Oracle 19c New Features

TriveraTech
TECHNOLOGY TRAINING

1. Introduction

Overview

- This course focuses on the new features and enhancements of Oracle Database.
- Oracle 19c New Features
- Hands-on practices emphasize functionality rather than test knowledge.

New Release Model

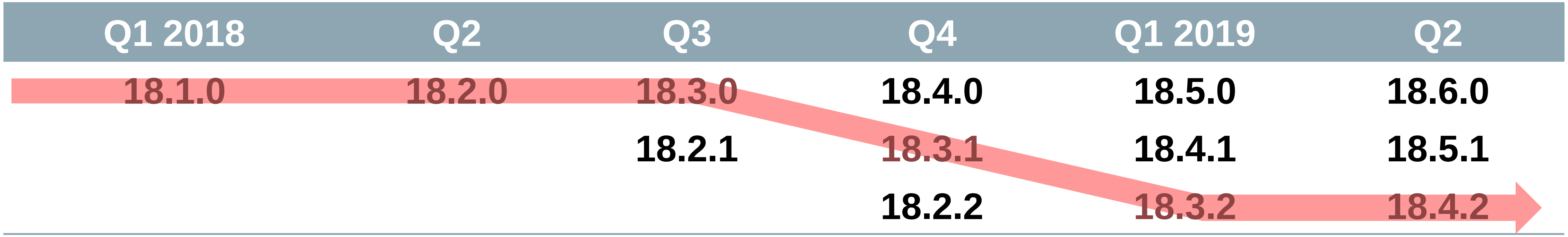
Oracle delivers annual releases and quarterly release updates.

- Oracle delivers releases yearly instead of on a multi-year cycle.
- Yearly releases improve database quality by reducing the number of software changes released at one time.
- Quarterly Release Update (RU) + Release Update Revision (RUR) improve the quality and experience of proactive maintenance. This model:
  - Gives best of PSUs combined with the best of Bundle Patches
  - Allows customers to update using RUs when they need fixes, and then switch to field proven RURs when their environment becomes stable
  - Enables customers to switch back and forth between RUs and RURs unlike PSUs and BPs
  - Contains all important security fixes
  - Eliminates tradeoff between security and stability
  - Shipped in January, April, July, and October as PSUs and BPs

New Version Numbering for Oracle Database

Version numbers use a three-digit format consisting of: ***Year.Update.Revision***

- ***Year*** is the last two digits of the year a release is delivered:
  - “18” stands for year 2018: Oracle Database 19c
- ***Update*** tracks release update (RU) or Beta releases.
- ***Revision*** tracks the associated release update revision (RUR) level (0, 1, 2).
  - 18.1.0 uses new version numbering for all RUs and RURs.
- Production databases start by using latest RU for fastest stabilization.
- When production DB stability is achieved, switch over to RURs seamlessly.



Q1 2018	Q2	Q3	Q4	Q1 2019	Q2
18.1.0	18.2.0	18.3.0	18.4.0	18.5.0	18.6.0
		18.2.1	18.3.1	18.4.1	18.5.1
			18.2.2	18.3.2	18.4.2

Practice 1: Overview

- 1-1: Discovering practices environment

2. Using Multitenant Enhancements

Objectives

After completing this module, you should be able to:

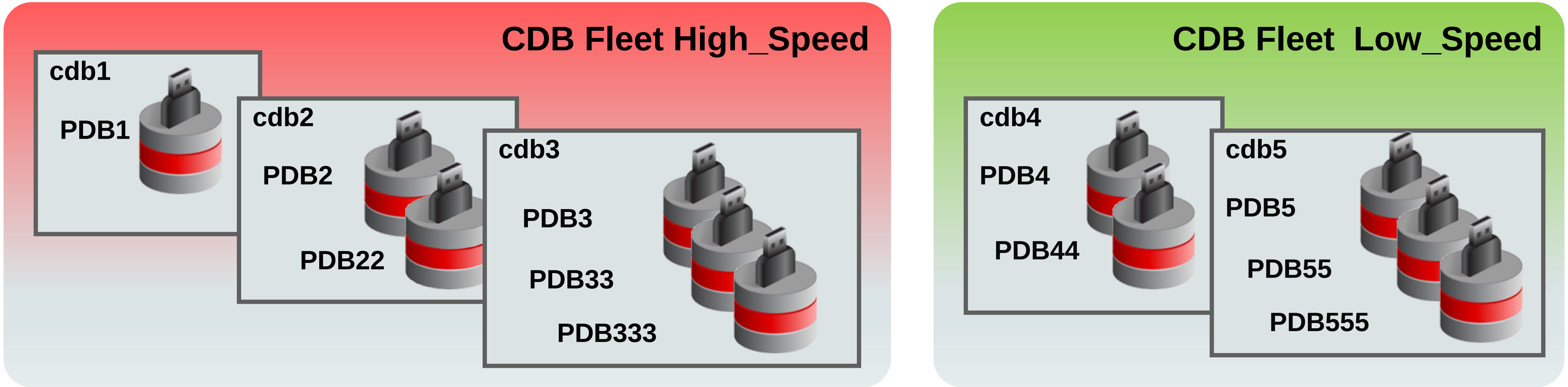
- Manage a CDB fleet
- Manage PDB snapshots
- Use a dynamic container map
- Explain lockdown profile inheritance
- Describe refreshable copy PDB switchover
- Identify the parameters used when instantiating a PDB on a standby
- Enable parallel statement queuing at PDB level
- Use DBCA to clone PDBs

CDB Fleet

19c

A CDB fleet is a collection of different CDBs that can be managed as one logical CDB:

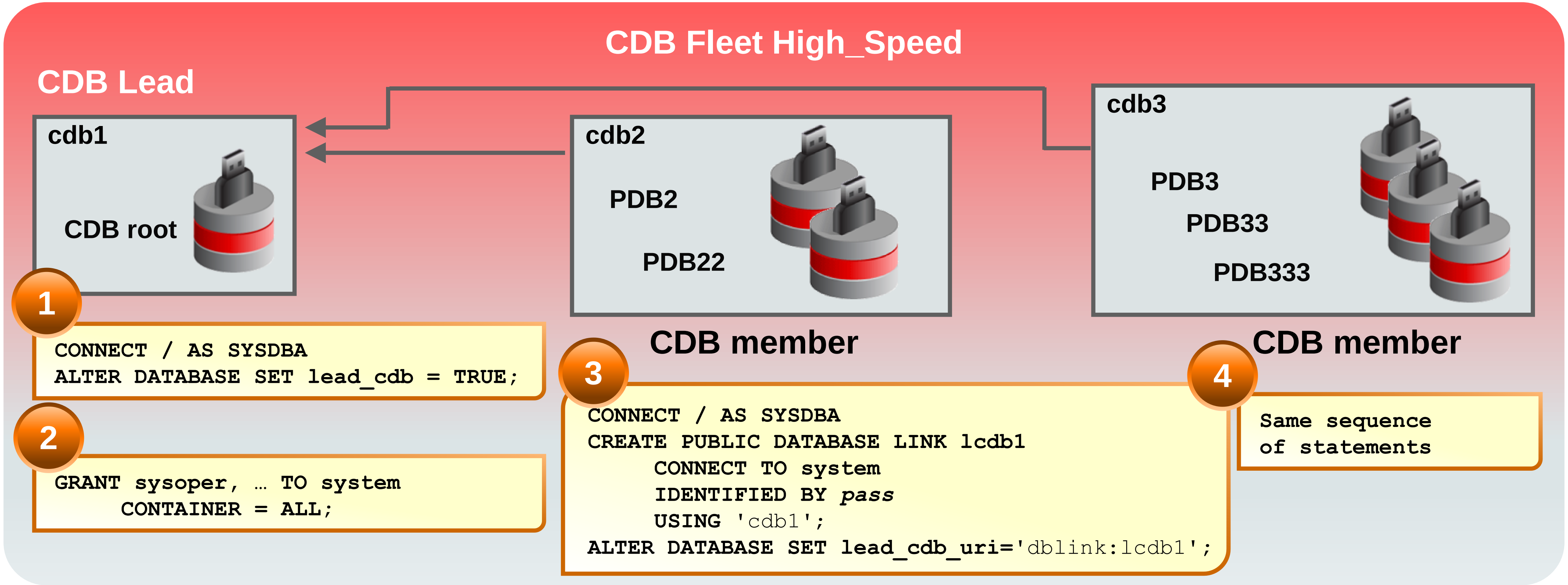
- To provide the underlying infrastructure for massive scalability and centralized management of many CDBs
- To provision more than the maximum number of PDBs for an application



- To manage appropriate server resources for PDBs, such as CPU, memory, I/O rate, and storage systems

CDB Lead and CDB Members

- The CDB lead in a fleet is the CDB from which you perform operations across the fleet.
- The CDB members of the fleet link to the CDB lead through a database link.



1

```
CONNECT / AS SYSDBA
ALTER DATABASE SET lead_cdb = TRUE;
```

2

```
GRANT sysoper, … TO system
      CONTAINER = ALL;
```

3

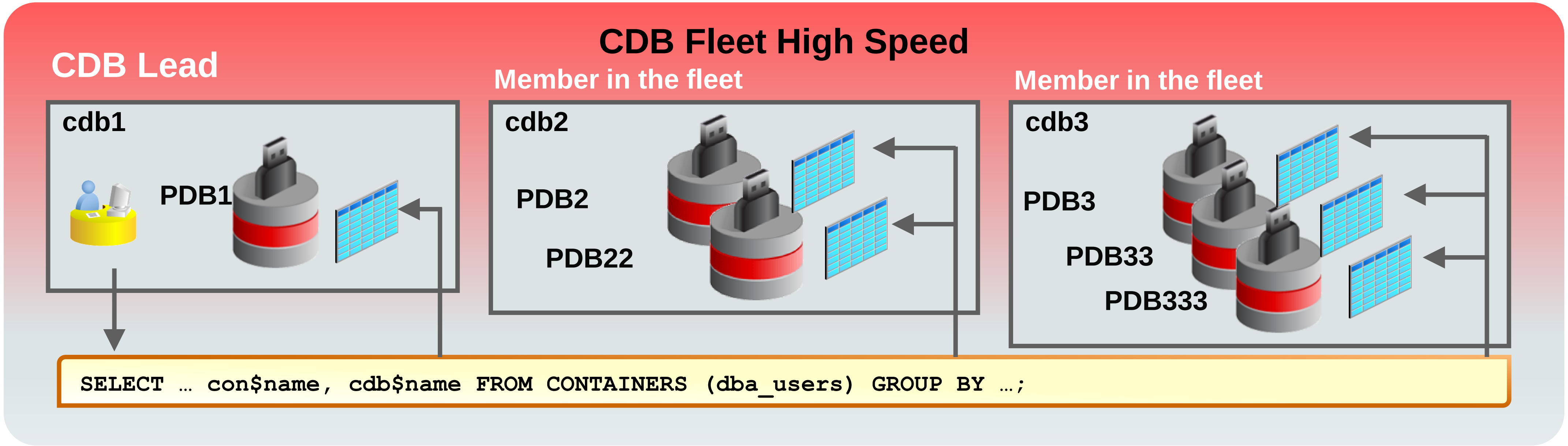
```
CONNECT / AS SYSDBA
CREATE PUBLIC DATABASE LINK lcdb1
      CONNECT TO system
      IDENTIFIED BY pass
      USING 'cdb1';
ALTER DATABASE SET lead_cdb_uri='dblink:lcdb1';
```

4

Same sequence of statements

Use Cases

- Monitoring and collecting diagnostic information across CDBs from the lead CDB
- Querying Oracle-supplied objects, such as DBA views, in different PDBs across the CDB fleet

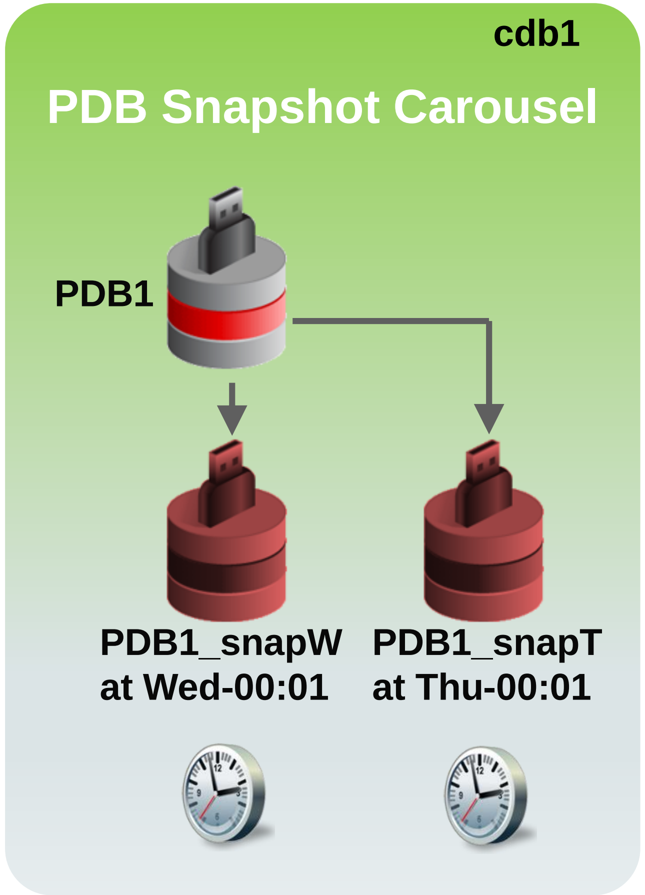


```
SELECT … con$name, cdb$name FROM CONTAINERS (dba_users) GROUP BY …;
```

- Serving as a central location where you can view information about and the status of all the PDBs across multiple CDBs

PDB Snapshot Carousel

A PDB snapshot is a named copy of a PDB at a specific point in time.



- Recovery extended beyond flashback retention period
- Reporting on historical data kept in snapshots
- Storage-efficient snapshot clones taken on periodic basis
- Maximum of eight snapshots for CDB and each PDB

Example:

On Friday, need to recover back to Wednesday.

- Restore `PDB1_snapW`.

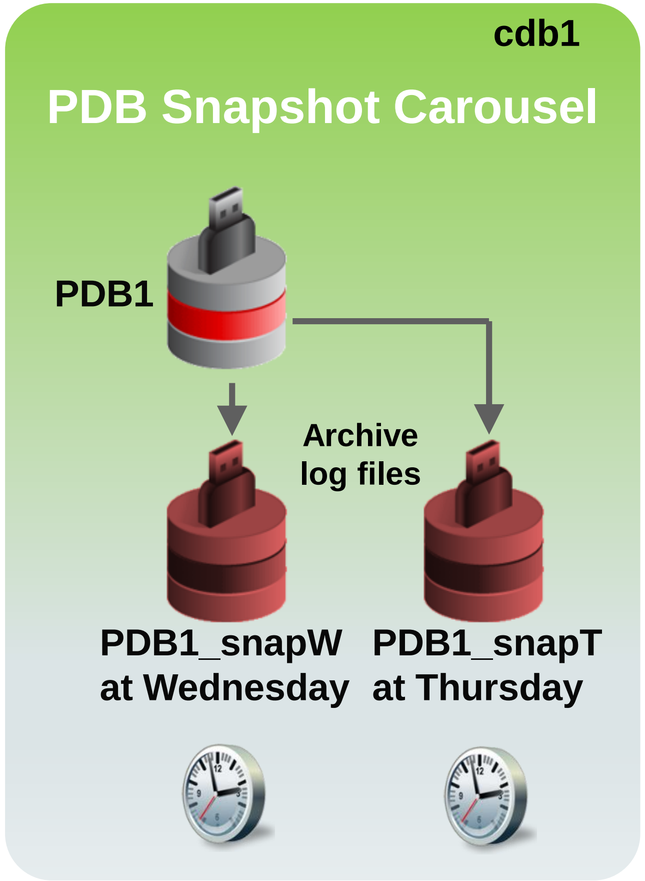
Creating PDB Snapshot

19c

```
DATABASE_PROPERTIES
 PROPERTY_NAME = MAX_PDB_SNAPSHOTS
 PROPERTY_VALUE  = 8
```

```
DBA_PDB_SNAPSHOTS
DBA_PDBS
    SNAPSHOT_MODE
```

To create PDB snapshots for a PDB:



1. Enable a PDB for PDB snapshots.

```
SQL> CREATE PLUGGABLE DATABASE pdb1 …
           SNAPSHOT MODE MANUAL;
```

```
SQL> ALTER PLUGGABLE DATABASE pdb1
           SNAPSHOT MODE EVERY 24 HOURS;
```

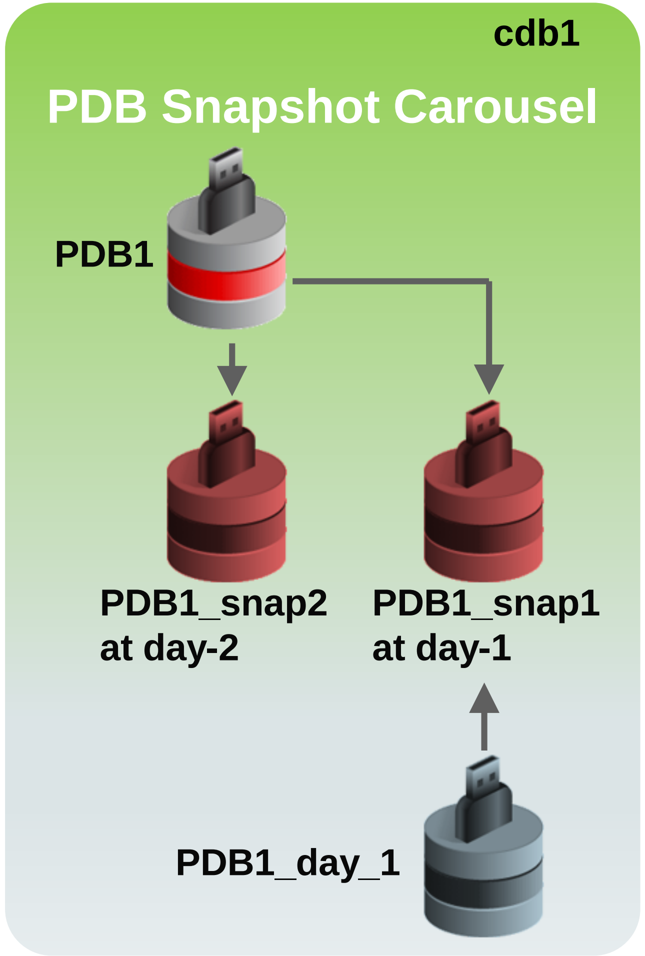
2. You can create multiple manual PDB snapshots of a PDB.

```
SQL> ALTER PLUGGABLE DATABASE pdb1
           SNAPSHOT pdb1_first_snap;
SQL> ALTER PLUGGABLE DATABASE pdb1
           SNAPSHOT pdb1_second_snap;
```

3. Disable snapshot creation for a PDB.

```
SQL> ALTER PLUGGABLE DATABASE pdb1 SNAPSHOT MODE NONE;
```

Creating PDBs Using PDB Snapshots



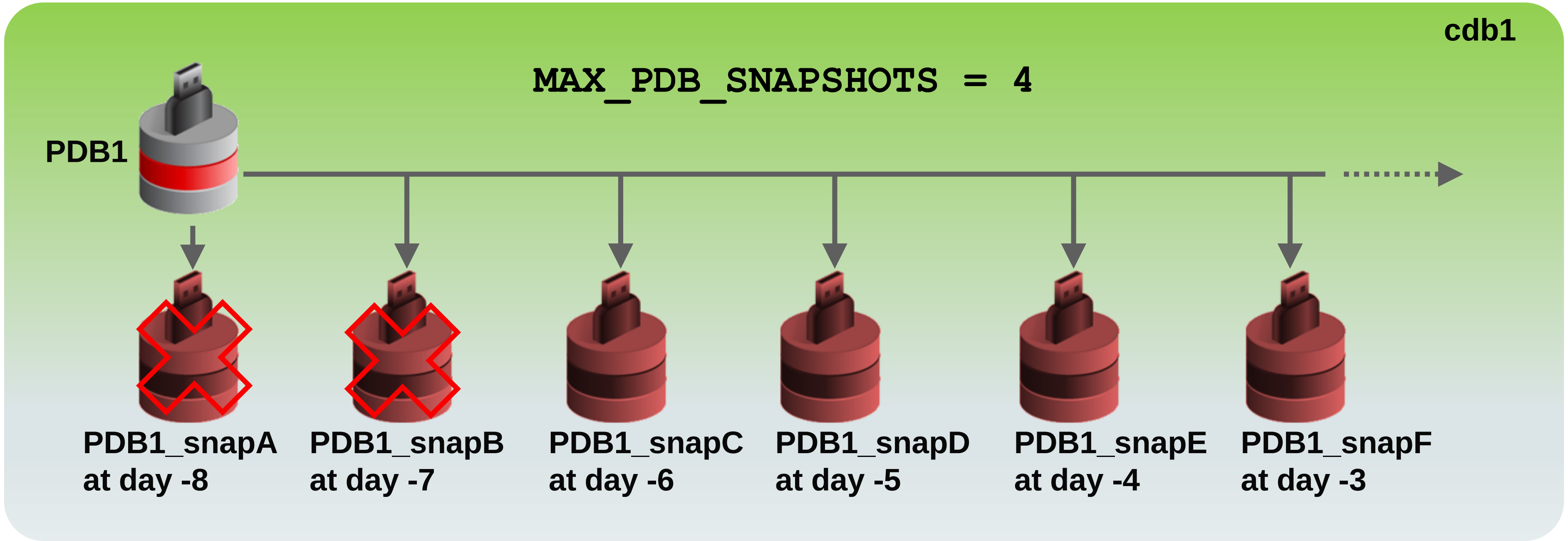
After a PDB snapshot is created, you can create a new PDB from it:

```
SQL> CREATE PLUGGABLE DATABASE pdb1_day_1 FROM pdb1
            USING SNAPSHOT <snapshot_name>;
```

```
SQL> CREATE PLUGGABLE DATABASE pdb1_day_2 FROM pdb1
            USING SNAPSHOT AT SCN <snapshot_SCN>;
```

Dropping PDB Snapshots

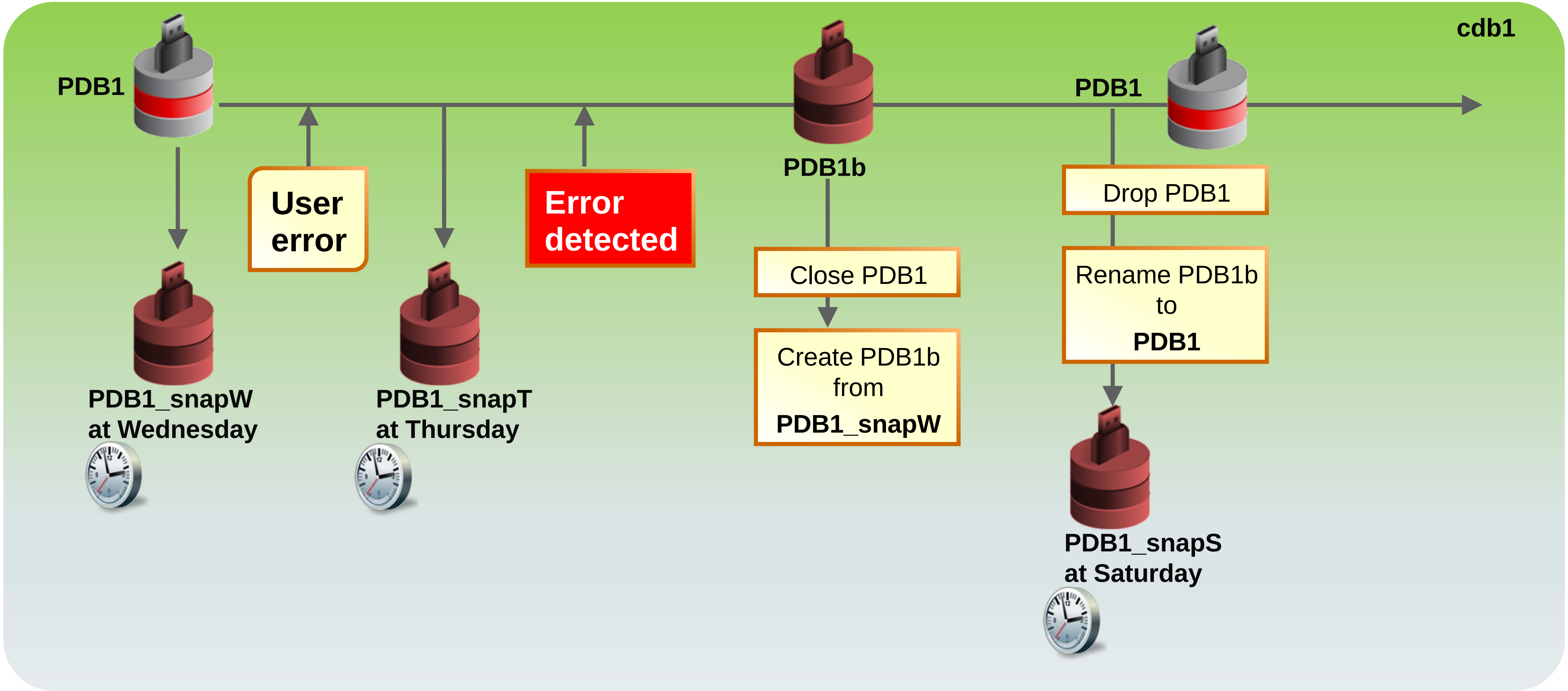
- Automatic PDB snapshot deletion when `MAX_PDB_SNAPSHOTS` is reached:



- Manual PDB snapshot deletion:

```
SQL> ALTER PLUGGABLE DATABASE pdb1 DROP SNAPSHOT pdb1_first_snap;
```

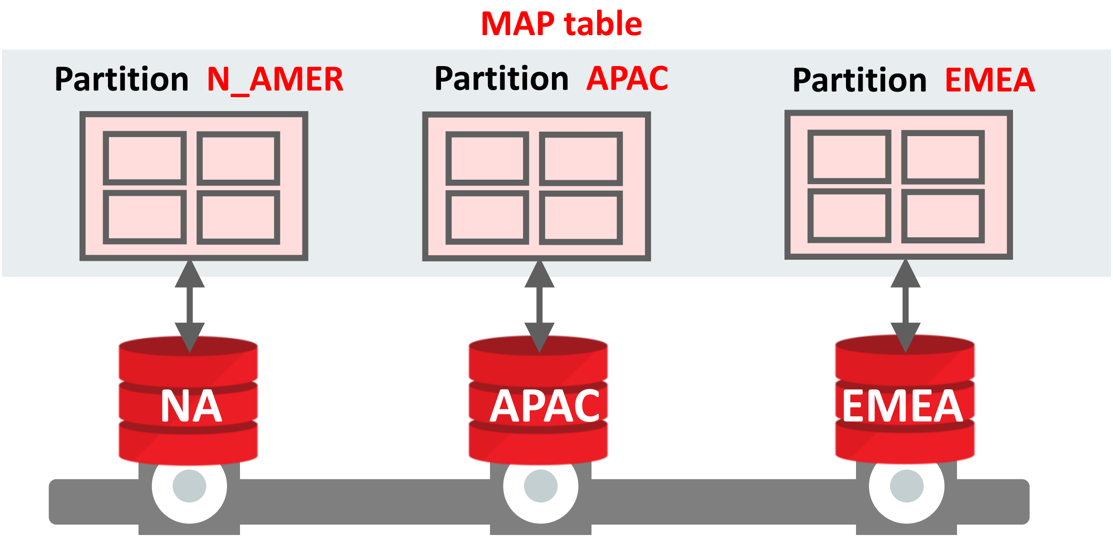
Flashbacking PDBs Using PDB Snapshots



Container Map

12c

- Define a PDB-based partition strategy based on the values stored in a column.
- Select a column that is commonly used and never updated.
  - Time Identifier (versus creation_date)/Region Name
- Set the database property `CONTAINER_MAP` in the application root.



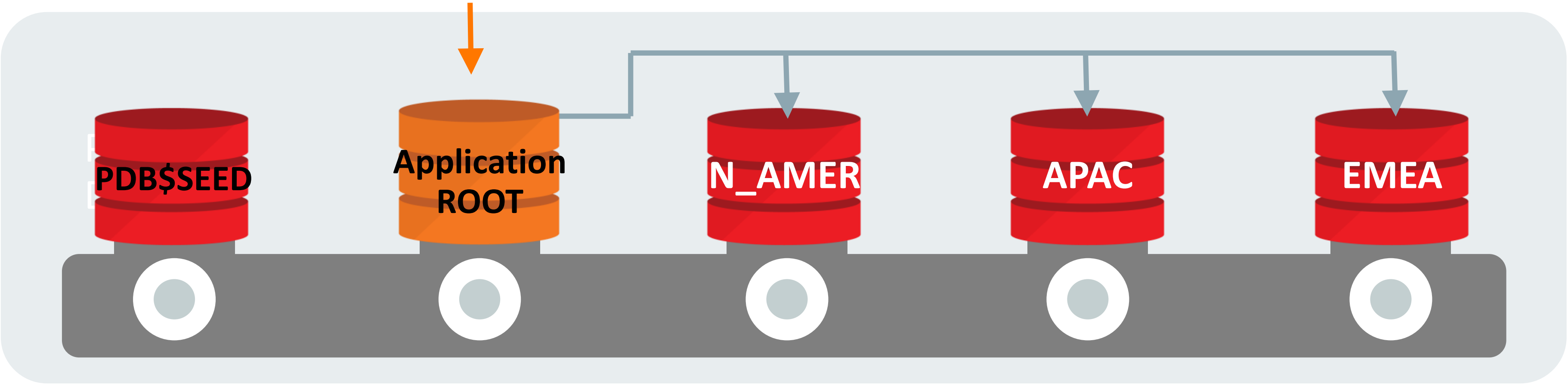
Each PDB corresponds to data for a particular partition.

DATABASE_PROPERTIES
PROPERTY_NAME = CONTAINER_MAP
PROPERTY_VALUE = app.tabapp
DESCRIPTION = value of container mapping table

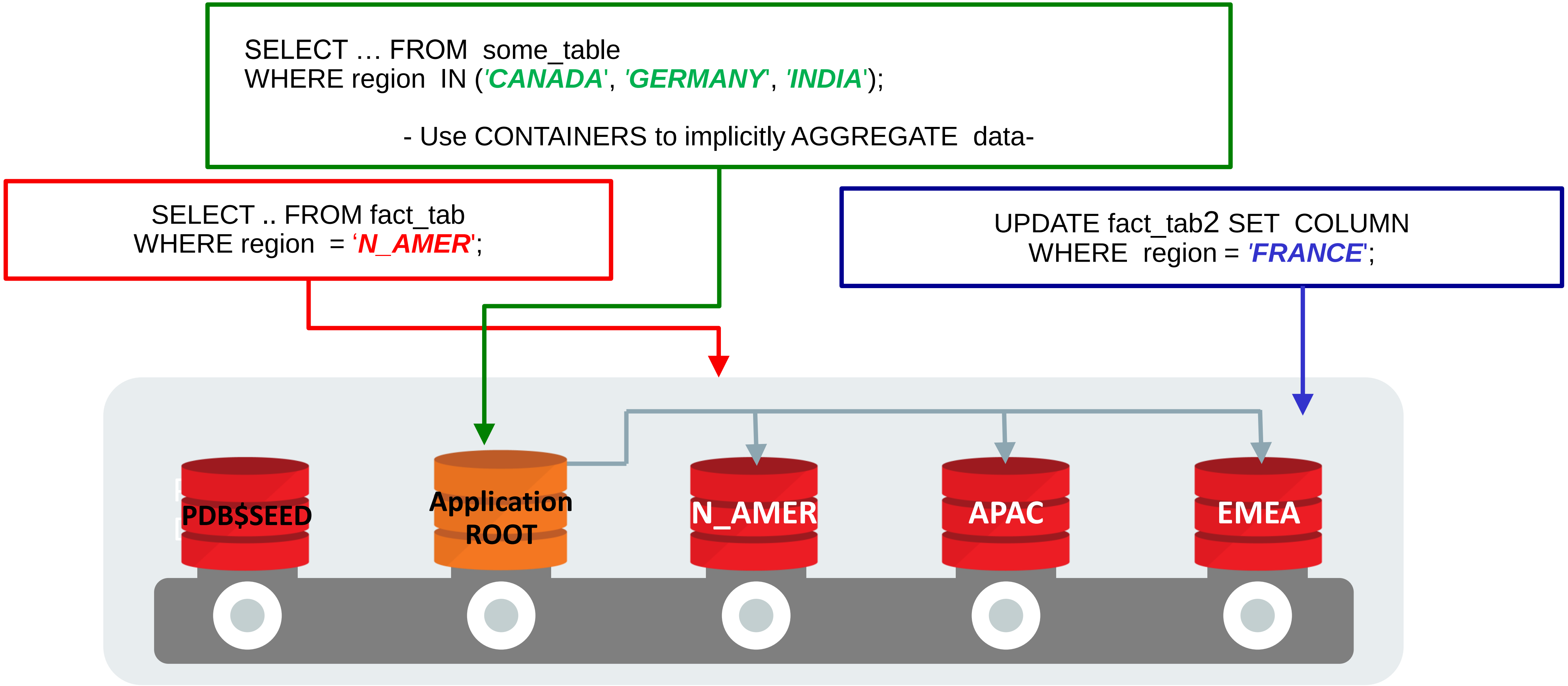
Container Map: Example

```
CREATE TABLE tab1 (region …, …);
CREATE TABLE tab2 (…, region …);

CREATE TABLE app1.app_map ( columns …, region  VARCHAR2(20))
PARTITION BY LIST (region)
 (PARTITION N_AMER VALUES ('TEXAS','CALIFORNIA','MEXICO','CANADA'),
  PARTITION EMEA VALUES ('UK', 'FRANCE', 'GERMANY'),
  PARTITION APAC VALUES ('INDIA', 'CHINA', 'JAPAN'));


ALTER PLUGGABLE DATABASE SET CONTAINER_MAP = 'app1.app_map';
ALTER TABLE tab1 ENABLE container_map;
```

DBA_TABLES
CONTAINER_MAP_OBJECT = YES



Query Routed Appropriately



```
SELECT … FROM  some_table
WHERE region  IN ('CANADA', 'GERMANY', 'INDIA');
```

- Use CONTAINERS to implicitly AGGREGATE data-

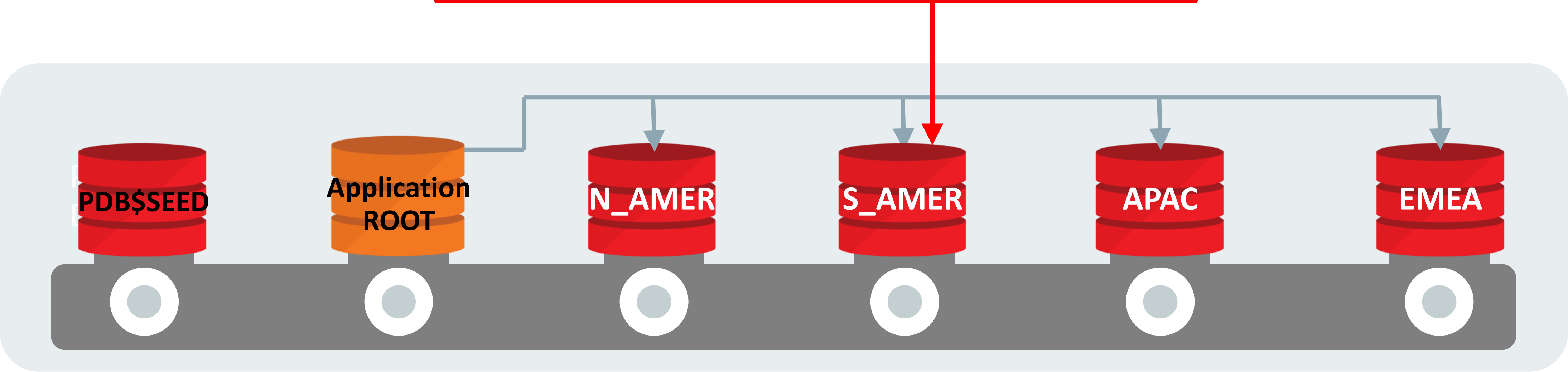
```
SELECT .. FROM fact_tab
WHERE region  = 'N_AMER';
```

```
UPDATE fact_tab2 SET  COLUMN
WHERE  region = 'FRANCE';
```

Dynamic Container Map

```
CREATE PLUGGABLE DATABASE s_amer …
       CONTAINER_MAP UPDATE (ADD PARTITION s_amer VALUES ('PERU','ARGENTINA'));
```

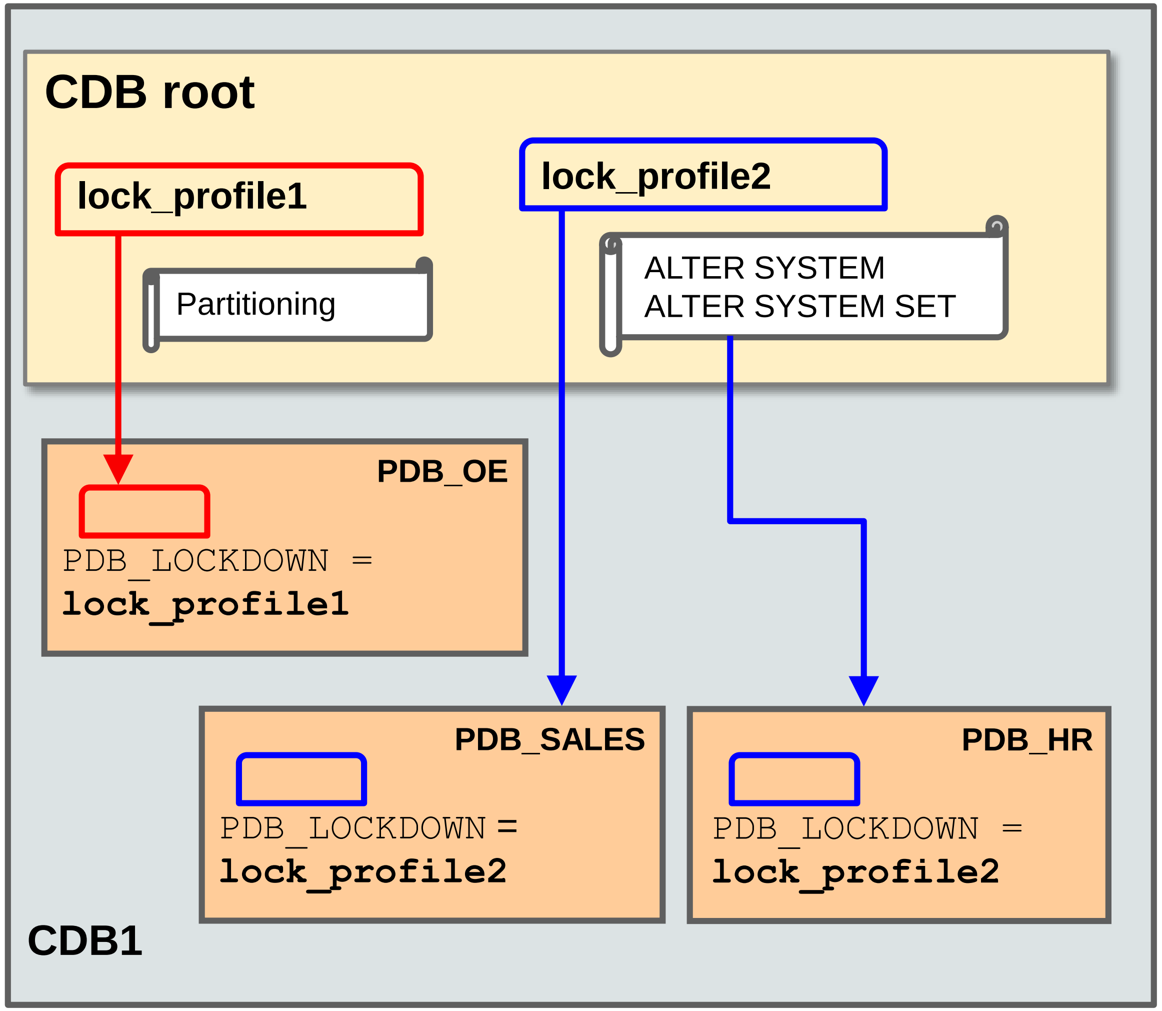
SELECT .. FROM fact_tab WHERE region = '*S_AMER*';



```
CREATE PLUGGABLE DATABASE s_amer_peru …
       CONTAINER_MAP UPDATE (SPLIT PARTITION s_amer
              INTO (partition s_amer ('ARGENTINA'), partition s_amer_peru));
```

Restricting Operations with Lockdown Profile

12c

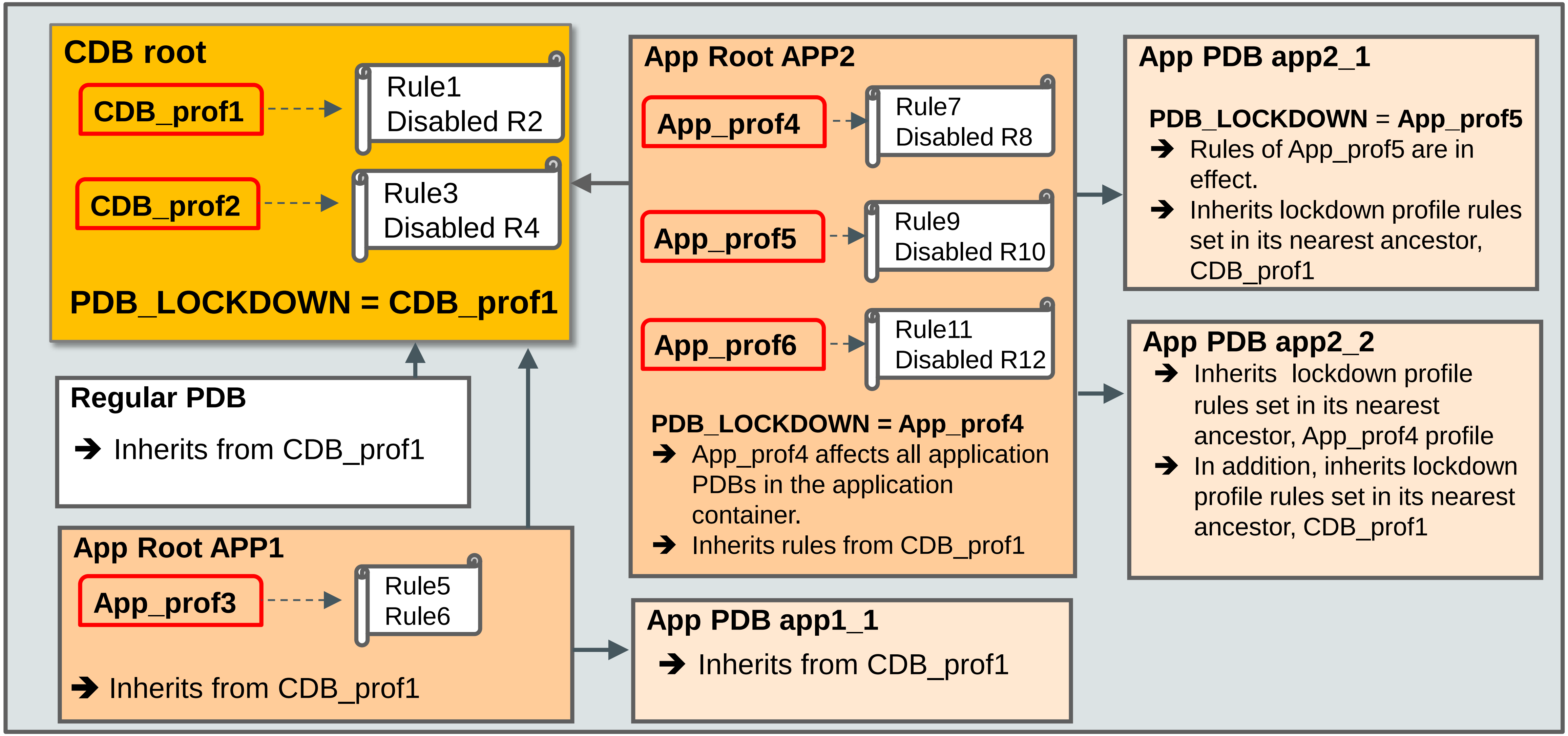


Operations, features, and options used by users connected to a given PDB can be disallowed.

1. You can create PDB lockdown profiles from the CDB root only.
2. You can define restrictions through enabled and disabled:
   - Statements and clauses
   - Features
   - Options
3. Setting the `PDB_LOCKDOWN` parameter to a PDB lockdown profile sets it for all PDBs.
4. Optionally, set the `PDB_LOCKDOWN` parameter to another PDB lockdown profile for a PDB.
5. Restart the PDBs.

19c

Lockdown Profiles Inheritance



CDB root

CDB_prof1 → Rule1 Disabled R2

CDB_prof2 → Rule3 Disabled R4

PDB_LOCKDOWN = CDB_prof1

Regular PDB

➔ Inherits from CDB_prof1

App Root APP1

App_prof3 → Rule5 Rule6

➔ Inherits from CDB_prof1

App Root APP2

App_prof4 → Rule7 Disabled R8

App_prof5 → Rule9 Disabled R10

App_prof6 → Rule11 Disabled R12

PDB_LOCKDOWN = App_prof4

➔ App_prof4 affects all application PDBs in the application container.

➔ Inherits rules from CDB_prof1

App PDB app1_1

➔ Inherits from CDB_prof1

App PDB app2_1

PDB_LOCKDOWN = **App_prof5**

➔ Rules of App_prof5 are in effect.

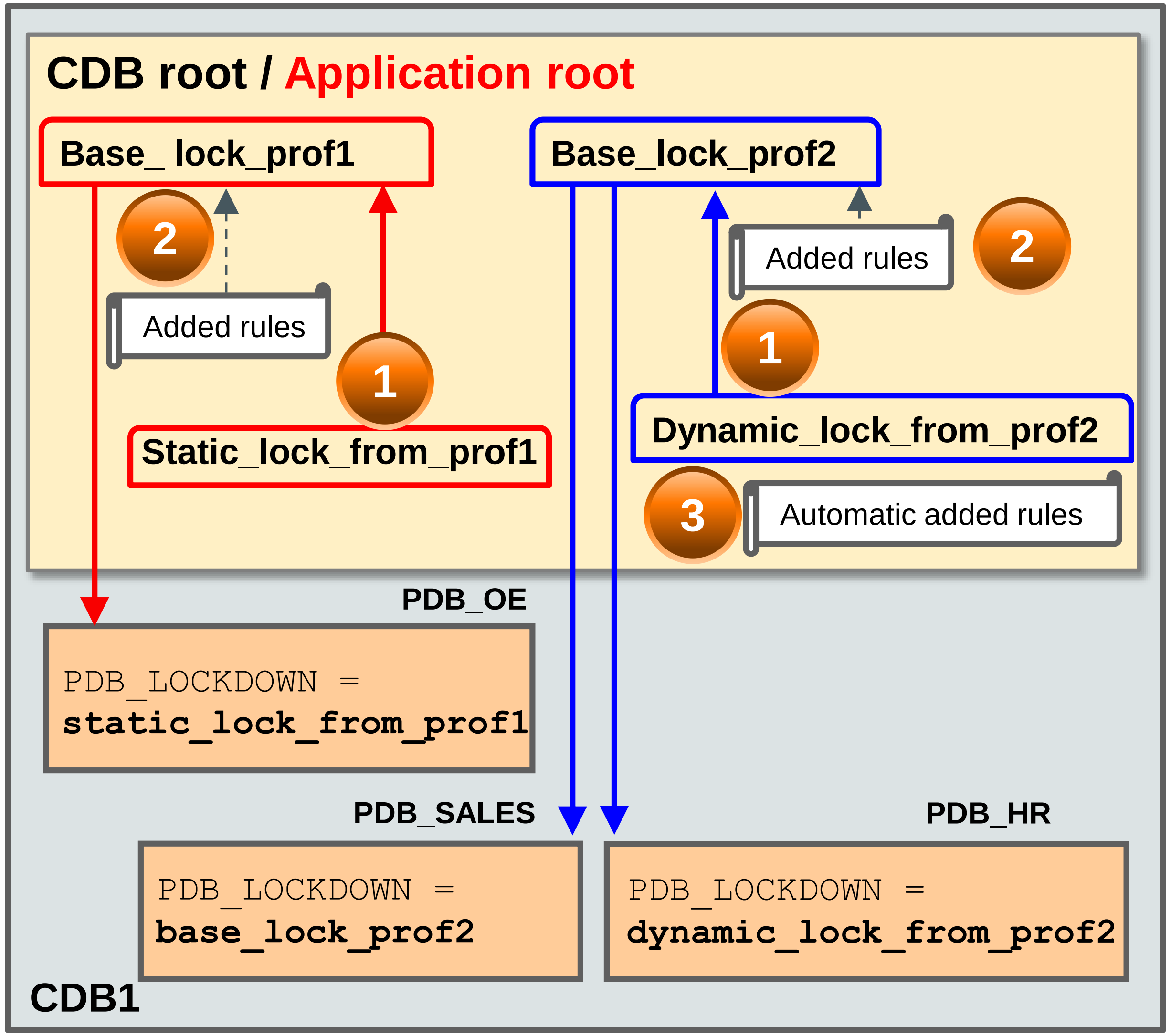
➔ Inherits lockdown profile rules set in its nearest ancestor, CDB_prof1

App PDB app2_2

➔ Inherits lockdown profile rules set in its nearest ancestor, App_prof4 profile

➔ In addition, inherits lockdown profile rules set in its nearest ancestor, CDB_prof1

Static and Dynamic Lockdown Profiles



There are two ways to create lockdown profiles by using an existing profile:

- Static lockdown profiles:

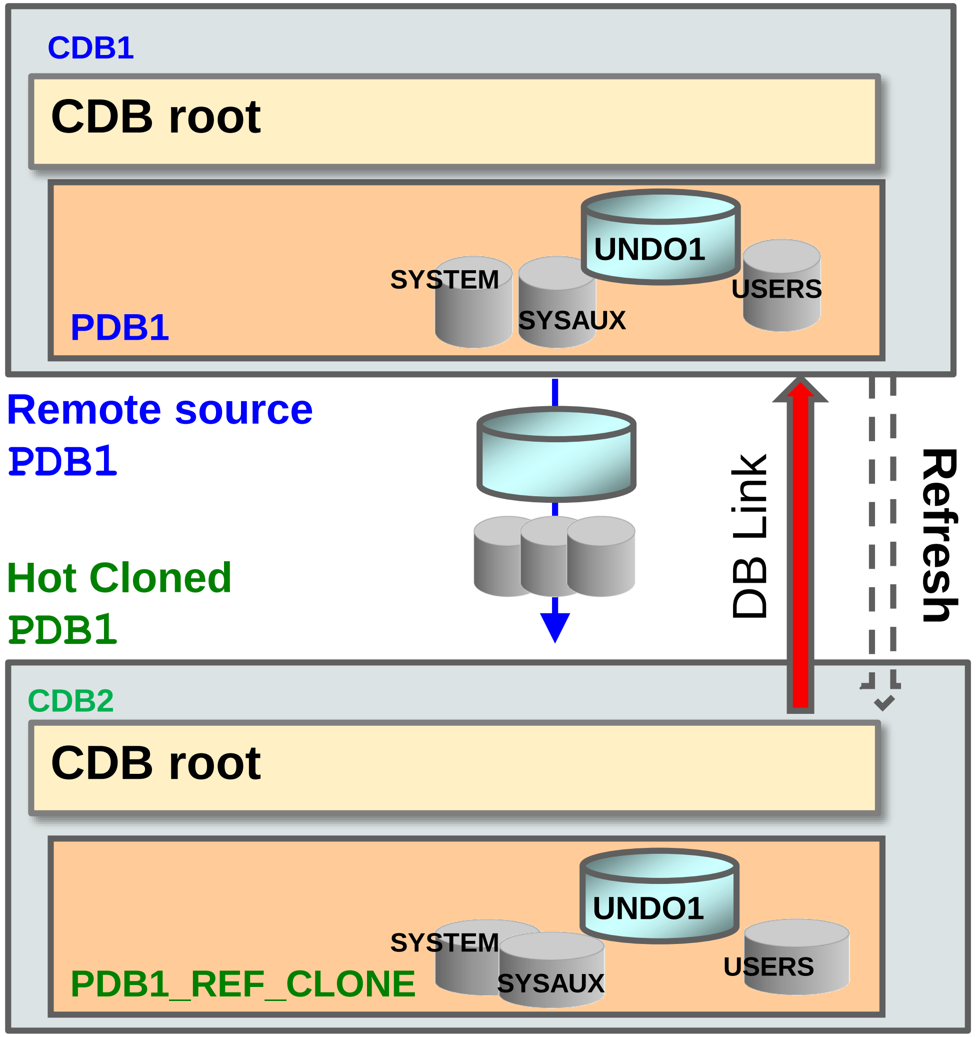
```
SQL> CREATE LOCKDOWN PROFILE prof3
       FROM base_lock_prof1;
```

- Dynamic lockdown profiles:

```
SQL> CREATE LOCKDOWN PROFILE prof4
       INCLUDING base_lock_prof2;
```

Refreshable Cloned PDB

12c



Remote source PDB still up and fully functional:

1. Connect to the target `CDB2` root to create the database link to `CDB1`.
2. Switch the shared UNDO mode to local UNDO mode in both CDBs.
3. Clone the remote `PDB1` to `PDB1_REF_CLONE`.
4. Open `PDB1_REF_CLONE` in read/write mode.

Incremental refreshing => Open `PDB1_REF_CLONE` in RO mode**:**

- Manual
- Automatic (predefined interval)

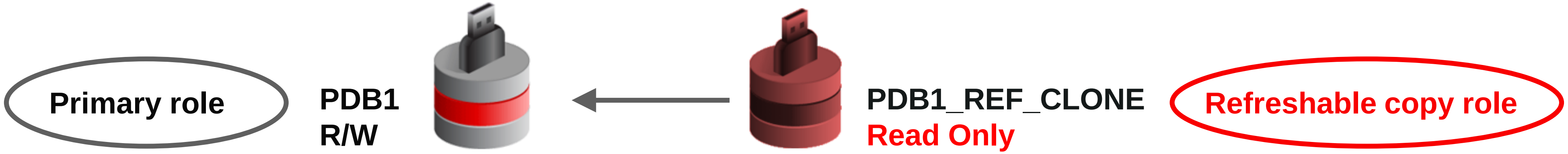
```
SQL> CREATE PLUGGABLE DATABASE pdb1_ref_clone
     FROM   pdb1@link_pdb_source_for_clone
     REFRESH MODE EVERY 2 MINUTES;
```

19c

Switching Over a Refreshable Cloned PDB

Switchover at the PDB level:

1. A user creates a refreshable clone of a PDB.



Primary role

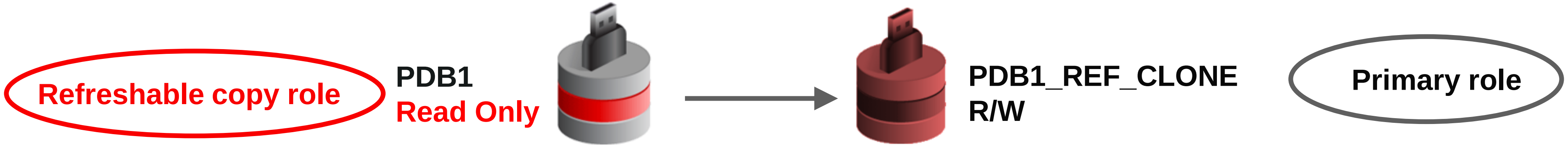
PDB1
R/W

PDB1_REF_CLONE
Read Only

Refreshable copy role

2. The roles can be reversed: the refreshable clone can be made the primary PDB.
   – The new primary PDB can be opened in read/write mode.
   – The primary PDB becomes the refreshable clone.

```
SQL> CONNECT sys@PDB1 AS SYSDBA
SQL> ALTER PLUGGABLE DATABASE REFRESH MODE EVERY 6 HOURS
          FROM pdb1_ref_clone@link_cdb_source_for_clone SWITCHOVER;
```



Refreshable copy role

PDB1
Read Only

PDB1_REF_CLONE
R/W

Primary role

Unplanned Switchover

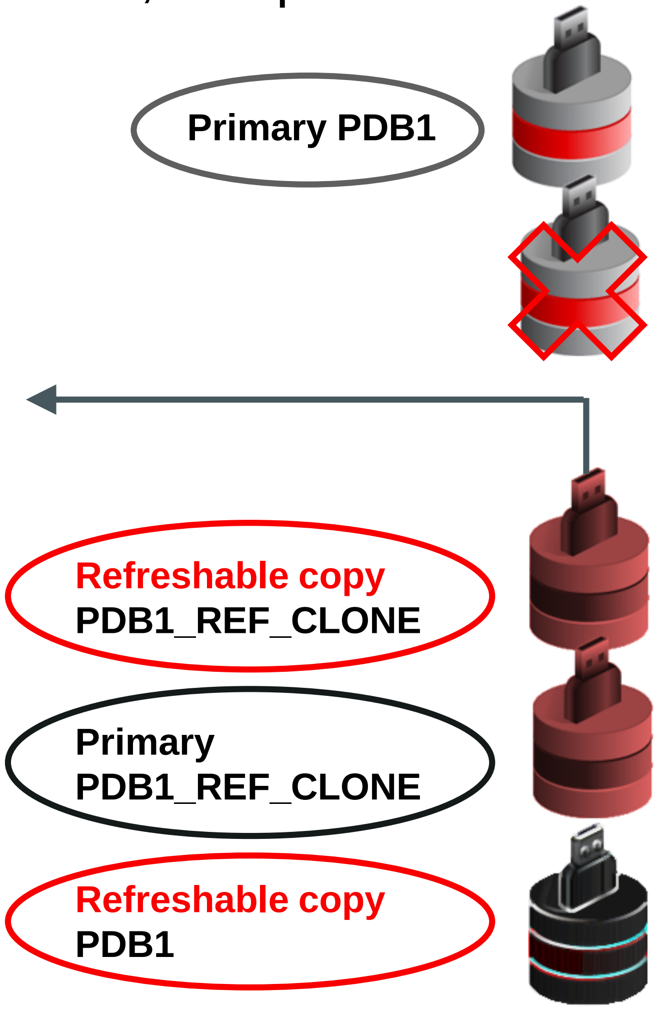
When a PDB with an associated refreshable clone encounters an issue, complete an unplanned switchover:

1. Close the primary PDB.
2. Archive the current redo log file.

3. Drop the primary PDB.
4. Copy the archive redo log files to a new folder.

5. Set the destination for the archive redo log files.
6. Refresh the refreshable clone PDB.

7. Disable the refresh mode of the refreshable clone PDB.
8. Open the refreshed PDB that became the new primary PDB.
9. Optionally, create a new refreshable clone.



Instantiating a PDB on a Standby

Creating a PDB on a primary CDB:

12c From an XML file: Copy the data files specified in the XML file to the standby database.

19c Use the `STANDBY_PDB_SOURCE_FILE_DIRECTORY` parameter to specify a directory location on the standby where source data files for instantiating the PDB may be found
➔ Data files are automatically copied.

12c As a clone from another PDB: Copy the data files belonging to the source PDB to the standby database.

19c Use the `STANDBY_PDB_SOURCE_FILE_DBLINK` parameter to specify the name of a database link which is used to copy the data files from the source PDB to which the database link points.
➔ The file copy is automatically done only if the database link points to the source PDB, and the source PDB is open in read-only mode.

19c

PDB-Level Parallel Statement Queuing

- Possible issues of parallel statements queuing in a PDB:

```
PARALLEL_DEGREE_POLICY = AUTO | ADAPTIVE
```

  - Not sufficient parallel servers available
  - Parallel statements queued for a long time
- A PDB DBA can make parallel statement queuing work just as it does in a non-CDB.
  - Disable parallel statement queuing at CDB level: `PARALLEL_SERVERS_TARGET` = 0.
  - Set the `PARALLEL_SERVERS_TARGET` initialization parameter for individual PDBs.
  - Kill a runaway SQL operation:

```
SQL> ALTER SYSTEM CANCEL SQL '272,31460';
```

  - Dequeue a parallel statement:

```
SQL> EXEC dbms_resource_manager.dequeue_parallel_statement()
```

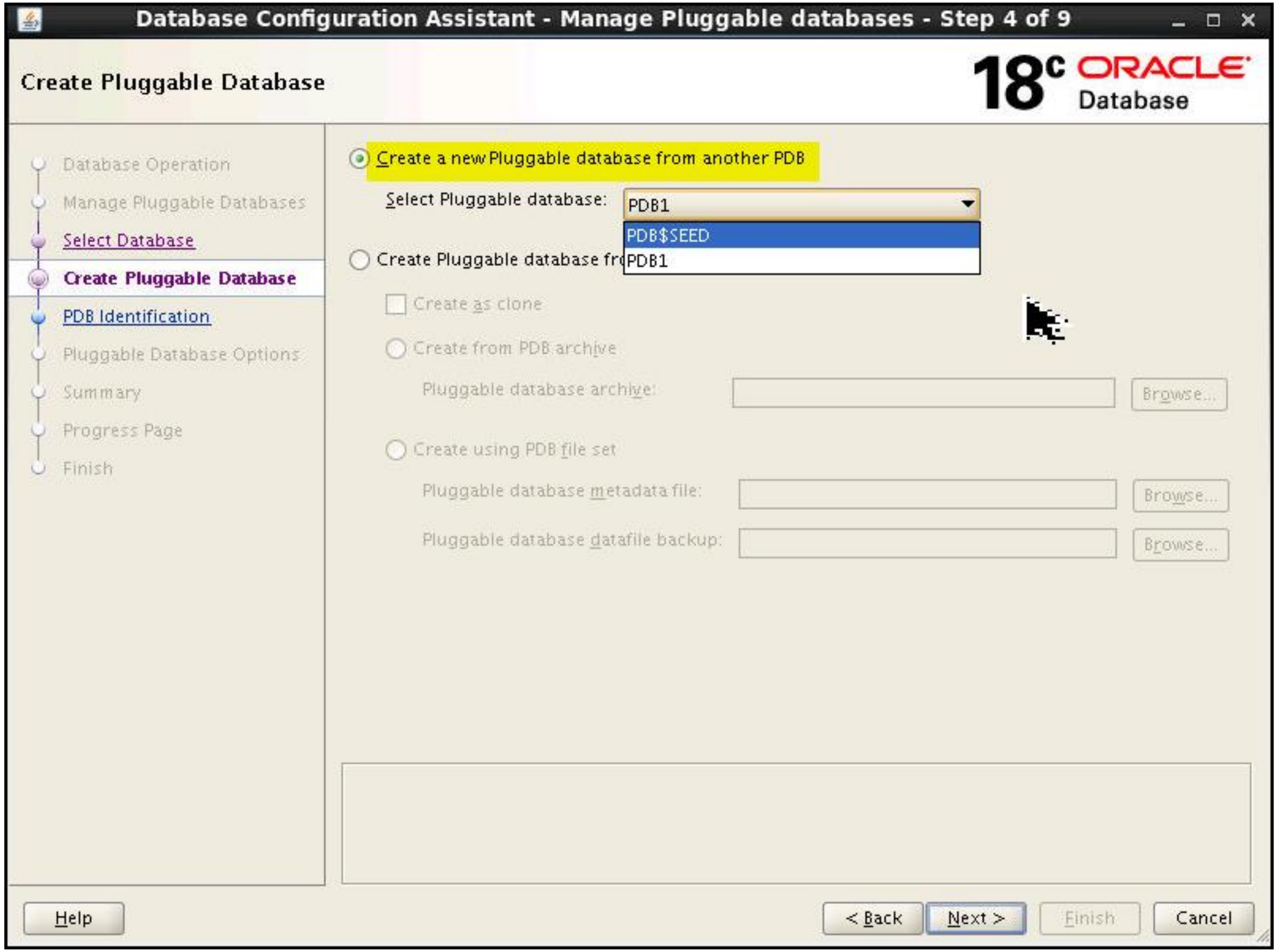
  - Define the action when dequeuing: `PQ_TIMEOUT_ACTION` plan directive

19c

PDB-Level Parallel Statement Queuing: CPU_COUNT

- If CPU_COUNT is set at the PDB level, the maximum DOP generated by AutoDOP queries are the PDB's CPU_COUNT.
- Similarly, the default value for PARALLEL_MAX_SERVERS and PARALLEL_SERVERS_TARGET are computed based on the PDB's CPU_COUNT.

Using DBCA to Clone PDBs


Database Configuration Assistant - Manage Pluggable databases - Step 4 of 9
Create Pluggable Database
18c ORACLE Database
Database Operation
Manage Pluggable Databases
Select Database
Create Pluggable Database
PDB Identification
Pluggable Database Options
Summary
Progress Page
Finish
Help
< Back
Next >
Finish
Cancel
Browse...
Browse...
Browse...

● Create a new Pluggable database from another PDB

Select Pluggable database: PDB1 (PDB$SEED, PDB1)

○ Create Pluggable database fr

□ Create as clone

○ Create from PDB archive

Pluggable database archive:

○ Create using PDB file set

Pluggable database metadata file:

Pluggable database datafile backup:

- Clones PDB in hot mode
- Creates the datafiles directory for the new PDB
- Opens the new PDB

Summary

In this lesson, you should have learned how to:

- Manage a CDB fleet
- Manage PDB snapshots
- Use a dynamic container map
- Explain lockdown profile inheritance
- Describe refreshable copy PDB switchover
- Identify the parameters used when instantiating a PDB on a standby
- Enable parallel statement queuing at PDB level
- Use DBCA to clone PDBs

Practice 2: Overview

- 2-1: Managing a CDB fleet
- 2-2: Managing and using PDB snapshots
- 2-3: Using a dynamic container map
- 2-4: Using static and dynamic lockdown profiles
- 2-5: Switching over refreshable cloned PDBs

3. Managing Security

Objectives

After completing this module, you should be able to:

- Create schema-only accounts
- Isolate a new PDB keystore
- Convert a PDB to run in isolated or united mode
- Migrate PDB keystore between keystore types
- Create user-defined TDE master keys
- Protect fixed-user database link passwords
- Export and import fixed-user database links with encrypted passwords
- Configure encryption of sensitive data in Database Replay files
- Perform Database Replay capture and replay in a database with Database Vault
- Explain Enterprise users integration with Active Directory

Schema-Only Account

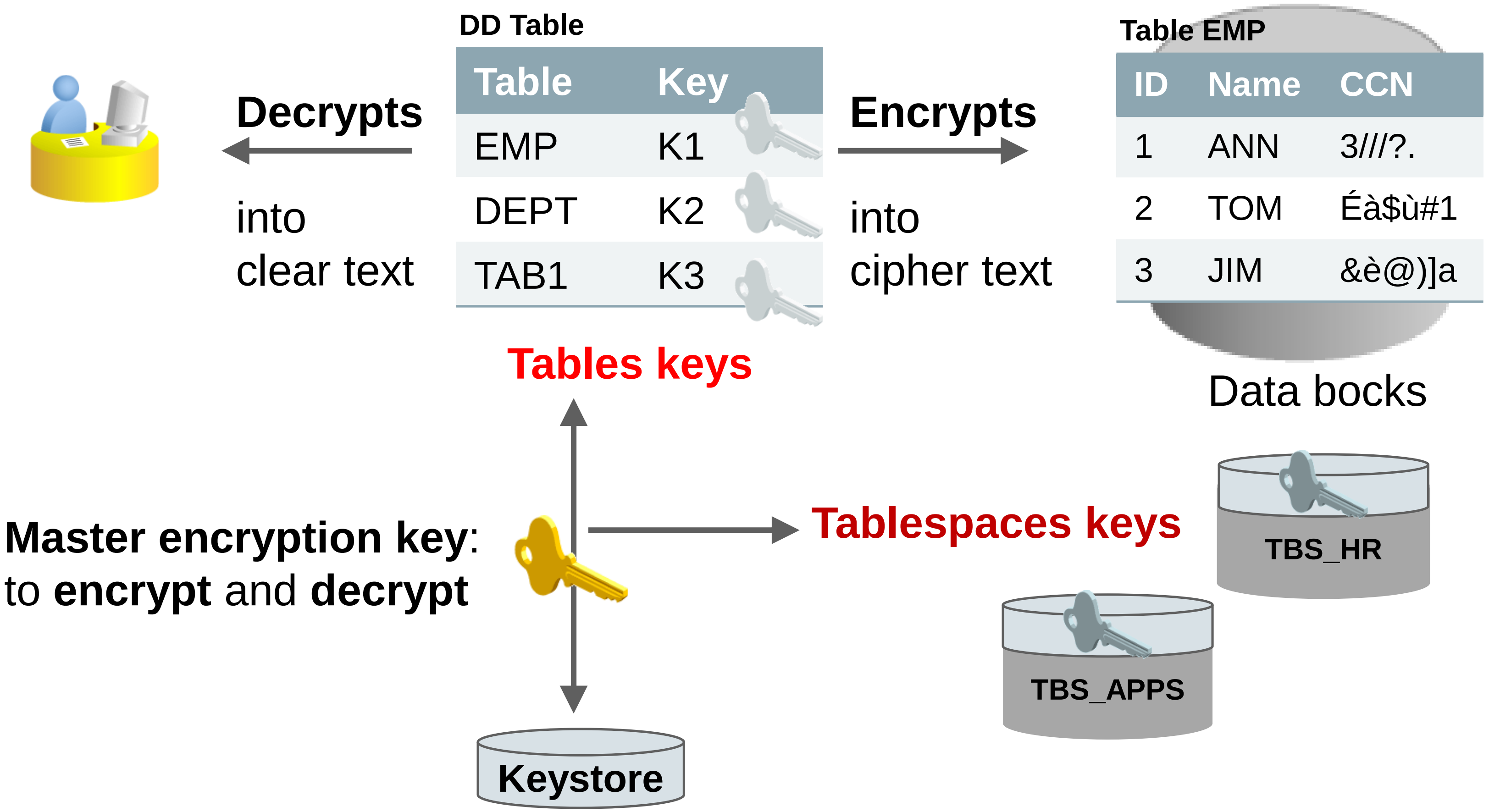
Ensure that user cannot log in to the instance:

- Enforce data access through the application.
- Secure schema objects.
  - Prevent objects from being dropped by the connected schema.
- Use the `NO AUTHENTICATION` clause.
  - Can be replaced by `IDENTIFIED BY VALUES`
- A schema-only account cannot be:
  - Granted system administrative privileges
  - Used in database links

```
DBA_USERS
  AUTHENTICATION_TYPE = NONE | PASSWORD
```

Encrypting Data Using Transparent Data Encryption

12c



Decrypts

into
clear text

DD Table

Table	Key
EMP	K1
DEPT	K2
TAB1	K3

Encrypts

into
cipher text

Table EMP

ID	Name	CCN
1	ANN	3///?.
2	TOM	Éà$ù#1
3	JIM	&è@)]a

Data bocks

Tables keys

Master encryption key:
to **encrypt** and **decrypt**

Tablespaces keys

TBS_HR

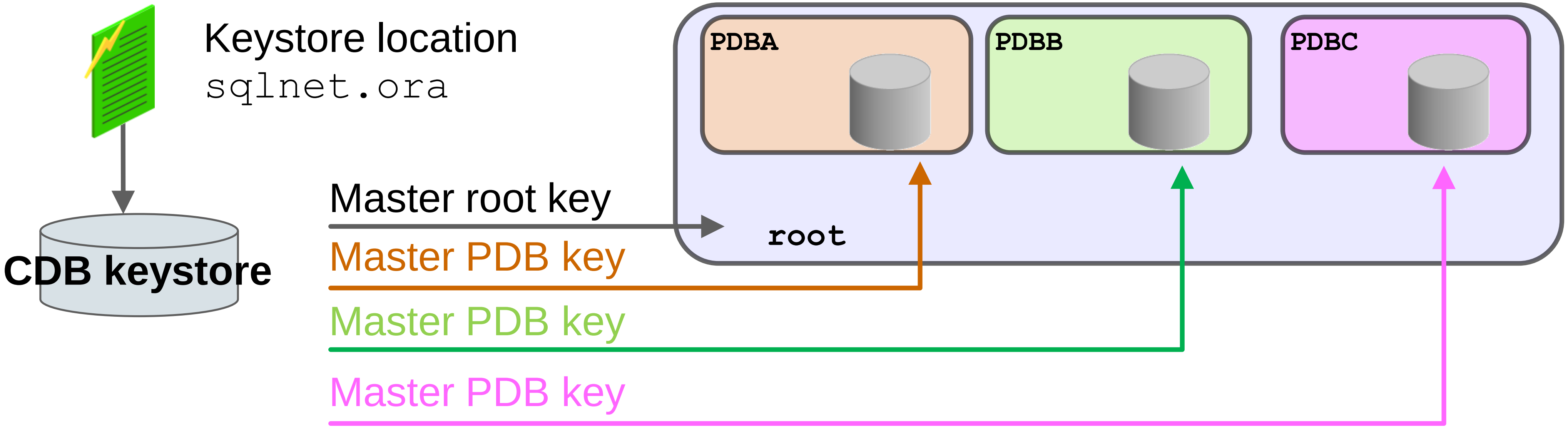
TBS_APPS

Keystore

Managing Keystore in CDB and PDBs

12c

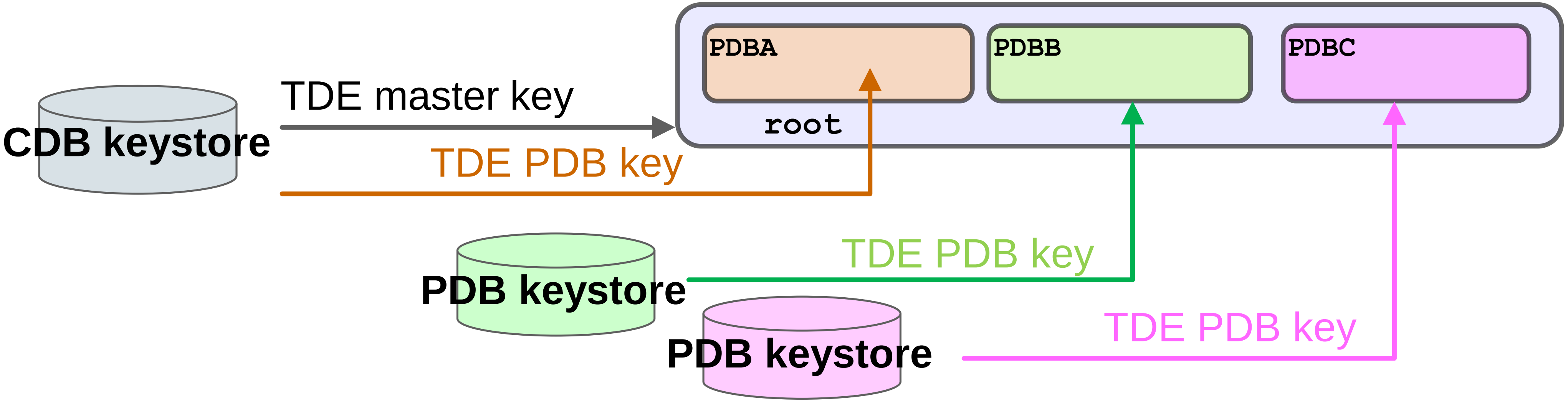
- There is one single keystore for CDB and all PDBs.
- There is one master key per PDB to encrypt PDB data.
- The master key must be transported from the source database keystore to the target database keystore when a PDB is moved from one host to another.



Managing Keystore in CDB and PDBs

19c

- There is still one single keystore for CDB and optionally one keystore per PDB.
- There is still one master key per PDB to encrypt PDB data, stored in the PDB keystore.
- Modes of operation
  - United mode: PDB keys are stored in the unique CDB root keystore.
  - Isolated mode: PDBs keys are stored in their own keystore.
  - Mix mode: Some PDBs use united mode, some use isolated mode.



Keystore Management Changes for PDB

```
V$ENCRYPTION_WALLET
   ENCRYPTION_MODE = NONE
```

PDBs can optionally have their own keystore, allowing tenants to manage their own keys.

1. Define the shared location for the CDB root and PDB keystores:

```
SQL> ALTER SYSTEM SET wallet_root = /u01/app/oracle/admin/ORCL/tde_wallet;
```

2. Define the default PDB keystore type for each future isolated PDB, and then define a different file type in each isolated PDB if necessary:

```
SQL> ALTER SYSTEM SET tde_configuration = 'KEYSTORE_CONFIGURATION=FILE';
```

   – United: ➔ WALLET_ROOT/*<component>*/ewallet.p12

     CDB root and **PDBA** /u01/app/oracle/admin/ORCL/tde_wallet/tde/ewallet.p12

   – Isolated: ➔ WALLET_ROOT/*<pdb_guid>*/*<component>*/ewallet.p12

     PDBB /u01/app/oracle/admin/ORCL/tde_wallet/51FE2A4899472AE6/tde/ewallet.p12

     PDBC /u01/app/oracle/admin/ORCL/tde_wallet/7893AB8994724ZC8/tde/ewallet.p12

Defining the Keystore Type

Values of keystore types allowed:

- FILE
- OKV (Oracle Key Vault)
- HSM (Hardware Security Module)
- FILE|OKV: Reverse-migration from OKV to FILE has occurred
- FILE|HSM: Reverse-migration from HSM to FILE has occurred
- OKV|FILE: Migration from FILE to OKV has occurred
- HSM|FILE: Migration from FILE to HSM has occurred

In isolated mode, when the CDB is in mounted state:

```
SQL> STARTUP MOUNT
SQL> ALTER SYSTEM SET tde_configuration='CONTAINER=pdb1; KEYSTORE_CONFIGURATION=FILE';
```

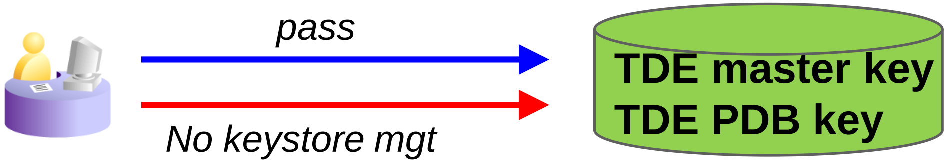
Isolating a PDB Keystore

```
V$ENCRYPTION_WALLET
   ENCRYPTION_MODE = ISOLATED
```

1. Create / open the CDB root keystore:

```
SQL> ADMINISTER KEY MANAGEMENT CREATE KEYSTORE
                          IDENTIFIED BY <united_keystore_pass> ;
```

2. Connect as the PDB security admin to the newly created PDB to:
   a. Create the PDB keystore



WALLET_ROOT/<pdb_guid>/tde/ewallet.p12

```
SQL> ADMINISTER KEY MANAGEMENT CREATE KEYSTORE
                          IDENTIFIED BY <isolated_keystore_pass> ;
```

   b. Open the PDB keystore

```
SQL> ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN
                          IDENTIFIED BY <isolated_keystore_pass> ;
```

   c. Create the TDE PDB key in the PDB keystore

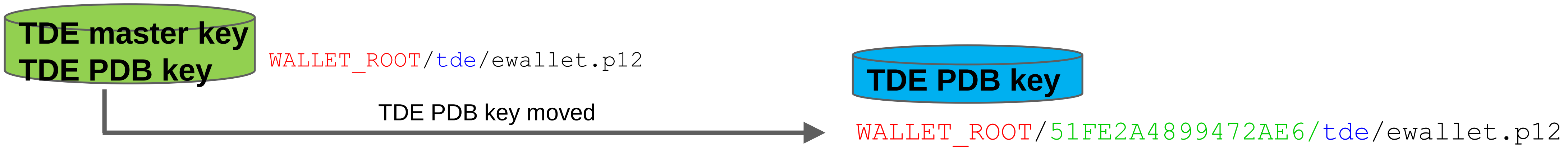
```
SQL> ADMINISTER KEY MANAGEMENT SET KEY IDENTIFIED BY <isolated_keystore_pass>
                          WITH BACKUP;
```

Converting a PDB to Run in Isolated Mode

```
V$ENCRYPTION_WALLET
   ENCRYPTION_MODE = ISOLATED
```

1. In the CDB root:
   a. Create a common user to act as the security officer
   b. Grant the ADMINISTER KEY MANAGEMENT privilege commonly to
2. Connect as the security officer to the PDB and create the keystore in the PDB.

```
SQL> ADMINISTER KEY MANAGEMENT ISOLATE KEYSTORE
         IDENTIFIED BY <isolated_keystore_password>
         FROM ROOT KEYSTORE IDENTIFIED BY [EXTERNAL STORE | <united_keystore_password>]
         WITH BACKUP;
```



TDE master key
TDE PDB key

WALLET_ROOT/tde/ewallet.p12

TDE PDB key moved

TDE PDB key

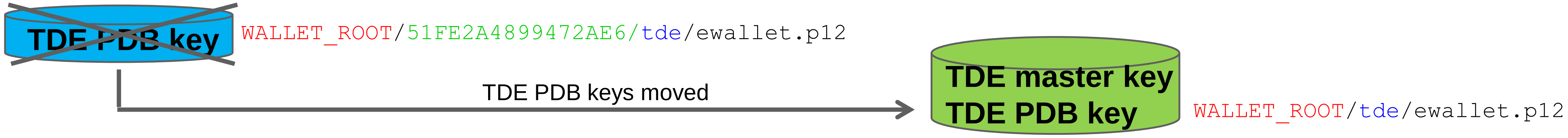
WALLET_ROOT/51FE2A4899472AE6/tde/ewallet.p12

Converting a PDB to Run in United Mode

```
V$ENCRYPTION_WALLET
   ENCRYPTION_MODE = UNITED
```

1. In the CDB root:
   a. The security officer of the CDB exists.
   b. The security officer of the CDB is granted the `ADMINISTER KEY MANAGEMENT` privilege commonly.
2. Connect as the security officer to the PDB and unite the TDE PDB key with those of the CDB root.

```
SQL> ADMINISTER KEY MANAGEMENT UNITE KEYSTORE
          IDENTIFIED BY <isolated_keystore_password>
          WITH ROOT KEYSTORE IDENTIFIED BY [EXTERNAL STORE | <united_keystore_password>]
          [WITH BACKUP [USING <backup_id>]];
```



~~TDE PDB key~~
WALLET_ROOT/51FE2A4899472AE6/tde/ewallet.p12
TDE PDB keys moved
TDE master key
TDE PDB key
WALLET_ROOT/tde/ewallet.p12

Migrating a PDB Between Keystore Types

To migrate a PDB from using wallet as the keystore to using Oracle Key Vault if the PDB is running in isolated mode:

1. Upload the TDE encryption keys from the isolated keystore to Oracle Key Vault using a utility.
2. Set the `TDE_CONFIGURATION` parameter of the PDB to the appropriate value:

```
SQL> ALTER SYSTEM SET tde_configuration = 'KEYSTORE_CONFIGURATION=OKV';
```

Creating Your Own TDE Master Encryption Key

- Create and then use your own TDE master encryption key by providing raw binary data:

```
SQL> ADMINISTER KEY MANAGEMENT CREATE KEY
       '10203040506032F88967A5419662A6F4E460E892318E307F017BA048707B402493C'
        USING ALGORITHM 'SEED128' FORCE KEYSTORE
        IDENTIFIED BY "WELcome_12" WITH BACKUP;
```

```
SQL> ADMINISTER KEY MANAGEMENT USE KEY
        'ARAgMEBQYHCAERITFBUWFxgAAAAAAAAAAAAAAAAAAAAAAAAAAAAA'
        IDENTIFIED BY "WELcome_12" WITH BACKUP;
```

- Or, create and activate your TDE master encryption key:

```
SQL> ADMINISTER KEY MANAGEMENT SET KEY
        '102030406070801112131415161718:3D432109DF1967062A6F4E460E892319c'
        USING ALGORITHM 'SEED128'
        IDENTIFIED BY "WELcome_12" WITH BACKUP;
```

Protecting Fixed-User Database Links Obfuscated Passwords

How to prevent an intruder from decrypting an obfuscated database link password?

- 12c Passwords for DB links are stored obfuscated in the database.
- 19c Passwords for DB links are not exported, being replaced with ‘x’.
  - Set the `COMPATIBLE` initialization parameter to 18.1.0.0.
  - Open the CDB root keystore and PDB isolated keystores if necessary.
  - Enable credentials encryption in the dictionary: at CDB root or PDB level:

```
SQL> ALTER DATABASE DICTIONARY ENCRYPT CREDENTIALS;
```

  - Display the enforcement status of the credentials encryption:

```
DICTIONARY_CREDENTIALS_ENCRYPT
   ENFORCEMENT = ENABLED | DISABLED
```

  - Display the usability of encrypted passwords of database links:

```
DBA_DB_LINKS
   VALID = YES | NO
```

Importing Fixed-User Database Links Encrypted Passwords

In the dump file of Data Pump export:

12c ➔ Obfuscated values for DB link passwords

- Passwords not protected
- Unless exported with `ENCRYPTION_PWD_PROMPT=YES`

19c If credentials encryption enabled in the dictionary:

➔ Invalid value for DB link password in the dump file

➔ Warning message during export and import

```
$ expdp …
ORA-39395: Warning: object SYSTEM.TEST requires password reset after import
```

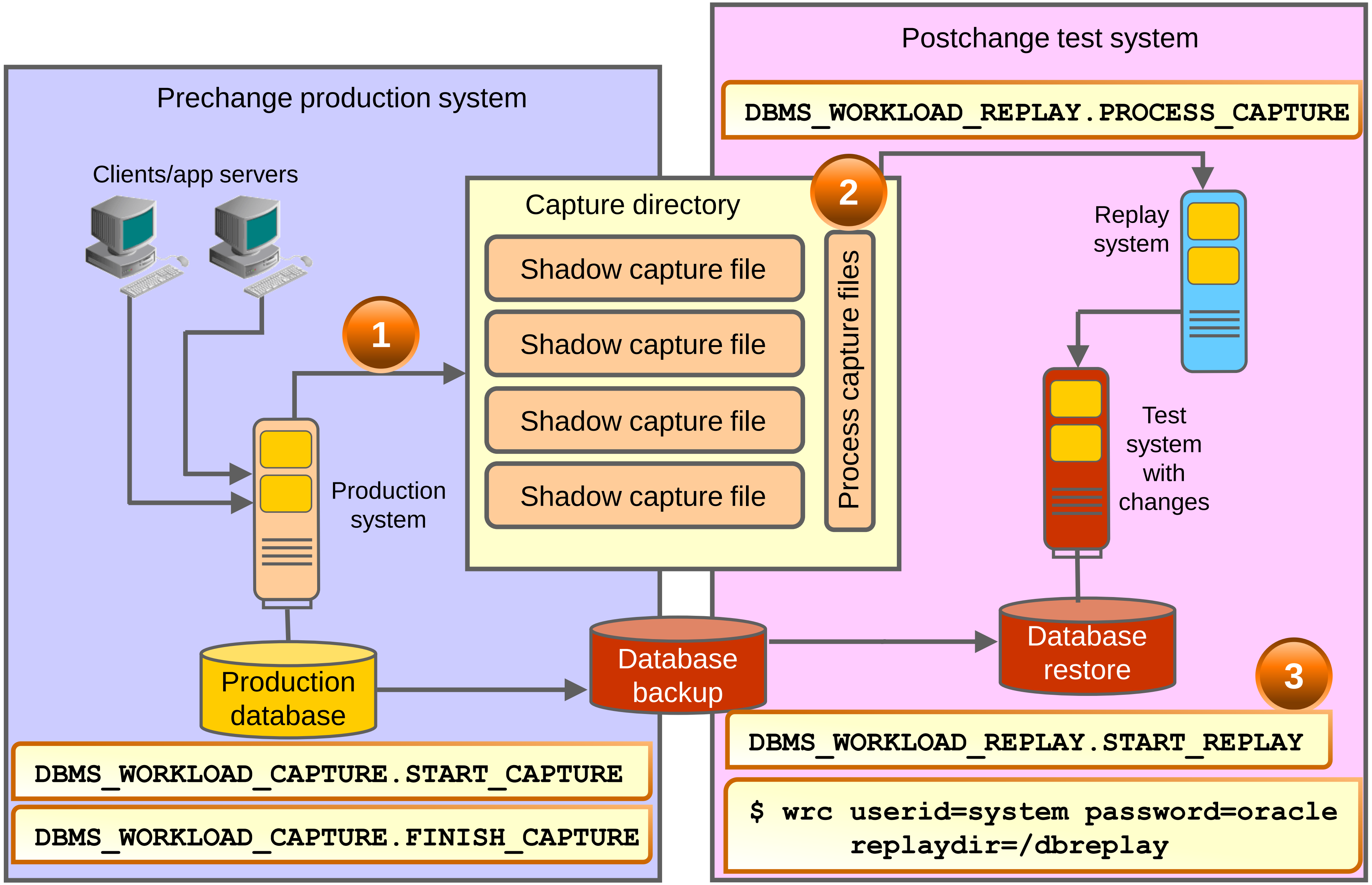
➔ Reset password for the DB link after import

```
SQL> ALTER DATABASE LINK lk1 CONNECT TO test IDENTIFIED BY <reset_password>;
```

DB Replay: The Big Picture

In capture files, clear byte strings for:

- Connection
- SQL text
- Bind values



```
DBMS_WORKLOAD_REPLAY.PROCESS_CAPTURE
```

```
DBMS_WORKLOAD_CAPTURE.START_CAPTURE
```

```
DBMS_WORKLOAD_CAPTURE.FINISH_CAPTURE
```

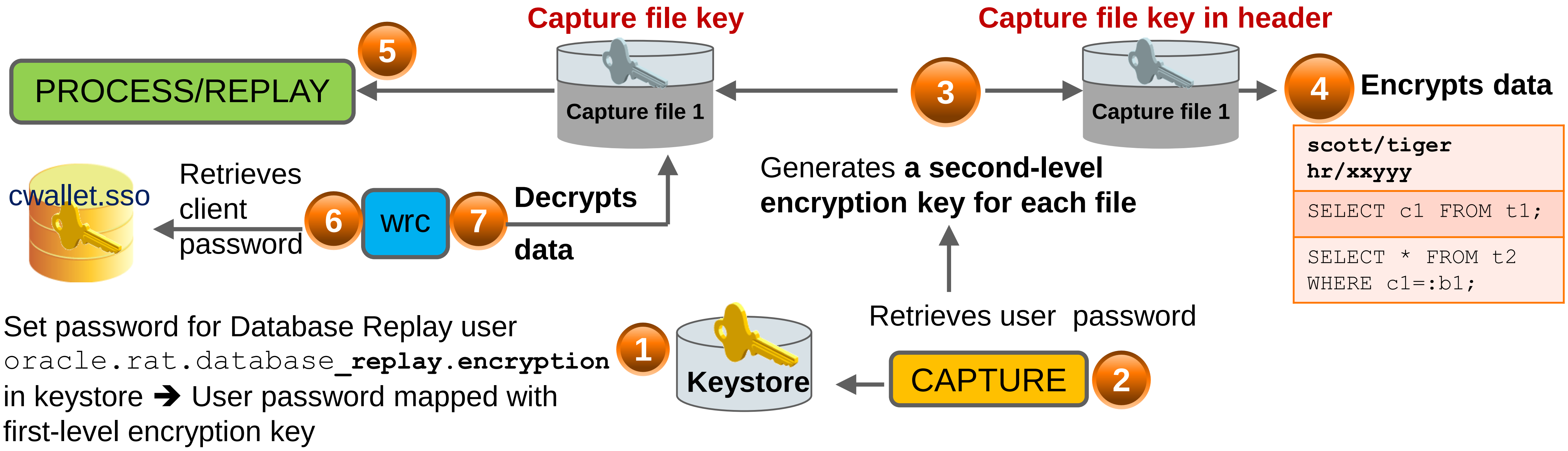
```
DBMS_WORKLOAD_REPLAY.START_REPLAY
```

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

Encryption of Sensitive Data in Database Replay Files

19c

- Protect data in capture files with TDE encryption.
- The keystore is used to:
  - Store the `oracle.rat.database_replay.encryption` user password
  - Retrieve the `oracle.rat.database_replay.encryption` user password during workload capture, process, and replay



Generates **a second-level encryption key for each file**

```
scott/tiger
hr/xxyyy
```

```
SELECT c1 FROM t1;
```

```
SELECT * FROM t2
WHERE c1=:b1;
```

Set password for Database Replay user
`oracle.rat.database_replay.encryption`
in keystore ➔ User password mapped with first-level encryption key

Capture Setup for DB Replay

```
DBA_WORKLOAD_CAPTURES
   ENCRYPTION = AES256
```

1. Set a password for the Database Replay user `oracle.rat.database_replay.encryption` in the keystore.

```
SQL> ADMINISTER KEY MANAGEMENT
           ADD SECRET '<replaypass>'
           FOR CLIENT 'oracle.rat.database_replay.encryption'
           IDENTIFIED BY <pass> WITH BACKUP;
```

2. Start the capture by using an encryption algorithm.

```
SQL> exec DBMS_WORKLOAD_CAPTURE.START_CAPTURE( NAME => 'OLTP_peak', -
                                               DIR => 'OLTP', -
                                               ENCRYPTION => 'AES256')
```

3. Stop the capture.

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

Process and Replay Setup for DB Replay – Phase 1

4. Process the capture after moving the capture files to the testing server environment.

```
SQL> exec DBMS_WORKLOAD_REPLAY.PROCESS_CAPTURE(capture_dir => 'OLTP')
```

5. Initialize the replay after setting various replay parameters.

```
SQL> exec DBMS_WORKLOAD_REPLAY.INITIALIZE_REPLAY(replay_name => 'R',
replay_dir => 'OLTP')
```

6. Prepare the replay to start.

```
SQL> exec DBMS_WORKLOAD_REPLAY.PREPARE_REPLAY ()
```

Process and Replay Setup for DB Replay – Phase 2

7. On the client side, set up a client-side keystore including the `oracle.rat.database_replay.encryption` client password.

```
$ mkdir /tmp/replay_encrypt_cwallet
$ mkstore -wrl /tmp/replay_encrypt_cwallet -create
$ mkstore -wrl /tmp/replay_encrypt_cwallet -createEntry
'oracle.rat.database_replay.encryption' "replaypass"
```

8. Start the replay clients.

```
$ wrc REPLAYDIR=/tmp/dbreplay USERID=system WALLETDIR= tmp/replay_encrypt_cwallet
Workload Replay Client: …
Wai
```

```
$ wrc REPLAYDIR=/tmp/dbreplay USERID=system WALLETDIR= tmp/replay_encrypt_cwallet
Workload Replay Client: …
Wait for the replay to start (21:47:01)
```

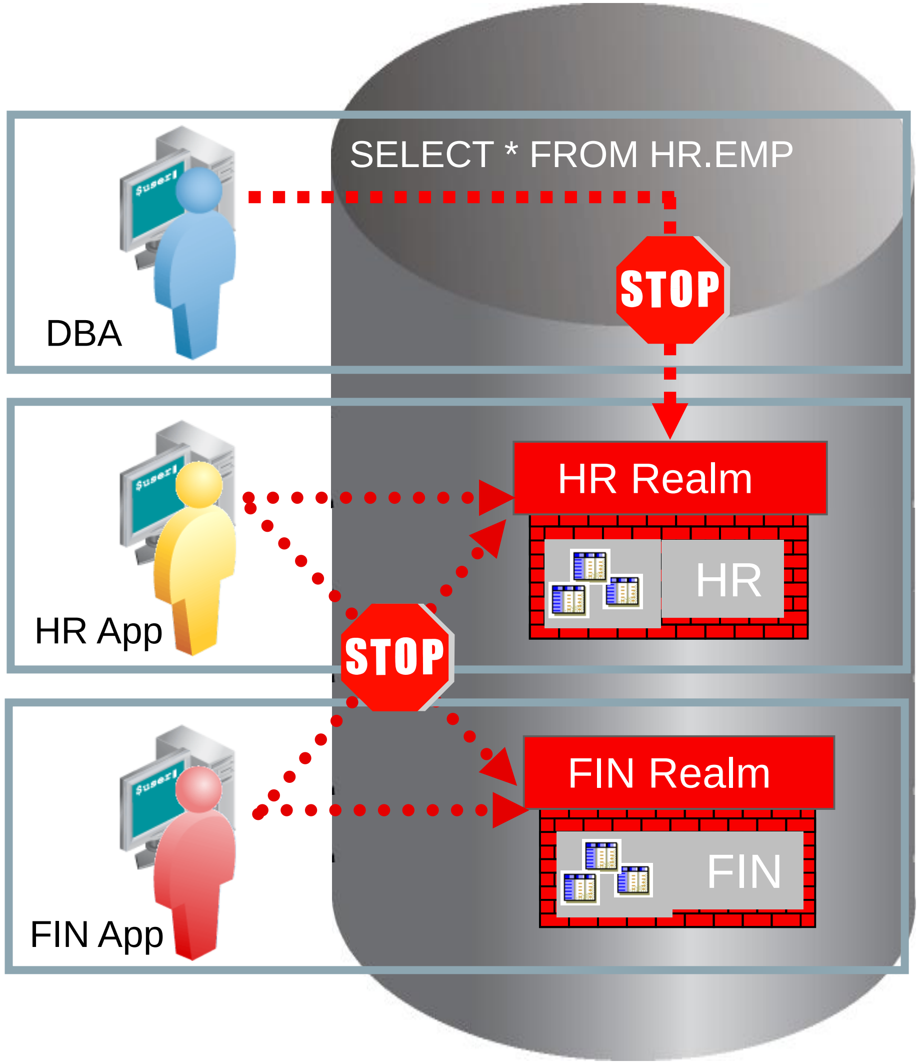
9. Start replaying for the clients waiting in step 8.

```
SQL> exec DBMS_WORKLOAD_REPLAY.START_REPLAY ()
```

12c

Oracle Database Vault: Privileged User Controls

- Database DBA is blocked from viewing HR data:
  - Compliance and protection from insiders
- HR App Owner is blocked from viewing FIN data:
  - Eliminates security risks from server consolidation



Database Vault: Access Control Components

The following components of Database Vault provide highly configurable access control:

- Realms and authorization types
  - Participant
  - Owner
- Command rules
- Rule sets
- Secure application roles
- Factors

1. The DBA can view the ORDERS table data.

```
SQL> SELECT order_total FROM oe.orders
     WHERE customer_id = 101;

ORDER_TOTAL
-----------
    78279.6
```

2. The security manager protects the OE.ORDERS table with a realm.
3. The DBA can no longer view the ORDERS table data.

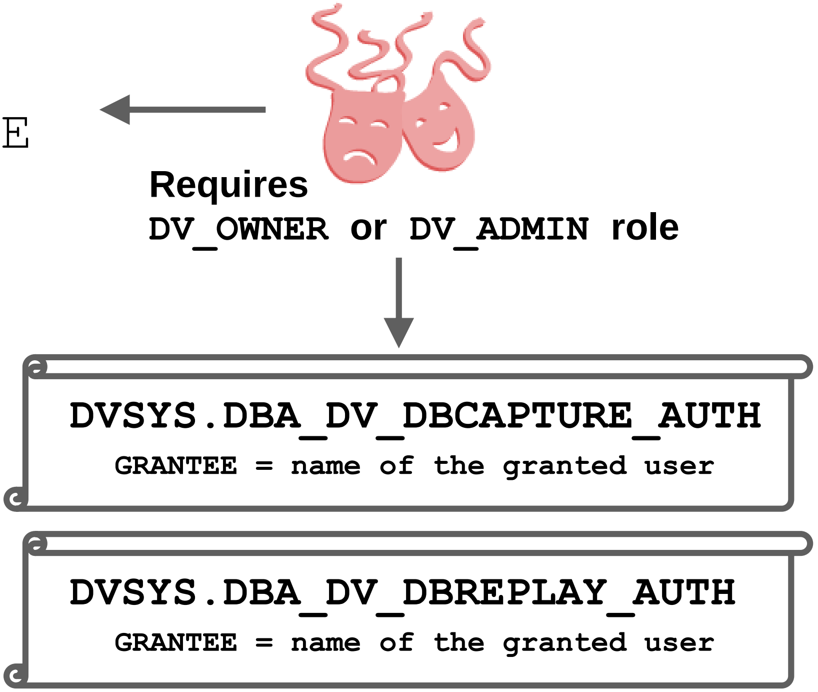
```
SQL> SELECT order_total FROM oe.orders
     WHERE customer_id = 101;
SELECT order_total FROM oe.orders
                           *
ERROR at line 1:
ORA-01031: insufficient privileges
```

DB Replay Capture and Replay with Database Vault

19c

New realm authorization types to allow users to run DB Replay capture and replay:

- `DBCAPTURE` authorization type
- `DBREPLAY` authorization type
- Managed using Database Vault admin procedures:
  - `DVSYS.DBMS_MACADM.AUTHORIZE_DBCAPTURE`
  - `DVSYS.DBMS_MACADM.UNAUTHORIZE_DBCAPTURE`
  - `DVSYS.DBMS_MACADM.AUTHORIZE_DBREPLAY`
  - `DVSYS.DBMS_MACADM.UNAUTHORIZE_DBREPLAY`



Requires `DV_OWNER` or `DV_ADMIN` role

`DVSYS.DBA_DV_DBCAPTURE_AUTH`

`GRANTEE = name of the granted user`

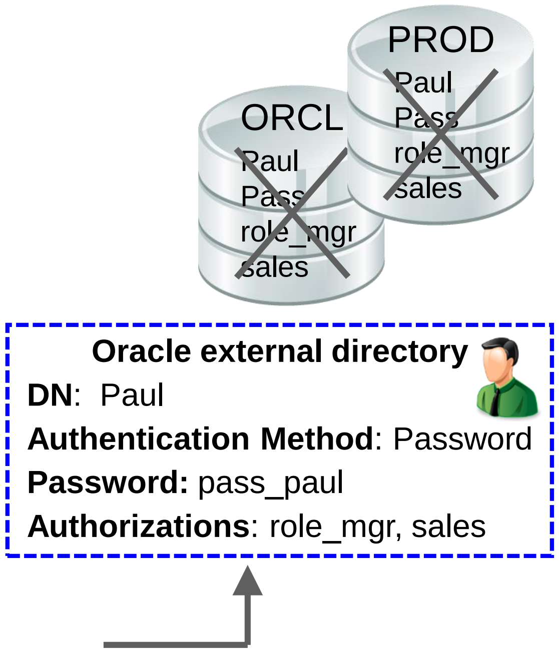
`DVSYS.DBA_DV_DBREPLAY_AUTH`

`GRANTEE = name of the granted user`

Authenticating and Authorizing Users with External Directories

External directories store user credentials and authorizations in a central location (LDAP-compliant directory, such as OID, OUD, and OVD).

- Eases administration through centralization
- Enables single-point authentication
- Eliminates the need for client-side wallets



Oracle external directory

DN: Paul

Authentication Method: Password

Password: pass_paul

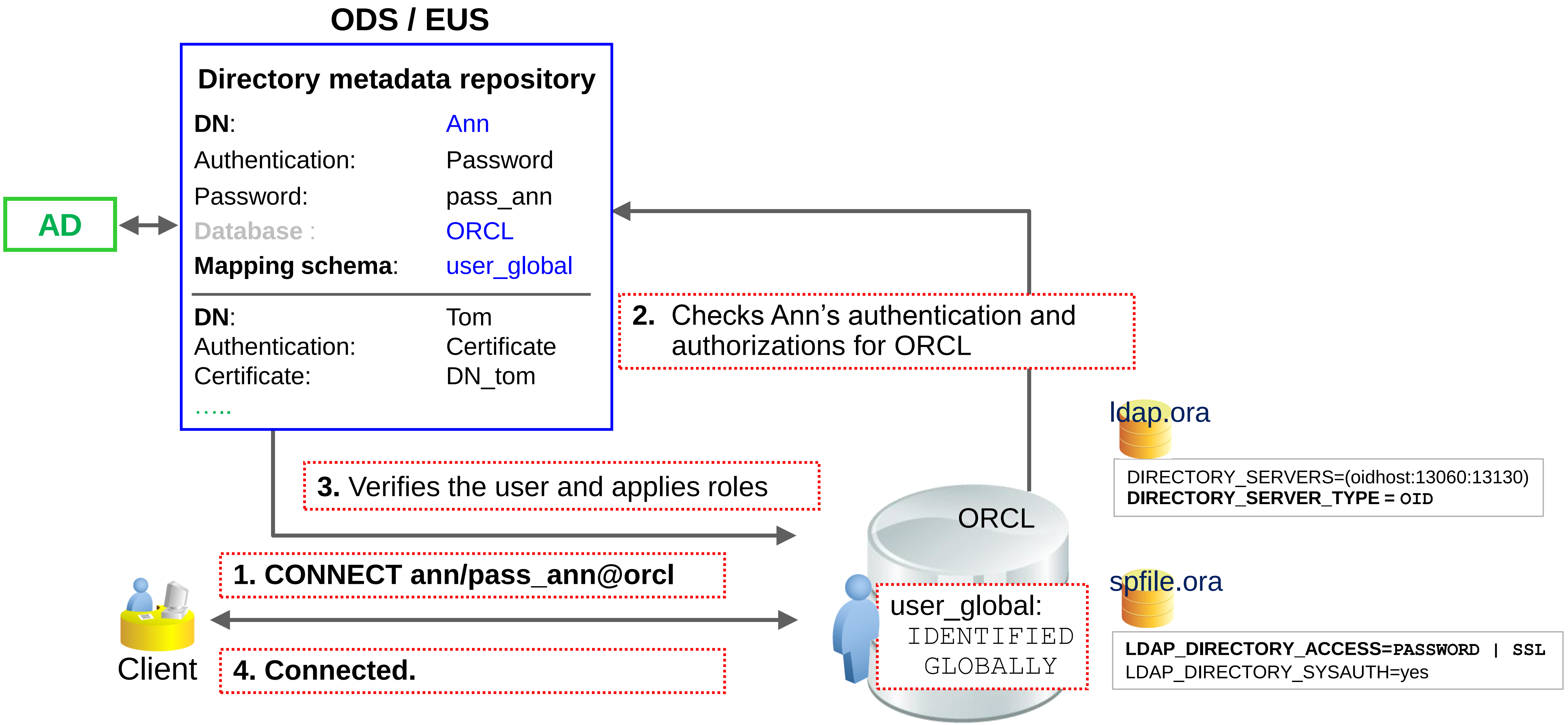
Authorizations: role_mgr, sales

Example:

- User changes job roles.
- Security administrator changes
  the user roles in Oracle Virtual Directory.
- No changes are made to the services that the user accesses.

Architecture

12c



ODS / EUS

Directory metadata repository

DN:	Ann
Authentication:	Password
Password:	pass_ann
Database :	ORCL
Mapping schema:	user_global
DN:	Tom
Authentication:	Certificate
Certificate:	DN_tom
…..	

1. CONNECT ann/pass_ann@orcl

2. Checks Ann’s authentication and authorizations for ORCL

3. Verifies the user and applies roles

4. Connected.

```
user_global:
 IDENTIFIED
  GLOBALLY
```

ldap.ora

```
DIRECTORY_SERVERS=(oidhost:13060:13130)
DIRECTORY_SERVER_TYPE = OID
```

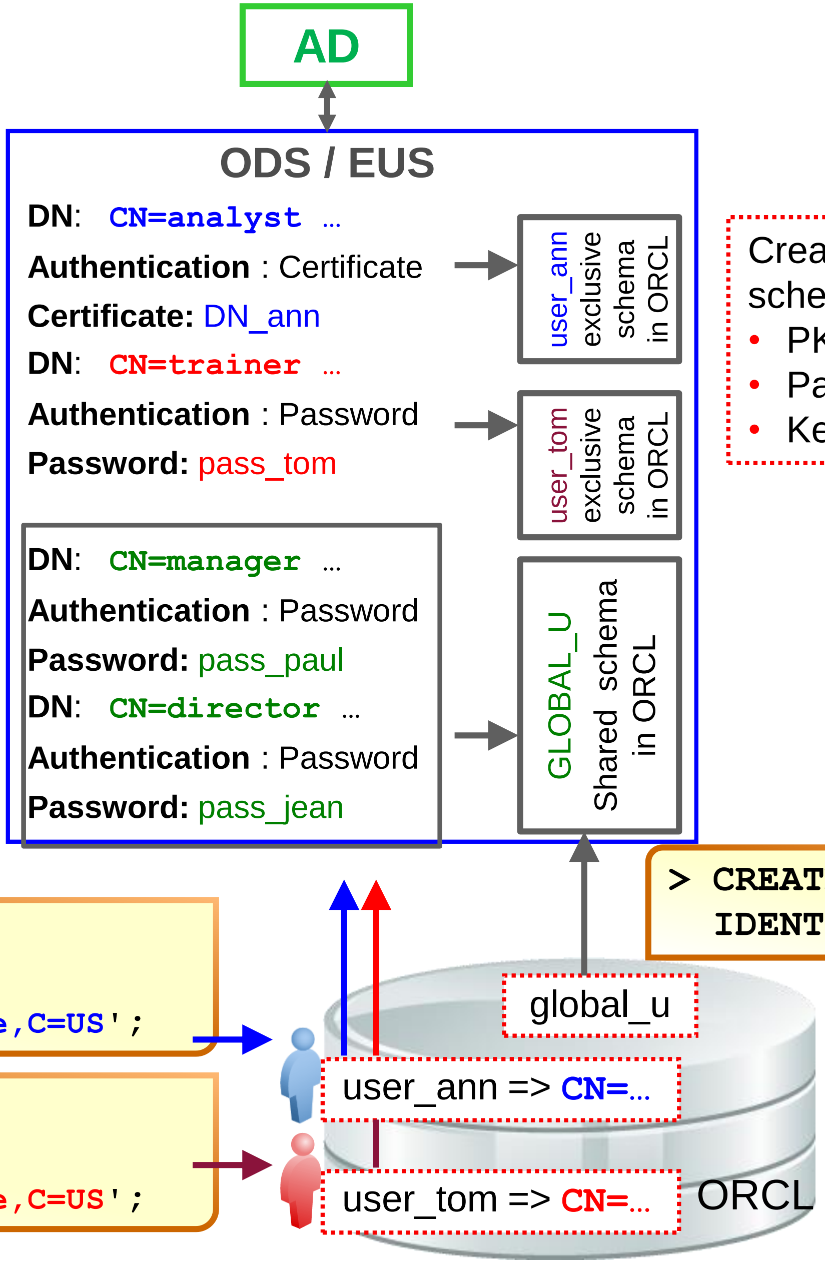
spfile.ora

```
LDAP_DIRECTORY_ACCESS=PASSWORD | SSL
LDAP_DIRECTORY_SYSAUTH=yes
```

EUS and AD

Create **exclusive global** schemas authenticated by:

- PKI certificates
- Passwords
- Kerberos tickets



AD

ODS / EUS

DN: CN=analyst …
Authentication : Certificate
Certificate: DN_ann
DN: CN=trainer …
Authentication : Password
Password: pass_tom

DN: CN=manager …
Authentication : Password
Password: pass_paul
DN: CN=director …
Authentication : Password
Password: pass_jean

user_ann exclusive schema in ORCL
user_tom exclusive schema in ORCL
GLOBAL_U Shared schema in ORCL
global_u
user_ann => CN=…
user_tom => CN=…
ORCL

Create **shared global** schemas authenticated by:

- PKI certificates
- Passwords
- Kerberos tickets

```
> CREATE USER user_ann
IDENTIFIED GLOBALLY AS
'CN=analyst,OU=div1,O=oracle,C=US';
```

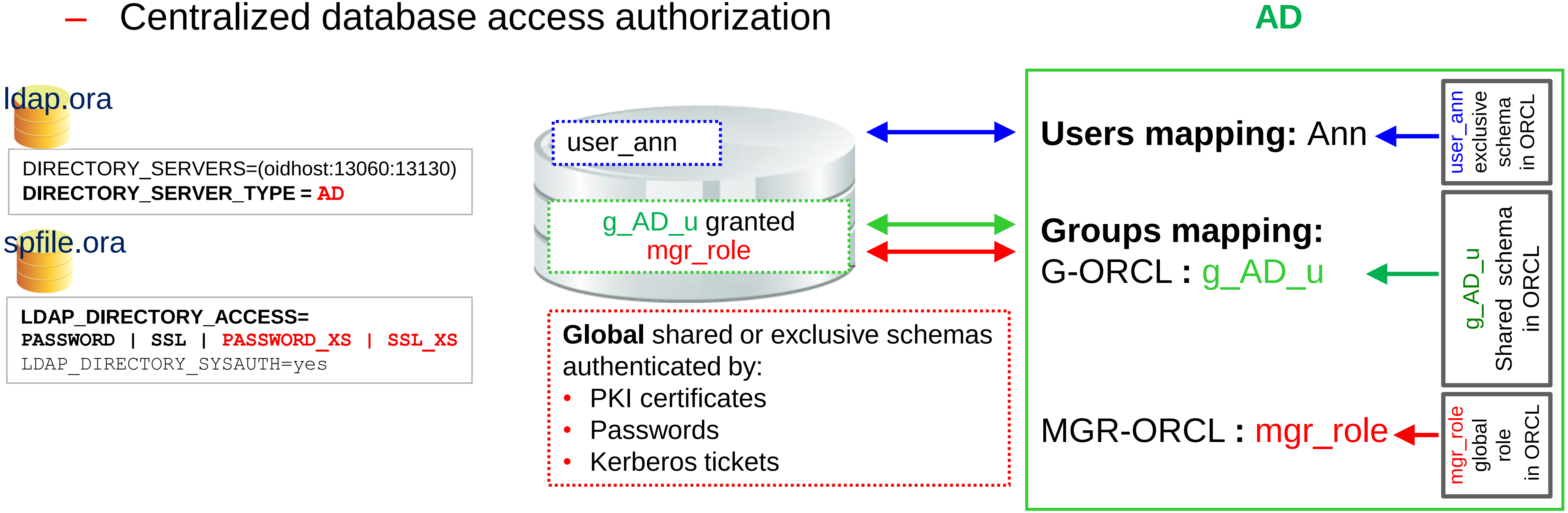
```
> CREATE USER user_tom
IDENTIFIED GLOBALLY AS
'CN=trainer,OU=div2,O=oracle,C=US';
```

```
> CREATE USER global_u
  IDENTIFIED GLOBALLY;
```

CMU and AD

```
DBA_USERS
DBA_ROLES
   EXTERNAL_NAME
```

- 12c Deploy and synchronize database user credentials and authorizations with ODS/EUS first.
- 19c Deploy database user credentials and authorizations directly in Active Directory with Centrally Managed Users (CMU):
  - Centralized database user authentication
  - Centralized database access authorization



AD

ldap.ora

```
DIRECTORY_SERVERS=(oidhost:13060:13130)
DIRECTORY_SERVER_TYPE = AD
```

spfile.ora

```
LDAP_DIRECTORY_ACCESS=
PASSWORD | SSL | PASSWORD_XS | SSL_XS
LDAP_DIRECTORY_SYSAUTH=yes
```

user_ann

g_AD_u granted mgr_role

Global shared or exclusive schemas authenticated by:

- PKI certificates
- Passwords
- Kerberos tickets

Users mapping: Ann

user_ann exclusive schema in ORCL

Groups mapping:

G-ORCL : g_AD_u

g_AD_u Shared schema in ORCL

MGR-ORCL : mgr_role

mgr_role global role in ORCL

Choosing Between EUS and CMU

		EUS	CMU
Simplified Implementation			✓
Authentication	Password, Kerberos, PKI certificates	✓	✓
	Enforce directory account policies	✓	✓
Authorization	Role authorization	✓	✓
	Administrative users	✓	✓
	Shared DB schema mapping	✓	✓
	Exclusive schema mapping	✓	✓
Enterprise Domains	Current User trusted DB link	✓	
	Integrated with Oracle Label Security, XDB	✓	
	Consolidated reporting and management of data access	✓	

Summary

In this lesson, you should have learned how to:

- Create schema-only accounts
- Isolate a new PDB keystore
- Convert a PDB to run in isolated or united mode
- Migrate PDB keystore between keystore types
- Create user-defined TDE master keys
- Protect fixed-user database link passwords
- Export and import fixed-user database links with encrypted passwords
- Configure encryption of sensitive data in Database Replay files
- Perform Database Replay capture and replay in a database with Database Vault
- Explain Enterprise users integration with Active Directory

Practice 3: Overview

- 3-1: Creating schema-only accounts
- 3-2: Managing PDB keystores
- 3-3: Creating user-defined TDE master keys
- 3-4: Exporting and importing fixed-user database links
- 3-5: Encrypting sensitive data in DB Replay files

4. Using RMAN Enhancements

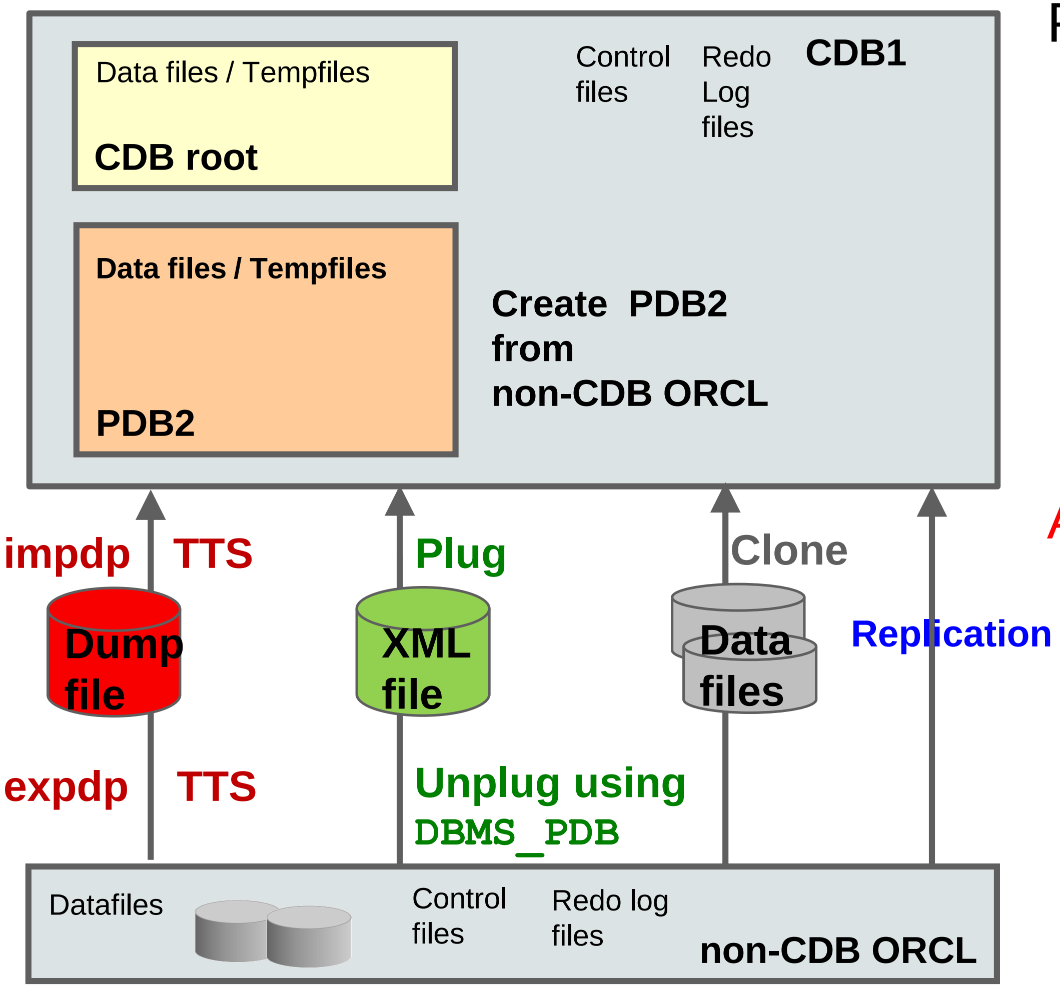
Objectives

After completing this lesson, you should be able to:

- Reuse preplugin backups after conversion of a non-CDB to a PDB
- Reuse preplugin backups after plugging/relocating a PDB into another CDB
- Duplicate an active PDB into an existing CDB
- Duplicate a CDB as encrypted
- Duplicate a CDB as decrypted
- Recover a standby database from primary

Migrating a Non-CDB to a CDB

12c



Possible methods:

- Data Pump (TTS or TDB or full export/import)
- Plugging (XML file definition with `DBMS_PDB`)
- Cloning
- Replication

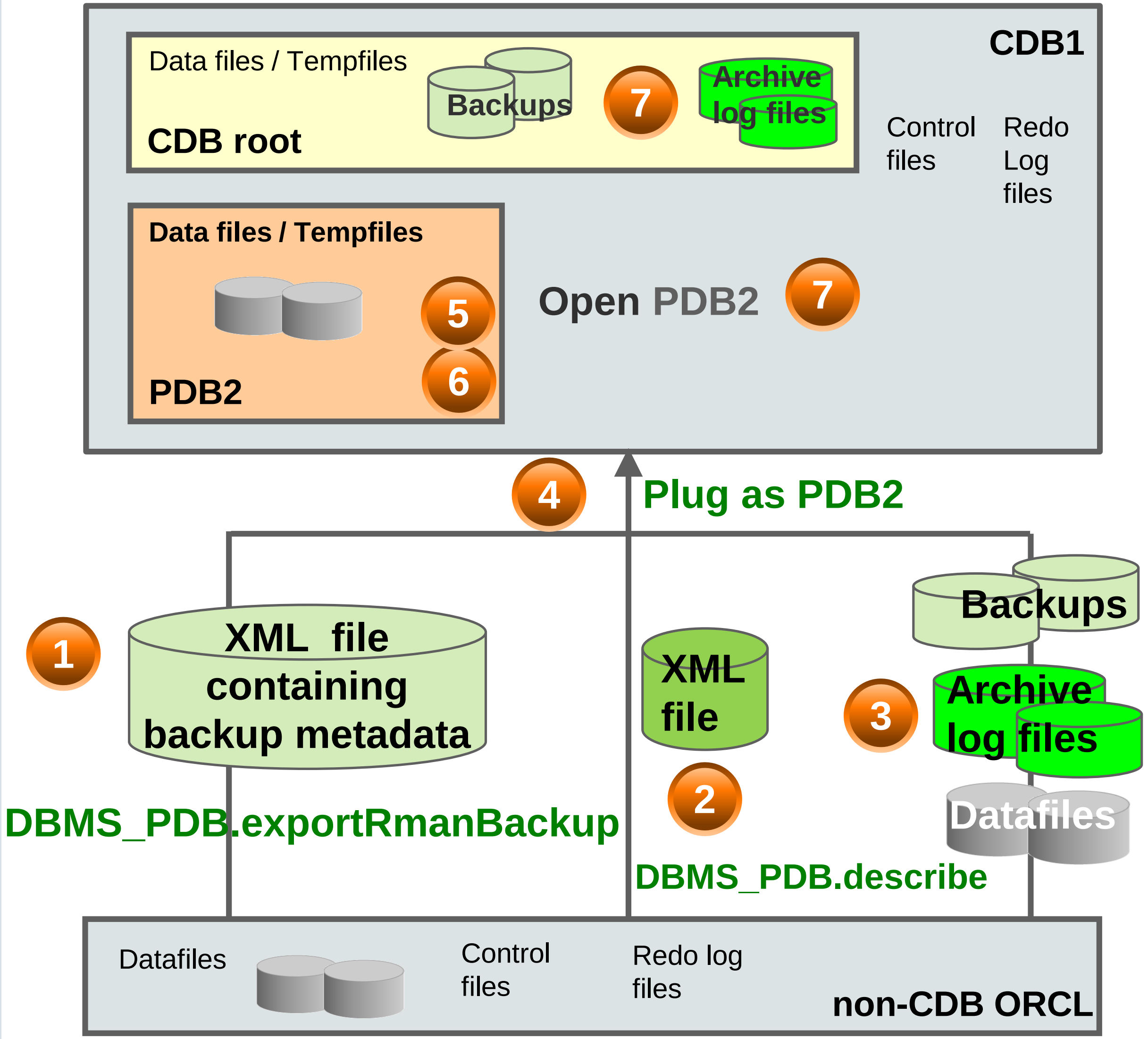
After conversion:

- Is it possible to recover the PDB back in time before the non-CDB was converted?
- Are the non-CDB backups transported with the non-CDB?

?

Migrating a Non-CDB and Transporting Non-CDB Backups to a CDB

19c



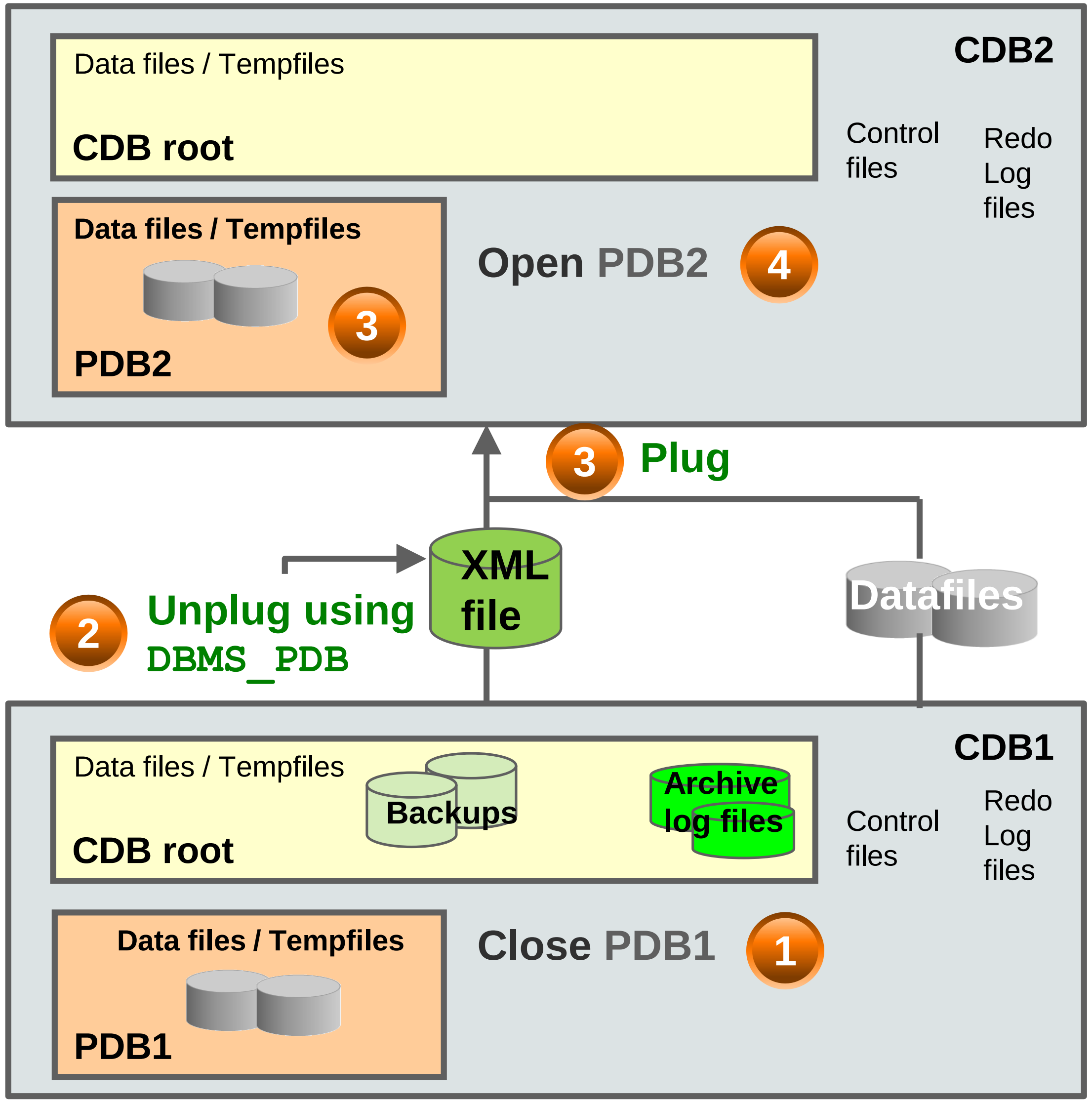
1. Export backups metadata with `DBMS_PDB.exportRmanBackup`.
2. Unplug the non-CDB using `DBMS_PDB.describe`.
3. Archive the current redo log file.
4. Transfer data files including backups to the target CDB.
5. Plug using the XML file.
6. Execute the `noncdb_to_pdb.sql` script.
7. Open PDB. This automatically imports backups metadata into the CDB dictionary.

Restore/recover the PDB with preplugin backups:

1. Catalog the archived redo log file.
2. Restore PDB using preplugin backups.
3. Recover PDB using preplugin backups.

Relocating/Plugging a PDB into Another CDB

12c



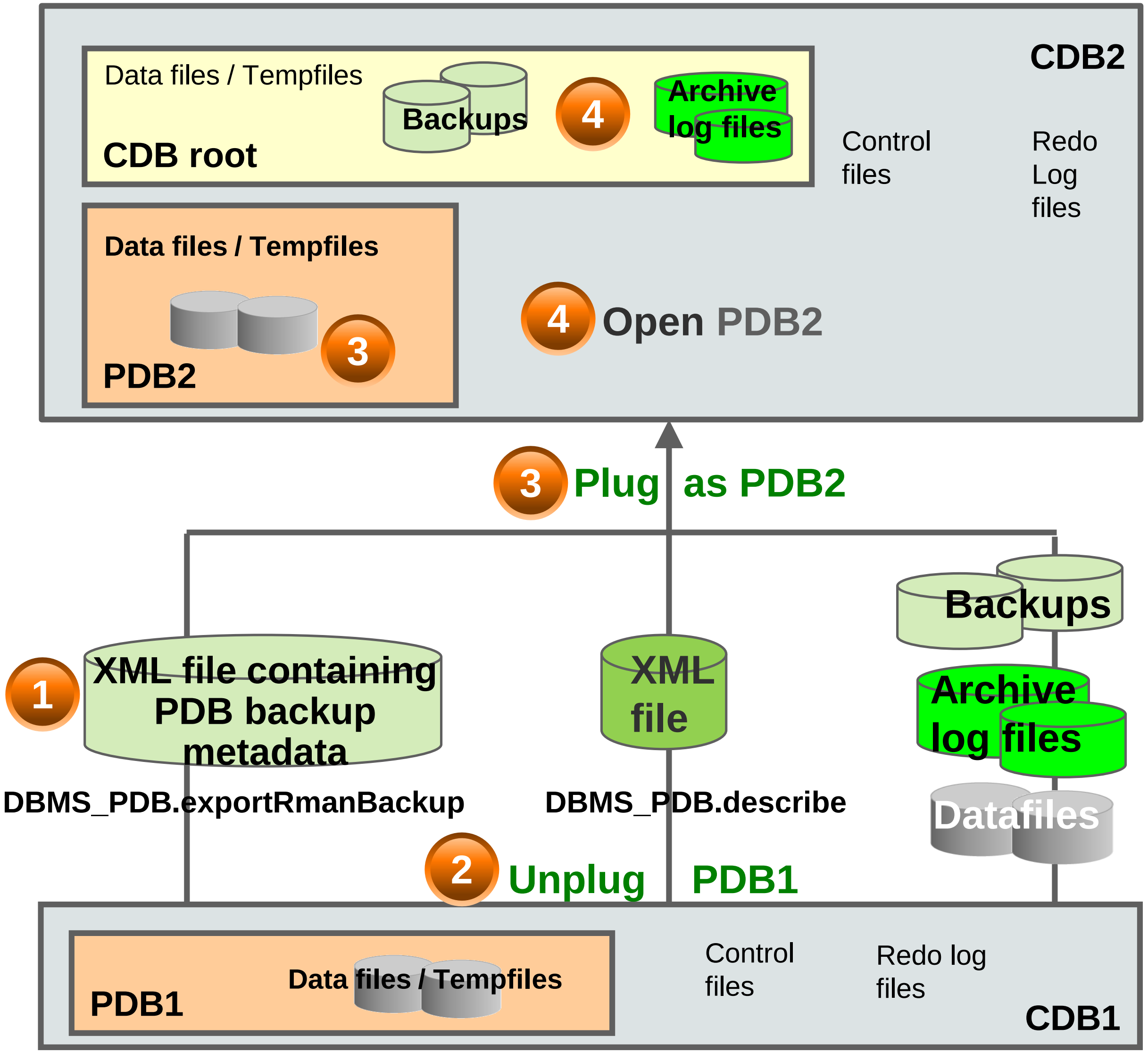
After relocating/plugging the PDB into another CDB:

- Is it possible to recover the PDB back in time before it was relocated/unplugged?
- Are the PDB backups transported with the relocated/unplugged PDB?

?

Plugging a PDB and Transporting PDB Backups to a CDB - 1

19c



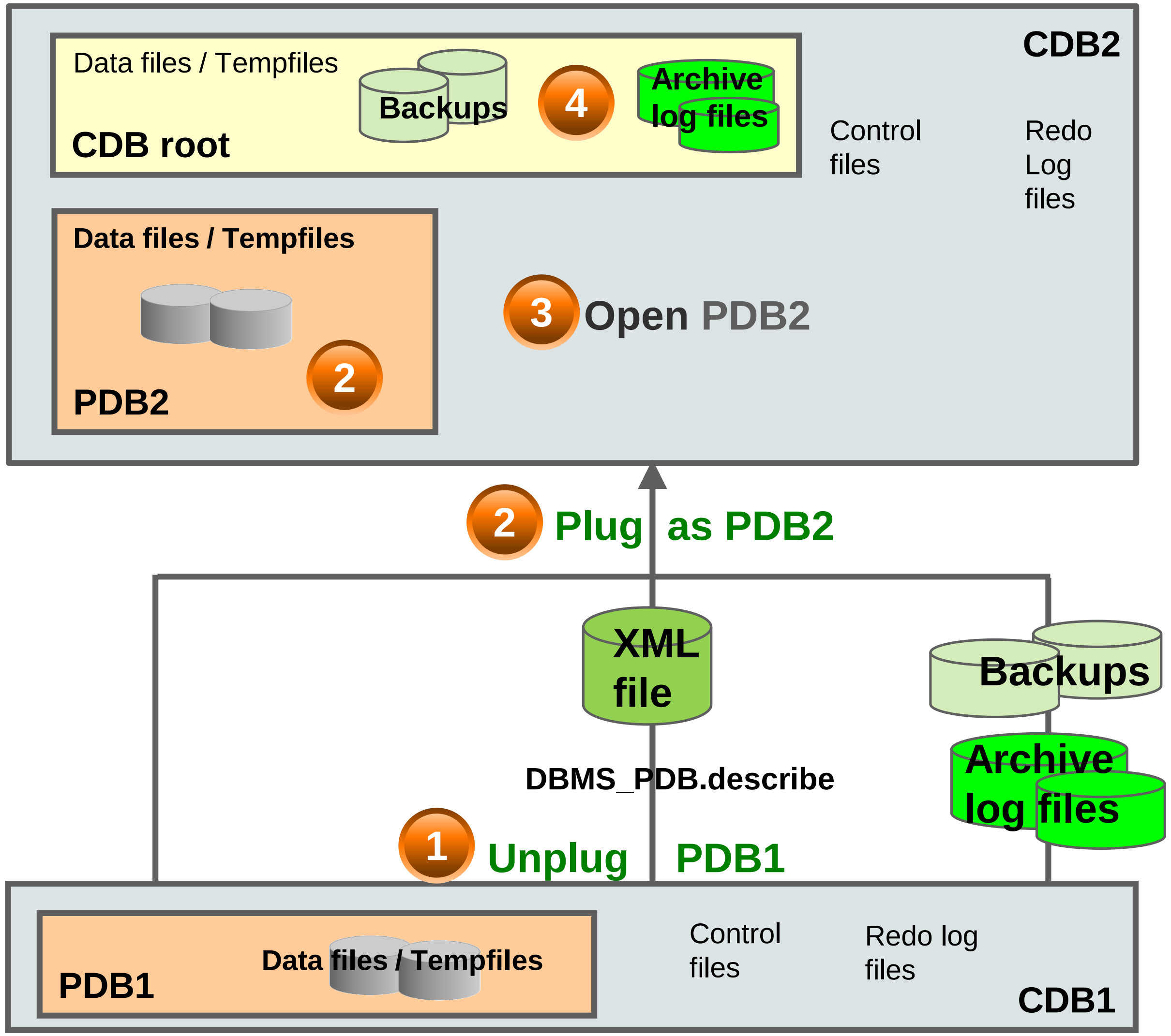
1. Export backups metadata by using `DBMS_PDB.exportRmanBackup`.
2. Unplug the PDB by using `DBMS_PDB.describe`.
3. Transfer the data files including backups to the target CDB.
4. Plug using the XML file.
5. Open PDB. This automatically imports backups metadata into the CDB dictionary.

Then you can restore/recover the PDB by using the transported backups:

1. Restore PDB using preplugin backups.
2. Recover PDB using preplugin backups.

Plugging a PDB and Transporting PDB Backups to a CDB - 2

19c



1. Unplug the PDB with `DBMS_PDB.describe`.
2. Transfer the datafiles including backups to the target CDB.
3. Plug using the XML file.
4. Open PDB.
5. Catalog preplugin backups into CDB.

Then you can restore/recover the PDB using the transported backups:

1. Restore PDB using preplugin backups.
2. Recover PDB using preplugin backups.

Using PrePlugin Backups

Use the `PrePlugin` option to perform RMAN operations using preplugin backups.

- Restore a PDB from its preplugin backups cataloged in the target CDB.

```
RMAN> RESTORE PLUGGABLE DATABASE pdb_noncdb FROM PREPLUGIN;
```

- Recover a PDB from its preplugin backups until the datafile was plugged in.

```
RMAN> RECOVER PLUGGABLE DATABASE pdb_noncdb FROM PREPLUGIN;
```

- Check whether preplugin backups and archive log files are cataloged in the target CDB.

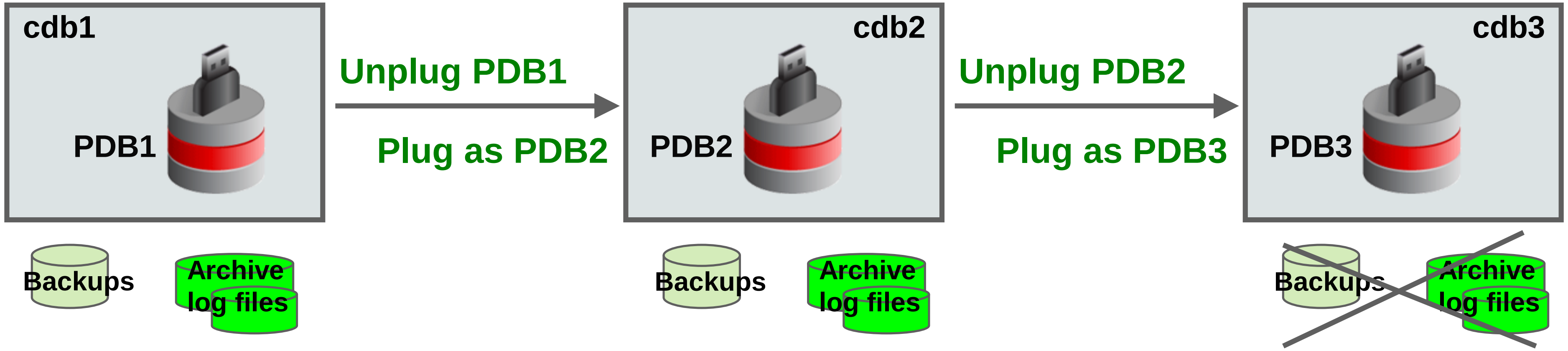
```
RMAN> SET PREPLUGIN CONTAINER pdb1;
RMAN> LIST PREPLUGIN BACKUP;
RMAN> LIST PREPLUGIN ARCHIVELOG ALL;
RMAN> LIST PREPLUGIN COPY;
```

- Verify that cataloged preplugin backups are available on disk.

```
RMAN> CROSSCHECK PREPLUGIN BACKUP;
RMAN> DELETE PREPLUGIN BACKUP;
```

To Be Aware Of

- The source and destination CDBs must have COMPATIBLE set to 18.1 or higher to create/restore/recover preplugin backups.
- In case of plugging in a non-CDB, the non-CDB must use ARCHIVELOG mode.
- The target CDB does not manage preplugin backups.
  - Use CROSSCHECK and DELETE commands to manage the preplugin backups.
- A RESTORE using preplugin backups can restore datafiles from one PDB only.
- Backups taken by the source cdb1 are visible in target cdb2 only.



Example

```
RMAN> SET PREPLUGIN CONTAINER pdb1;
RMAN> CATALOG PREPLUGIN ARCHIVELOG '/u03/app/…/o1_mf_1_8_dnqwm59v_.arc';
RMAN> RUN { RESTORE PLUGGABLE DATABASE pdb1 FROM PREPLUGIN;
            RECOVER PLUGGABLE DATABASE pdb1 FROM PREPLUGIN;
          }
RMAN> RECOVER PLUGGABLE DATABASE pdb1;
```

Cloning Active PDB into Another CDB Using DUPLICATE

Use the DUPLICATE command to create a copy of a PDB or subset of a PDB.

12c Duplicate a CDB or PDBs or PDB tablespaces in active mode to a fresh auxiliary instance.

```
RMAN> DUPLICATE TARGET DATABASE TO cdb1 PLUGGABLE DATABASE pdb1b;
```

19c Duplicate a PDB or PDB tablespaces in active mode to an existing opened CDB.

- Set the COMPATIBLE initialization parameter to 18.1.
- Clone only one PDB at a time.
- Set the destination CDB in RW mode.
- Set the REMOTE_RECOVERY_FILE_DEST initialization parameter in the destination CDB to the location where to restore foreign archive log files.

```
RMAN> DUPLICATE PLUGGABLE DATABASE pdb1 AS pdb2 FROM ACTIVE DATABASE
                          DB_FILE_NAME_CONVERT ('cdb1', 'cdb2');
```

Example: 1

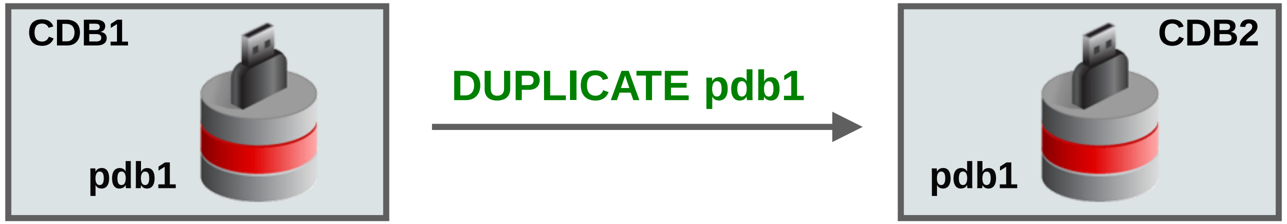
To duplicate `pdb1` from `CDB1` into `CDB2`:

1. Set the `REMOTE_RECOVERY_FILE_DEST` initialization parameter in `CDB2`.

```
SQL> ALTER SYSTEM SET REMOTE_RECOVERY_FILE_DEST='/dir_to_restore_archive log files';
```

2. Connect to the source (`TARGET` for `DUPLICATE` command): `CDB1`
3. Connect to the existing `CDB2` that acts as the auxiliary instance:

```
rman TARGET sys/password@cdb1 AUXILIARY sys/password@cdb2
```



4. Start duplicate.

```
RMAN> DUPLICATE PLUGGABLE DATABASE pdb1 TO cdb2 FROM ACTIVE DATABASE;
```

Example: 2

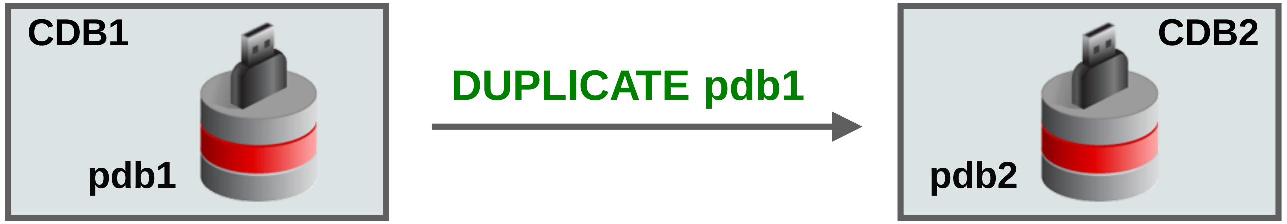
To duplicate pdb1 from CDB1 into CDB2:

1. Set the REMOTE_RECOVERY_FILE_DEST initialization parameter in CDB2.

```
SQL> ALTER SYSTEM SET REMOTE_RECOVERY_FILE_DEST='/dir_to_restore_archive log files';
```

2. Connect to the source (TARGET for DUPLICATE command): CDB1
3. Connect to the existing CDB2 that acts as the auxiliary instance:

```
rman TARGET sys/password@cdb1 AUXILIARY sys/password@cdb2
```



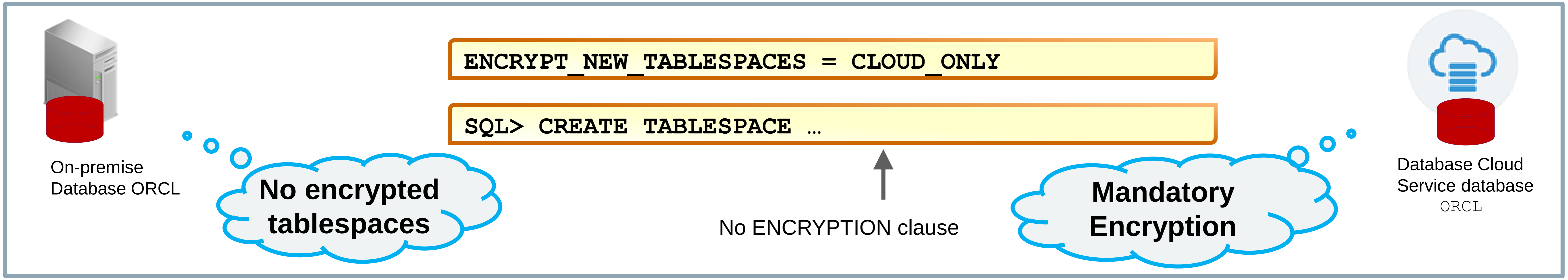
4. Start duplicate.

```
RMAN> DUPLICATE PLUGGABLE DATABASE pdb1 AS pdb2 TO cdb2 FROM ACTIVE DATABASE;
```

Duplicating On-Premise CDB as Cloud Encrypted CDB

Duplicating an on-premise CDB to the Cloud:

- Any newly created tablespace is encrypted in the Cloud CDB.



```
ENCRYPT_NEW_TABLESPACES = CLOUD_ONLY
```

```
SQL> CREATE TABLESPACE …
```

On-premise
Database ORCL

No encrypted tablespaces

No ENCRYPTION clause

Mandatory Encryption

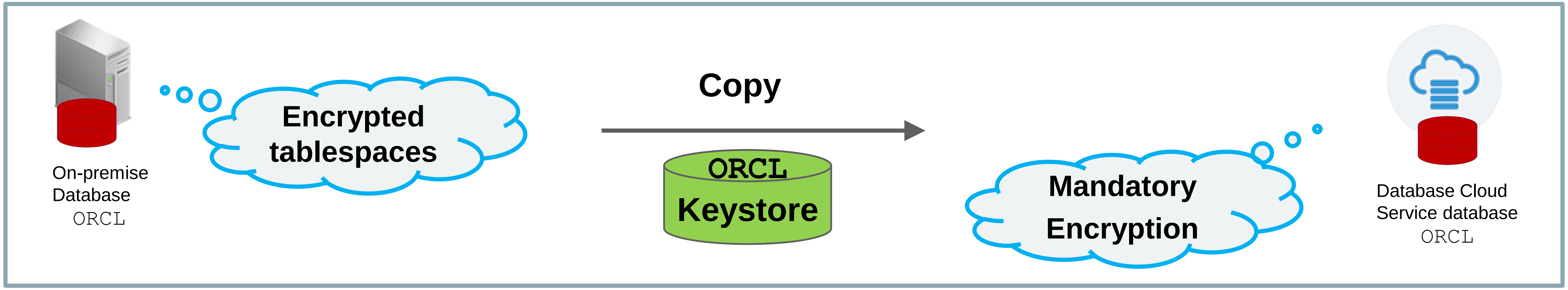
Database Cloud
Service database
ORCL

- The Cloud CDB holds a keystore because this is the default behavior on Cloud.
- All forms of normal duplication are compatible:
  - Active duplication
  - Backup-based duplication
  - Targetless duplicate

Duplicating On-Premise Encrypted CDB as Cloud Encrypted CDB

Duplicating an on-premise CDB with encrypted tablespaces to the Cloud:

1. Tablespaces of the source CDB need to be decrypted.

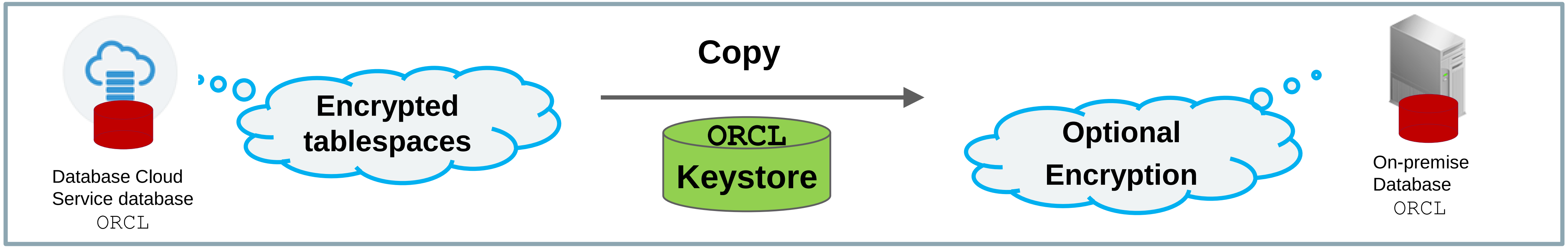


2. Restored tablespaces are re-encrypted in the Cloud CDB.
   - Requires the master TDE key from the source CDB keystore
   - Requires the source keystore to be copied and opened at the destination CDB

```
RMAN> SET DECRYPTION WALLET OPEN IDENTIFIED BY password;
RMAN> DUPLICATE DATABASE TO orcl FROM ACTIVE DATABASE AS ENCRYPTED;
```

Duplicating Cloud Encrypted CDB as On-Premise CDB

1. Tablespaces of the source CDB are necessarily encrypted.



2. Restored tablespaces need to be decrypted to be created:
   – Requires the TDE master key from the source CDB keystore
   – Requires the source keystore to be copied and opened at the destination CDB

```
RMAN> SET DECRYPTION WALLET OPEN IDENTIFIED BY password;
RMAN> DUPLICATE DATABASE TO orcl FROM ACTIVE DATABASE AS DECRYPTED;
```

Automated Standby Synchronization from Primary

12c A standby database might lag behind the primary for various reasons like:

- Unavailability or insufficient network bandwidth between primary and standby database
- Unavailability of standby database
- Corruption/accidental deletion of archive redo data on primary

➔ Manually restore the primary controlfile on standby after use of `RECOVER FROM SERVICE`.

19c `RECOVER FROM SERVICE` automatically rolls a standby forward:

1. Remember all datafile names on the standby.
2. Restart standby in nomount.
3. Restore controlfile from primary.
4. Mount standby database.
5. Rename datafiles from stored standby names.
6. Restore new datafiles to new names.
7. Recover standby.

Summary

In this lesson, you should have learned how to:

- Reuse preplugin backups after conversion of a non-CDB to a PDB
- Reuse preplugin backups after plugging a PDB into another CDB
- Duplicate an active PDB into an existing CDB
- Duplicate a CDB as encrypted
- Duplicate a CDB as decrypted
- Recover a standby database from primary

Practice 4: Overview

- 4-1: Recovering a plugged non-CDB using preplugin backups
- 4-2: Recovering a plugged PDB using preplugin backups
- 4-3: Duplicating a PDB into an existing CDB
- 4-4: Duplicating an on-premise CDB for Cloud

5. Using General Database Enhancements

Objectives

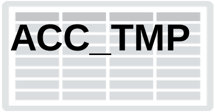
After completing this lesson, you should be able to:

- Manage private temporary tables
- Use the Data Pump Import `CONTINUE_LOAD_ON_FORMAT_ERROR` option of the `DATA_OPTIONS` parameter
- Perform online modification of partitioning and subpartitioning strategy
- Perform online `MERGE` partition and subpartition
- Generate batched DDL by using the `DBMS_METADATA` package
- Benefit from Unicode 9.0 support

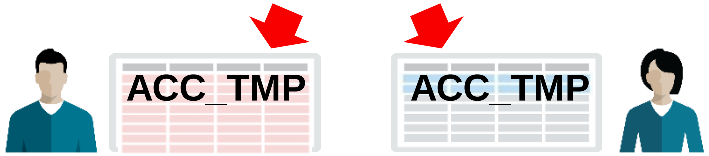
Global Temporary Tables

12c

- The definition of global temporary tables is visible by all sessions.



- Each session can see and modify only its own data.



```
SQL> CREATE GLOBAL TEMPORARY TABLE hr.employees_temp
            AS SELECT * FROM hr.employees;
```

- Global temporary tables retain data only for the duration of a transaction or session.
- DML locks are not acquired on the data.
- You can create indexes, views, and triggers on global temporary tables.
- Global temporary tables are created by using the `GLOBAL TEMPORARY` clause.

19c

Private Temporary Tables

USER_PRIVATE_TEMP_TABLES

Private Temporary Tables (PTTs) exist only for the session that creates them.

- You can create a PTT with the `CREATE PRIVATE TEMPORARY TABLE` statement.
- Table name must start with `ORA$PTT_` :

```
PRIVATE_TEMP_TABLE_PREFIX = ORA$PTT_
```

```
SQL> CREATE PRIVATE TEMPORARY TABLE ORA$PTT_mine (c1 DATE, … c3 NUMBER(10,2));
```

- The `CREATE PRIVATE TEMPORARY TABLE` statement does not commit a transaction.
- Two concurrent sessions may have a PTT with the same name but different shape.

ORA$PTT_mine
ORA$PTT_mine

- PTT definition and contents are automatically dropped at the end of a session or transaction.

```
SQL> CREATE PRIVATE TEMPORARY TABLE ORA$PTT_mine (c1 DATE …)
            ON COMMIT PRESERVE DEFINITION;
```

```
SQL> DROP TABLE ORA$PTT_mine;
```

Import with the CONTINUE_LOAD_ON_FORMAT_ERROR option

12c When import detects a format error in the data stream, it aborts the load.

- All table data for the current operation is rolled back.
- Solution: Either re-export and re-import or recover as much of the data as possible from the file with this corruption.

19c Importing with the CONTINUE_LOAD_ON_FORMAT_ERROR option:

- Detects a format error in the data stream while importing data
- Instead of aborting the import operation, resumes loading data at the next granule boundary
- Recovers at least some data from the dump file
- Is ignored for network mode import

```
$ impdp hr TABLES = employees DUMPFILE = dpemp DIRECTORY = dirhr
           DATA_OPTIONS = CONTINUE_LOAD_ON_FORMAT_ERROR
```

Online Partition and Subpartition Maintenance Operations

Improve the high-availability of data by supporting an online implementation of a number of frequently used DDLs:

- 11g `CREATE INDEX / ALTER TABLE ADD COLUMN | ADD CONSTRAINT`
- 12c `DROP INDEX / ALTER INDEX UNUSABLE / ALTER TABLE DROP CONSTRAINT | SET COLUMN UNUSED | MOVE | MOVE PARTITION | SPLIT PARTITION | MODIFY` nonpartitioned to partitioned | `MOVE PARTITION INCLUDING ROWS`

```
SQL> ALTER INDEX hr.i_emp_ix UNUSABLE ONLINE;
```

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

- 19c `ALTER TABLE MODIFY` to allow repartitioning a table, add or remove subpartitioning
- 19c `ALTER TABLE MERGE PARTITION`

Online Modification of Partitioning and Subpartitioning Strategy

- Prevents concurrent DDLs on the affected table, until the operation completes
- `ONLINE` clause: Does not hold a blocking X DML lock on the table being modified
- No tablespace defined for the partitions; defaults to the original table’s tablespace
- The `UPDATE INDEXES` clause:
  - Changes the partitioning state of indexes and storage properties of the indexes being converted
  - Cannot change the columns on which the original list of indexes are defined
  - Cannot change the uniqueness property of the index or any other index property
    - No tablespace defined for indexes:
      - Local indexes after the conversion collocate with the table partition.
      - Global indexes after the conversion reside in the same tablespace of the original global index on the nonpartitioned table.

Online Modification of Subpartitioning Strategy: Example

Before online conversion:

```
SQL> CREATE TABLE sales (prodno NUMBER NOT NULL, custno NUMBER, time_id DATE,
                         … qty_sold NUMBER(10,2), amt_sold NUMBER(10,2))
     PARTITION BY RANGE (time_id)
     (PARTITION s_q1_17 VALUES LESS THAN (TO_DATE('01-APR-2017','dd-MON-yyyy')),
      PARTITION s_q2_17 VALUES LESS THAN (TO_DATE('01-JUL-2017','dd-MON-yyyy')), …);
```

```
SQL> CREATE INDEX i1_custno ON sales (custno) LOCAL;
SQL> CREATE UNIQUE INDEX i2_time_id ON sales (time_id);
SQL> CREATE INDEX i3_prodno ON sales (prodno);
```

Online conversion:

```
SQL> ALTER TABLE sales MODIFY PARTITION BY RANGE (time_id)
     SUBPARTITION BY HASH (custno) SUBPARTITIONS 8
     (PARTITION s_q1_17 VALUES LESS THAN (TO_DATE('01-APR-2017','dd-MON-yyyy')),
      PARTITION s_q2_17 VALUES LESS THAN (TO_DATE('01-JUL-2017','dd-MON-yyyy')), …)
     ONLINE UPDATE INDEXES (i1_custno LOCAL, i2_time_id GLOBAL PARTITION BY
            RANGE (time_id) ( PARTITION ip1 VALUES LESS THAN (MAXVALUE)));
```

Online MERGE Partition and Subpartition: Example

Before partition merging operation:

```
SQL> CREATE TABLE sales
       (prod_id NUMBER, cust_id NUMBER, time_id DATE, channel_id NUMBER,
        promo_id NUMBER, quantity_sold NUMBER(10,2), amount_sold NUMBER(10,2))
     PARTITION BY RANGE (time_id) INTERVAL (100)
     ( PARTITION p1 VALUES LESS THAN (100),
       PARTITION p2 VALUES LESS THAN (500));
```

```
SQL> CREATE INDEX i1_time_id ON sales (time_id) LOCAL TABLESPACE tbs_2;
SQL> CREATE INDEX i2_promo_id ON sales (promo_id) GLOBAL TABLESPACE tbs_2;
```

Online partition merging operation:

```
SQL> ALTER TABLE sales MERGE PARTITIONS jan17,feb17,mar17 INTO PARTITION q1_17
                         COMPRESS UPDATE INDEXES ONLINE;
```

- The online operation can also be performed on subpartitions.

Batched DDL from DBMS_METADATA Package

The DBMS_METADATA.SET_TRANSFORM_PARAM procedure identifies differences between two tables.

- 12c Generates one ALTER TABLE statement for every difference found
- 19c One single ALTER TABLE statement for all the differences related to scalar columns
  - No change in behavior for LOB and complex types
  - ALTER TABLE simplified patches

```
DECLARE
…
   DBMS_METADATA.SET_TRANSFORM_PARAM (th, 'BATCH_ALTER_DDL', TRUE);
…
/
ALTER TABLE "APP1"."TEST1" ADD ("Y" VARCHAR2(40), "T" VARCHAR2(30), "Z" DATE)
ALTER TABLE "APP1"."TEST1" RENAME TO "TEST2"
```

Unicode 9.0 Support

- Unicode is an evolving standard.
- Unicode 9.0 was recently released:
  - 11 new code blocks
  - 7,500 new characters
  - 6 new language scripts
  - 72 new emoji characters
- Oracle Database 12c R2 (12.2.0.2) has been updated to use the 9.0.0 standard.
  - Both AL32UTF8 and AL16UTF16 are updated.
  - Oracle Globalization Development Kit (GDK) for Java is updated.

Summary

In this lesson, you should have learned how to:

- Manage private temporary tables
- Use the Data Pump Import `CONTINUE_LOAD_ON_FORMAT_ERROR` option of the `DATA_OPTIONS` parameter
- Perform online modification of partitioning and subpartitioning strategy
- Perform online `MERGE` partition and subpartition
- Generate batched DDL by using the `DBMS_METADATA` package
- Benefit from Unicode 9.0 support

Practice 5: Overview

- 5-1: Managing private temporary tables
- 5-2: Using the Data Pump Import `CONTINUE_LOAD_ON_FORMAT_ERROR` option
- 5-3: Converting a `HASH` partitioned table to a `RANGE` partitioned table, online
- 5-4: Converting a `LIST` partitioned table on two keys to a `LIST AUTOMATIC` partitioned table on one key, online
- 5-5: Converting a `LIST AUTOMATIC` partitioned table to a `LIST AUTOMATIC` partitioned table with `SUBPARTITIONING`, online
- 5-6: Merging partitions of a partitioned table, online
- 5-7: Using batched DDL

6. Improving Performance

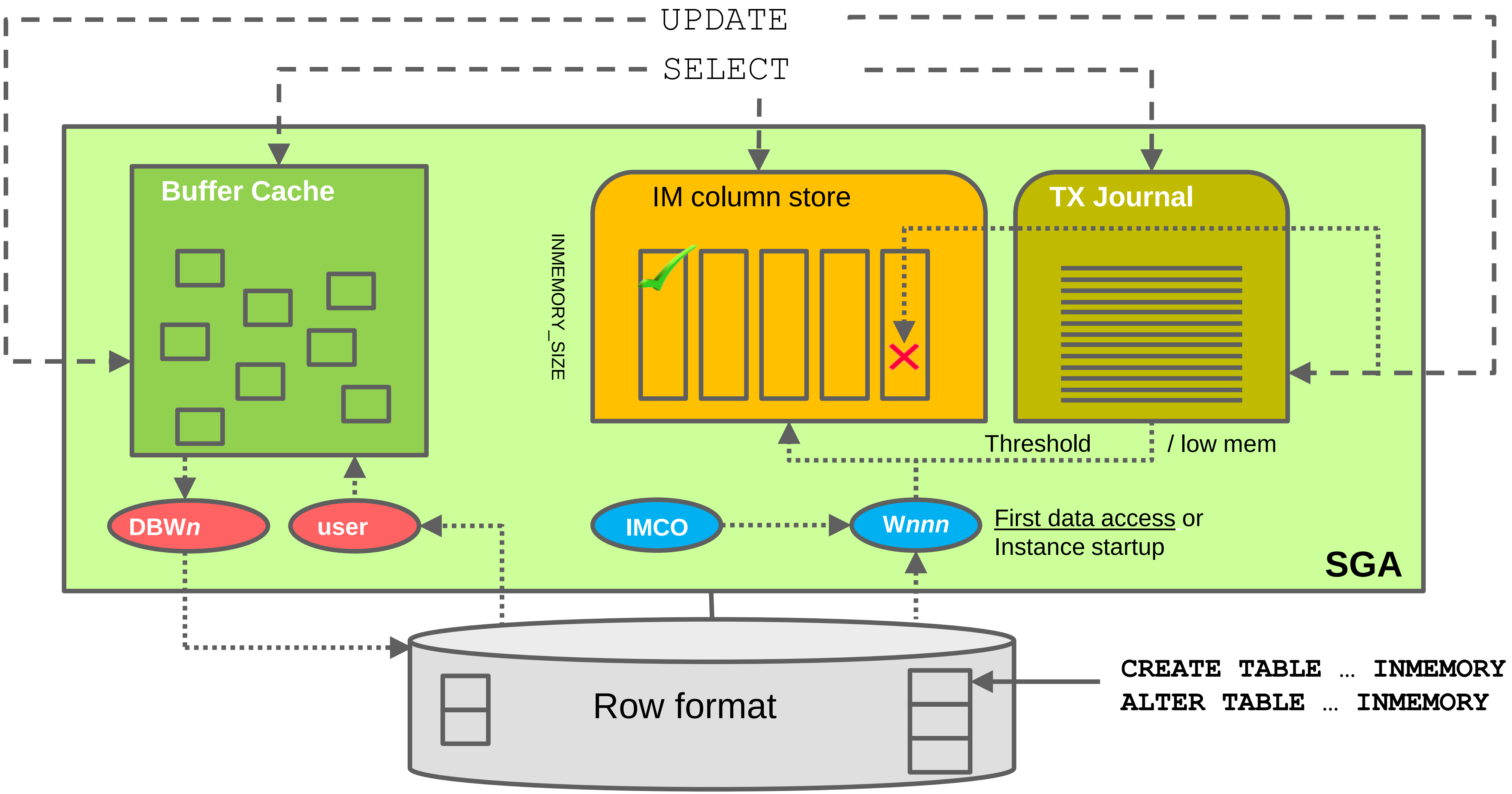
Objectives

After completing this lesson, you should be able to:

- Configure and use Automatic In-Memory
- Configure the window capture of In-Memory expressions
- Describe the Memoptimized Rowstore feature and use in-memory hash index structures
- Describe the new SQL Tuning Set package
- Describe the concurrency of SQL execution of SQL Performance Analyzer tasks
- Describe SQL Performance Analyzer result set validation
- Describe a SQL Exadata-aware profile

In-Memory Column Store: Dual Format of Segments in SGA

12c



```
CREATE TABLE … INMEMORY
ALTER TABLE … INMEMORY
```

Deploying the In-Memory Column Store

12c

1. Verify the database compatibility value.

```
COMPATIBLE = 12.2.0.0.0
```

2. Configure the IM column store size.

```
INMEMORY_SIZE = 100G
```

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

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

Setting In-Memory Object Attributes

3. Enable or disable objects to be populated into the IM column store.
   – IMCUs are initialized and populated at query access time 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 when the database is opened.

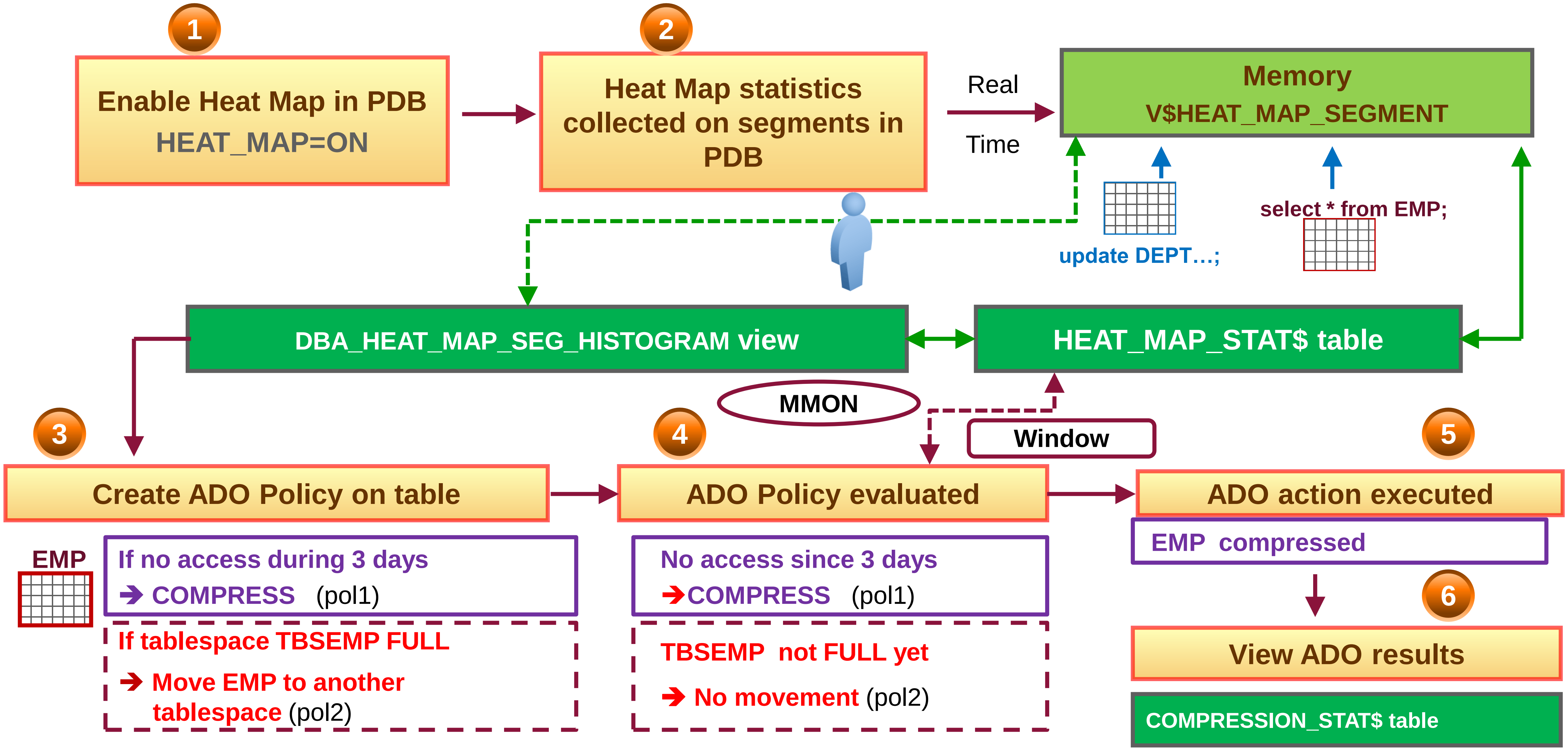
```
SQL> CREATE TABLE test (…) INMEMORY PRIORITY CRITICAL;
```

   – Use `MEMCOMPRESS` to define the compression level.

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

12c

Managing Heat Map and Automatic Data Optimization Policies



1 Enable Heat Map in PDB
HEAT_MAP=ON

2 Heat Map statistics collected on segments in PDB

Real Time

Memory
V$HEAT_MAP_SEGMENT

update DEPT…;

select * from EMP;

DBA_HEAT_MAP_SEG_HISTOGRAM view

HEAT_MAP_STAT$ table

MMON

Window

3 Create ADO Policy on table

EMP

If no access during 3 days
➔ COMPRESS (pol1)

If tablespace TBSEMP FULL
➔ Move EMP to another tablespace (pol2)

4 ADO Policy evaluated

No access since 3 days
➔COMPRESS (pol1)

TBSEMP not FULL yet
➔ No movement (pol2)

5 ADO action executed

EMP compressed

6 View ADO results

COMPRESSION_STAT$ table

Creating ADO In-Memory Policies

- Types of ADO In-Memory policies based on heat map statistics:

  ```
  HEAT_MAP = ON
  ```

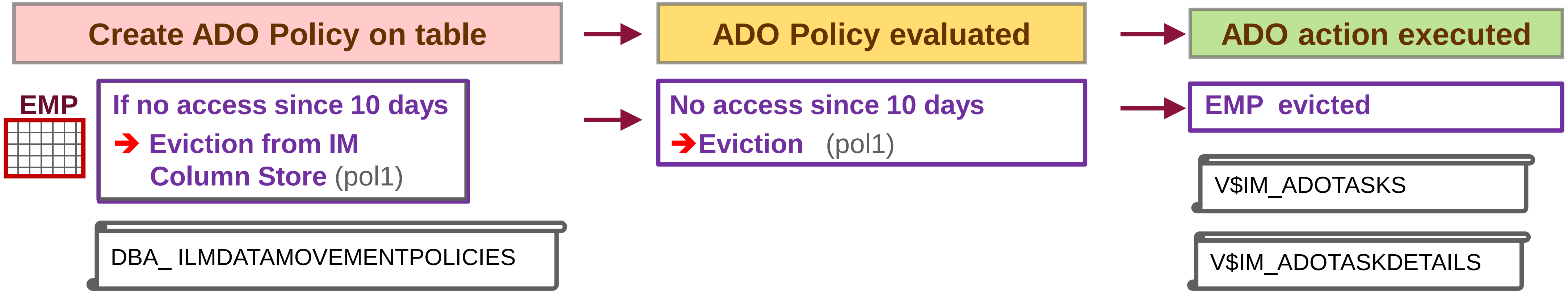
  – Define policy to set IM attribute.

  – Define policy to unset IM attribute ➔ Eviction from IM column store.

  – Define policy to modify IM compression.

```
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;
```

- Define an anticipated time for evicting IM segments from the IM column store:



Create ADO Policy on table → ADO Policy evaluated → ADO action executed

EMP

If no access since 10 days ➔ Eviction from IM Column Store (pol1) → No access since 10 days ➔Eviction (pol1) → EMP evicted

DBA_ ILMDATAMOVEMENTPOLICIES

V$IM_ADOTASKS

V$IM_ADOTASKDETAILS

Automatic In-Memory: Overview

19c

Before Automatic In-Memory was introduced, the DBA had to:

- Define when in-memory segments should be populated into the IM column store
- Define ADO IM policies to evict / populate IM segments from or into the IM column store

AIM automates the management of the IM column store by using heat map statistics:

- Ensures that the “working data set” is in the IM column store at all times
- Moves IM segments in and out of the IM column store

Benefits:

- Automatic actions: Makes the management of the IM column store easier
- Automatic eviction: Increases effective IM column store capacity
- Improved performance: Keeps as much of the working data set in memory as possible

AIM Action

- Increase the effective capacity of the IM column store by evicting inactive IM segments with priority `NONE` from the IM column store under memory pressure.
- Evict at segment level:
  - According to the amount of time that an IM segment has been inactive
  - According to the window of time used by AIM to determine the statistics for decision-making
- Populate hot data.

Note: ADO IM policies override AIM considerations.

Configuring Automatic In-Memory

- Activate heat map statistics:

```
SQL> ALTER SYSTEM SET heat_map = ON;
```

- Set the initialization parameter:

```
SQL> ALTER SYSTEM SET INMEMORY_AUTOMATIC_LEVEL = MEDIUM SCOPE = BOTH;
```

- Use the `DBMS_INMEMORY_ADMIN.AIM_SET_PARAMETER` procedure to configure the sliding stats window in days:

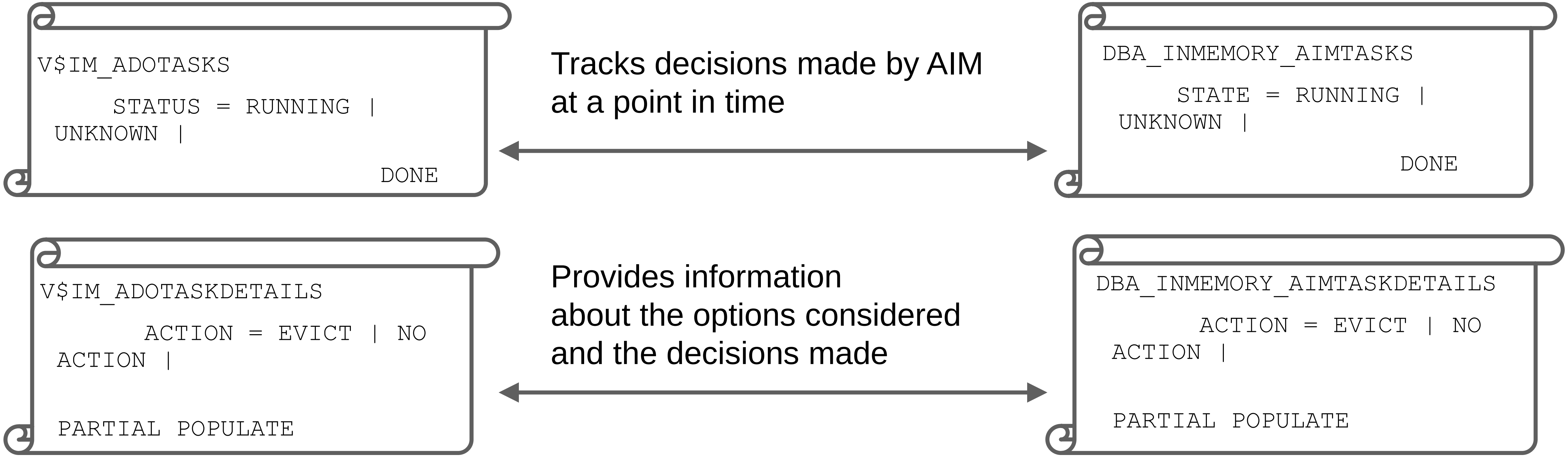
```
SQL> EXEC dbms_inmemory_admin.aim_set_parameter (-
          parameter => dbms_inmemory_admin.AIM_STATWINDOW_DAYS , -
          value => 1)
```

- Use the `DBMS_INMEMORY_ADMIN.AIM_GET_PARAMETER` procedure to get the current values of the AIM parameters.

Diagnostic Views

19c

What are the decisions and actions made by AIM?



V$IM_ADOTASKS

STATUS = RUNNING | UNKNOWN | DONE

Tracks decisions made by AIM at a point in time

DBA_INMEMORY_AIMTASKS

STATE = RUNNING | UNKNOWN | DONE

V$IM_ADOTASKDETAILS

ACTION = EVICT | NO ACTION | PARTIAL POPULATE

Provides information about the options considered and the decisions made

DBA_INMEMORY_AIMTASKDETAILS

ACTION = EVICT | NO ACTION | PARTIAL POPULATE

Populating In-Memory Expression Results

DBA_EXPRESSION_STATISTICS
SNAPSHOT =
LATEST | CUMULATIVE [WINDOW

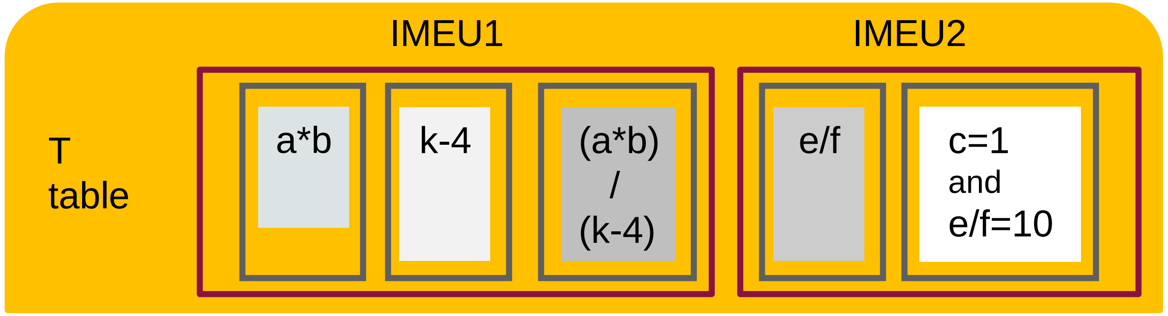
Two types of IMEs: *optimizer-defined expressions* and *user-defined virtual columns*

- Query performance improved by caching:
  - Results of frequently evaluated query expressions
  - Results of user-defined virtual columns

```
INMEMORY_EXPRESSIONS_USAGE = ENABLE
```

```
INMEMORY_VIRTUAL_COLUMNS = ENABLE
```

- Results stored in IM expression units (IMEUs) for subsequent reuse



- Candidates detected by and eligible according to expression statistics

12c

```
SQL> exec DBMS_INMEMORY_ADMIN.IME_CAPTURE_EXPRESSIONS ('CURRENT')
```

19c

```
SQL> exec DBMS_INMEMORY_ADMIN.IME_CAPTURE_EXPRESSIONS ('WINDOW')
```

Populating In-Memory Expression Results Within a Window

1. Open a window:

```
SQL> exec DBMS_INMEMORY_ADMIN.IME_OPEN_CAPTURE_WINDOW()
```

2. Let workload run.

Optionally, get the current capture state of the expression capture window and the time stamp of the most recent modification.

```
SQL> exec DBMS_INMEMORY_ADMIN.IME_GET_CAPTURE_STATE( P_CAPTURE_STATE, -
                                                     P_LAST_MODIFIED)
```

3. Close the window:

```
SQL> exec DBMS_INMEMORY_ADMIN.IME_CLOSE_CAPTURE_WINDOW()
```

4. Populate all the hot expressions captured in the window into the IM column store:

```
SQL> exec DBMS_INMEMORY_ADMIN.IME_CAPTURE_EXPRESSIONS ('WINDOW')
```

Memoptimized Rowstore

19c

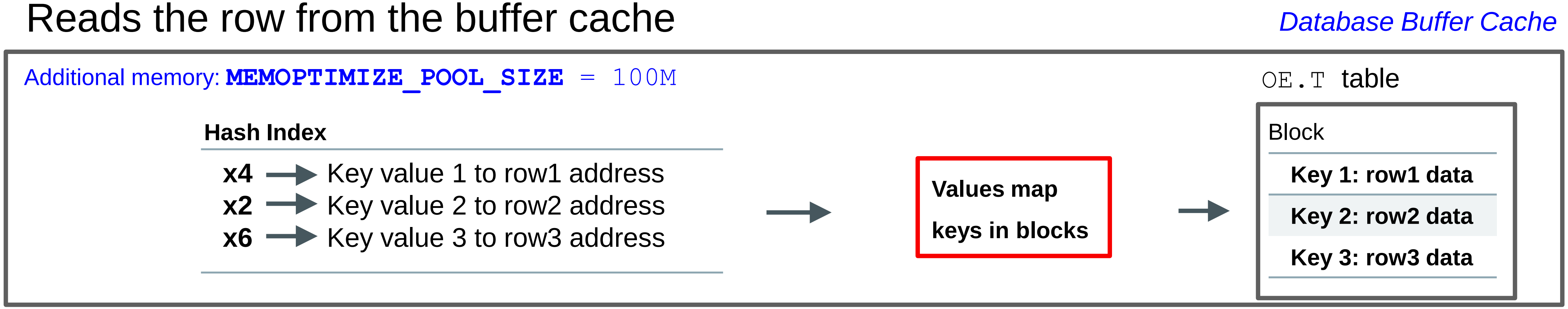
Fast ingest and query rates for thousands of devices from the Internet requires:

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

In-Memory Hash Index

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

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


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

- Enable tables for `MEMOPTIMIZE FOR READ`

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

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

DBA_TABLES
DBA_TAB_PARTITIONS
DBA_TAB_SUBPARTITIONS
DBA_OBJECT_TABLES
MEMOPTIMIZE_READ = ENABLED
MEMOPTIMIZE_WRITE = DISABLED

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

DBMS_SQLTUNE Versus DBMS_SQLSET Package

Package	**SQL tuning task management:** • Create / drop tuning task. • Execute tuning task. • Display advisor recommendations.	**SQL Profile management:** • Accept SQL profile. • Drop / alter SQL profile. • Manipulate staging tables.	**STS management:** • Create / drop STS. • Populate STS. • Query STS content. • Manipulate staging tables.
12c DBMS_SQLTUNE	✓	✓	✓
19c DBMS_SQLSET			✓

SQL Tuning Sets: Manipulation

12c SQL Tuning Set functionality is available only if one of the following conditions exist:

- Tuning Pack is enabled.
- Real Application Testing (RAT) option is installed.

19c SQL Tuning Set functionality is available for free with Oracle DB Enterprise Edition.

- A new DBMS_SQLSET package is available to create, edit, drop, populate, and query STS and manipulate staging tables.

```
SQL> EXEC dbms_sqlset.create_sqlset | delete_sqlset | update_sqlset |
                          drop_sqlset
```

```
SQL> EXEC dbms_sqlset.capture_cursor_cache | load_sqlset
```

```
SQL> EXEC dbms_sqlset.create_stgtab | pack_stgtab | unpack_stgtab |
                          remap_stgtab
```

- The new package is not part of the tuning pack or RAT option.

SQL Performance Analyzer

- Targeted users: DBAs, QAs, application developers
- Helps predict the impact of system changes on SQL workload response time:
  - Database upgrades
  - Implementation of tuning recommendations
  - Schema / database parameter changes
  - Statistics gathering
  - OS and hardware changes
- Builds different versions of SQL workload performance (SQL execution plans and execution statistics)
- Re-executes SQL statements serially *12c* or concurrently *19c*
- Analyzes performance differences
- Offers fine-grained performance analysis on individual SQL

Using SQL Performance Analyzer

1. Capture SQL workload on production.
2. Transport the SQL workload to a test system.
3. Build “before-change” performance data.
4. Make changes.
5. Build “after-change” performance data.
6. Compare results from steps 3 and 5.
7. Tune regressed SQL.

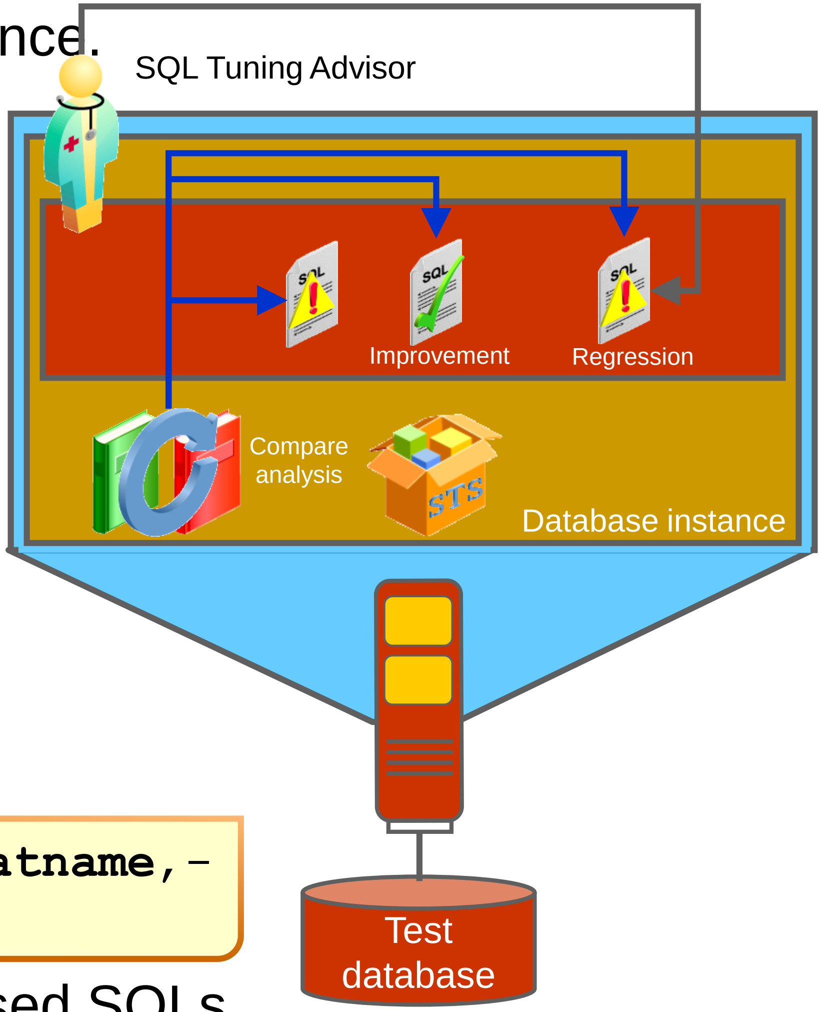
Steps 6-7: Comparing / Analyzing Performance and Tuning Regressed SQL

- Rely on user-specified metrics to compare SQL performance.
- Calculate impact of change on individual SQLs and SQL workload.
- Use SQL execution frequency to define a weight of importance.
- Detect improvements, regressions, and unchanged performance.
- Detect changes in execution plans.

19c Validate that the same result set was returned during the initial SPA test and during subsequent tests.

```
SQL> EXEC dbms_sqlpa.set_analysis_task_parameter(:atname,-
                    'COMPARE_RESULTSET', 'FALSE')
```

- Recommend running SQL Tuning Advisor to tune regressed SQLs.



SQL Performance Analyzer: PL/SQL Example

1. Create the tuning task:

```
EXEC :tname:= dbms_sqlpa.create_analysis_task( -
              sqlset_name => 'MYSTS', task_name => 'MYSPA')
```

2. Set task execution parameters:

```
EXEC dbms_sqlpa.set_analysis_task_parameter( :tname, 'TEST_EXECUTE_DOP', 4)
```

3. Execute the task to build the before-change performance data:

```
EXEC dbms_sqlpa.execute_analysis_task(task_name => :tname, -
   execution_type => 'TEST EXECUTE', execution_name => 'before')
```

4. Produce the before-change report:

```
SELECT dbms_sqlpa.report_analysis_task(task_name => :tname,
   type=>'text', section=>'summary') FROM dual;
```

SQL Performance Analyzer: PL/SQL Example

After making your changes:

5. Create the after-change performance data:

```
EXEC dbms_sqlpa.execute_analysis_task(task_name => :tname, -
   execution_type => 'TEST EXECUTE', execution_name => 'after')
```

6. Generate the after-change report:

```
SELECT dbms_sqlpa.report_analysis_task(task_name => :tname,
   type=>'text', section=>'summary') FROM dual;
```

7. Set task comparison parameters and compare the task executions:

```
EXEC dbms_sqlpa.set_analysis_task_parameter(:tname, 'COMPARE_RESULTSET', 'TRUE')
```

```
EXEC dbms_sqlpa.execute_analysis_task(task_name => :tname,
              execution_type => 'COMPARE PERFORMANCE')
```

8. Generate the analysis report:

```
SELECT dbms_sqlpa.report_analysis_task(task_name => :tname,
   type=>'text', section=>'summary') FROM dual;
```

SQL Exadata-Aware Profile

19c

SQL Tuning Advisor provides better execution plans, faster SQL, less resource usage, and enhanced performance of Exadata systems by using new algorithms.

- SQL Tuning Advisor executes a new analysis to determine if any of these system statistics are not up to date:
  - I/O Seek Time
  - Multi-Block Read Count (MBRC)
  - I/O Transfer Speed
- If any of the system statistics are found to be stale and gathering them improves the performance of the SQL being tuned, SQL Tuning Advisor recommends an Exadata-aware SQL profile.
- Accepting such a profile impacts performance of only the SQL being tuned and not any of the other SQLs.

Summary

In this lesson, you should have learned how to:

- Configure and use Automatic In-Memory
- Configure the window capture of in-memory expressions
- Describe the Memoptimized Rowstore feature and use in-memory hash index structures
- Describe the new SQL Tuning Set package
- Describe the concurrency of SQL execution of SQL Performance Analyzer tasks
- Describe the SQL Performance Analyzer result set validation
- Describe a SQL Exadata-aware profile

Practice 6: Overview

- 6-1: Configuring and Using AIM
- 6-2: Tracking IM Expressions Within a Capture Window

7. Handling Enhancements in Big Data and Data Warehousing

Objectives

After completing this lesson, you should be able to:

- Query inlined external tables
- Manage in-memory external tables
- Use the new query capabilities of the Analytics view
- Create and use polymorphic table functions
- Use new functions for approximate Top-N queries

12c

Querying External Tables

1. Create the external table before querying against the external table.

```
CREATE TABLE ext_emp  (id NUMBER,  …, email  VARCHAR2(25))
 ORGANIZATION EXTERNAL
 ( TYPE ORACLE_LOADER  DEFAULT DIRECTORY ext_dir
   ACCESS PARAMETERS
              ( records delimited by newline
                badfile ext_dir:'empxt%a_%p.bad'
                logfile ext_dir:'empxt%a_%p.log'
                fields terminated by ','  missing field
values are null
                (emp_id, first_name, last_name, job_id) )
    LOCATION ('empext1.dat') )
 REJECT LIMIT UNLIMITED;
```

/extdir/empext1.dat

```
SQL> SELECT * FROM ext_emp;
```

- DEFAULT DIRECTORY
- ACCESS PARAMETERS
  - DISCARDFILE, BADFILE, LOGFILE
- LOCATION
- REJECT LIMIT

2. Modify parameters during a query: no need to alter the external table definition.

```
SQL> SELECT * FROM ext_emp EXTERNAL MODIFY ( ACCESS PARAMETERS ( BADFILE ext_dir:'empxt2%a_%p.bad',
                                                                 LOGFILE ext_dir:'empxt2%a_%p.log')
                                           LOCATION ('empext2.dat'));
```

/extdir/empext2.dat

Querying Inlined External Tables

19c

Increasing the flexibility and ease of SQL access, inlined external tables:

- Are similar to inline views
- Allow the runtime definition of an external table as part of a SQL query statement
- Transparently access data outside the Oracle database
- Simplify the access of external data by a simpler and more efficient code

```
SQL> SELECT ext_emp.id FROM EXTERNAL ( (id NUMBER, …, email VARCHAR2(25))
            TYPE ORACLE_LOADER
            DEFAULT DIRECTORY ext_dir
            ACCESS PARAMETERS
               ( records delimited by newline
                 badfile ext_dir:'empxt%a_%p.bad'  logfile ext_dir:'empxt%a_%p.log'
                 fields terminated by ','  missing field values are null
                  (id, …, email)
               )
            LOCATION ('empext1.dat') ────────→  /extdir/empext1.dat
            REJECT LIMIT UNLIMITED
          ) ext_emp;
```

Database In-Memory Support for External Tables

19c

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

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

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

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

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

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

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

Analytic Views

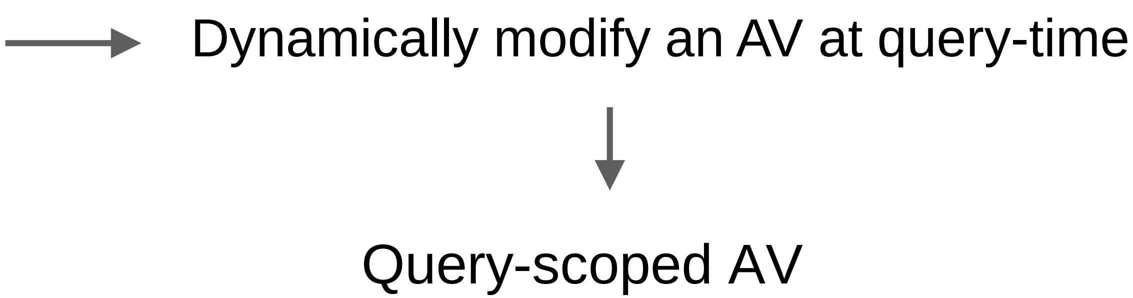
Analytic views (AVs) are metadata-only objects defined over standard tables or views:

- They provide a hierarchical organization and analytical hierarchy-aware calculations.
- They are queried via SQL or the `DBMS_MDX_ODBO` package.

12c Microsoft's Multidimensional Expression (MDX) provides features that are not accessible in SQL query.

19c Enhancing the AV SQL query capabilities adds support for two such features:

- Query-scoped calculations
- Filter-before aggregate



Dynamically modify an AV at query-time

Query-scoped AV

Visual Totals Versus Non-Visual Totals

12c

Table FACT_SALES → Totals of sales with MDX

TIME	SALES
Q1-2016	10
Q2-2016	20
Q3-2016	30
Q4-2016	40

TIME	Visual SALES	Non-Visual SALES
2016	**30**	**100**
Q1-2016	10	10
Q2-2016	20	20

→ SQL equivalent

Predicates specified in the WHERE clause simply reduce the rows returned, but do not impact the aggregated measure data.

No SQL equivalent

If a node has any descendants in the WHERE clause, only those descendants are used to aggregate up to that node.

- SQL querying an AV always produces the non-visual result.
- Visual totals with MDX:
  - If the selection includes the year 2016 but none of its descendants, the data for 2016 is aggregated from all of its leaves.
  - If the selection includes year 2016 and in addition includes Q1-2016 and Q2-2016, the data for 2016 using only those quarters is aggregated.

Filter-Before Aggregate Predicates and Calculated Measures

19c

How can SQL produce visual totals in queries?

- Use filter-before aggregate predicates based on hierarchy navigation.

Non-visual totals → Visual totals

TIME	SALES
2016	**100**
Q1-2016	10
Q2-2016	20

TIME	SALES
2016	**30**
Q1-2016	10
Q2-2016	20

with SQL filter-before aggregate predicate

1. Each hierarchy may specify a filter-before aggregate predicate, which serves to filter the leaves of that hierarchy before aggregating the measures (keeps *`Q1-2016`* and *`Q2-2016`* leaves).
2. The predicate for a given hierarchy specifies some set of hierarchy members.
3. The fact rows are then filtered to include only the leaf descendants of those members.

- Define calculated measures within a `SELECT` statement.
- Optionally, use a `WHERE` clause to further filter the output.

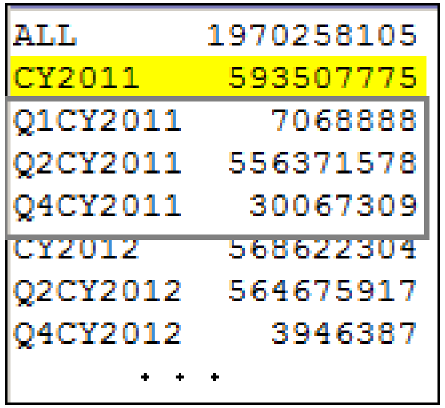
Query-Scoped Calculations Using Hierarchy-Based Predicates

Set the `COMPATIBLE` initialization parameter to 18.0.0 at least.

```
COMPATIBLE = 18.0.0.0.0
```

- Without filter-before aggregate predicates:

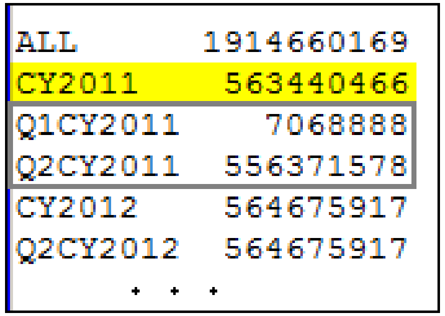
```
SELECT time_hier.member_name, sales FROM av.sales_av  HIERARCHIES(time_hier)
WHERE time_hier.level_name IN ('ALL','YEAR','QUARTER')
ORDER BY time_hier.hier_order;
```



```
ALL         1970258105
CY2011       593507775
Q1CY2011       7068888
Q2CY2011     556371578
Q4CY2011      30067309
CY2012       568622304
Q2CY2012     564675917
Q4CY2012       3946387
       . . .
```

- Using filter-before aggregate predicates:

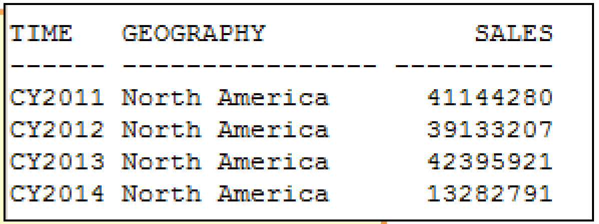
```
SELECT time_hier.member_name, sales FROM ANALYTIC VIEW (
            USING sales_av HIERARCHIES(time_hier)
            FILTER FACT (time_hier TO level_name = 'MONTH'
                          AND  TO_CHAR(month_end_date,'Q') IN (1, 2)
                          )
            )
WHERE time_hier.level_name IN ('ALL','YEAR','QUARTER')
ORDER BY time_hier.hier_order;
```



```
ALL         1914660169
CY2011       563440466
Q1CY2011       7068888
Q2CY2011     556371578
CY2012       564675917
Q2CY2012     564675917
       . . .
```

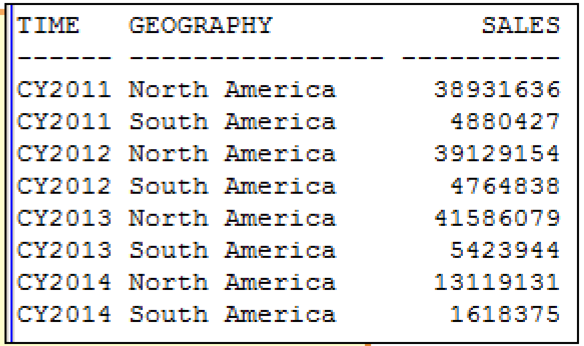
Using Multiple Hierarchy-Based Predicates

```
SELECT time_hier.member_name AS time,
       geography_hier.member_name AS geography, sales
FROM ANALYTIC VIEW (
     USING sales_av HIERARCHIES(time_hier, geography_hier)
     FILTER FACT ( time_hier TO level_name = 'QUARTER' AND (quarter_name
                             like 'Q1%' OR quarter_name like 'Q2%'),
                   geography_hier TO level_name = 'COUNTRY'
                             AND country_name in ('Mexico','Canada'))
                   )
WHERE time_hier.level_name IN ('YEAR') AND geography_hier.level_name = 'REGION'
ORDER BY time_hier.hier_order;
```



```
TIME   GEOGRAPHY             SALES
------ ---------------- ----------
CY2011 North America      41144280
CY2012 North America      39133207
CY2013 North America      42395921
CY2014 North America      13282791
```

```
SELECT time_hier.member_name AS time,
       geography_hier.member_name AS geography, sales
FROM ANALYTIC VIEW (
     USING sales_av HIERARCHIES(time_hier, geography_hier)
     FILTER FACT ( time_hier TO level_name = 'QUARTER' AND (quarter_name
                             like 'Q1%' OR quarter_name like 'Q2%'),
                   geography_hier TO level_name = 'COUNTRY'
                             AND country_name in ('Mexico', 'Canada', 'Chile'))
                   )
WHERE time_hier.level_name IN ('YEAR') AND geography_hier.level_name = 'REGION'
ORDER BY time_hier.hier_order;
```

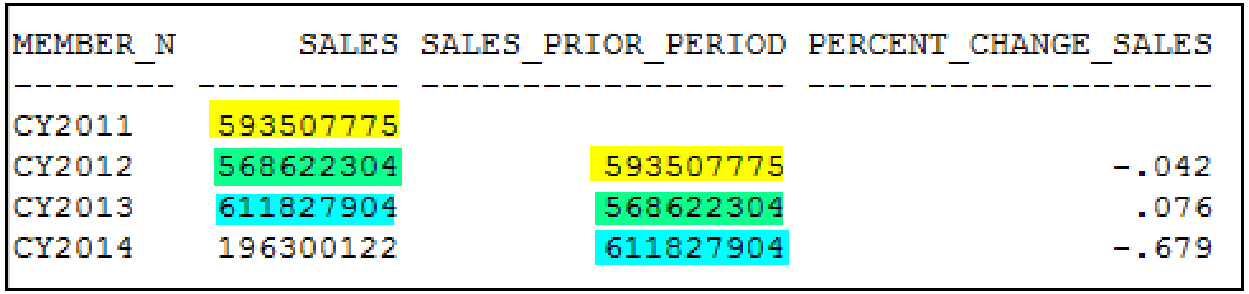


```
TIME   GEOGRAPHY             SALES
------ ---------------- ----------
CY2011 North America      38931636
CY2011 South America       4880427
CY2012 North America      39129154
CY2012 South America       4764838
CY2013 North America      41586079
CY2013 South America       5423944
CY2014 North America      13119131
CY2014 South America       1618375
```

Query-Scoped Calculations Using Calculated Measures

- To define calculated measures:

```
SELECT time_hier.member_name, sales, sales_prior_period,
       ROUND(sales_prior_period_pct_change,3) AS percent_change_sales
 FROM ANALYTIC VIEW (
     USING sales_av HIERARCHIES(time_hier)
           ADD MEASURES (
            sales_prior_period AS (LAG(sales) OVER (HIERARCHY time_hier OFFSET 1)),
            sales_prior_period_pct_change AS (LAG_DIFF_PERCENT(sales) OVER
                                          (HIERARCHY time_hier OFFSET 1))
                        )
                    )
WHERE time_hier.level_name = 'YEAR'
ORDER BY time_hier.hier_order;
```

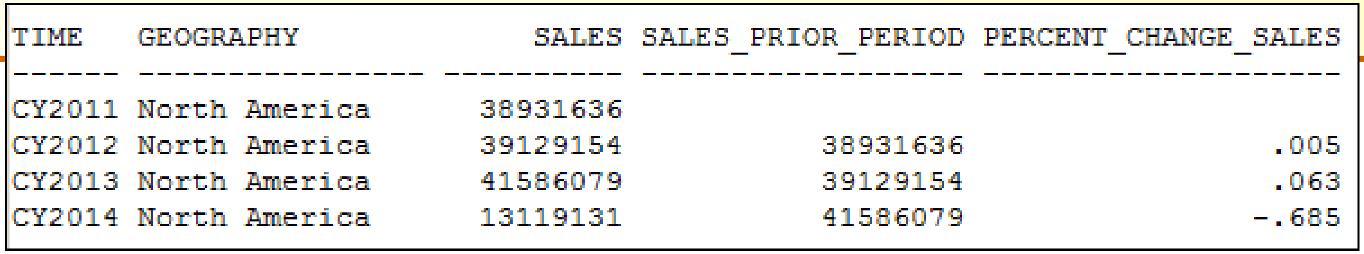


```
MEMBER_N      SALES SALES_PRIOR_PERIOD PERCENT_CHANGE_SALES
-------- ---------- ------------------ --------------------
CY2011    593507775
CY2012    568622304          593507775                -.042
CY2013    611827904          568622304                 .076
CY2014    196300122          611827904                -.679
```

Using Hierarchy-Based Predicates and Calculated Measures

Combine FILTER FACT and ADD MEASURES:

```
SELECT time_hier.member_name AS time, geography_hier.member_name AS geography, sales,
       sales_prior_period,
       ROUND(sales_prior_period_pct_change,3) AS percent_change_sales
FROM ANALYTIC VIEW (
     USING sales_av HIERARCHIES(time_hier, geography_hier)
     FILTER FACT    ( time_hier TO level_name = 'QUARTER' AND (quarter_name
                              like 'Q1%' OR quarter_name like 'Q2%'),
                      geography_hier TO level_name = 'COUNTRY'
                              AND country_name in ('Mexico','Canada'))
     ADD MEASURES   (
             sales_prior_period AS (LAG(sales) OVER (HIERARCHY time_hier OFFSET 1)),
             sales_prior_period_pct_change AS (LAG_DIFF_PERCENT(sales) OVER
                                              (HIERARCHY time_hier OFFSET 1)))
                   )
WHERE time_hier.level_name = 'YEAR' AND geography_hier.level_name = 'REGION'
ORDER BY time_hier.hier_order;
```

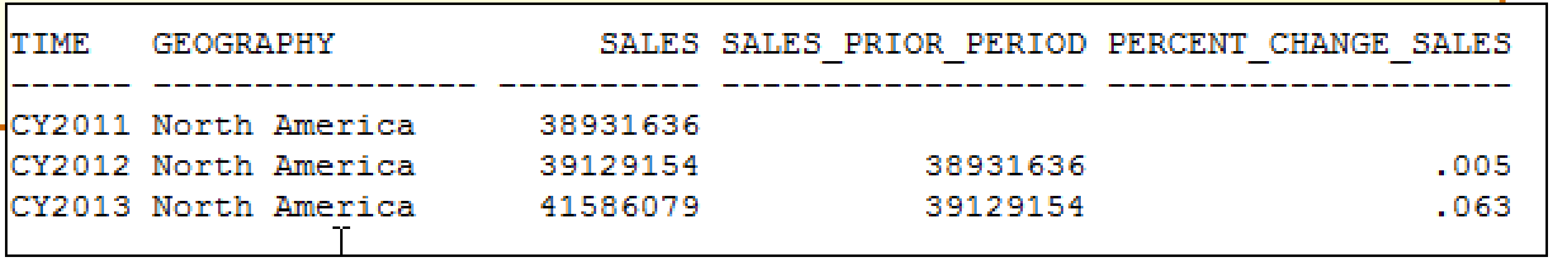


```
TIME   GEOGRAPHY              SALES SALES_PRIOR_PERIOD PERCENT_CHANGE_SALES
------ ---------------- ---------- ------------------ --------------------
CY2011 North America      38931636
CY2012 North America      39129154           38931636                 .005
CY2013 North America      41586079           39129154                 .063
CY2014 North America      13119131           41586079                -.685
```

Using Hierarchy-Based Predicates and Calculated Measures

Combine `FILTER FACT`, `ADD MEASURES` and `WHERE` clause:

```
SELECT time_hier.member_name AS time, geography_hier.member_name AS geography, sales,
       sales_prior_period,
       ROUND(sales_prior_period_pct_change,3) AS percent_change_sales
FROM ANALYTIC VIEW (
     USING sales_av HIERARCHIES(time_hier, geography_hier)
     FILTER FACT   ( time_hier TO level_name = 'QUARTER' AND (quarter_name
                               like 'Q1%' OR quarter_name like 'Q2%'),
                     geography_hier TO level_name = 'COUNTRY'
                               AND country_name in ('Mexico','Canada'))
     ADD MEASURES   (
            sales_prior_period AS (LAG(sales) OVER (HIERARCHY time_hier OFFSET 1)),
            sales_prior_period_pct_change AS (LAG_DIFF_PERCENT(sales) OVER
                                     (HIERARCHY time_hier OFFSET 1)))
                    )
WHERE time_hier.level_name = 'YEAR' AND geography_hier.level_name = 'REGION'
AND   sales > 30000000
ORDER BY time_hier.hier_order;
```



```
TIME   GEOGRAPHY               SALES SALES_PRIOR_PERIOD PERCENT_CHANGE_SALES
------ ---------------- ---------- ------------------ --------------------
CY2011 North America      38931636
CY2012 North America      39129154           38931636                 .005
CY2013 North America      41586079           39129154                 .063
```

Polymorphic Table Functions

- A table function (TF) is a function that returns a collection of rows that can be called from the from-clause of a SQL query block.

```
SQL> SELECT * FROM TF_NOOP(emp);
```

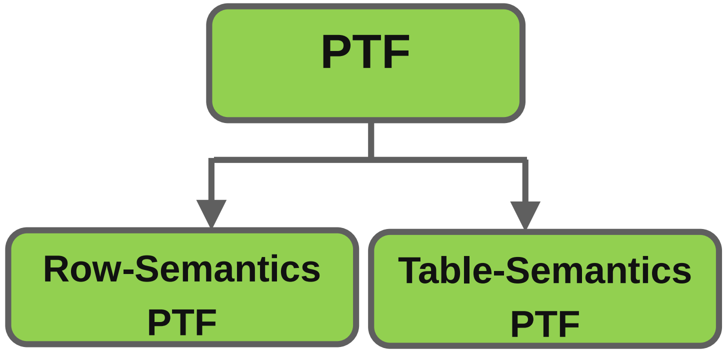
- A polymorphic table function (PTF) is a TF whose return type is determined by the arguments to the PTF.

```
SQL> SELECT * FROM PTF_NOOP(emp);
```

Goal

- Provide a framework for DBAs and developers for writing PTFs that are simple to use and have an efficient and scalable implementation inside the RDBMS.
  1. The DBA or developer creates the PTF.
  2. A SQL writer invokes the PTF in a query.
  3. RDBMS compiles and executes the PTF.

Row-Semantics and Table-Semantics PTFs



```
SQL> SELECT * FROM PTF_NOOP(emp);
```

DBA_PROCEDURES
    POLYMORPHIC = NULL | TABLE | ROW

PTFs take a table as an argument.

- Use Row Semantic (RS) PTF when the new columns can be determined by just looking at any single row.
- Use Table Semantic (TS) PTF when the new columns can be determined by looking at not only the current row but also some state that summarizes previously processed rows.
- A PTF is invoked from the from-clause of a SQL query block like existing TFs.
- The table-argument can be one of the following:
  - A schema-level object that is allowed in a from-clause
  - A with-clause query
- A new `EXPLAIN` operation appears in plans: `POLYMORPHIC TABLE FUNCTION`

Components to Create PTFs

A PTF is composed of two parts:

- The PL/SQL package that contains functions/procedures for the PTF implementation
  - Required function `DESCRIBE`: Returns the new table “shape”
  - Required procedure `FETCH_ROWS`: For a given subset of rows, produces the new associated column values
- The PL/SQL function that names the PTF and then associates it with the implementation package

Steps to Create PTFs

1. Create the package containing the `DESCRIBE` function and `FETCH_ROWS` procedure.

```
CREATE OR REPLACE PACKAGE change_case_p AS
  function  Describe(tab IN OUT DBMS_TF.Table_t, new_case varchar2)
            return DBMS_TF.describe_t;
  procedure Fetch_Rows(new_case varchar2);
END change_case_p;
/
CREATE OR REPLACE PACKAGE BODY change_case_p AS …
```

2. Create the function.

```
CREATE OR REPLACE FUNCTION change_case (tab TABLE, new_case varchar2) return TABLE
                  pipelined ROW | TABLE POLYMORPHIC USING change_case_p;
```

- Specify exactly one formal argument of type TABLE.
- Specify the return type of the PTF as TABLE.
- Specify the type of the PTF function (`ROW` or `TABLE POLYMORPHIC`).
- Indicate which package contains the actual PTF implementation.

DBMS_TF Routines

- `DBMS_TF.Table_t`: Descriptor for the PTF input table
- `DBMS_TF.Columns_New_t`: Descriptor for the table produced by the PTF
- `DBMS_TF.Row_Set_t`: Input rowset
- `DBMS_TF.Row_Set_t`: Output rowset
- `DBMS_TF.Get_Row_Set`: Computes the input rowset into the output
- `DBMS_TF.Put_Row_Set`: Provides the output rowset

How Does RDBMS Compile and Execute PTFs?

- The PTF query is a single cursor.
- Only columns of interest are passed to the PTF. The PTF row-source does the “row stitching”.
  1. The PTF row-source keeps the input row-set in memory and passes the relevant columns to the PTF.
  2. The PTF processes the input columns to produce the output columns.
  3. The PTF row-source finally “stitches” them to produce the new rows that are returned to the parent row-source.
- Predicates, projections, or partitioning are pushed into the underlying table/query (where semantically possible).
- Parallelism is based on type of PTF and query-specified partitioning (if any).

Exact Top-N Query Processing: SQL Row-Limiting Clause

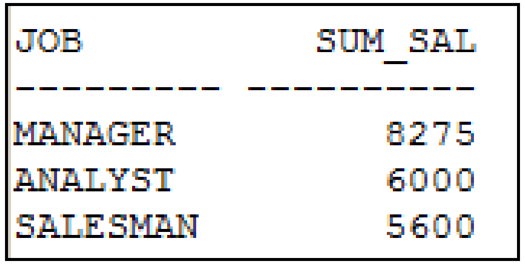
12c

In data analysis applications, users need to find the most frequent values.

Example: Find the top three job titles contributing to the most payroll expenses.

- Use a row-limiting clause to limit the rows returned by the query.
  - Specify the number of rows to return with the `FETCH FIRST/NEXT` keywords.
  - Specify the percentage of rows to return with the `PERCENT` keyword.

```
SQL> SELECT job, SUM(sal) sum_sal FROM emp
     GROUP BY job
     ORDER BY sum_sal DESC
     FETCH FIRST 3 ROWS ONLY ;
```



```
JOB         SUM_SAL
--------- ----------
MANAGER        8275
ANALYST        6000
SALESMAN       5600
```

- Queries that order data and limit row output are referred to as `Top-N` queries.
- The `Top-N` queries return exact results.
- `FETCH FIRST n ROWS ONLY` applies to global ordering.

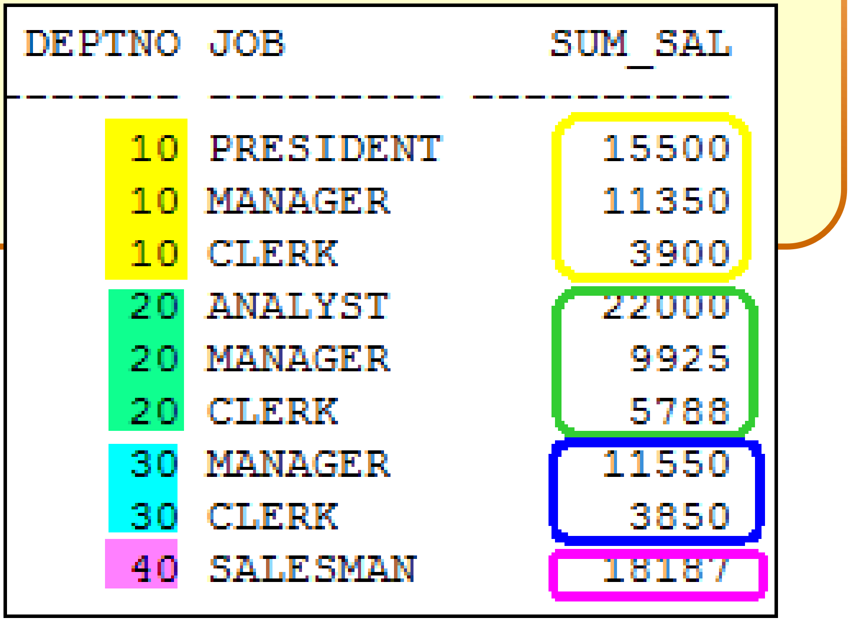
Exact Top-N Query Processing: Rank Window Function

12c

Example: Find the top three job titles contributing to the most payroll expenses within each department.

- Use a rank window function and a nested query.

```
SQL> SELECT deptno, job, sum_sal
     FROM (SELECT deptno, job, SUM(sal) sum_sal, RANK() OVER
                                (PARTITION BY deptno ORDER BY sum(sal) DESC)
                  sum_sal_rank
           FROM emp
           GROUP BY deptno, job)
     WHERE sum_sal_rank <= 3;
```



```
DEPTNO JOB          SUM_SAL
------ --------- ----------
    10 PRESIDENT      15500
    10 MANAGER        11350
    10 CLERK           3900
    20 ANALYST        22000
    20 MANAGER         9925
    20 CLERK           5788
    30 MANAGER        11550
    30 CLERK           3850
    40 SALESMAN       18187
```

- The rank window queries return exact results.
- The rank window function applies to groups of rows ordering.

Approximate Top-N Query Processing

12c

When aggregation functions and analytic functions sort large volumes of data, exact `Top-N` queries require lots of memory and are time consuming.

- Approximate query processing is much faster.
- It is useful for situations where a tolerable amount of error is acceptable.

```
APPROX_FOR_AGGREGATION = true
```

Automatically replaces exact query processing for aggregation queries with approximate query processing.

```
APPROX_FOR_COUNT_DISTINCT = true
```

Automatically replaces `COUNT (DISTINCT expr)` queries with `APPROX_COUNT_DISTINCT` queries.

```
APPROX_FOR_PERCENTILE =
      NONE
      PERCENTILE_CONT | PERCENTILE_CONT DETERMINISTIC | PERCENTILE_DISC | PERCENTILE_DISC
      DETERMINISTIC | ALL | ALL DETERMINISTIC
```

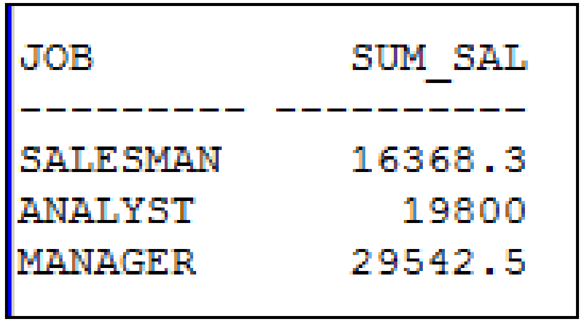
- `APPROX_FOR_PERCENTILE` converts exact percentile functions to their approximate percentile function counterparts.

19c

Approximate Top-N Query Processing

- Use the new approximate functions, APPROX_COUNT and APPROX_SUM to replace their exact counterparts of the exact version, COUNT and SUM.
- For each APPROX_COUNT / APPROX_SUM that appears in the SELECT list, a corresponding APPROX_RANK function in the HAVING clause is required.

```
SQL> SELECT job, 0.9 * APPROX_SUM(sal)
     FROM emp
     GROUP BY job
     HAVING APPROX_RANK(ORDER BY APPROX_SUM(sal) desc) <= 3;
```

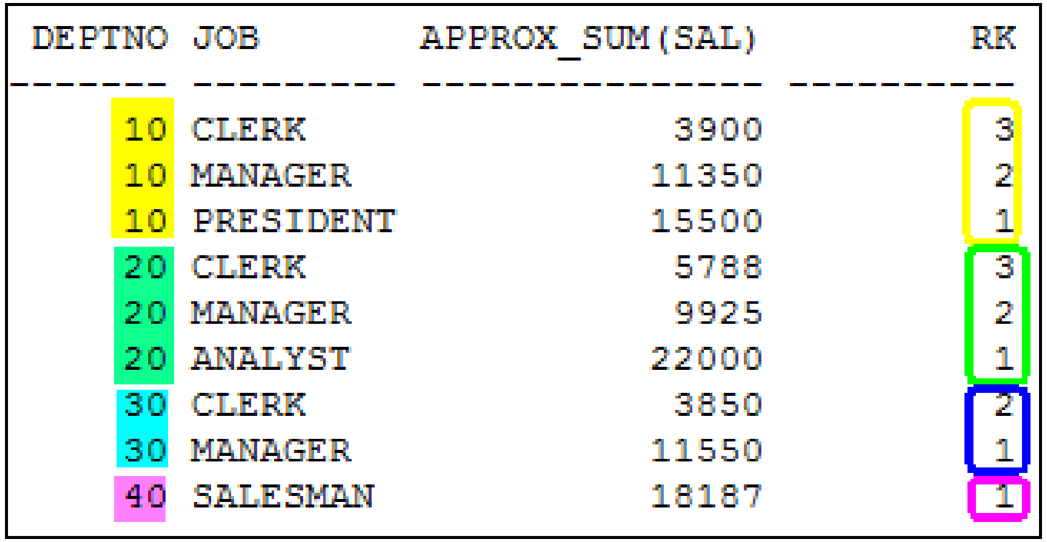


```
JOB              SUM_SAL
--------- ----------
SALESMAN     16368.3
ANALYST        19800
MANAGER      29542.5
```

Approximate Top-N Query Processing: Example 1

- Find the jobs that are among the top 10 in terms of total salary per department.

```
SQL> SELECT deptno, job, APPROX_SUM(sal),
          APPROX_RANK(partition by deptno ORDER BY APPROX_SUM(sal) desc) Rk
     FROM emp
     GROUP BY deptno, job
     HAVING APPROX_RANK(partition by deptno ORDER BY APPROX_SUM(sal) desc)
            <= 10;
```



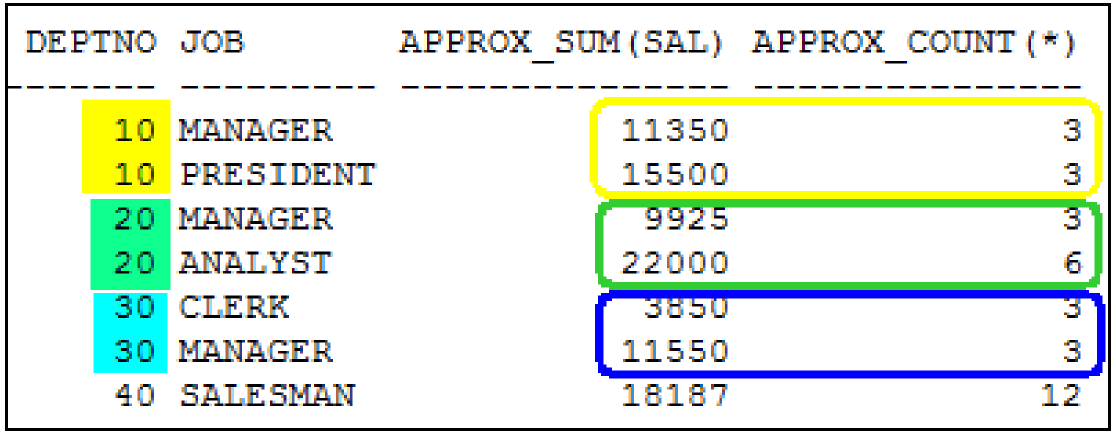
```
DEPTNO JOB       APPROX_SUM(SAL)         RK
------- --------- --------------- ----------
     10 CLERK                3900          3
     10 MANAGER             11350          2
     10 PRESIDENT           15500          1
     20 CLERK                5788          3
     20 MANAGER              9925          2
     20 ANALYST             22000          1
     30 CLERK                3850          2
     30 MANAGER             11550          1
     40 SALESMAN            18187          1
```

Approximate Top-N Query Processing: Example-2

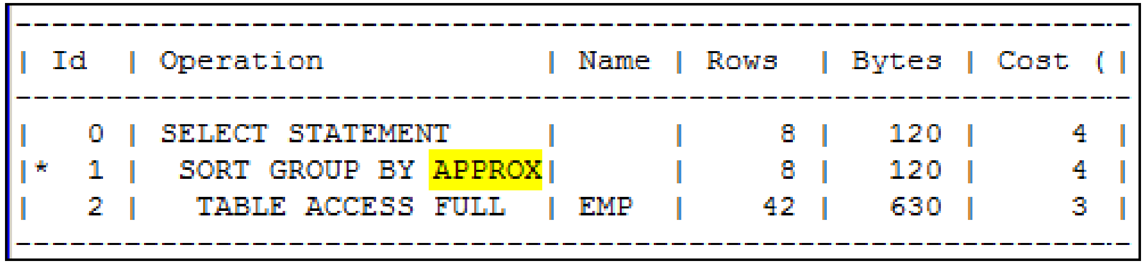
- Find the jobs that are among the top 2 in terms of total salary, and among the top 3 in terms of number of employees holding the job titles per department.

```
SQL> SELECT deptno, job, APPROX_SUM(sal), APPROX_COUNT(*)
     FROM emp GROUP BY deptno, job
     HAVING APPROX_RANK(partition by deptno order by APPROX_SUM(sal) desc)
            <= 2
     AND APPROX_RANK(partition by deptno order by APPROX_COUNT(*) desc) <= 3;
```

- There can be multiple approximate functions in the `SELECT` list ➔ For each approximate function, there has to be a corresponding predicate in the `HAVING` clause.



```
DEPTNO JOB       APPROX_SUM(SAL) APPROX_COUNT(*)
------ --------- --------------- ---------------
    10 MANAGER             11350               3
    10 PRESIDENT           15500               3
    20 MANAGER              9925               3
    20 ANALYST             22000               6
    30 CLERK                3850               3
    30 MANAGER             11550               3
    40 SALESMAN            18187              12
```

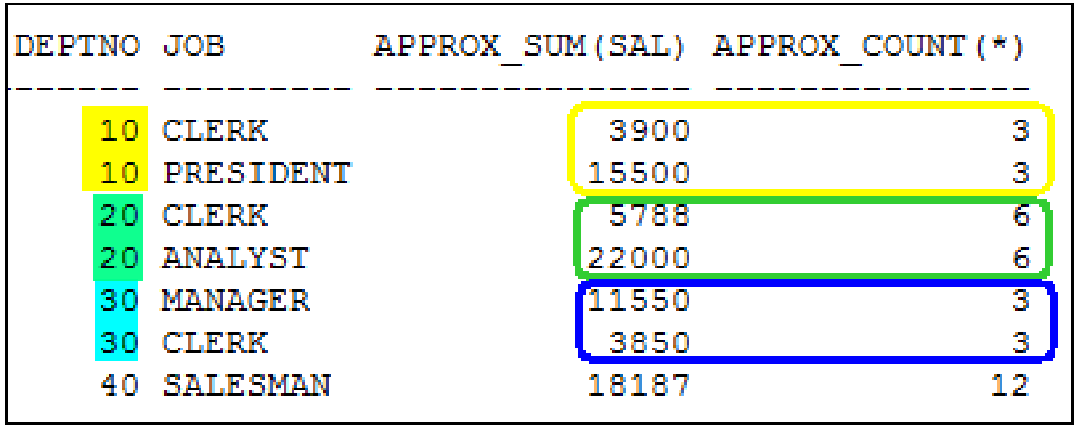


```
-------------------------------------------------------------------
| Id  | Operation              | Name | Rows  | Bytes | Cost ( |
-------------------------------------------------------------------
|   0 | SELECT STATEMENT       |      |     8 |   120 |     4  |
|*  1 |  SORT GROUP BY APPROX  |      |     8 |   120 |     4  |
|   2 |   TABLE ACCESS FULL    | EMP  |    42 |   630 |     3  |
-------------------------------------------------------------------
```

Approximate Top-N Query Processing: Example-3

- Find the jobs that are among the top 3 in terms of total salary, and among the top 2 in terms of number of employees holding the job titles per department.

```
SQL> SELECT deptno, job, APPROX_SUM(sal), APPROX_COUNT(*)
     FROM emp
     GROUP BY deptno, job
     HAVING APPROX_RANK(partition by deptno order by APPROX_SUM(sal) desc)
            <= 3
     AND APPROX_RANK(partition by deptno order by APPROX_COUNT(*) desc)
            <= 2;
```

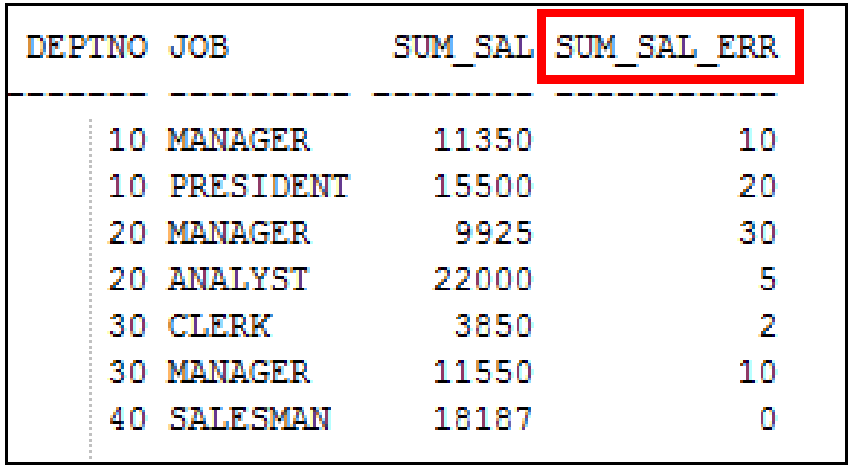


```
DEPTNO JOB       APPROX_SUM(SAL) APPROX_COUNT(*)
------ --------- --------------- ---------------
    10 CLERK                3900               3
    10 PRESIDENT           15500               3
    20 CLERK                5788               6
    20 ANALYST             22000               6
    30 MANAGER             11550               3
    30 CLERK                3850               3
    40 SALESMAN            18187              12
```

Approximate Top-N Query Processing: Example-4

- Report the accuracy of the approximate aggregate by using the `MAX_ERROR` attribute.

```
SQL> SELECT deptno, job, APPROX_SUM(sal) sum_sal,
                         APPROX_SUM(sal, 'MAX_ERROR') sum_sal_err
     FROM emp
     GROUP BY deptno, job
     HAVING APPROX_RANK(partition by deptno order by APPROX_SUM(sal) desc)
            <= 2;
```



```
DEPTNO JOB          SUM_SAL SUM_SAL_ERR
------- --------- -------- -----------
     10 MANAGER      11350          10
     10 PRESIDENT    15500          20
     20 MANAGER       9925          30
     20 ANALYST      22000           5
     30 CLERK         3850           2
     30 MANAGER      11550          10
     40 SALESMAN     18187           0
```

Summary

In this lesson, you should have learned how to:

- Query inlined external tables
- Manage in-memory external tables
- Use the new query capabilities of the Analytic view
- Create and use polymorphic table functions
- Use new functions for approximate Top-N queries

Practice 7: Overview

- 7-1: Querying Inlined External Tables
- 7-2: Populating External Tables in an In-Memory Column Store
- 7-3: Using Hierarchy-Based Predicates and Calculated Measures on Analytic Views
- 7-4: Using Polymorphic Table Functions

8. Sharding Enhancements

Objectives

After completing this lesson, you should be able to describe:

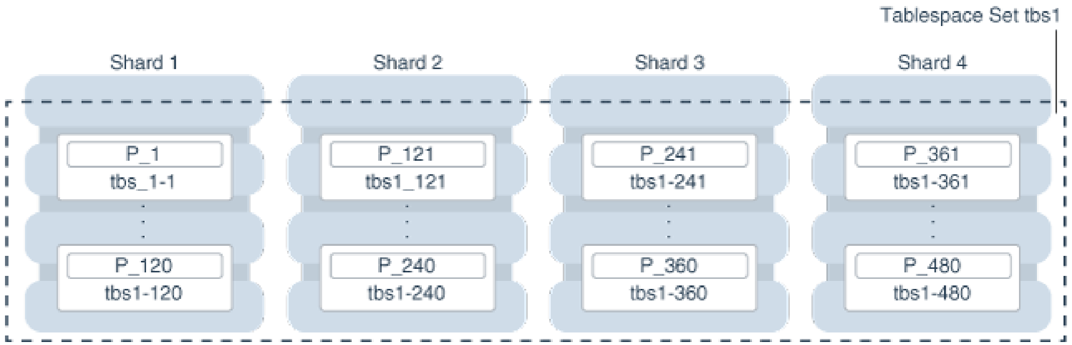
- User-defined sharding method
- Support for PDBs as shards
- Oracle GoldenGate enhancements for Oracle Sharding support
- Query System Objects Across Shards
- Setting multi-shard query data consistency level
- Sharding support for JSON, LOBs, and spatial objects
- Improved multi-shard query enhancements
- Where to find Oracle Sharding documentation in Oracle Database 19c

12c

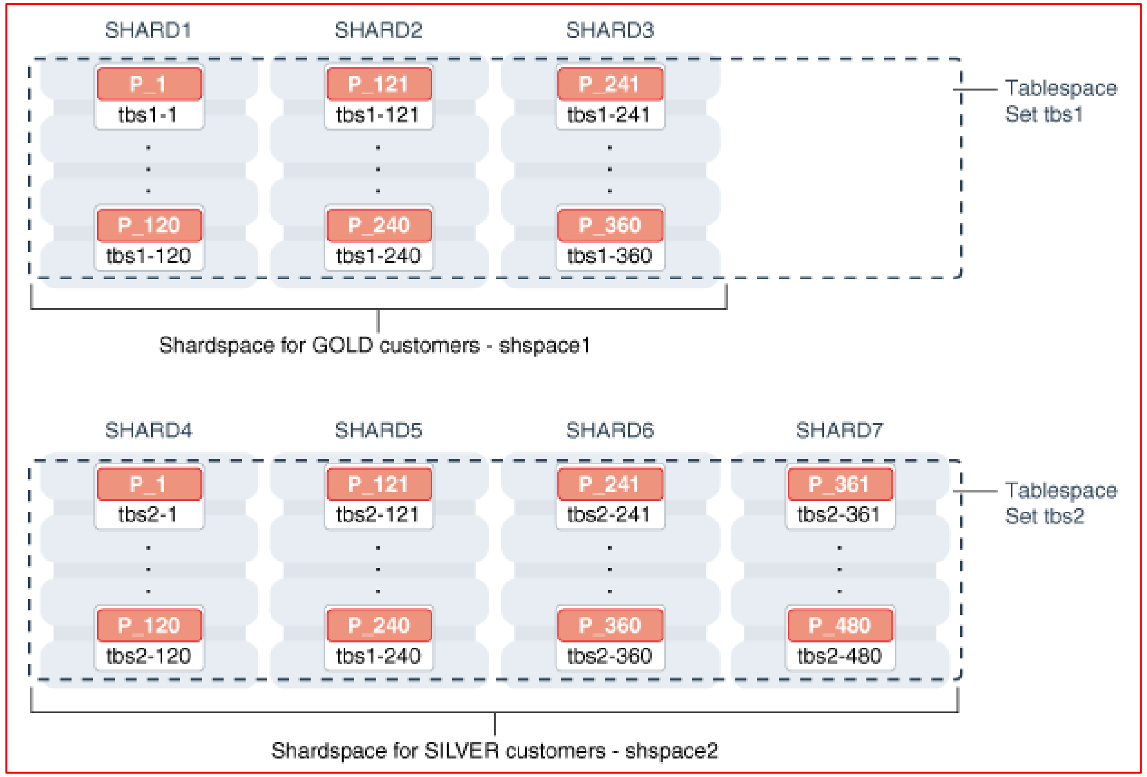
System-Managed and Composite Sharding Methods

Only two methods are supported in 12.2:

System-Managed Sharding:
Data is automatically distributed across shards using partitioning by consistent hash.



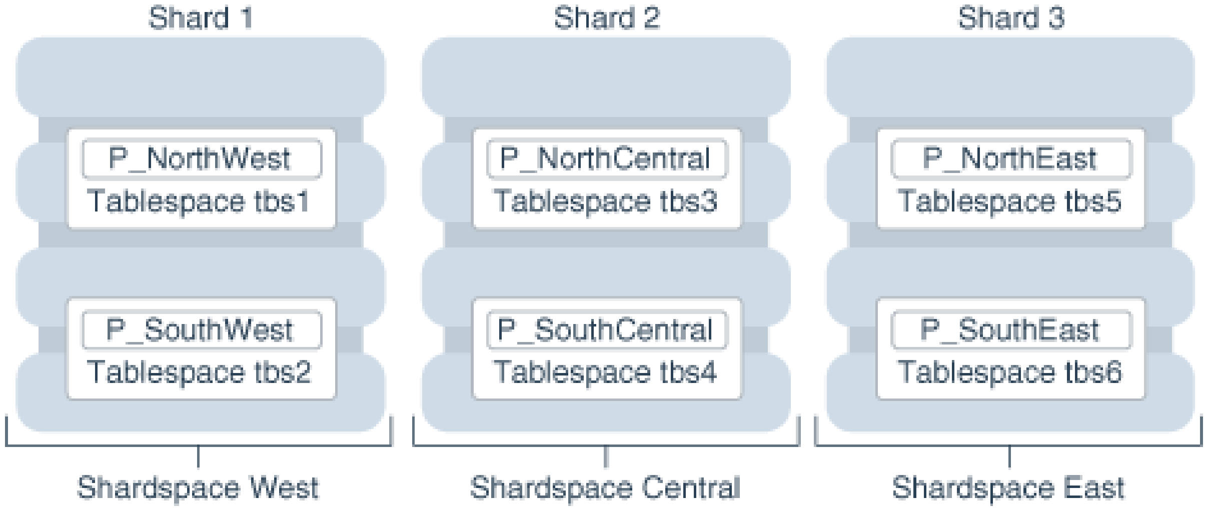
Composite Sharding:
Data is first partitioned by list or range across multiple shardspaces, and then further partitioned by consistent hash across multiple shards in each shardspace.



User-Defined Sharding Method

19c

This method enables users to define LIST- or RANGE-based sharding.



```
SQL> CREATE SHARDED TABLE accounts (id NUMBER, account_nb NUMBER, cust_id NUMBER,
                                    branch_id NUMBER, state VARCHAR(2), status VARCHAR2(1))
       PARTITION BY LIST (state)
         ( PARTITION p_northwest VALUES ('OR', 'WA') TABLESPACE ts1,
           …
           PARTITION p_northeast VALUES ('NY', 'VM', 'NJ') TABLESPACE ts5,
           PARTITION p_southeast VALUES ('FL', 'GA') TABLESPACE ts6 );
```

Support for PDBs as Shards

12c Sharded databases must consist of sharding catalogs and shards that can be:

- Single-instance database
- Oracle RAC–enabled stand-alone databases
- CDBs not supported

19c

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

```
GDSCTL> ADD CDB -connect db11 -pwd GSMUSER_password
GDSCTL> ADD SHARD -cdb db11 -connect connect_string –shardgroup shgrp1 -deploy_as active_standby -pwd
GSMUSER_password
```

Improved Oracle GoldenGate Support

12c

- Split chunks not supported

19c

- Split chunk support
- Automatic CDR support of tables with unique indexes/constraints

Query System Objects Across Shards

12c

- Shards managed individually
- No aggregate views from all shards

19c

`SHARDS()` clause and `shard_id`

```
SQL> SELECT sql_text, shard_id FROM SHARDS(sys.v$sql) a WHERE a.sql_id = '1234';
```

- Query performance views

```
SQL> SELECT shard_id, callspersec FROM SHARDS(v$servicemetric)
     WHERE  service_name LIKE 'oltp%' AND group_id = 10;
```

- Statistics collection

```
SQL> SELECT table_name, partition_name, blocks, num_rows
     FROM SHARDS(dba_tab_partition) p WHERE p.table_owner = :1;
```

Consistency Levels for Multi-Shard Queries

- 12c Multi-shard queries always used SCN synchronization and were resource intensive

- 19c New initialization parameter: `MULTISHARD_QUERY_DATA_CONSISTENCY`

```
SQL> ALTER SYSTEM SET MULTISHARD_QUERY_DATA_CONSISTENCY =
                                delayed_standby_allowed
                           SCOPE=SPFILE;
```

Sharding Support for JSON, LOBs, and Spatial Objects

19c

System-managed sharded databases:

- Create tablespace set for LOBs

```
SQL> CREATE TABLESPACE SET lobtss1;
```

- Include tablespace set for LOBs in parent table `CREATE`

```
SQL> CREATE SHARDED TABLE customers (CustId VARCHAR2(60) NOT NULL, …
                                     image       BLOB,
            CONSTRAINT pk_customers PRIMARY KEY (CustId),
            CONSTRAINT json_customers CHECK (CustProfile IS JSON))
     TABLESPACE SET TSP_SET_1
     LOB(image) STORE AS (TABLESPACE SET LOBTSS1)
     PARTITION BY CONSISTENT HASH (CustId) PARTITIONS AUTO;
```

Sharding Support for JSON, LOBs, and Spatial Objects

19c

Composite sharded databases:

- Create tablespace sets for LOBs

```
SQL> CREATE TABLESPACE SET LOBTSS1 IN SHARDSPACE cust_america ... ;
SQL> CREATE TABLESPACE SET LOBTSS2 IN SHARDSPACE cust_europe ... ;
```

- Include tablespace sets for LOBs in parent table `CREATE`

```
SQL> CREATE SHARDED TABLE customers ( CustId VARCHAR2(60) NOT NULL, … image BLOB,
                  CONSTRAINT pk_customers PRIMARY KEY (CustId),
                  CONSTRAINT json_customers CHECK (CustProfile IS JSON))
        PARTITIONSET BY LIST (GEO) PARTITION BY CONSISTENT HASH (CustId)
        PARTITIONS AUTO (PARTITIONSET america VALUES ('AMERICA')
                                      TABLESPACE SET tsp_set_1
                         LOB(image) STORE AS (TABLESPACE SET LOBTSS1),
                         PARTITIONSET europe VALUES ('EUROPE')
                                      TABLESPACE SET tsp_set_2
                         LOB(image) STORE AS (TABLESPACE SET LOBTSS2));
```

Sharding Support for JSON, LOBs, and Spatial Objects

19c

User-defined sharded databases:

- Create tablespace sets for LOBs

```
SQL> CREATE TABLESPACE lobts1 … IN SHARDSPACE shspace1;
SQL> CREATE TABLESPACE lobts2 … in shardspace shspace2;
```

- Include tablespaces for LOBs in parent `CREATE` table

```
SQL> CREATE SHARDED TABLE customers (CustId VARCHAR2(60) NOT NULL, …
                                     image       BLOB,
                    CONSTRAINT pk_customers PRIMARY KEY (CustId),
                    CONSTRAINT json_customers CHECK (CustProfile IS JSON))
      PARTITION BY RANGE (CustId)
      ( PARTITION ck1 values less than ('m') tablespace ck1_tsp
                      LOB(image) store as (TABLESPACE LOBTS1),
        PARTITION ck2 values less than (MAXVALUE) tablespace ck2_tsp
                      LOB(image) store as (tablespace LOBTS2));
```

Improved Multi-Shard Query Support

12c

- There are restrictions on query shapes.
- Only system-managed sharding is supported.

19c

- All query shapes supported
- System-managed, user-defined, and composite sharding methods supported
- Centralized execution plan display available
- Oracle supplied objects in queries
- Multi-column sharding keys supported
- SET operators supported

Oracle Sharding Documentation

12c

Oracle Sharding documentation contained in *Oracle Database Administrator's Guide*, Part VII "*Sharded Database Management*".

19c

Oracle Sharding documentation has its own book, *Oracle Database Using Oracle Sharding*, included in Oracle Database documentation library in Oracle Help Center.

Summary

In this lesson, you should have learned how to describe:

- User-defined sharding method
- Support for PDBs as shards
- Oracle GoldenGate enhancements for Oracle Sharding support
- Query System Objects Across Shards
- Setting multi-shard query data consistency level
- Sharding support for JSON, LOBs, and spatial objects
- Improved multi-shard query enhancements
- Where to find Oracle Sharding documentation in Oracle Database 19c

Practice 8: Overview

There are no practices for this lesson.

9. Database Sharding

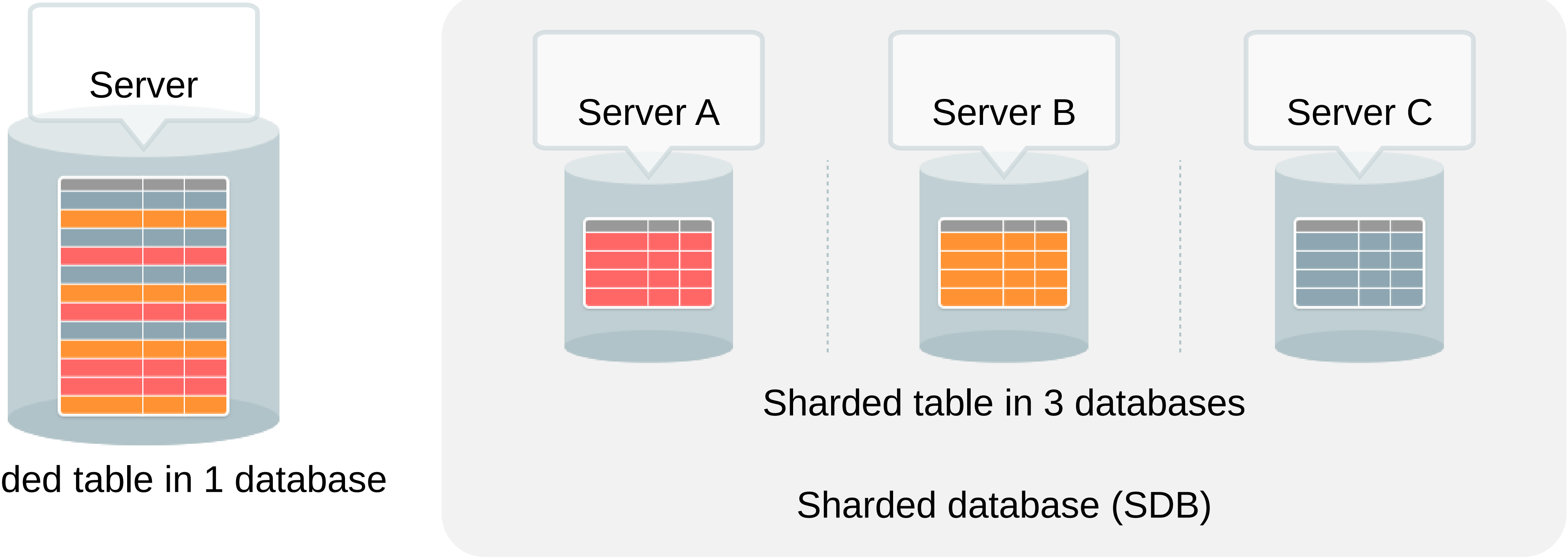
Objectives

After completing this lesson, you should be able to:

- Describe the challenges and benefits of a sharded database
- Describe sharded database architecture
- Configure a sharded database (SDB)

What Is Database Sharding?

- Shared-nothing architecture for scalability and availability
- Horizontally partitioned data across independent databases
- Loosely coupled data tier without clusterware



Unsharded table in 1 database

Sharded table in 3 databases

Sharded database (SDB)

Sharding: Benefits

- Extreme scalability by adding shards (independent databases)
- Fault containment by eliminating single points of failure
- Global data distribution with the ability to store particular data in a specific shard
- Rolling upgrades with independent availability of shards
- Simplicity of cloud deployment with different sized shards

Oracle Sharding: Advantages

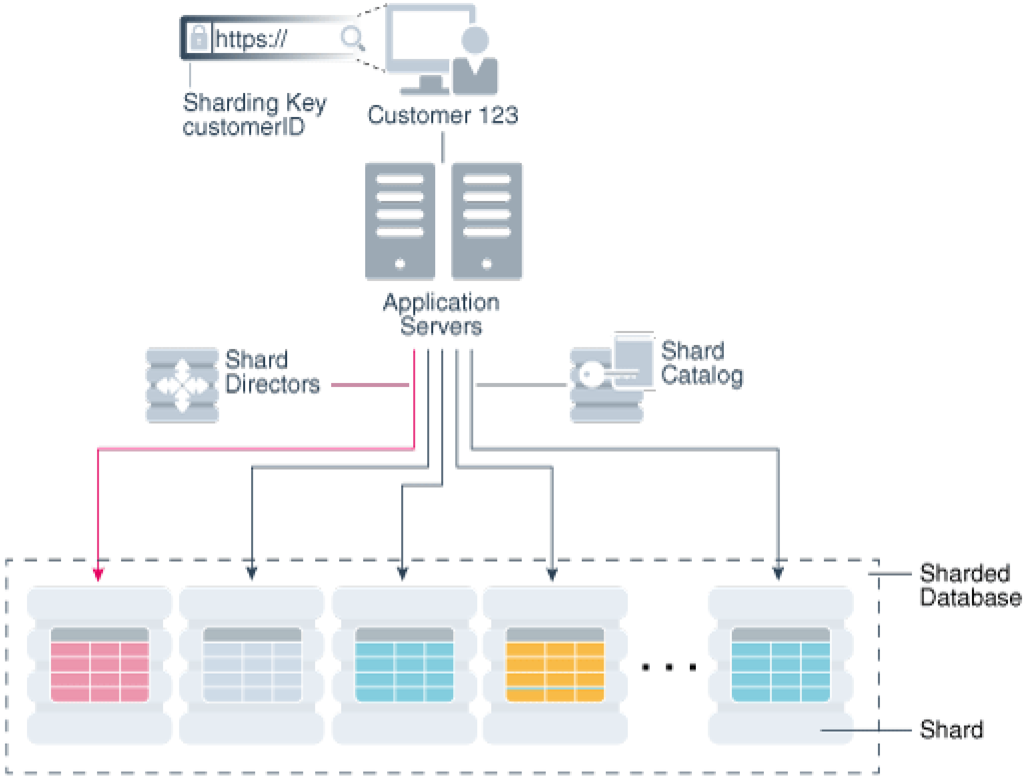
- Relational schemas
- Database partitioning
- ACID properties and read consistency
- SQL and other programmatic interfaces
- Complex data types
- Online schema changes
- Multicore scalability
- Advanced security
- Compression
- High availability features
- Enterprise-scale backup and recovery

Application Considerations for Sharding

- Available only in new database creations
- Intended for OLTP applications that:
  - Have a well-defined data model and data distribution strategy
  - Have a hierarchical tree structure data model with a single root table
  - Primarily access data by using a sharding key that is stable and with high cardinality
  - Generally access data associated with a single value for the sharding key
  - Use Oracle integrated connection pools (UCP, OCI, ODP.NET, and JDBC) to connect to the sharded database

Components of Database Sharding

- Sharded database (SDB)
- Shards
- Global service
- Shard catalog
- Shard directors
- Connection pools
- Management tools
  - GDSCTL
  - EMCC 13c



Shard Catalog

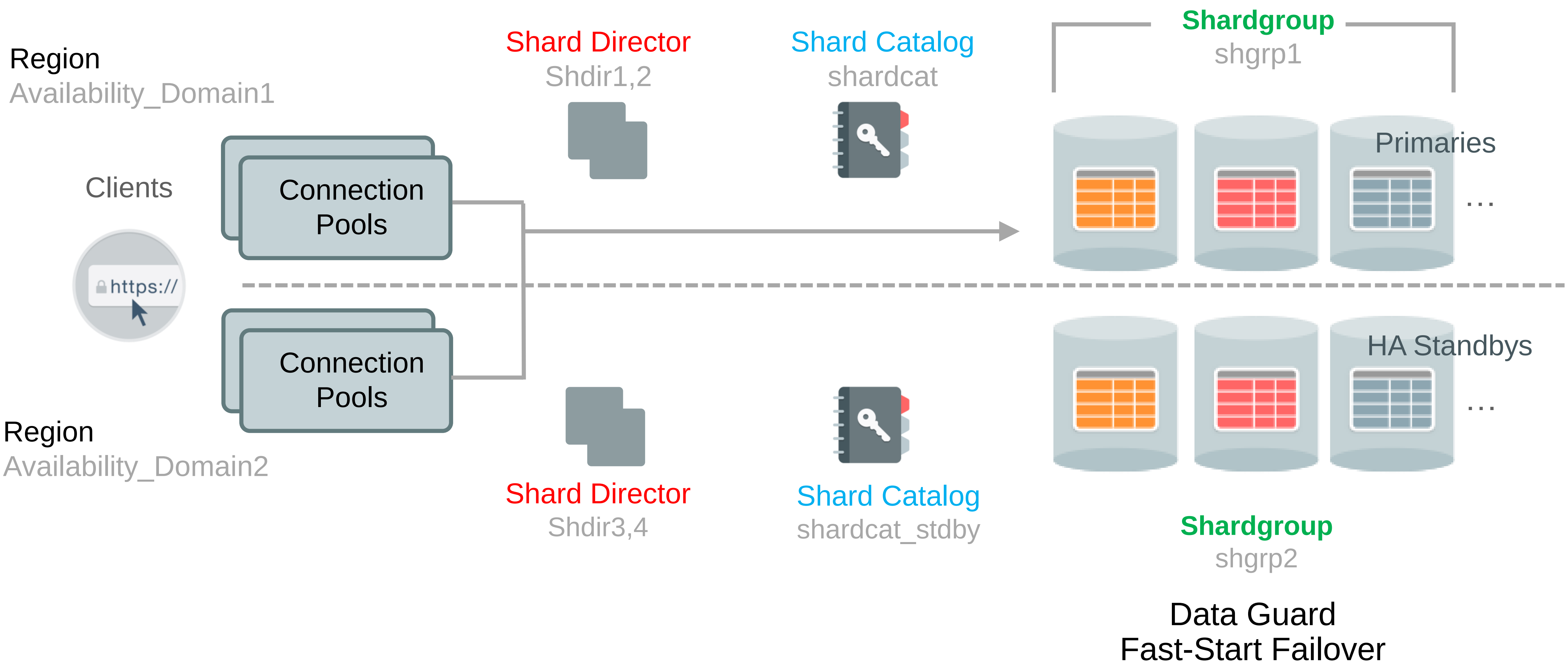
- Is an enhanced Global Data Services (GDS) catalog containing persistent sharding configuration data
- Is used to initiate all configuration changes
- Is used for connections for all DDL commands
- Contains the master copy of all duplicated tables
- Replicates changes to duplicated tables by using materialized views
- Acts as a query coordinator to process multi-shard queries

Shard Directors

The following are the key capabilities of shard directors:

- Maintaining runtime data about SDB configuration and availability of shards
- Measuring network latency between its own and other regions
- Acting as a regional listener for clients to connect to an SDB
- Managing global services
- Performing connection load balancing

Complete Deployment of a System-Managed SDB



Creating Sharded Tables

- Use a sharding key (partition key) to distribute partitions across shards at the tablespace level.
- The `NUMBER`, `INTEGER`, `SMALLINT`, `RAW`, `(N)VARCHAR`, `(N)CHAR`, `DATE`, and `TIMESTAMP` data types are supported for the sharding key.

```
SQL> CREATE SHARDED TABLE customers
         ( CustNo      NUMBER NOT NULL
         , Name        VARCHAR2(50)
         , Address     VARCHAR2(250)
         , CONSTRAINT RootPK PRIMARY KEY(CustNo) )
         PARTITION BY CONSISTENT HASH (CustNo)
         PARTITIONS AUTO TABLESPACE SET ts1;
```

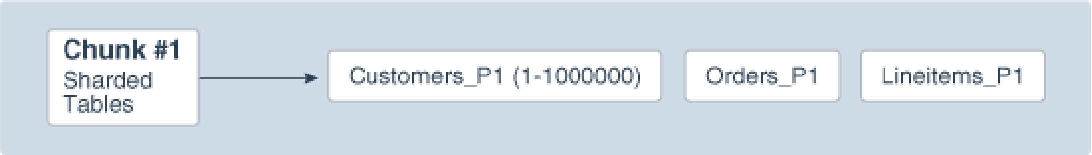
Sharded Table Family

- A set of tables sharded in the same way
- Only a single root table (table with no parent) per family
- Only a single table family per SDB
- Only a single sharding method (partitioning method) per SDB, that cannot be changed after creation

```
SQL> CREATE SHARDED TABLE Orders
         ( OrderNo   NUMBER NOT NULL
         , CustNo    NUMBER NOT NULL
         , OrderDate DATE
         , CONSTRAINT OrderPK PRIMARY KEY (CustNo, OrderNo)
         , CONSTRAINT CustFK  FOREIGN KEY (CustNo)
                      REFERENCES Customers(CustNo) )
         PARTITION BY REFERENCE (CustFK);
```

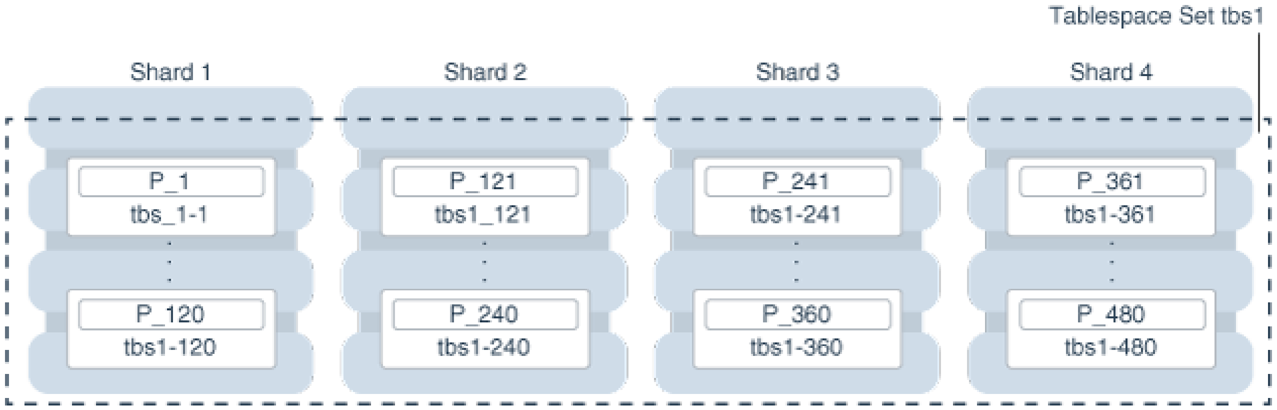
Partitions, Tablespaces, and Chunks

- Each partition of a sharded table is stored in a separate tablespace.
- The corresponding data value partitions of all the tables in a table family are always stored in the same shard.
  - Guaranteed when the tables in a table family are created in the same tablespace set
- The child tables of a table family can be stored in separate tablespace sets.
  - Uses chunks or groups of tablespaces that contain a single partition from each table with the corresponding partitions in the family



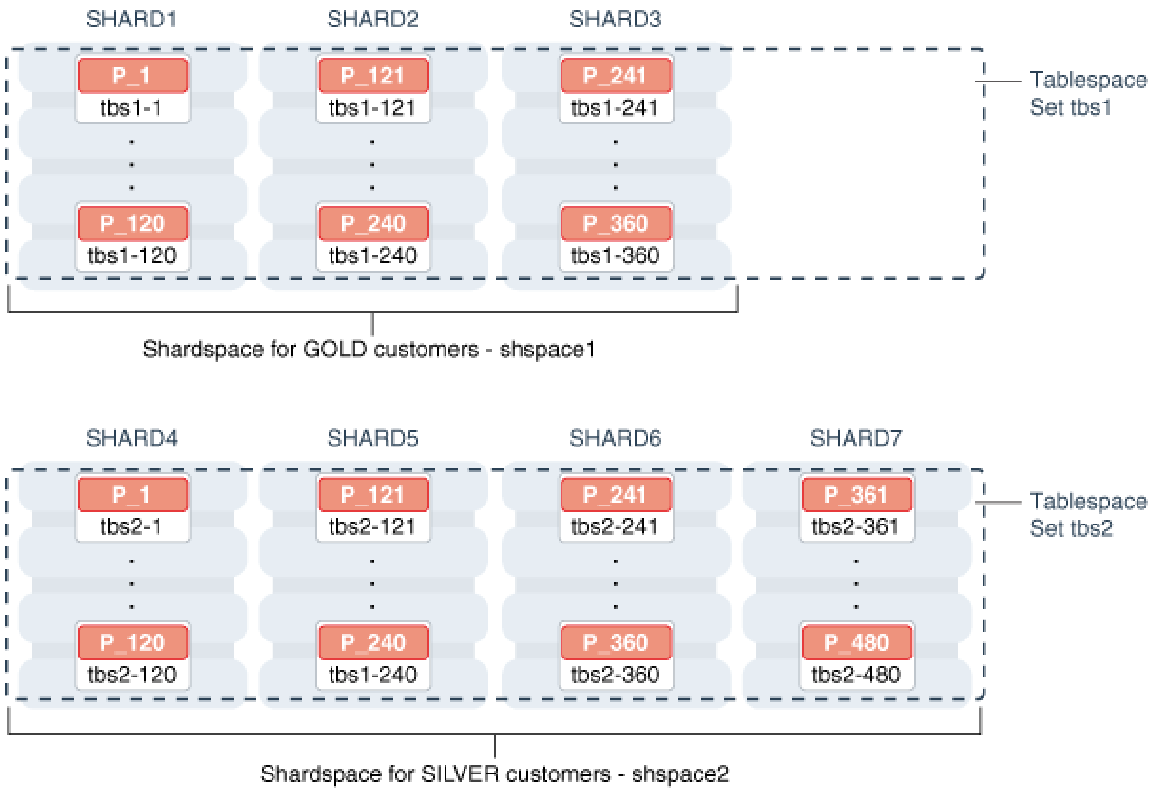
Sharding Methods: System-Managed Sharding

Data is automatically distributed across shards using partitioning by consistent hash.



Sharding Methods: Composite Sharding

Data is first partitioned by list or range across multiple shardspaces, and then further partitioned by consistent hash across multiple shards in each shardspace.



Duplicated Tables

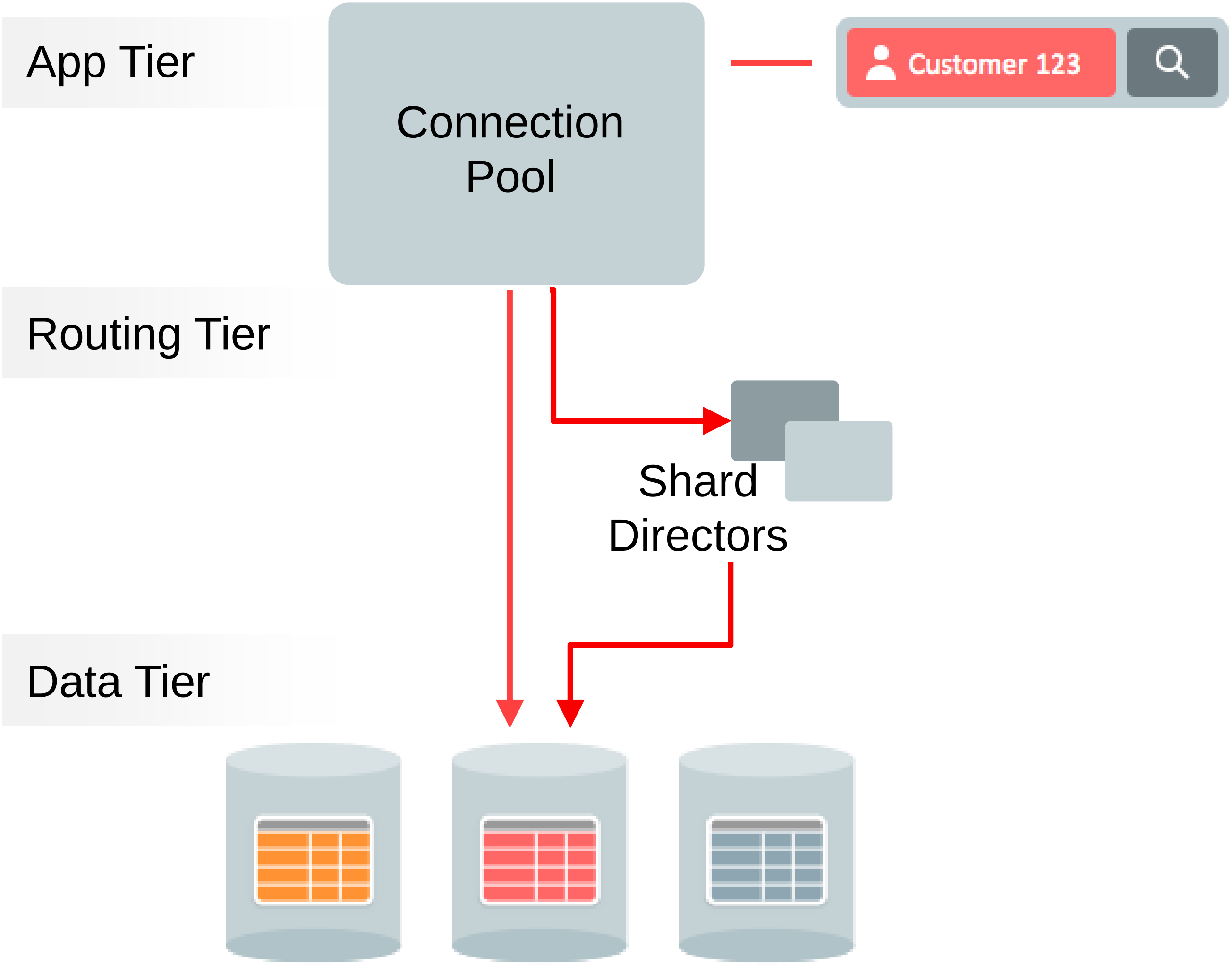
- Are nonsharded tables that duplicate data on all shards
- Help eliminate cross-shard queries
- Are created in the shard catalog
- Use materialized view replication
- Can be refreshed by using a refresh frequency (default 60 seconds) that is set with the `SHRD_DUPL_TABLE_REFRESH_RATE` initialization parameter
- Cannot be stored in tablespaces used for sharded tables

```
SQL> CREATE DUPLICATED TABLE Products
          ( StockNo     NUMBER PRIMARY KEY
          , Description VARCHAR2(20)
          , Price       NUMBER(6,2));
```

Routing in an Oracle Sharded Environment

- Direct Routing based on `sharding_key`
  - For OLTP workloads that specify `sharding_key` (for example, `customer_id`) during connect
  - Enabled by enhancements to mid-tier connection pools and drivers
- Proxy Routing via a coordinator (shard catalog)
  - For workloads that cannot specify `sharding_key` (as part of a connection)
  - For reporting, batch jobs
  - Queries spanning one or more or all shards

Direct Routing via Sharding Key



App Tier

Connection Pool

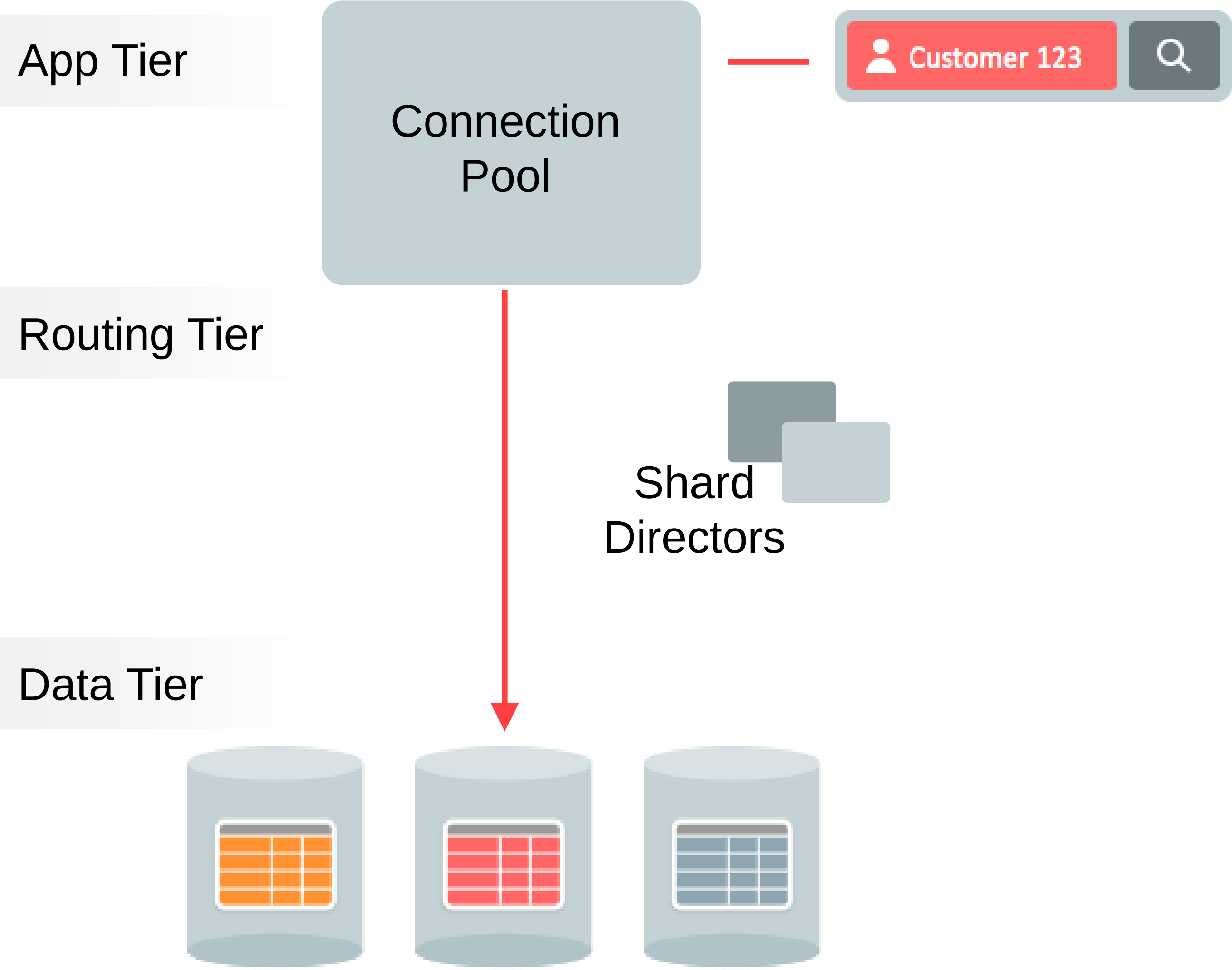
Customer 123

Routing Tier

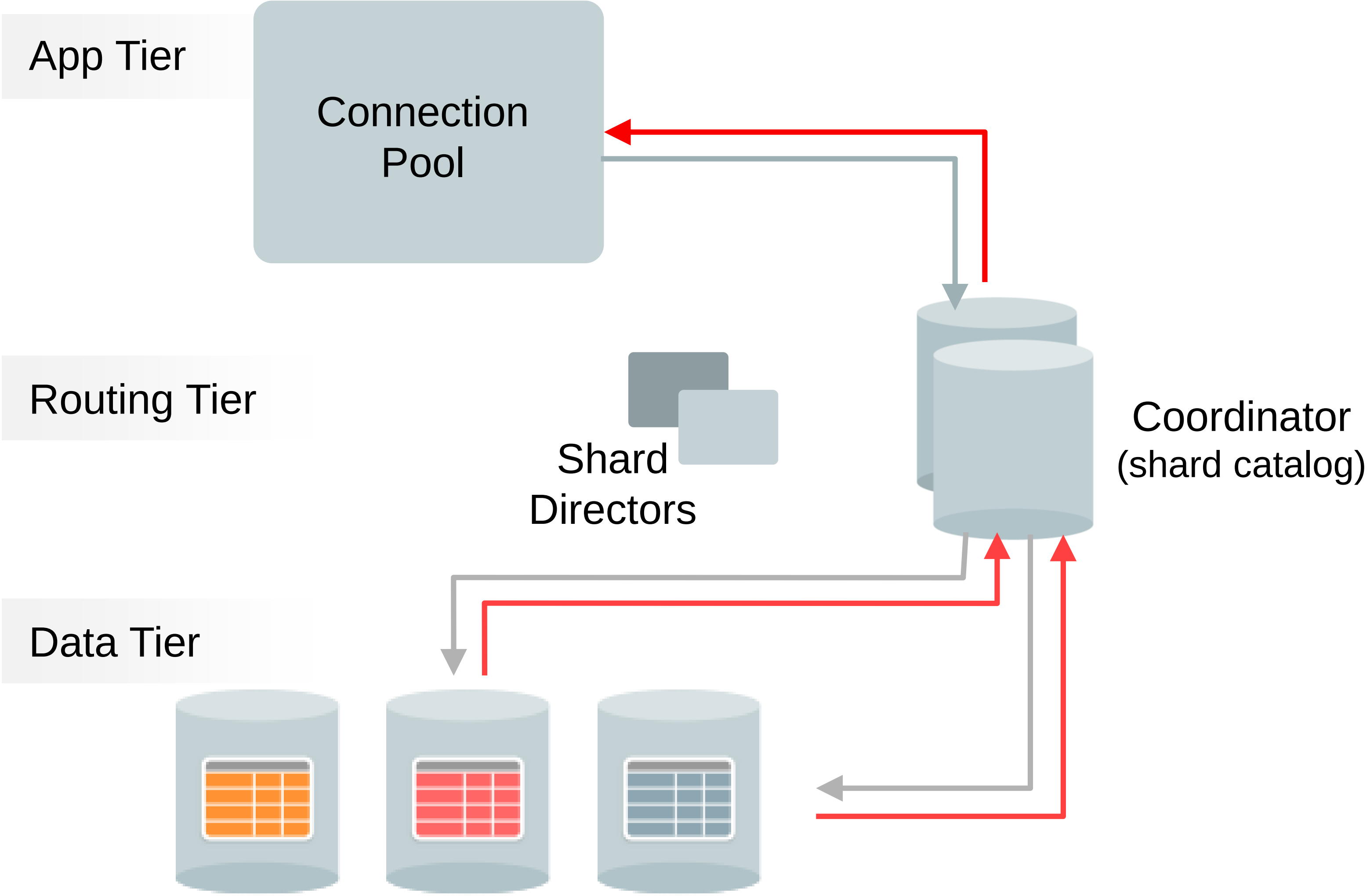
Shard Directors

Data Tier

Connection Pool as Shard Director



Proxy Routing: Limited to System Managed in 12.2.0.1



Lifecycle Management of SDB

- The DBA can manually move or split a chunk from one shard to another.
- When a new shard is added, chunks are automatically rebalanced.
- Before a shard is removed, chunks must be manually moved.
- Connection pools are notified (via ONS) about a split, a move, addition or removal of shards, auto-resharding, and read-only access operations.
- All shards can be patched with one command via `opatchauto`.
- EM supports monitoring and management of SDB.

Sharding Deployment Outline: DBA Steps

1. Create users and groups on all host servers.
2. Perform an Oracle-database-software-only installation on the shard catalog server and save a response file.
3. Perform silent installations of the Oracle database software only on all the shard hosts and the additional shard catalog host.
4. Install the Oracle Global Service manager software on all the shard director hosts.
5. Create a non-container database by using DBCA on the shard catalog host with Oracle Managed Files (required).
6. Configure the remote scheduler on the shard catalog host.
7. Register the remote scheduler on each shard host with the shard catalog host.

Sharding Deployment Outline: DBA Steps

8. Use GDSCTL on the shard catalog host to create a shard catalog.
9. Use GDSCTL on the shard catalog host to create and start the shard directors.
10. Create additional shard catalogs in a different region for high availability.
11. Define the primary shardgroup (region) by using GDSCTL connected to the shard director host.
12. Define the Active Data Guard standby shardgroup by using GDSCTL connected to the shard director host.
13. Define each shard host as belonging to the primary or standby shardgroup.

Sharding Deployment Outline: DBA Steps

14. Use GDSCTL connected to the shard director host to run the `DEPLOY` command, which:
    - Creates all primary and standby shard databases using DBCA
    - Enables archiving and flashback for all shards
    - Configures Data Guard Broker with Fast-Start Failover enabled
    - Starts observers on the standby group’s shard director
15. Use GDSCTL to add and start a global service that runs on all primary shards.
16. Use GDSCTL to add and start a global service for read-only workloads on all standby shards.
17. Use SQL*Plus connected to the shard catalog database to design the sharded schema model (developer steps).

Summary

In this lesson, you should have learned how to:

- Describe the challenges and benefits of a sharded database
- Describe sharded database architecture
- Configure a sharded database (SDB)