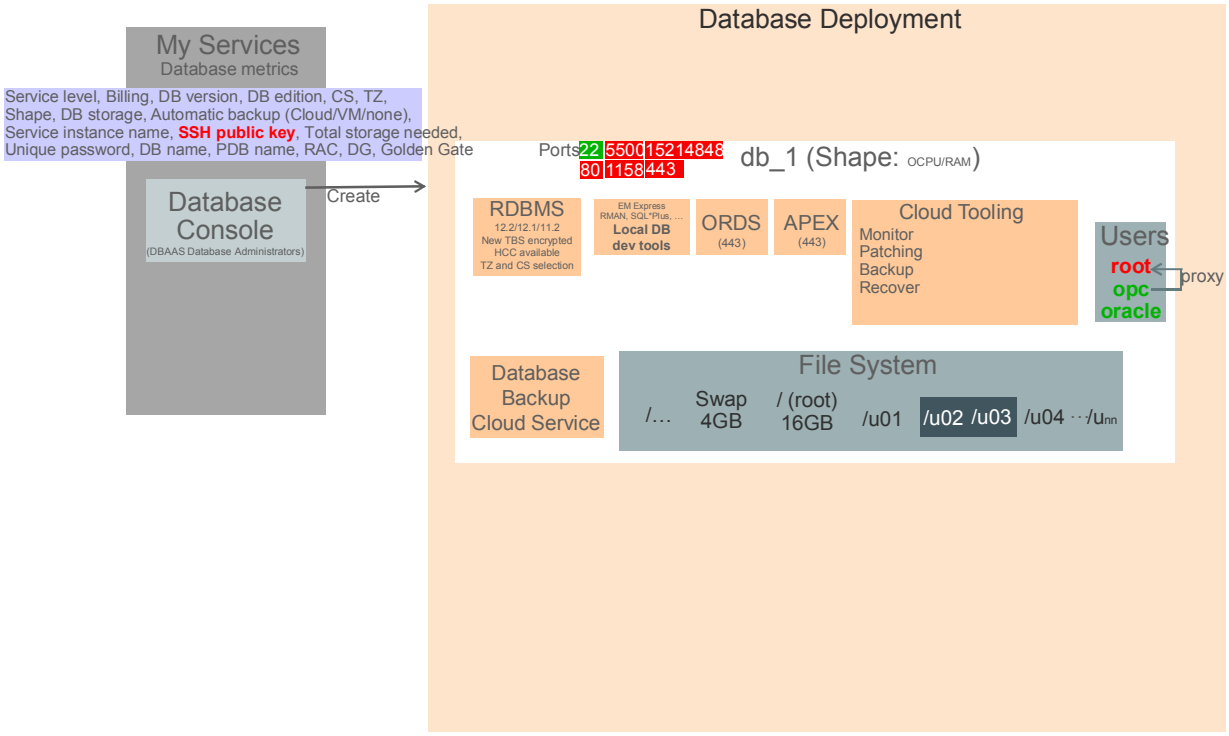
All this is configured automatically for you.

When creating a database deployment, you choose the amount of usable data storage you want for your database in gigabyte (GB) increments up to a maximum of 2 TB

(2048 GB).

After you create the database deployment, you can add more data storage as needed.

By adding more storage, you can create a database of up to 4.6 TB with local backups or up to 12 TB without local backups.



So, to create such a VM, all you need to do is to go to your Database Console, and answer few questions like the shape of your VM;

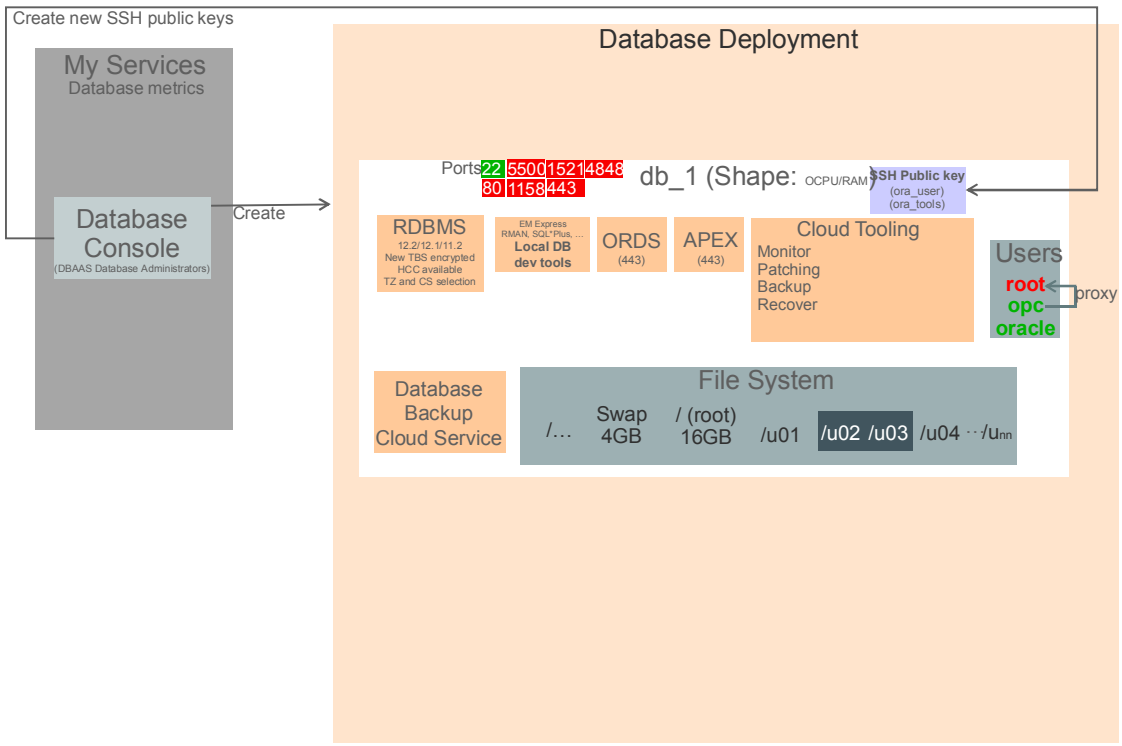
size of your database storage; where to store backups: locally, on storage cloud, both, or no backups at all.

You need to give your service instance a name, an SSH public key used in conjunction with a corresponding private key to securely connect to your VM once created through SSH communication. You can have the wizard create the key pair for you during deployment creation and download a zip file containing the public and private keys.

You specify also the total size for your filesystem, a unique password for all key database users and encryption password, a database name, and optionally a Pluggable database name.

You also have the possibility to select your time zone, character set, use RAC, Data Guard, or Golden Gate for replication.

You also have the possibility to include a PDB dedicated to demo new features.



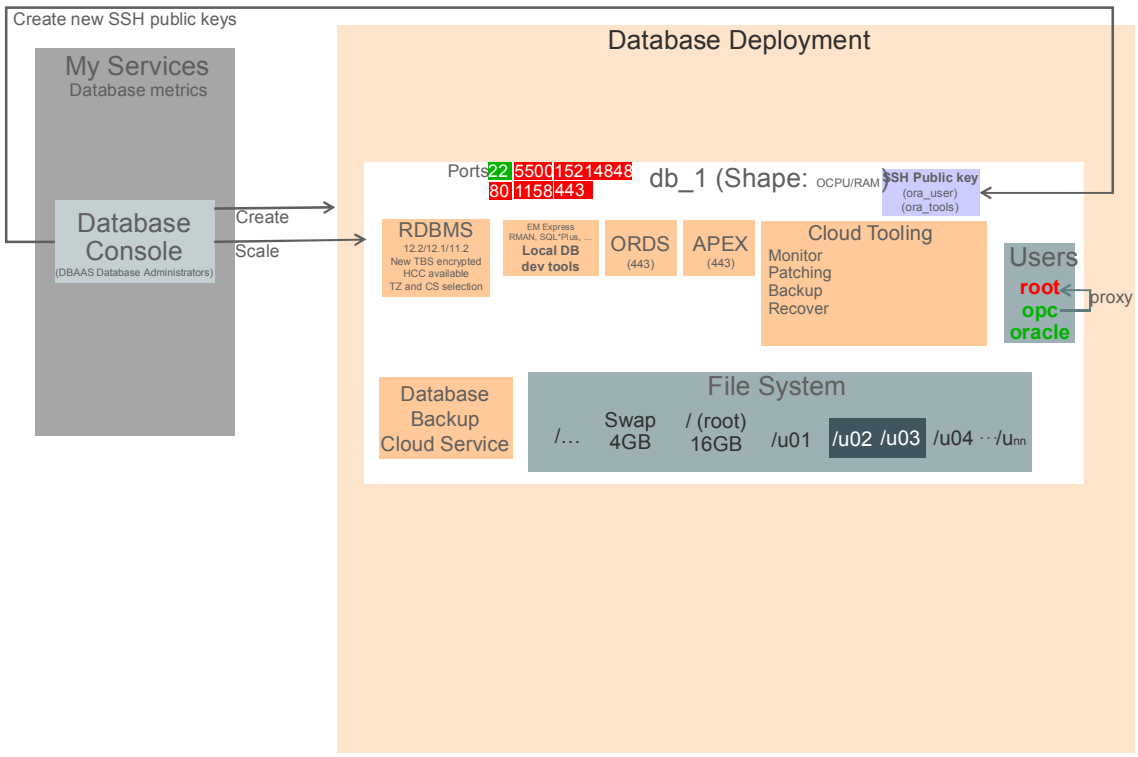
At the time you create your database deployment, you need to specify an SSH public key which will be used in conjunction with a corresponding private key to allow VM access.

By default this public key is called ora_user.

Another one (ora_tools) is also created to allow your My Services console to securely communicate with your VM to gather various utilization statistics and metrics for your database and compute resources.

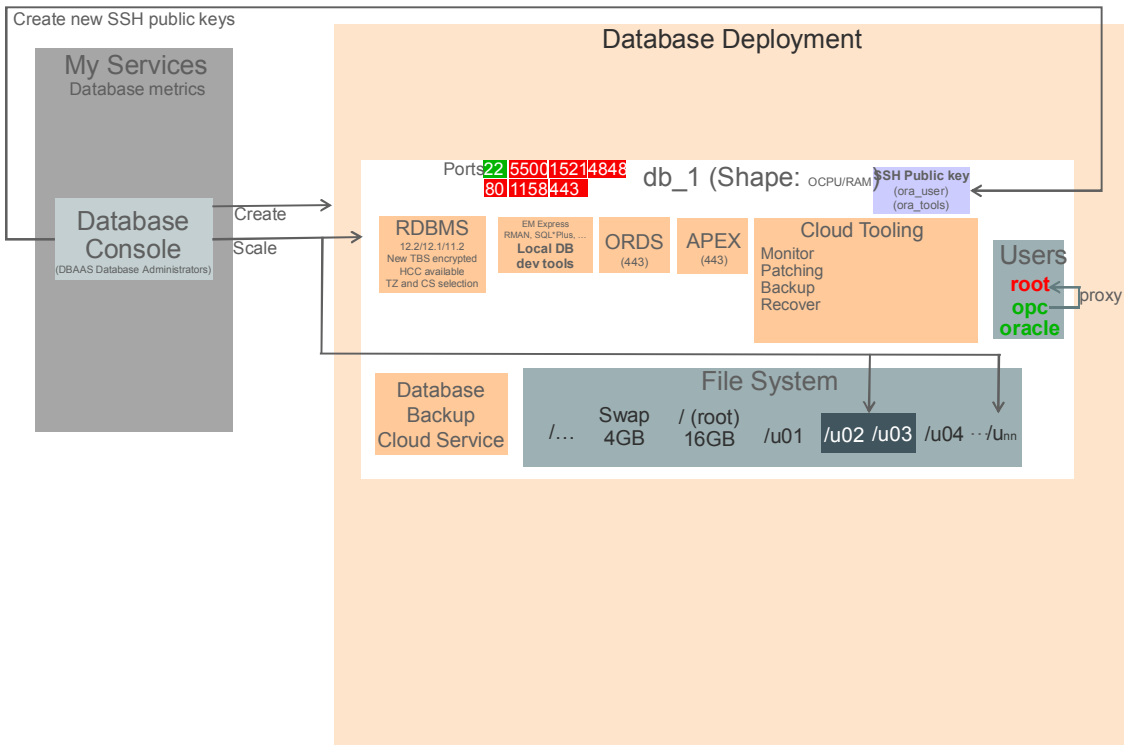
If there is a need, you can add new public keys for opc and oracle users to access your VM.

This can be done directly from the Database Console.



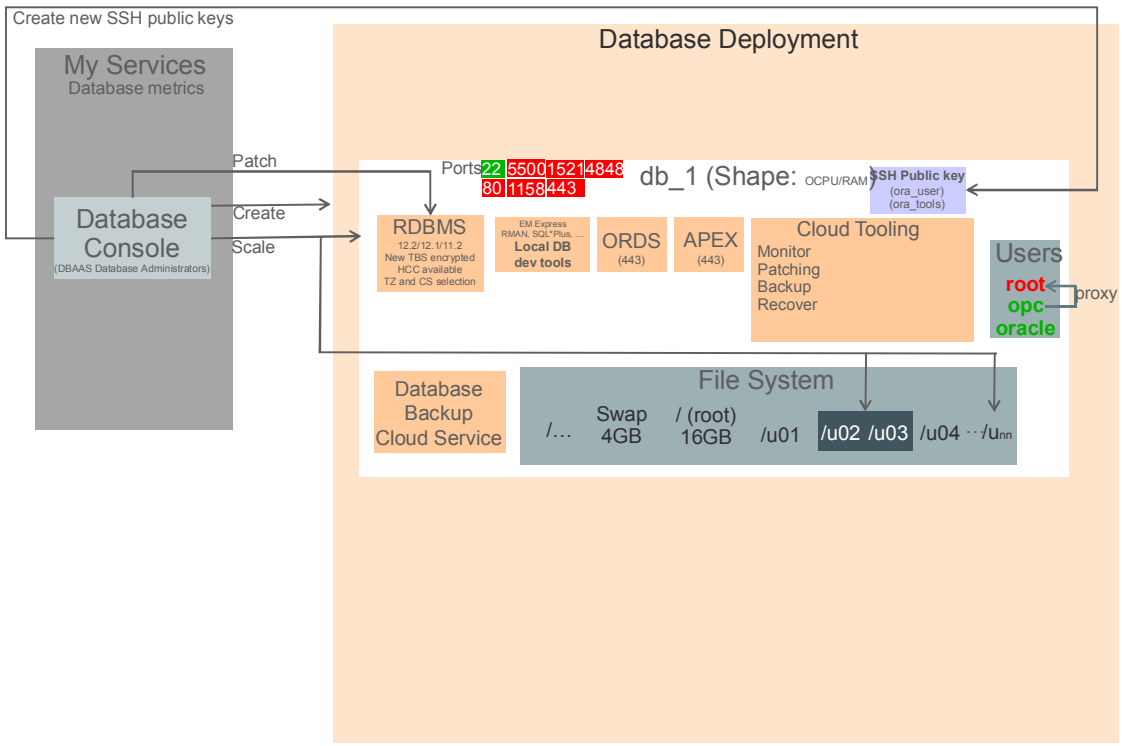
One important operation you can do directly from the Database Console is scaling your VM shape if you want to add more CPUs and memory to your database service.

This operation will temporally stop your service before the VM is recreated without loss of any of your data.

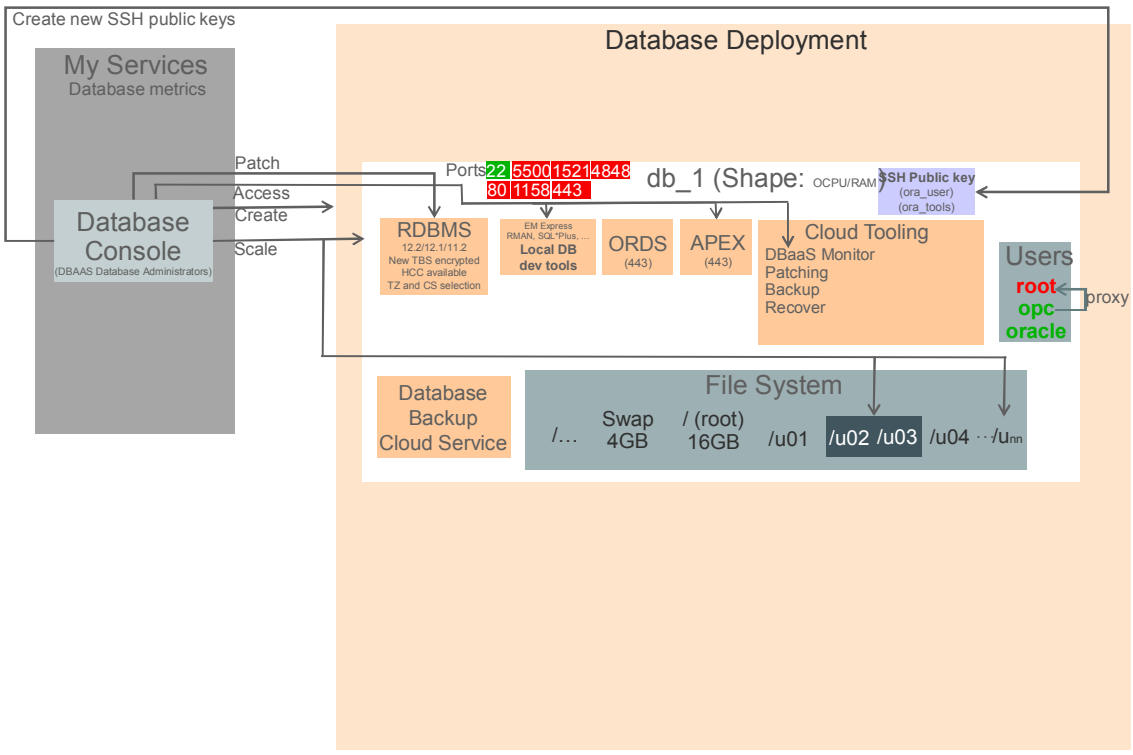


The second scaling operation you can do from the Database Console is scaling your storage if you need more space for your data, backups, or anything else.

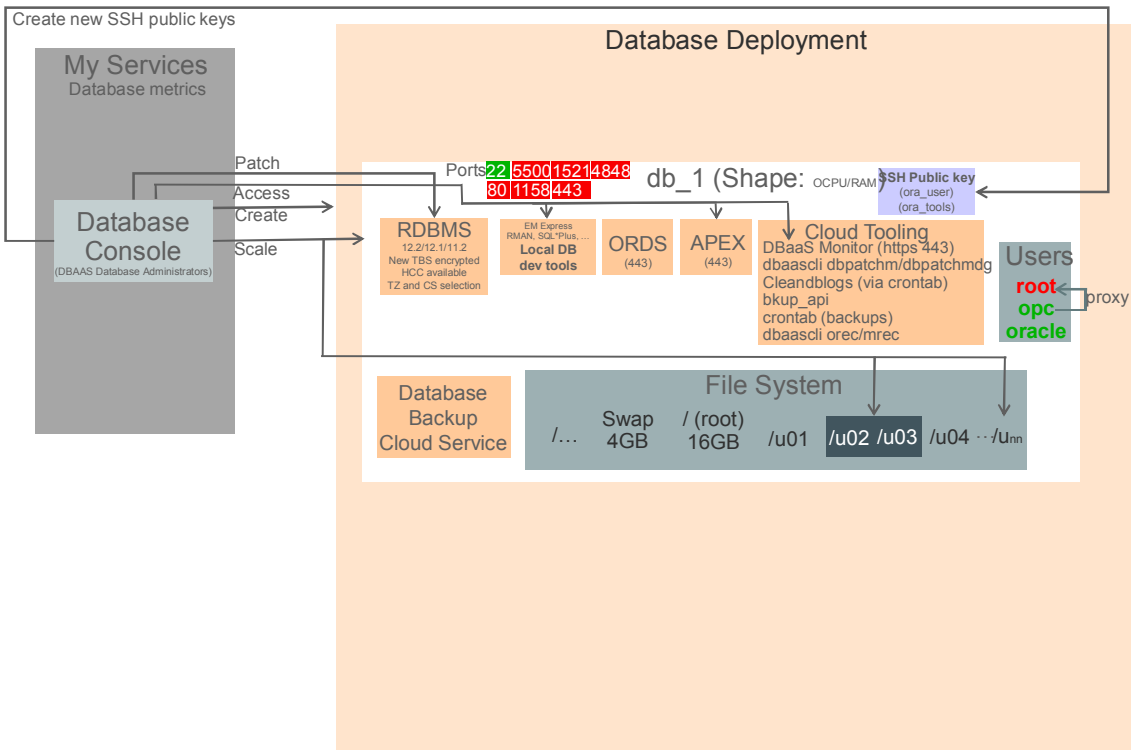
All this done by a simple click!



Another important operation that is automated from the Database Console is Database patching.



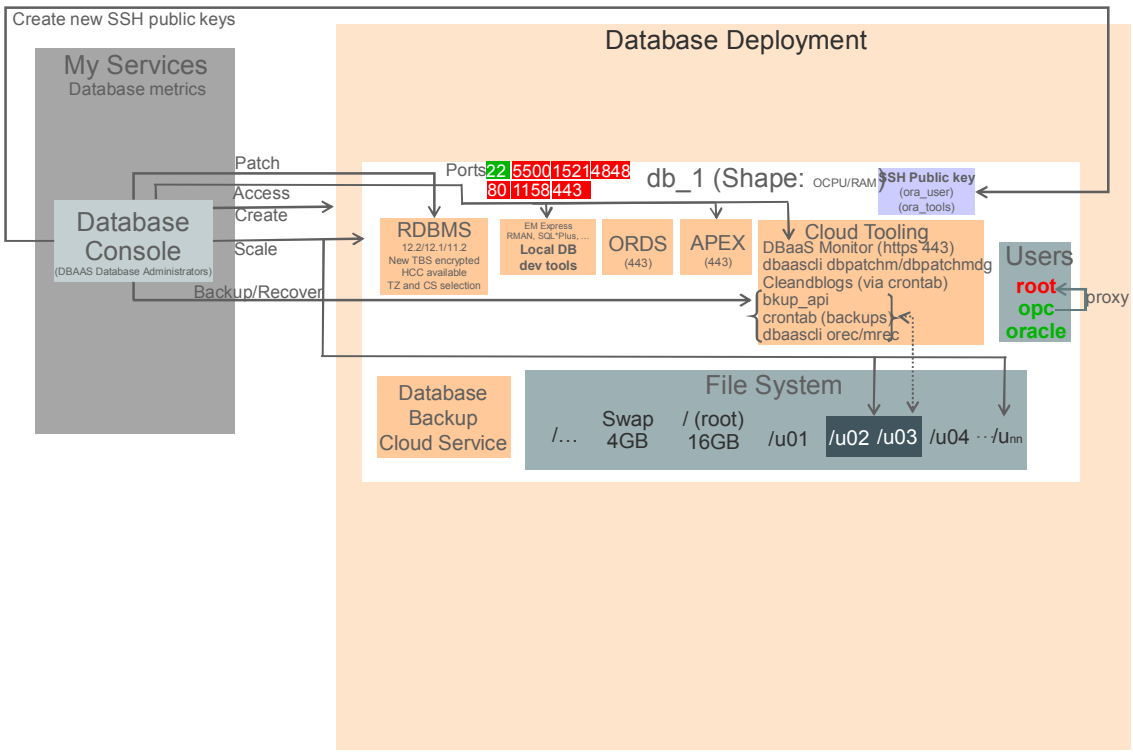
You also have the possibility to access directly from your Database console the EM Express console as well as the Apex console and the Database Monitor.



As for Cloud Tooling, you can see a list of tools you can use to help you manage your Database like dbaascli for patching, data guard, and recovering, mrec for last option recovery of your database, cleandblogs for purging log files, bkup_api for backups, crontab is used to automate backup creation and log purging.

For backup and recovery, these tools are based on Recovery Manager with a simple syntax for basic operations.

You also find a new web interface for monitoring your Database and VM called DBaaS Monitor.

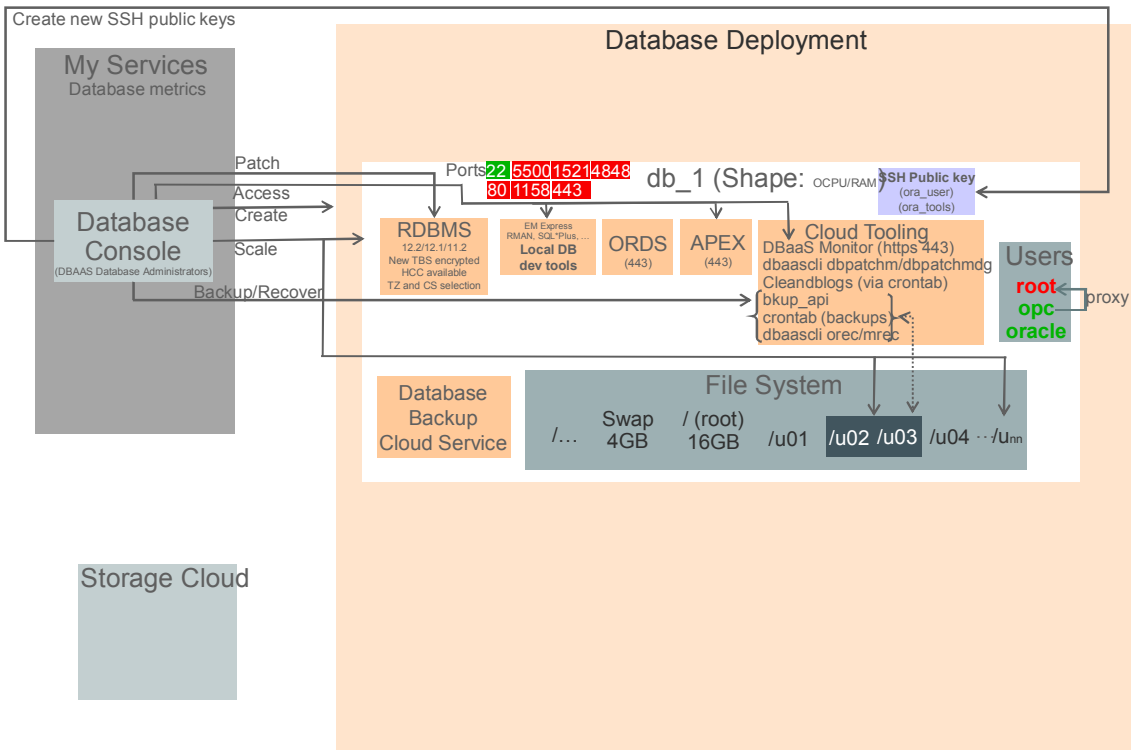


Regarding backup and recovery operations, you can create a backup or recover your database directly from the database console.

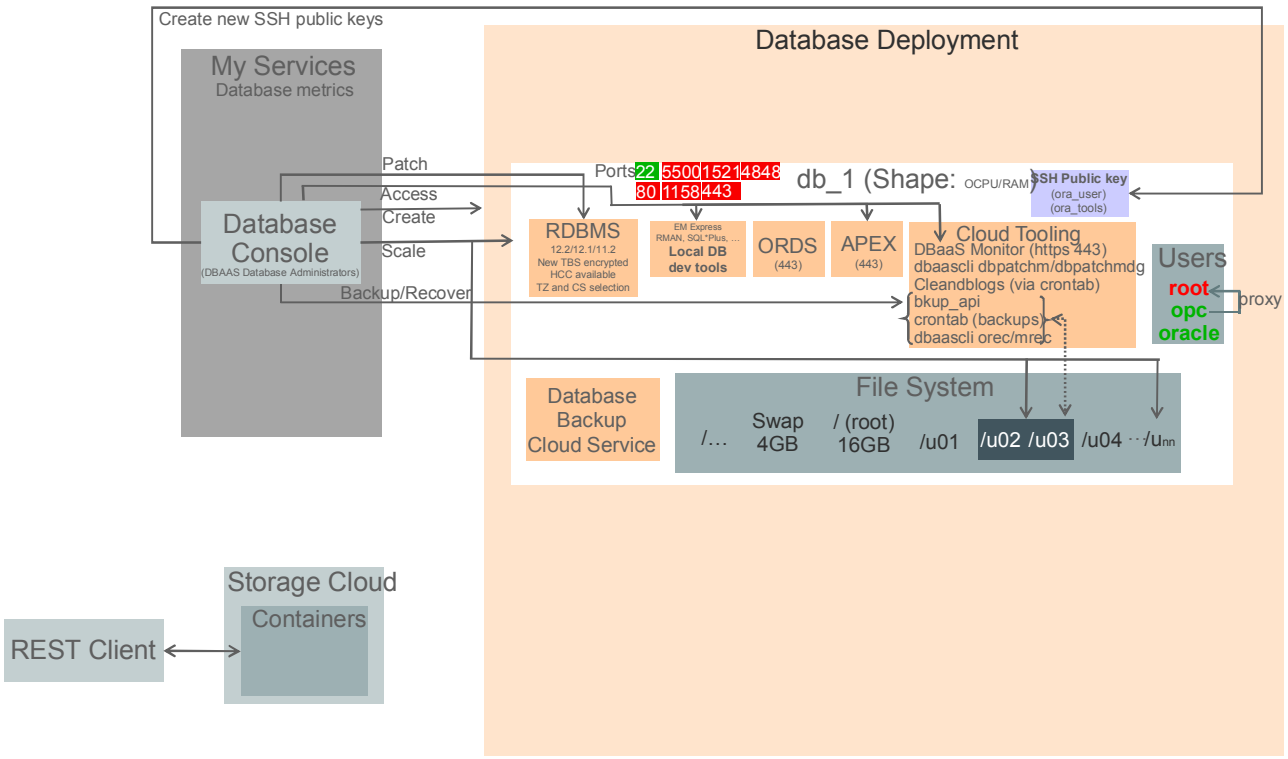
If you choose to backup to your local VM, backups are automatically generated for your database files, database configuration files, as well as important OS configuration files.

They are stored in /u03 using an incremental policy you can change.

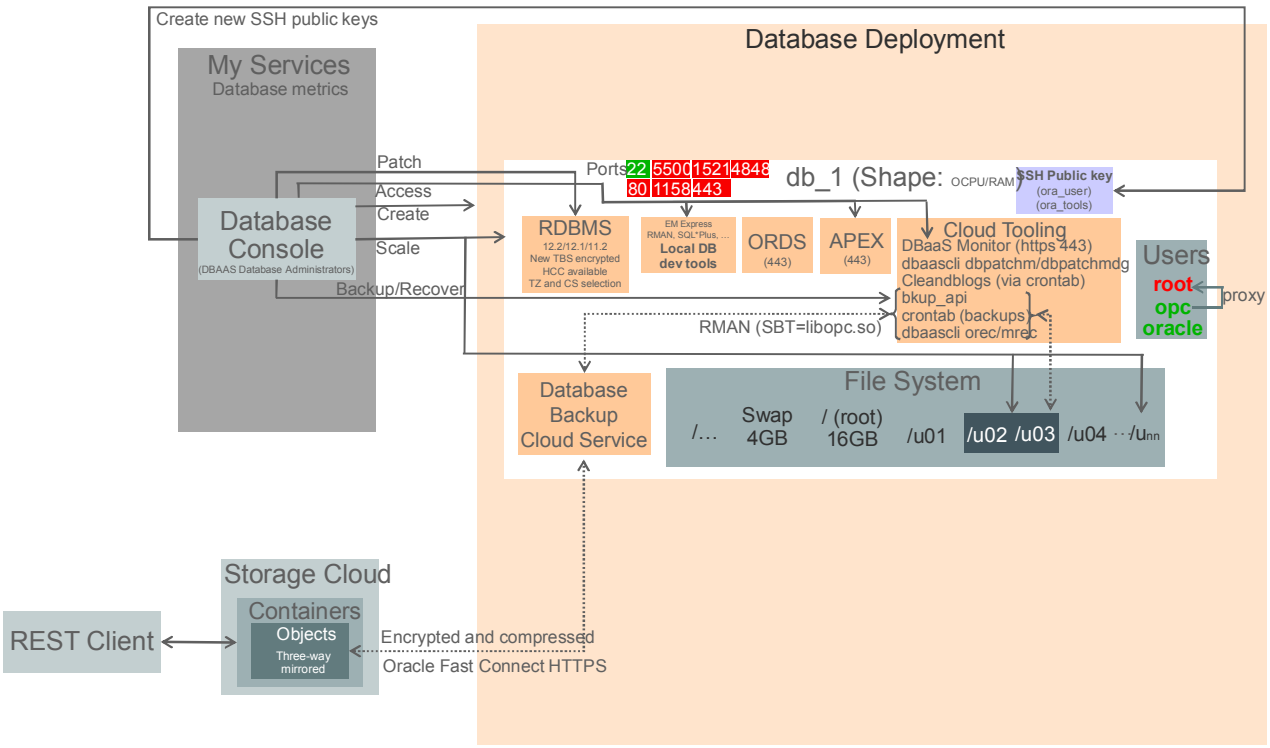
The retention period is 7 days by default on your local storage.



In addition to block storage provided by the Compute Block Storage layer, Oracle also provides a longer term type of storage capability that can be managed independently of your database and called Storage Cloud Service.



You interfere with your Storage Cloud Service mainly from your browser using REST APIs by managing what are called containers.

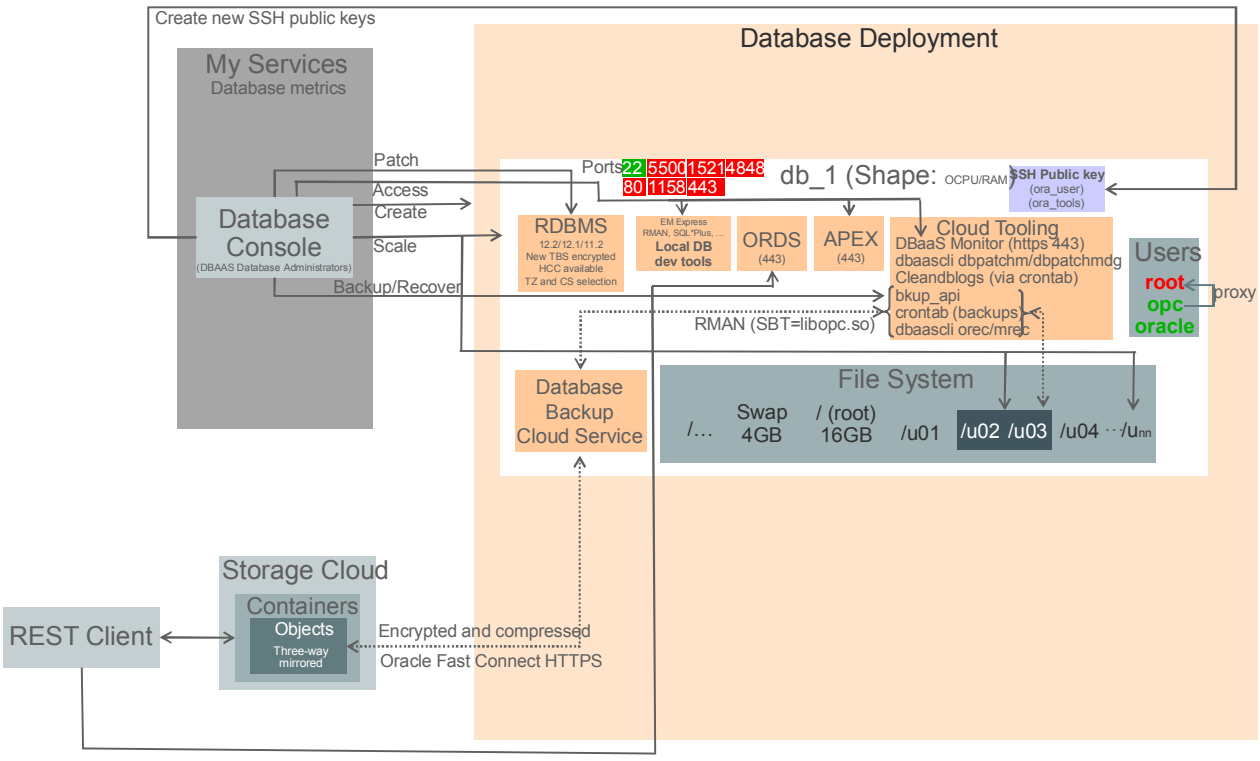


If you configured your database backups to also use Storage Cloud Service, then you must either have created a new container before you can create your DBaaS instance, or you can have the wizard create the container for you during deployment creation.

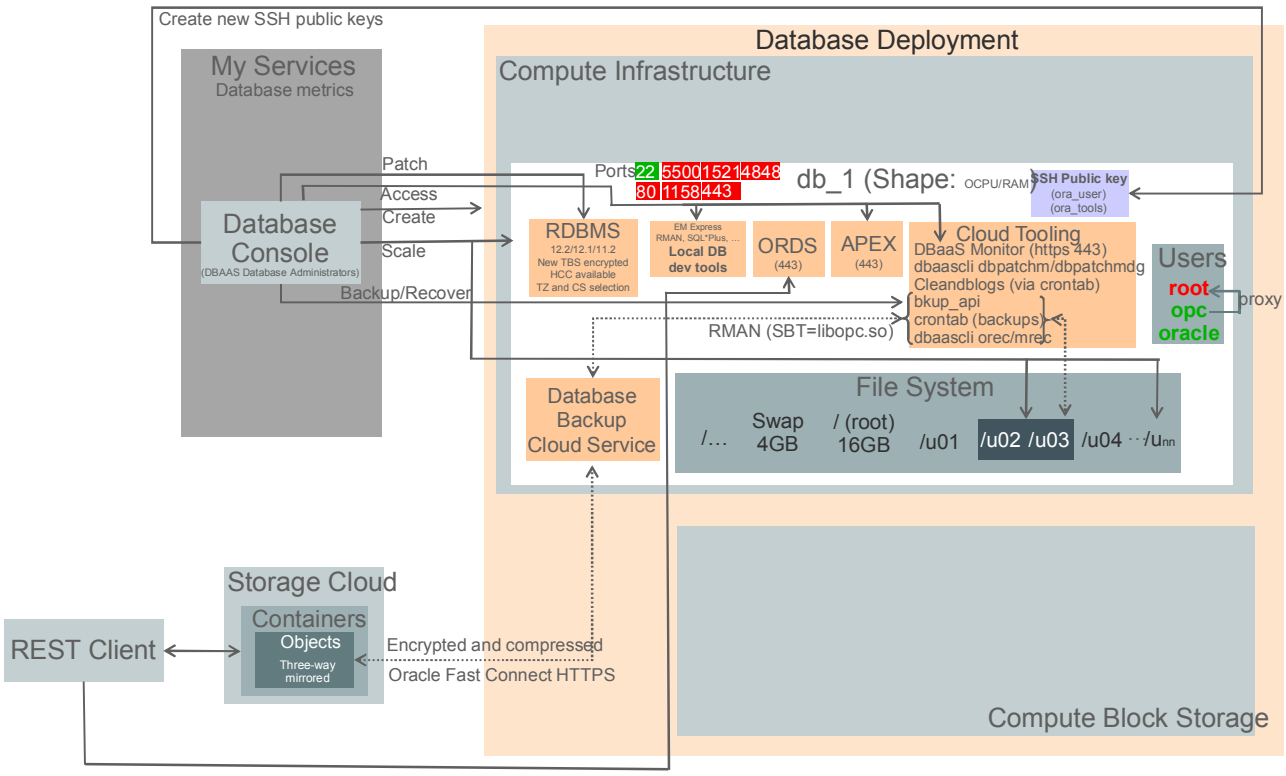
This container will then be used automatically by your backup jobs to create object storage inside your container to store your database backkups up to 30 days.

Note that all generated backups are encrypted on both local and cloud storage.

The special SBT library libopc.so is automatically used and configured for RMAN to access Cloud Storage through Cloud Tooling.

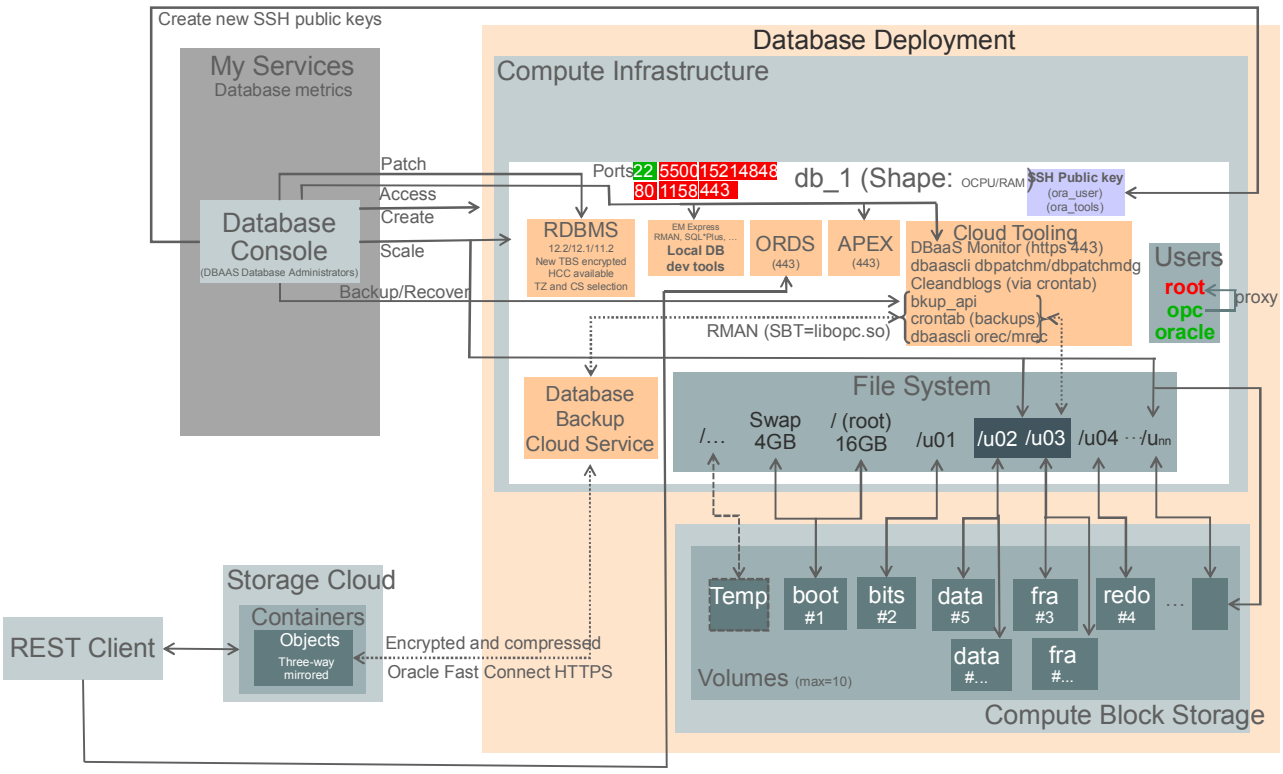


You can also use your REST client tool to access ORDS to manage your database deployment using REST APIs.



Looking in more details, your DBaaS instance is comprised of a VM allocate from the Compute Infrastructure and an underlying block storage from where its filesystem is created.

This additional resource is part of what is called Compute Block Storage.



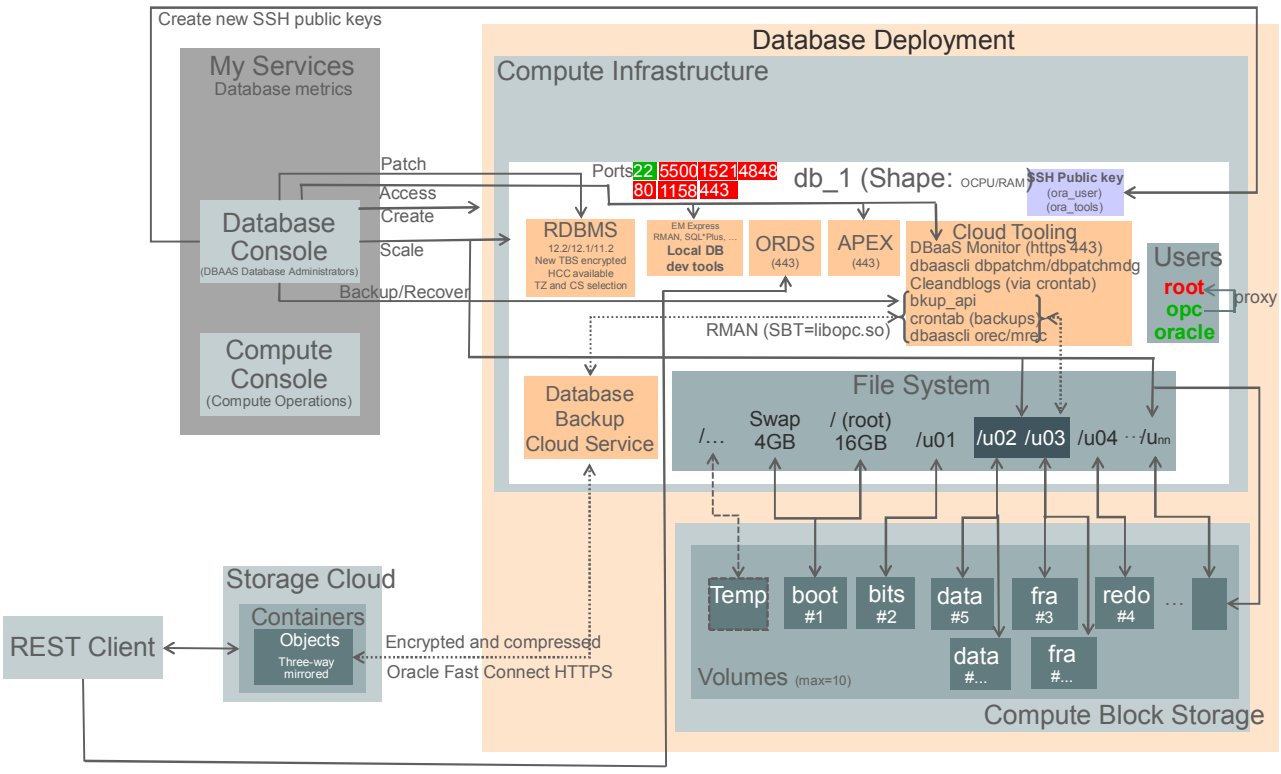
The goal of the Compute Block Storage layer is to allocate volumes you can attach to your VM.

A maximum of 10 volumes can be attached to a single VM, and by default, when you create your DBaaS instance, five volumes are created:

boot which is 20GB, bits 30GB, data depending on the size you specified for your data, fra which is by default 1.7 times the data size if you are using backups, and redo which is 10GB.

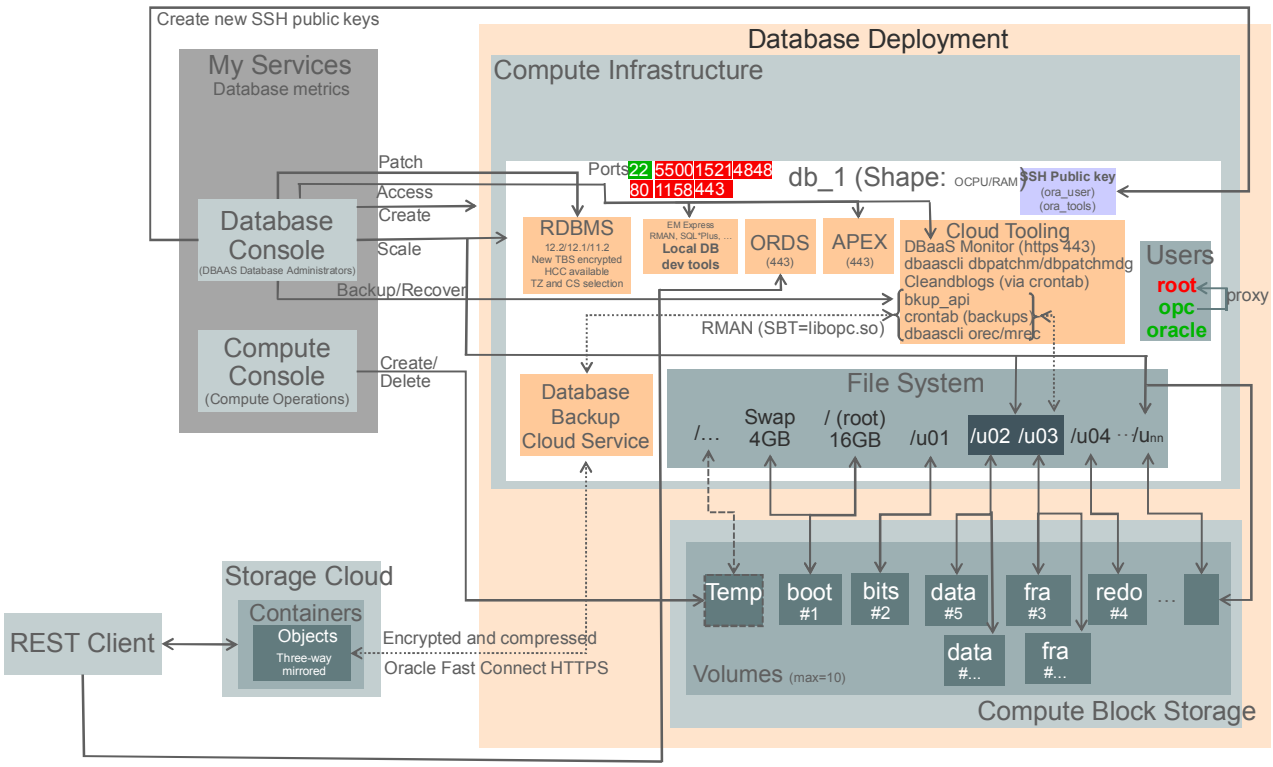
These volumes are automatically attached to your VM, and then formatted and mounted appropriately.

When scaling storage from your database console, you can add new volumes that will be mounted in new directories, or you can extend your data and fra storage by adding new volumes to /u02 or /u03.

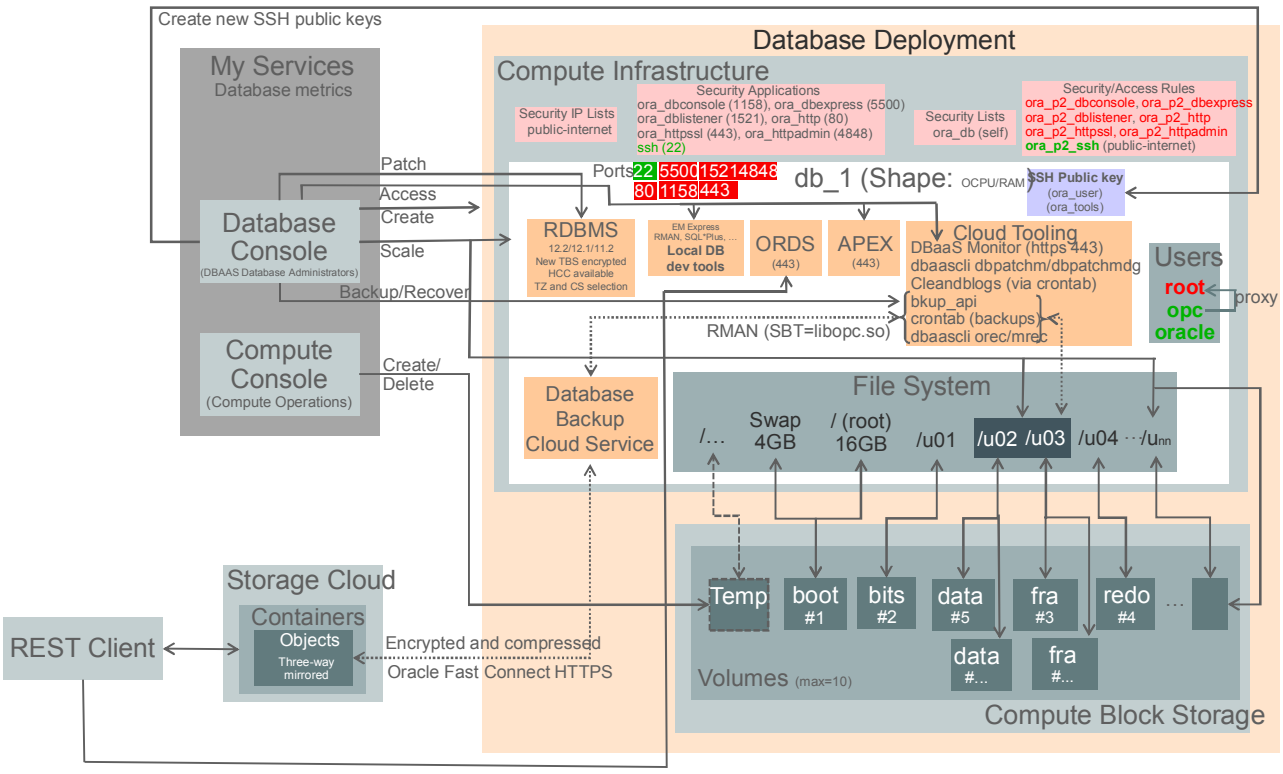


For some of the Compute resources like volumes and network controls, you can manage them directly from another console called the Compute Console.

You can access this console as long as you have enough privileges like the Compute Operations role.



From the Compute Console, you can add volumes you will mount to your filesystem after creation. This is considered a temporary need as the volume is not automatically attached to your file system.



For network controls, you can manage Security Rules that control ports access to your VM by other computers on the Internet.

As shown on the diagram a number of Security Rules are pre-defined to control access to specific ports used by various Oracle software like EM Express, database listener, Database Control, or important protocols like HTTP, HTTS, and SSH.

By default all those pre-defined Security Rules are disabled except the SSH one allowing any computer to communicate through SSH.

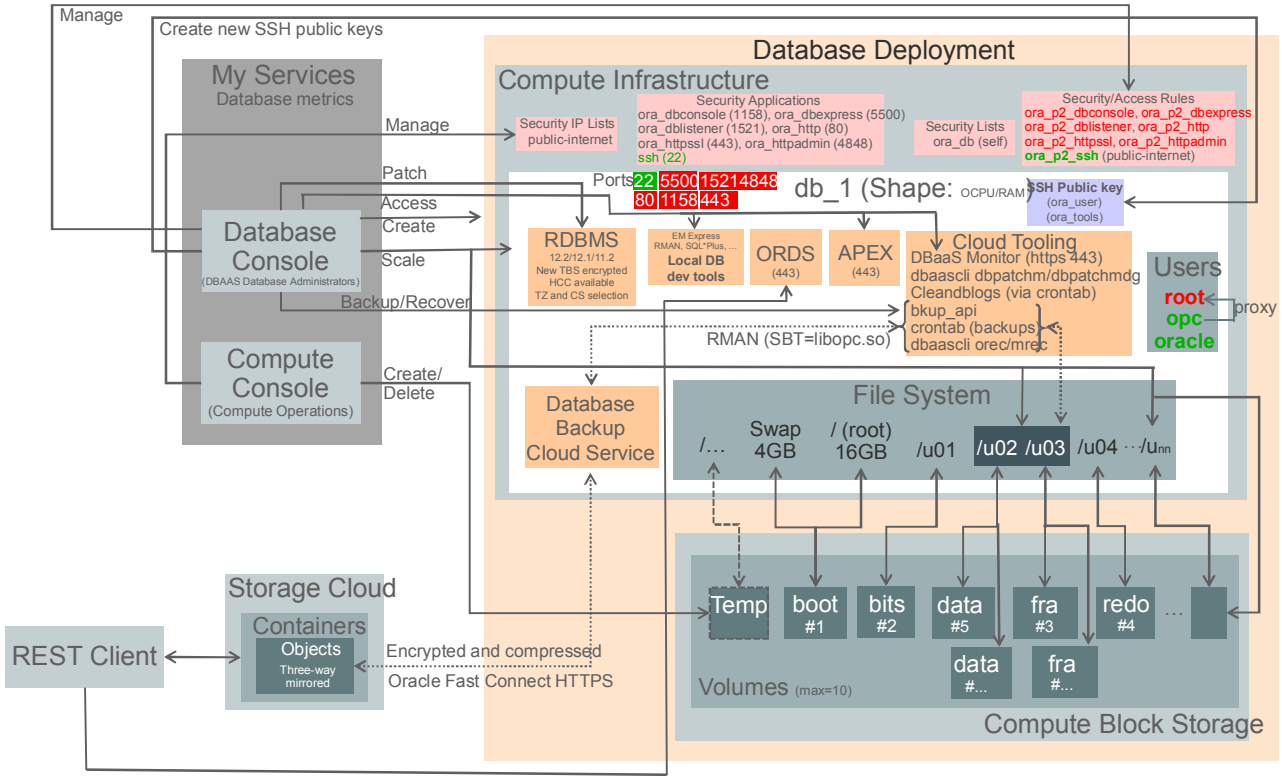
You can define Security IP Lists to group a set of computers outside of Oracle Public Cloud used in your Security Rules.

Security Lists are groups of Oracle Public Cloud VMs between which communication is allowed or not.

With these three entities, you can construct efficient firewalls between your service VMs and the outside world.

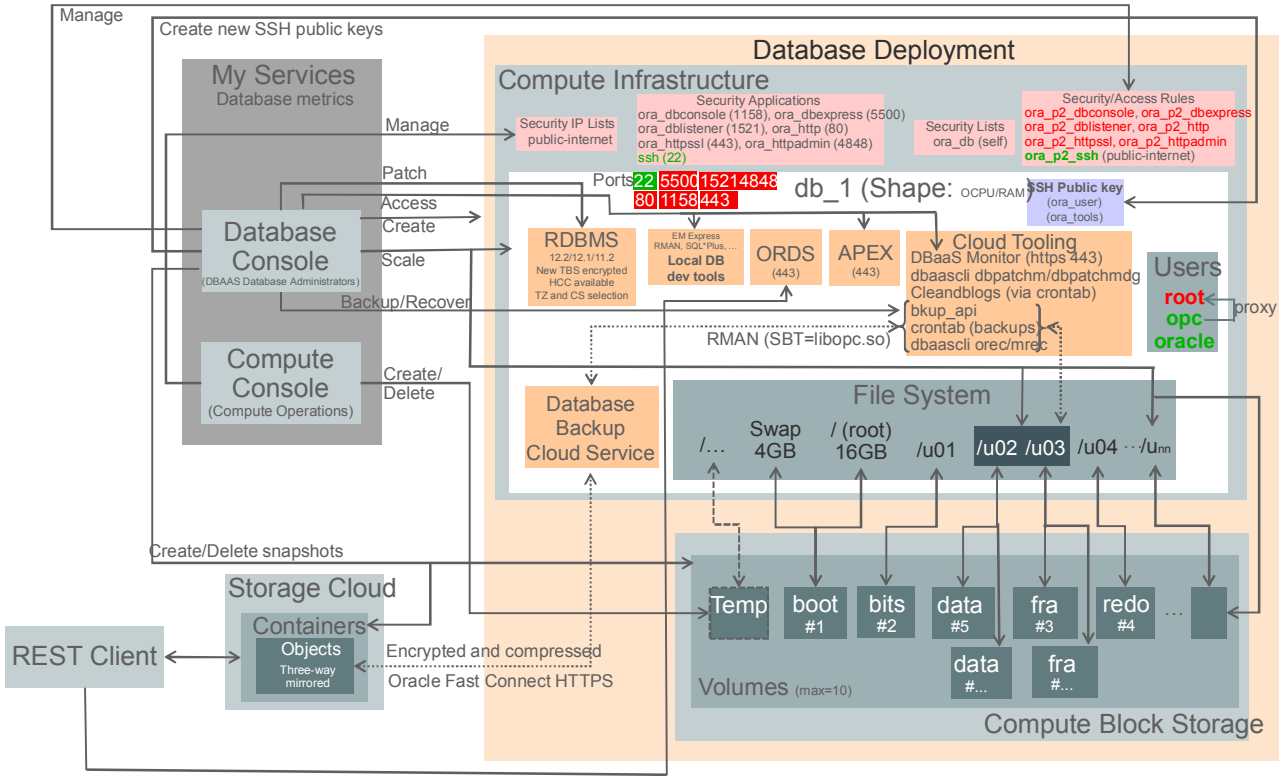
When you create a security rule, you can specify a security list or security IP list as a source and a security list as the destination in that security rule.

You also specify the security application (protocol), and you enable it or not.



From the Database Console, you can manage Security Rules (also called Access Rules).

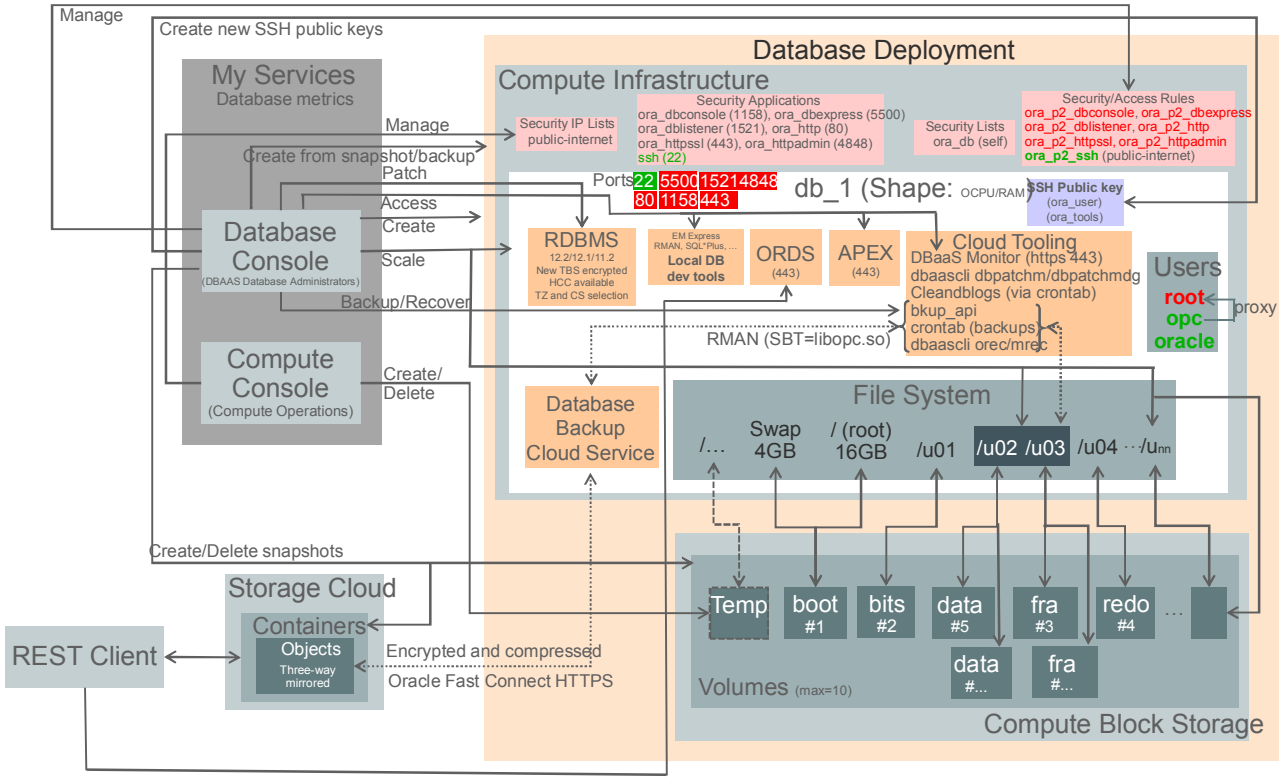
From your Compute Console, you can manage all network controls.



From the Database Console, you can create storage snapshots, which you can then use to create new database deployments called “linked clones”.

When you create a storage snapshot, the database deployment is put into maintenance status and a snapshot of all the storage volumes for the deployment is taken to your Storage Cloud Container.

Using the “copy on write” technology that Oracle Compute Cloud Service supports for storage volume snapshots, the file data on the linked-clone deployment can change without changing the snapshot itself. Thus, you can create several linked clones from the same snapshot to use for application testing or branched application development work.



You can create an DBCS Database deployment from a snapshot you have taken of another database deployment in the same identity domain.

The created deployment is called a “linked clone” because its storage is linked to the snapshot’s storage.

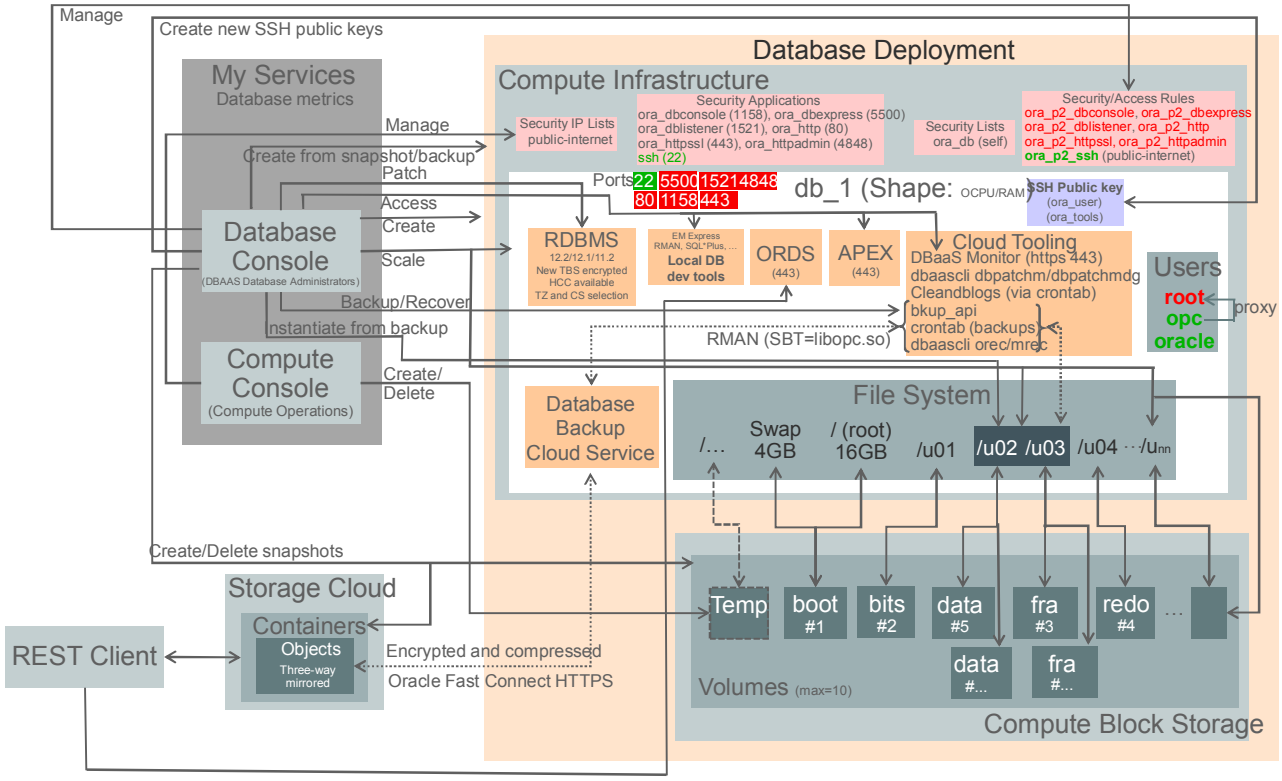
As you step through the creation wizard, you will note that several options are not selectable; for example, Service Level, Software Release and Software Edition.

Such options are not selectable because their values are determined from the snapshot upon which the linked-clone is based.

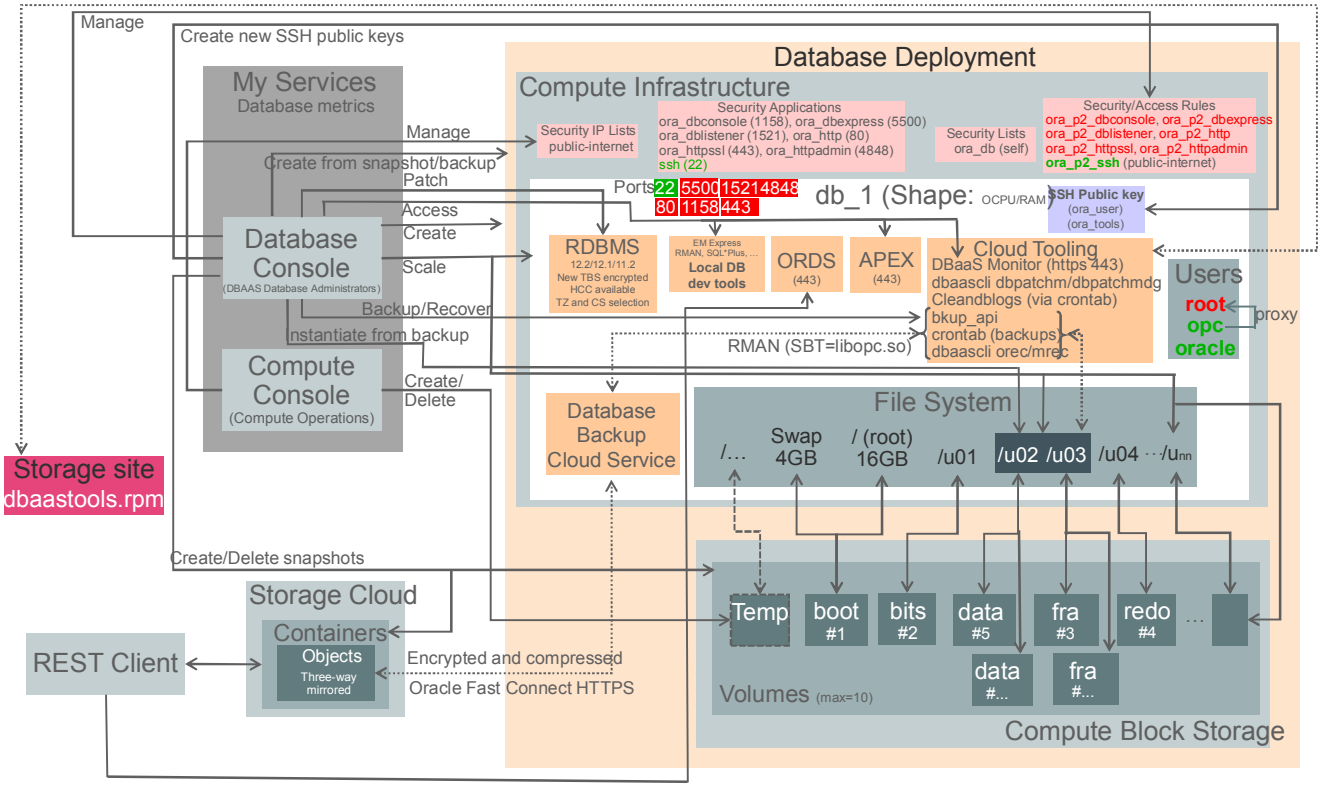
Other options are required like a new service name, and an SSH public key.

You can change the other selectable options from their defaults if you want to; for example, Shape and Backup Destination.

You can also use the Oracle Database Cloud Service creation wizard to perform an instantiate-from-backup operation as it creates a database deployment, provided that database recovered from the backup will be smaller than 2 TB (1.2 TB if you plan to back up the new deployment to both cloud and local storage).



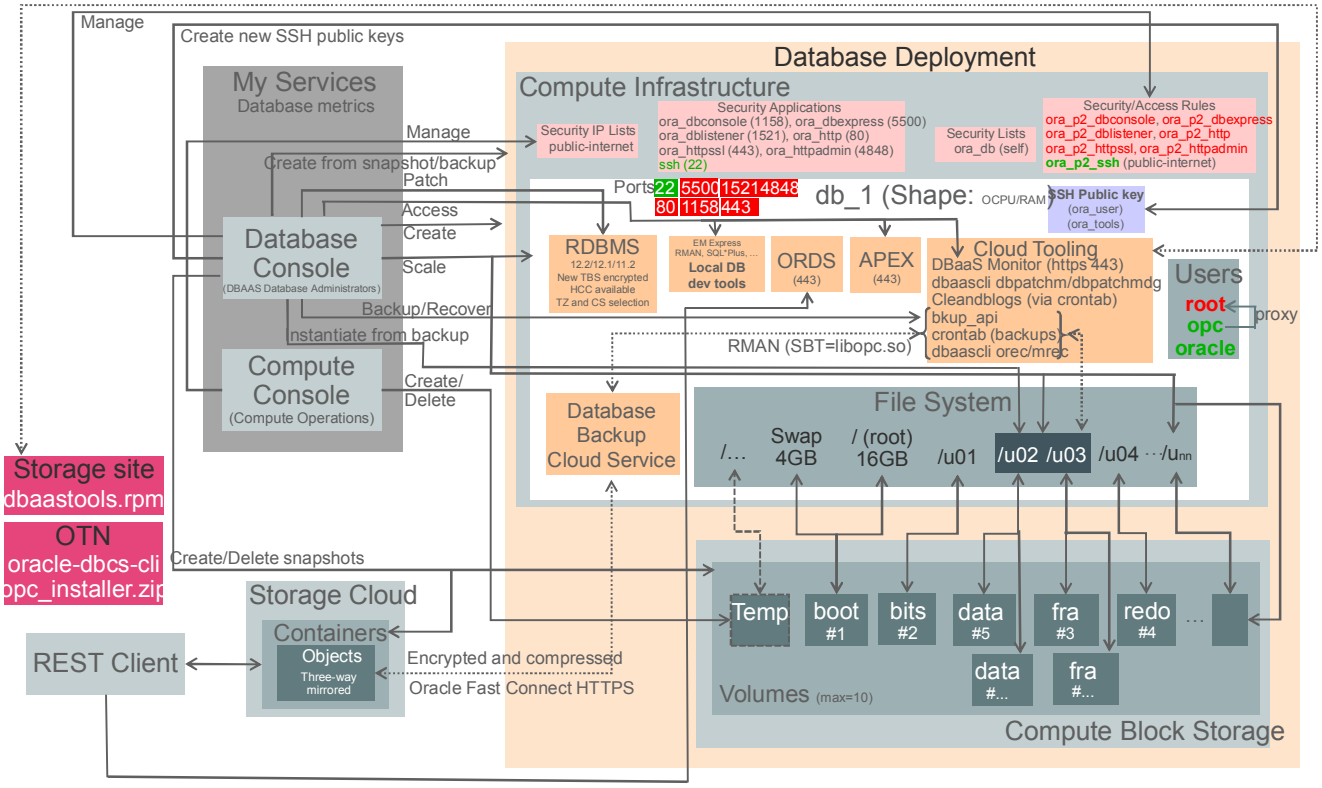
You create a Database Cloud Service database deployment hosting a single-instance database and then you replace the newly created database using another database's backup to Oracle Database Backup Cloud Service. This technique is often called "instantiate from backup" and the database from which the backup was made is called the "source database".



From time to time, you have the possibility to update your Cloud Tooling to take advantage of the latest functionalities.

This operation is as simple as updating the dbaastools.rpm file you can download to your VM from the special storage site using wget command.

You can also use the dbpatchm subcommand of the dbaascli utility to update the cloud tooling.



The last thing I'd like to discuss about this DBaaS architecture diagram is the possibility you have to install the oracle-dbcs-cli utility directly on your Linux on-premises servers.

You can download this utility from OTN, and you install it by unziping it.

This utility allows you to connect to Oracle Cloud and perform a variety of life-cycle and administration operations on your DBaaS instances.

In addition, you have the possibility to backup your on-premises databases to Oracle Database Backup Cloud Service.

To do that, you first need to install the Oracle Database Cloud Backup Module.

You'll download the opc_installer.zip module from Oracle Technology Network (OTN) and install it on your database server.

Major Differences - 1

Operation type	On-Premises database	Cloud database
Backed up files	Database files + controlfiles + SPFILE	All database files + SPFILE + password file and others from /home/oracle/bkup/dbcfg.spec and /home/oracle/bkup/oscfg.spec files
Backup destination	Single or Cloud	•Local compute node storage •Oracle Storage Cloud Service container
Backups	Manual or manual scheduling: RMAN> backup	Automatic: bkup_api
Installation	Manual •Oracle Database 11g or 12c •Database creation	Automatic •Oracle Database 11g or 12c •Pre-created database
Location for database files and backups	Manual	Automatic
Log and diagnostics files cleanup	None	Automatic, using a configuration file
Monitoring tools	EM Express, EM Cloud Control, SQL Developer	DBaaS Monitor, EM Express, EM Cloud Control, SQL Developer
Oracle Database 12c	Non-CDBs and CDBs	Only CDBs
Patch discovery	•None •Oracle Support •EM Cloud Control	GUI tool: Oracle Database Cloud Service console

The table in the slide lists the main differences between on-premises databases and Cloud database deployments.

Major Differences - 2

Operation type	On-Premises database	Cloud database
Port access	Automatic configuration via dbca •EM Express •EM Cloud Control •Listener registration	Automatic configuration via pre-defined security rules to enable when required
Recovery	RMAN> `recover`	`dbaascli orec`
Storage allocation	Manual Unix commands	GUI tool: Oracle Database Cloud Service console
Tablespace encryption	None by default	Default encryption (TDE) for user-defined tablespaces: Initialization parameter `encrypt_new_tablespaces = cloud_only`
Types of server connection	All types (password, SSH ...)	SSH
Upgrade	GUI tool: dbua	None
User and group	`oracle` user and `oinstall` group	`oracle` and `opc` users, and `oinstall` group

The table in the slide lists the main difference sbetween on-premises databases and Cloud database deployments.

Migration Methods for On-Premises Databases to Cloud - 1

Method	On-Premises 11g database to Cloud 11g database	On-Premises 11g database to Cloud 12c PDB	On-Premises 12c non-CDB to Cloud 12c PDB	On-Premises 12c PDBs to Cloud 12c PDB
Data Pump Conventional Export/Import	Y	Y	Y	Y
Data Pump Transportable Tablespace	Y	Y	Y	Y
Data Pump Full Transportable	N	Y	Y	Y
RMAN Transportable Tablespace with Data Pump	Y	Y	Y	Y
RMAN CONVERT Transportable Tablespace with Data Pump	Y	Y	Y	Y
RMAN Cross-Platform Transportable Tablespace Backup Sets	N	N	Y	Y
RMAN Cross-Platform Transportable PDB	N	N	N	Y

Some of the characteristics and factors to consider when choosing a migration method are:

- On-premises database version
- Oracle Database Cloud database version
- On-premises host operating system and version
- On-premises database character set
- Quantity of data, including indexes
- Data types used in the on-premises database
- Storage for data staging
- Acceptable length of system outage
- Network bandwidth

To determine which migration methods are applicable to your migration scenario, gather the following information:

- Database version of your on-premises database
- For on-premises Oracle Database 12*c* Release 1 databases, the architecture of the database (multitenant or non-CDB)
- Endian format (byte ordering) of your on-premises database's host platform
- Database character set of your on-premises database and your Database Cloud Service database
- Database version of your Database Cloud Service database

Migration Methods for On-Premises Databases to Cloud - 2

Method	On-Premises 11g database to Cloud 11g database	On-Premises 11g database to Cloud 12c PDB	On-Premises 12c non-CDB to Cloud 12c PDB	On-Premises 12c PDBs to Cloud 12c PDB
Unplugging/Plugging	N	N	Y	Y
Remote Cloning	N	N	Y	Y
SQL Developer and SQL*Loader to Migrate Selected Objects	N	N	Y	Y
SQL Developer and INSERT Statements to Migrate Selected Objects	N	N	Y	Y

The following methods are applicable when migrating an Oracle Database 11*g* or 12*c* on-premises databases to an Oracle Database 11*g* or 12*c* database in the Cloud.

See *Migrating from Oracle Database 11g to Oracle Database 11g in the Cloud (http://www.oracle.com/pls/topic/lookup?ctx=cloud&id=CSDBI-GUID-24C35B32-0AB1-4602-A2B1-E298516835C6)* for additional information.

See *Migrating from Oracle Database 11g to Oracle Database 12c in the Cloud* (*http://www.oracle.com/pls/topic/lookup?ctx=cloud&id=CSDBI-GUID-811C8688-EBA5-454A-A35B-8F06DCC4D271*) for additional information.

See *Migrating from Oracle Database 12c Non-CDB to Oracle Database 12c in the Cloud* (*http://www.oracle.com/pls/topic/lookup?ctx=cloud&id=CSDBI-GUID-FC3576DD-5DA8-4204-9B2C-6F1399B55DB2*) for additional information.

See *Migrating from Oracle Database 12c CDB to Oracle Database 12c in the Cloud* (*http://www.oracle.com/pls/topic/lookup?ctx=cloud&id=CSDBI-GUID-EE77DD02-C036-469E-BB41-BDFF166D450B*) for additional information.

Summary

In this lesson, you should have learned how to:

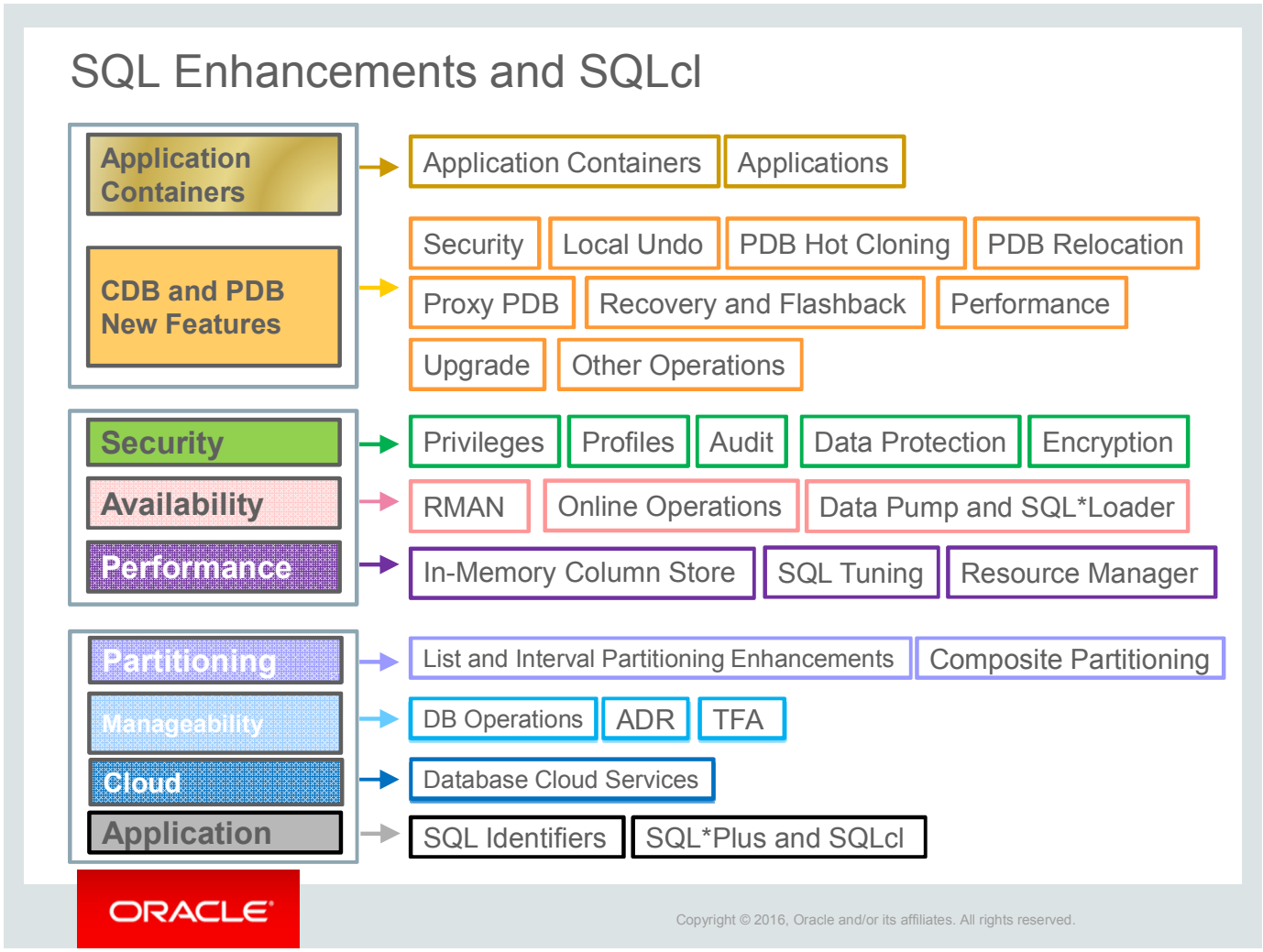
- Describe the main services of Oracle Database Cloud Services (DBCS)
- Describe the components of the DBCS and their interactions
- Define migration methods for on-premises databases to Cloud

Practice 22: Overview

- 22-1: Creating a database deployment.
- 22-2: Connecting to the database deployment compute node.
- 22-3: Performing an on-demand backup.
- 22-4: Monitoring your database deployment by using DBaaS Monitor.

23

SQL and SQLcl



This lesson describes new commands in SQL*Plus and the commands of the new SQLcl utility. It also describes SQL identifier new lengths and explains the purpose of the new `VALIDATE_CONVERSION` function.

Objectives

After completing this lesson, you should be able to:

- Recall SQL*Plus commands in the same session from memory
- Use the commands of the new SQLcl utility
- Describe SQL identifier new lengths
- Explain the new `VALIDATE_CONVERSION` function

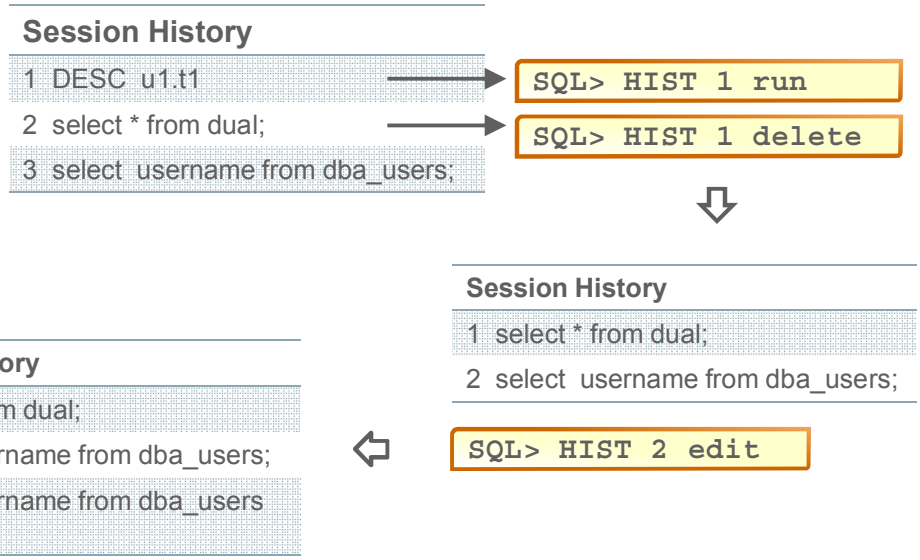
To get detailed information about how to perform any of the operations explained in this lesson, refer to the following guides in the Oracle documentation:

- *SQL*Plus User's Guide and Reference Release 2 (12.2)*
- *SQLcl – The New SQL*Plus*
- *Video: Oracle SQL Developer Meets SQL*Plus*
- *Oracle SQLcl Slidedeck: Overview of our new command line interface*
- *The Modern Command Line*
- *SQLcl: The new challenger for the SQL*Plus crown*
- *Kris' blog*

SQL*Plus History Command

Run, edit, or delete previously used commands from the history list in the current session.

- SQL*Plus
- SQL
- PL/SQL



SQL*Plus now provides the ability to reissue commands that have been previously executed in your session from the time the history is activated.

This functionality is similar to the shell history command available on UNIX platform command line shells.

The history is not activated by default in a new session. Use the `SET HISTORY ON` command to enable history. Then select the line number in the command history list to either re-execute the command, edit the command, or delete the command from the history list.

SQLcl Utility

SQLcl is a new Java-based command-line interface based on the script engine in Oracle SQL Developer for Oracle Database.

- Delivers a more modern way of working on the command line
- Introduces new commands and features missing from SQL*Plus
  - Inline editor
  - SQL execution history and recall
  - Query result formatting
  - User-friendly text editing
  - Simplified commands for DBAs

To install SQLcl, download SQLcl from the Oracle SQL Developer product page on the Oracle Technology Network.

SQLcl supports connections via EZConnect, TNS, LDAP, TWO_TASK, and more—and all without an Oracle client installed or configured.

SQLcl Inline Editor

- Use arrow keys to move:
  - Back ⇦
  - Up ⇧
  - Down ⇩
  - All around your text directly at the cursor ⇨
- Use quick navigation keys:
  - **Ctrl+ W** Go to the top of the buffer.
  - **Ctrl+ A** Go to the start of a line.
  - **Ctrl+ E** Go to the end of a line.
  - **Ctrl+ S** Go to the bottom of the buffer.

In `SQLcl`, you can now use the arrow keys to move back, up, down, and all around the text directly at the cursor and can also hold down the Backspace key to delete the query beyond the current line of your buffer.

As you arrow up through the text, the current line is marked with an asterisk by the line number. When your edits are complete, you can use the Ctrl+R key sequence to execute the entire text.

This history keeps only the statements that execute successfully.

SQLcl History

```
SQLcl> history
```

Session1 user1 History

```
1  DESC  u1.t1
2  select * from dual;
3  select  username from dba_users;
```

```
SQLcl> exit
```

Session3 user1 History

```
1  select * from dual;
2  select  username from dba_users;
3  truncate table u1.t1 ;
...
99 update a  set c1=c1*2;
100 insert into a values (1);
```

```
SQL> h 2
1*  select  username from dba_users;
SQL> r 2
1*  select  username from dba_users;
USERNAME
------------------------------
SYS
...
SQL>
```


History is enabled by default in `SQLcl` sessions. `SQLcl` stores the previous 100 statements or scripts.

Additionally, the query history is maintained from one session to the next, and older queries age out as the history limit of 100 entries is met. The query history is accessible with the `history` command and also by using up and down arrow keys.

In addition to the `history` command, the following commands provide more information and operations:

- `history usage`: View the history usage for each command. If a command is often used, you can create aliases.
- `history time`: View the time spent executing each statement.
- `history clear`: Clear the history.

As you use the arrow keys to navigate the history list, `SQLcl` paints the text of the query at the command prompt. After you have recalled a statement, you can also edit it by using the arrow keys.

SQLcl Formatting

```
SQL> help set sqlformat
SET SQLFORMAT
  SET SQLFORMAT {csv, html ,xml, json, ansiconsole, insert, loader,
                 fixed, default}
```

```
SQL> SET SQLFORMAT json
SQL>
SQL> select * from hr.employees fetch first 1 rows only;
{"items":[ {"employee_id":198,"first_name":"Donald","last_
name":"OConnell","email":"DOCONNEL","phone_number":"650.50
7.9833","hire_date":"21-JUN-
99","job_id":"SH_CLERK","salary":2600,"manager_id":124,"de
partment_id":50}]}
```

In SQL*Plus, you write custom code to transform your query output into comma- or tab-delimited format.

In SQLcl, you can return your query output in the desired format by using the `SET SQLFORMAT` command. The formats supported in SQLcl include the ones listed by:

```
SQL> help set sqlformat
SET SQLFORMAT
  SET SQLFORMAT {
csv,html,xml,json,ansiconsole,insert,loader,fixed,default}
```

When you define a format, all query output will be formatted as requested. To disable the formatting, run the `SET SQLFORMAT` command without a format.

SQLcl Commands

- ALIAS
- CD
- CTAS
- DDL
- FORMAT
- INFO
- LOAD
- NET
- REPEAT
- SSHTUNNEL
- TNSPING

```
SQL> help ctas
```

```
SQL> alias myname=show user;
SQL> myname
USER is "SYSTEM »
```

```
SQL> DDL sh.sales
CREATE TABLE "SH"."SALES"
   (       "PROD_ID" NUMBER NOT NULL ENABLE,
           "CUST_ID" NUMBER NOT NULL ENABLE, …
PARTITION "SALES_Q4_2003"  …
TABLESPACE "USERS" ) ;
ALTER INDEX "SH"."SALES_TIME_BIX"  MODIFY PARTITION
"YEAR_1998" UNUSABLE;
SQL>
```

```
SQL> INFO sh.sales
TABLE: SALES
            LAST ANALYZED:2016-05-17 22:02:30.0
            ROWS          :741745
            SAMPLE SIZE   :741745
            INMEMORY      :
…
Columns
NAME          DATA TYPE     NULL  DEFAULT COMMENTS
 PROD_ID      NUMBER        No              FK to products
AMOUNT_SOLD   NUMBER(10,2)  No              invoiced amt
Indexes
INDEX_NAME            UNIQUENESS  STATUS
SQL>
```

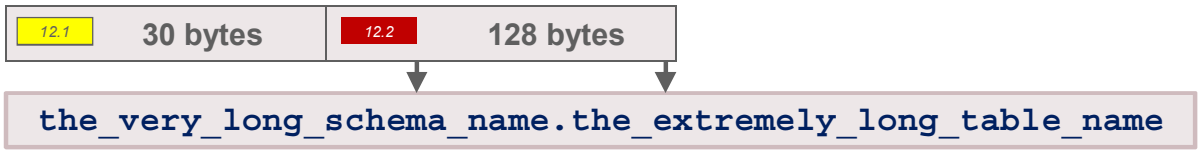

Several new SQLcl features and commands extend what is available in SQL*Plus.

The help command explains what each of the commands can do.

```
SQL> help ctas
CTAS
 ctas table new_table
Uses DBMS_METADATA to extract the DDL for the existing table
 Then modifies that into a create table as select * from
SQL> help repeat
repeat <iterations> <sleep>
   Repeats the current sql in the buffer the specified times with
   sleep intervals
   Maximum sleep is 120s
```

To get the full list of commands, use the help command – ALIAS, ARCHIVELOG, CD, COLUMN, COMPUTE, COPY, DEFINE, DEL, DESCRIBE, EDIT, EXECUTE, GET, HOST, INPUT, LIST, NOHISTORY, OERR, PASSWORD, PRINT, PROMPT, RESERVED WORDS, RUN, SAVE, SCRIPT, SET, SHOW, SHUTDOWN, SODA, SPOOL, START, STARTUP, STORE, TIMING, TTITLE, UNDEFINE, VARIABLE, WHENEVER, XQUERY.

SQL Identifier Length



- SQL*Plus formats the output based on the information returned from the server.

```
SQL> CREATE TABLE my_table_with_name_longer_than_30_bytes (
      column_with_name_longer_than_30_bytes_even_more_xxxxxxxx
      CHAR(10));

Table created.

SQL> DESC my_table_with_name_longer_than_30_bytes
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------
 COLUMN_WITH_NAME_LONGER_THAN_30_BYTES_EV           CHAR(10)
 EN_MORE_XXXXXXXX

SQL> CREATE USER my_preferred_user_with_a_very_long_name … ;
```


Increasing the SQL identifier limit eases migration from third-party databases to Oracle.

Although database migration scenarios rarely feature databases with tables or views with identifiers approaching 128 characters, multibyte database migration scenarios where table or view names exceed 30 bytes are now fairly common.

The length limit refers strictly to the length of each part of a multipart name like schema.table.column. Thus, in an example like schema1.table1.column1, the user, table, and column name can all be 128 bytes long, and the full multipart string would be 386 bytes long. If each part is quoted, there would be an additional two bytes for the double-quote for each part, so the full multipart string would be 392 bytes long.

The maximum length of identifiers can be applied to tables, table partitions, table subpartitions, views, materialized views, editions, sequences, constraints, synonyms, dimensions, clusters, SQL built-ins, indexes, indextypes, index partition, index subpartitions, and SQL operators, as well as to those defined by PL/SQL, procedures, functions, packages, package bodies, triggers, libraries, assemblies, types, and type bodies.

Database parameter names remain at 80 bytes.

Exceptions: Resource Manager plans, directives, consumer groups, ASM diskgroups, PDBs, and tablespaces

VALIDATE_CONVERSION Function

- Determines whether a given input can be converted to a given data type
- Helps identify column values that cannot be converted

```
SQL> SELECT * FROM tab1 WHERE VALIDATE_CONVERSION(tab1.c1 AS NUMBER) = 0;
```

Tab1.c1	Conversion possible?	Converted value
'1'	Yes: VALIDATE_CONVERSION(tab1.c1 AS NUMBER) = 1	1
'A'	No: VALIDATE_CONVERSION(tab1.c1 AS NUMBER) = 0	0
'0'	Yes: VALIDATE_CONVERSION(tab1.c1 AS NUMBER) = 1	1
''	No: VALIDATE_CONVERSION(tab1.c1 AS NUMBER) = 0	0

Given an expression and data type name, this new VALIDATE_CONVERSION SQL function returns 1 if the expression can be converted to the specified data type, or 0 otherwise. If the expression is a corrupted numeric value, this function also returns 0.

Using this function, it is possible to identify rows in a table where there is a conversion failure for a particular column.

The data type names that can be evaluated by the function are:

- NUMBER
- BINARY_FLOAT
- BINARY_DOUBLE
- DATE
- TIMESTAMP
- TIMESTAMP WITH TIMEZONE
- INTERVAL DAY TO SECOND
- INTERVAL YEAR TO MONTH

Summary

In this lesson, you should have learned how to:

- Recall SQL*Plus commands in the same session from memory
- Use the commands of the new SQLcl utility
- Describe SQL identifier new lengths
- Explain the new `VALIDATE_CONVERSION` function

Practice 23: Overview

- 23-1: Using the SQL*Plus `HISTORY` command and SQLcl utility
- 23-2: Using the new `VALIDATE_CONVERSION` function to identify rows in a table where there is a conversion failure

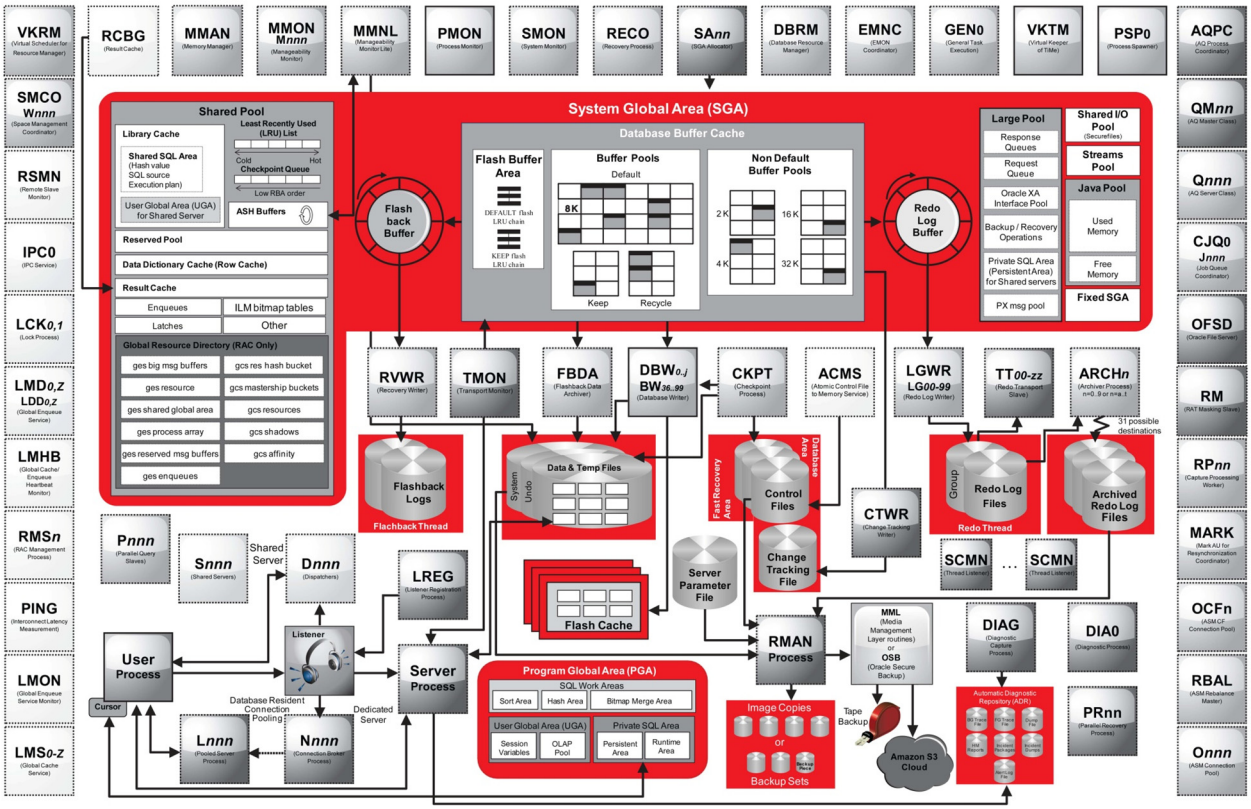
A

New Processes, Views, Parameters, Packages, and Privileges

For a complete list of processes, views, parameters, packages, and privileges on Oracle Database 12*c* new features, refer to the following guides in the Oracle documentation:

- *Oracle Database Reference 12c Release 1 (12.1)*
- *Oracle Database PL/SQL Packages and Types Reference 12c Release 1 (12.1)*
- *Oracle Database Security Guide 12c Release 1 (12.1)*

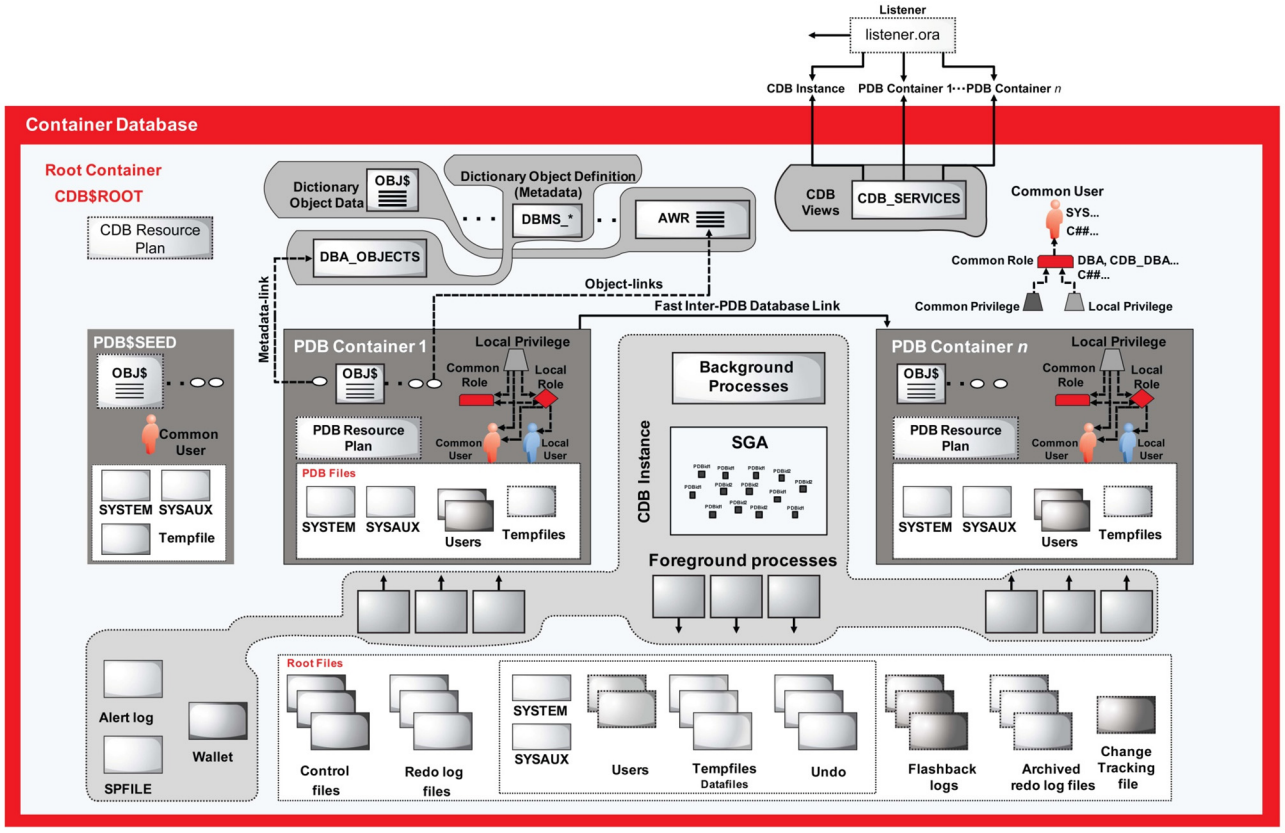
Instance and Database



Captions:

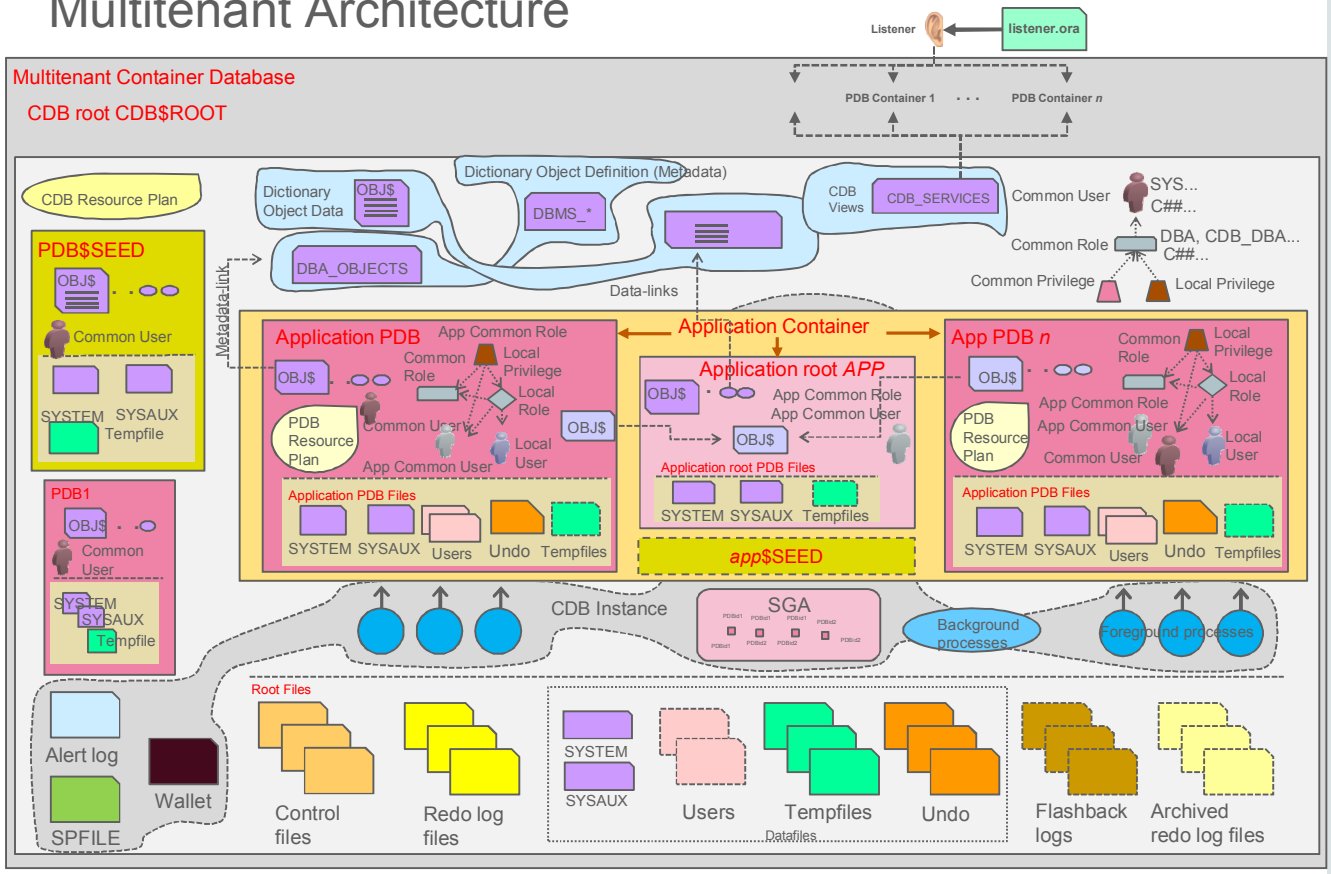
1. Circle elements represent Oracle processes. If they are surrounded by dotted lines (equal dotted elements), they can run either as threads or OS processes. If they are surrounded by a solid line, they can run as only OS processes. The SCMN case is an exception. When using the multiprocess, multithreaded architecture, each OS process running more than one Oracle process also runs a special thread called SCMN, that is basically an internal listener thread. All thread creation is routed through this thread.
2. Darker circle elements are new elements for DB 12*c*.
3. The two main different color nuances for circle elements are to make the distinction between RAC and non-RAC processes.
4. Files are represented by cylinders.
5. Storage location for those files is divided into three main areas: Fast Recover Area, Database Area, and Automatic Diagnostic Repository. Areas are designated by background rectangles by using three different colors. The exception is the server parameter file. If you find a file type part of two areas, it means that some corresponding files can be in both areas.
6. Inside one circle element, you may see two names. This is to indicate that the second (smaller letters) is a slave of the first.

Multitenant Architecture



Multitenant Architecture

12.2.0.1



CDB and PDB New Views and New Columns

- DBA_APPLICATIONS: Application created on top of application PDBs
- Other views related to applications:
  - DBA_APP_PATCHES
  - DBA_APP_STATEMENTS
  - DBA_APP_ERRORS
  - DBA_APP_VERSIONS
- New columns in V$CONTAINERS view:
  - APPLICATION_ROOT = YES|NO
  - APPLICATION_PDB = YES|NO
  - APPLICATION_SEED = YES|NO
  - APPLICATION_ROOT_CON_ID = <*con_id*>
  - APPLICATION_ROOT_CLONE = YES|NO
  - PROXY_PDB = YES|NO
  - PDB_COUNT = <*nbr*>
- New columns in V$PDBS view:
  - PDB_COUNT = <*nbr*>

CDB and PDB New Views and New Columns

- New columns or column values in CDB_PDBS view:
  - UPGRADE_PRIORITY = *<nbr>*
  - APPLICATION_ROOT = YES|NO
  - APPLICATION_PDB = YES|NO
  - APPLICATION_SEED = YES|NO
  - APPLICATION_ROOT_CON_ID = *<con_id>*
  - IS_PROXY_PDB = YES|NO
  - STATUS = RELOCATING | RELOCATED | REFRESHING
- New columns in DBA_TABLES view:
  - CONTAINER_MAP_OBJECT = YES|NO
  - CONTAINERS_DEFAULT = YES|NO
  - CONTAINER_MAP = YES|NO

CDB and PDB New Views and New Columns

- New values in `CDB_OBJECTS` view, column `SHARING = METADATA LINK | DATA LINK | EXTENDED DATA LINK | NONE`
- New column `INHERITED = YES|NO` in views:
  - `CDB_USERS`
  - `CDB_ROLES`
  - `CDB_TAB_PRIVS`
  - `CDB_SYS_PRIVS`
  - `CDB_ROLE_PRIVS`
  - `ROLE_ROLE_PRIVS`
  - `ROLE_SYS_PRIVS`
  - `ROLE_TAB_PRIVS`
  - `AUDIT_UNIFIED_POLICIES`
  - `DVSYS.DBA_DV_REALM`
  - `DVSYS.DBA_DV_REALM_OBJECT`

CDB and PDB New Views and New Columns

- New columns in DATABASE_PROPERTIES view:
  - PROPERTY_NAME = CONTAINER_MAP
  - LOCAL_UNDO_ENABLED = TRUE|FALSE
  - CONTAINERS_HOST = *<hostname>*
  - CONTAINERS_PORT = 1521
- CDB_LOCKDOWN_PROFILES
- V$RESTORE_POINT
- AWR_ROOT_PDB_IN_SNAP
- AWR_PDB_PARAMETER
- AWR_PDB_RSRC_PDB_METRIC
- AWR_PDB_SQL_SUMMARY
- V$RSRCPDBMETRIC/V$RSRCPDBMETRIC_HISTORY
- V$RSRC_PDB/V$RSRC_PDB_HISTORY
- DBA_HIST_RSRC_PDB_METRIC

CDB and PDB New Parameters and Packages

- Parameters:
  - DB_PERFORMANCE_PROFILE
  - PDB_LOCKDOWN
  - SGA_MIN_SIZE
  - MAX_PDBS
- Package: DVSYS.DBMS_MACADM.CREATE_REALM
  - REALM_SCOPE =>
    - DBMS_MACUTL.G_SCOPE_COMMON
    - DBMS_MACUTL.G_SCOPE_LOCAL
  - ENABLED =>
    - DBMS_MACUTL.G_SIMULATION
    - DBMS_MACUTL.G_YES
    - DBMS_MACUTL.G_NO

Security

- New column in UNIFIED_AUDIT_TRAIL view: RLS_INFO
- New columns in AUDIT_UNIFIED_ENABLED_POLICIES:
  - ENABLED_OPTION
  - ENTITY_NAME
  - ENTITY_TYPE
- New column values in DBA_PROFILES.RESOURCE_NAME = INACTIVE_ACCOUNT_TIME
- New columns in V$PWFILE_USERS:
  - ACCOUNT_STATUS
  - PASSWORD_PROFILE
  - LAST_LOGIN
  - LOCK_DATE
  - EXPIRY_DATE
  - EXTERNAL_NAME
  - AUTHENTICATION_TYPE
  - COMMON

Security: Privilege Analysis and Database Vault

- Privileges:
  - SYSRAC
  - INHERIT REMOTE PRIVILEGES
- New RUN_NAME column in Privilege Analysis views:
  - DBA_UNUSED_GRANTS
  - DBA_USED_OBJPRIVS_PATH
- Package DBMS_PRIVILEGE_CAPTURE:
  - ENABLE_CAPTURE(…, run_name)
  - GENERATE_RESULT(…, run_name)
- New Database Vault views:
  - DVSYS.DBA_DV_POLICY
  - DVSYS.DBA_DV_POLICY_OBJECT

Security: Oracle Data Redaction, TDE, and TSDP

- New view:
  - REDACTON_EXPRESSIONS
- Package DBMS_REDACT:
  - CREATE_POLICY_EXPRESSION
  - APPLY_POLICY_EXPR_TO_COL
- Parameter:
  - ENCRYPT_NEW_TABLESPACES = CLOUD_ONLY | DDL | ALWAYS
- Package DBMS_TSDP_PROTECT.ADD_POLICY:
  - DBMS_TSDP_PROTECT.FINE_GRAINED_AUDIT
  - DBMS_TSDP_PROTECT.COLUMN_ENCRYPTION

Data Pump and SQL*Loader

- Data Pump parameters:
  - `impdp … DATA_OPTIONS = VALIDATE_TABLE_DATA`
  - `impdp … DATA_OPTIONS = ENABLE_PARALLEL_PARTITION_LOAD`
  - `impdp … REMAP_DIRECTORY='''/dir1':'/dir2'''`
  - `impdp … NETWORK_LINK = dblink1 ACCESS_METHOD = DIRECT_PATH DATA_OPTIONS = ENABLE_NETWORK_COMPRESSION`
  - `expdp … DATA_OPTIONS = GROUP_PARTITION_TABLE_DATA`
- SQL*Loader parameter:
  - `sqlldr … CONTROL=test.ctl SDF_PREFIX=/LOBS/photos`

Performance

- New `ON_QUERY_COMPUTATION` column in `DBA_MVIEWS` view
- Parameter:
  - `CURSOR_INVALIDATION = DEFERRED | IMMEDIATE`
- Package: `DBMS_COMPRESSION.GET_COMPRESSION_RATIO`
  - New compression `DBMS_COMPRESSION.COMP_INDEX_ADVANCED_HIGH`
- Package: `DBMS_ILM`
  - New parameter in `STOP_ILM: P_JOBNAME`

Performance: In-Memory Column Store

- New view: `DBA_EXPRESSION_STATISTICS`
- Parameters:
  - `INMEMORY_EXPRESSIONS_USAGE = ENABLE`
  - `INMEMORY_VIRTUAL_COLUMNS = ENABLE`
  - `INMEMORY_EXPRESSIONS_CAPTURE = ENABLE`
- Package: `DBMS_INMEMORY_ADMIN.ENABLE_FASTSTART`

Performance: SQL Tuning

- New views:
  - DBA_SQL_MANAGEMENT_CONFIG
  - V$STATS_ADVISOR_RULES
- Package DBMS_SPM:
  - New task parameters in SET_EVOLVE_TASK_PARAMETER:
    - ALTERNATE_PLAN_SOURCE
    - ALTERNATE_PLAN_BASELINE
    - ALTERNATE_PLAN_LIMIT
  - New procedures:
    - CONFIGURE
    - LOAD_PLANS_FROM_AWR
- Package DBMS_SQLPA:
  - New task parameters in SET_ANALYSIS_TASK_PARAMETER:
    - EXECUTE_TRIGGERS
    - REPLACE_SYSDATE_WITH
    - NUM_ROWS_TO_FETCH

Performance: DB Replay

- New `PLSQL_MODE` column in `DBA_WORKLOAD_CAPTURES` view
- Package `DBMS_WORKLOAD_CAPTURE.START_CAPTURE`:
  - New capture parameter: `PLSQL_MODE`
- Package `DBMS_WORKLOAD_REPLAY.PROCESS_CAPTURE`:
  - New process parameter: `PLSQL_MODE`
- Package `DBMS_WORKLOAD_REPLAY.INITIALIZE_REPLAY`:
  - New initialize parameter: `PLSQL_MODE`

Performance: Resource Manager

- New `PGA_LIMIT_SESSIONS_KILLED` column in `V$RSRC_CONSUMER_GROUP` view
- New `SESSION_PGA_LIMIT` parameter in `DBMS_RESOURCE_MANAGER` package in procedures:
  - `CREATE_PLAN_DIRECTIVE`
  - `UPDATE_PLAN_DIRECTIVE`
  - `DELETE_PLAN_DIRECTIVE`

Performance: Online Operations

- New view:
  - V$ONLINE_REDEF
- New columns in DBA_REDEFINITION_STATUS view:
  - ERR_TXT
  - ACTION
  - REFRESH_DEP_MVIEWS
- New parameters in DBMS_REDEFINITION procedures:
  - START_REDEF_TABLE: ENABLE_ROLLBACK + REFRESH_DEP_MVIEWS
  - FINISH_REDEF_TABLE: DISABLE_ROLLBACK
- New procedures in DBMS_REDEFINITION package:
  - ABORT_ROLLBACK
  - ROLLBACK

Partitioning

- New columns in DBA_PART_TABLES:
  - AUTOLIST
  - INTERVAL_SUBPARTITION
  - AUTOLIST_SUBPARTITION