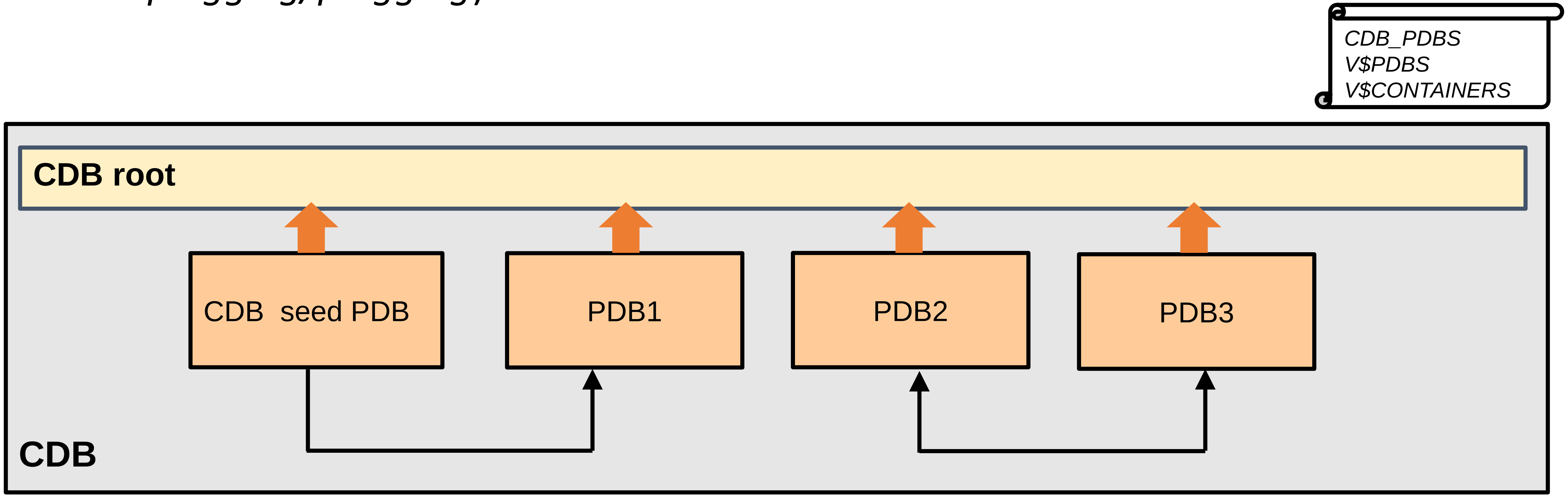
Application PDBs and Application Installation

Objectives

- After completing this lesson, you should be able to:
  - Describe application containers in CDBs
  - Explain the purpose of application root and application seed
  - Define application PDBs
  - Create application PDBs
  - Explain application installation on top of application containers
  - Install an application
  - Upgrade and patch applications
  - Describe the commonality concept in application contexts
  - Use a dynamic container map
  - Describe enhancements in various areas

Regular PDBs

- A regular PDB is a PDB within a CDB, storing data in objects independently of other PDBs.
- A regular PDB can be created from the CDB seed or from another PDB (*cloning* or *unplugging/plugging*).



PDBs and Applications

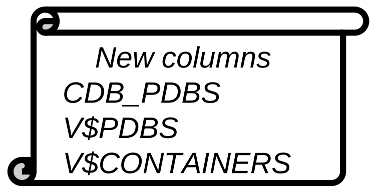
Applications in regular PDBs need to be upgraded or patched in the same CDB or across many CDBs.

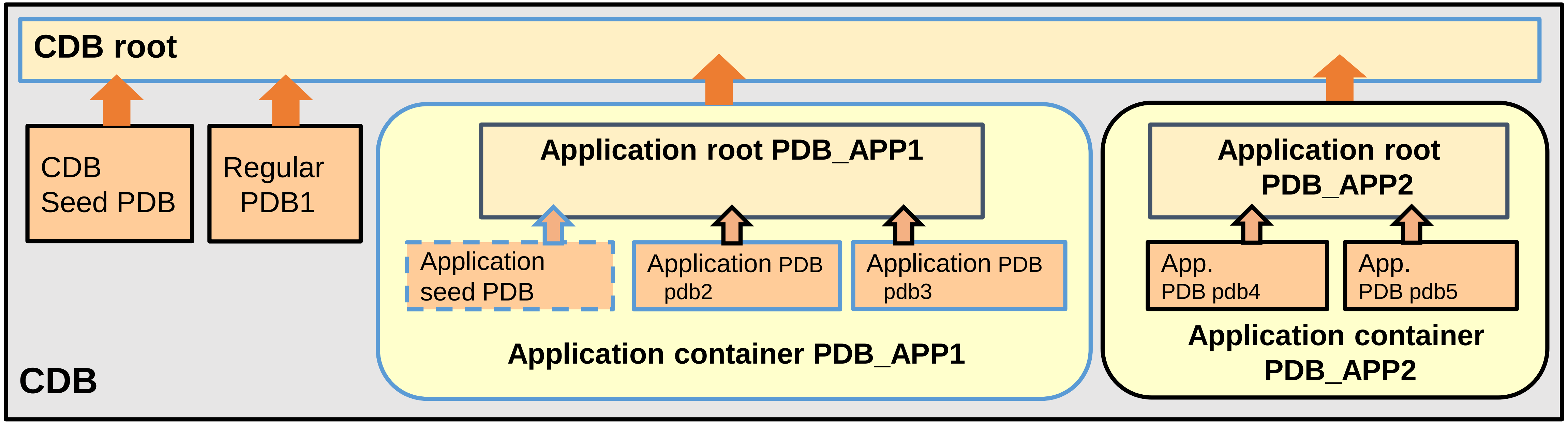
The upgrade script has to be executed in all regular PDBs individually.

No single master definition of application

Application Containers

- An application container is a collection of PDBs grouped together within a CDB to store data for an application.
  - The application root
  - An optional application seed
  - Application PDBs associated with the application root





Application Containers: Other Features

Application Master	Rapid Provisioning	Across CDBs	Report across tenants	Patching	Unplug/Plug
Metadata and common data shared across tenant PDBs	Instant provisioning of an Application PDB/Tenant (with a seed PDB)	Both local and remote PDBs can join an Application Container.	Container Data views for reporting across PDBs (*containers()* clause based)	Support for in-place simple patching	Support for Unplug/Plug upgrade across Application Root

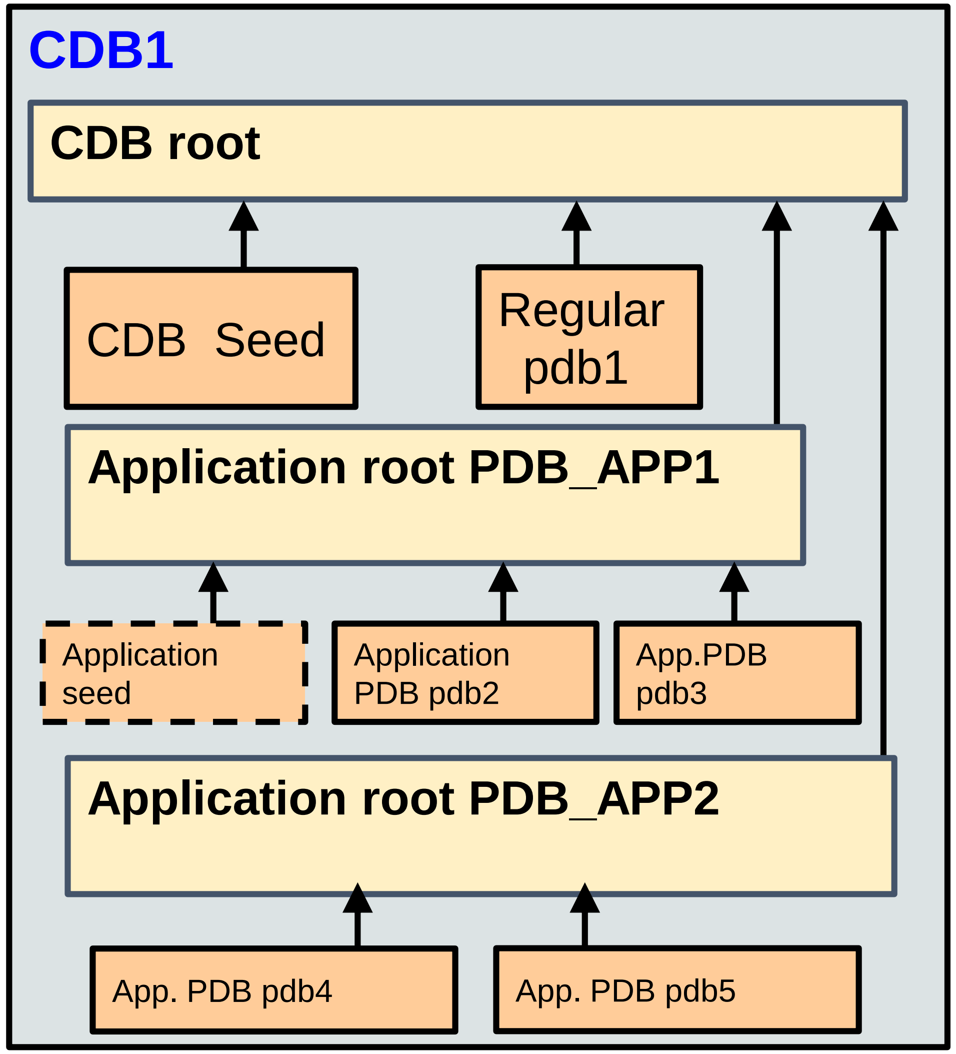
Types of Containers

New columns in

CDB_PDBS
V$PDBS
V$CONTAINERS

- The **CDB root container** (`CDB$ROOT`)
  - The first mandatory container created at CDB creation
  - Oracle system–supplied common objects and metadata
  - Oracle system–supplied common users and roles
- **Pluggable database containers** (PDBs)
  - The CDB seed (`PDB$SEED`)
    - The second mandatory container created at CDB creation
    - Oracle system–supplied common entities for new PDBs
  - Regular PDBs
  - Application containers
    - Application root PDB
    - Optional application seed PDB (*`application_container_root_name`*`$SEED`)
    - Application PDBs

Creating Application PDBs



1. Connect to the **CDB1** CDB root.
2. Create the **PDB_APP1** PDB as the application root.

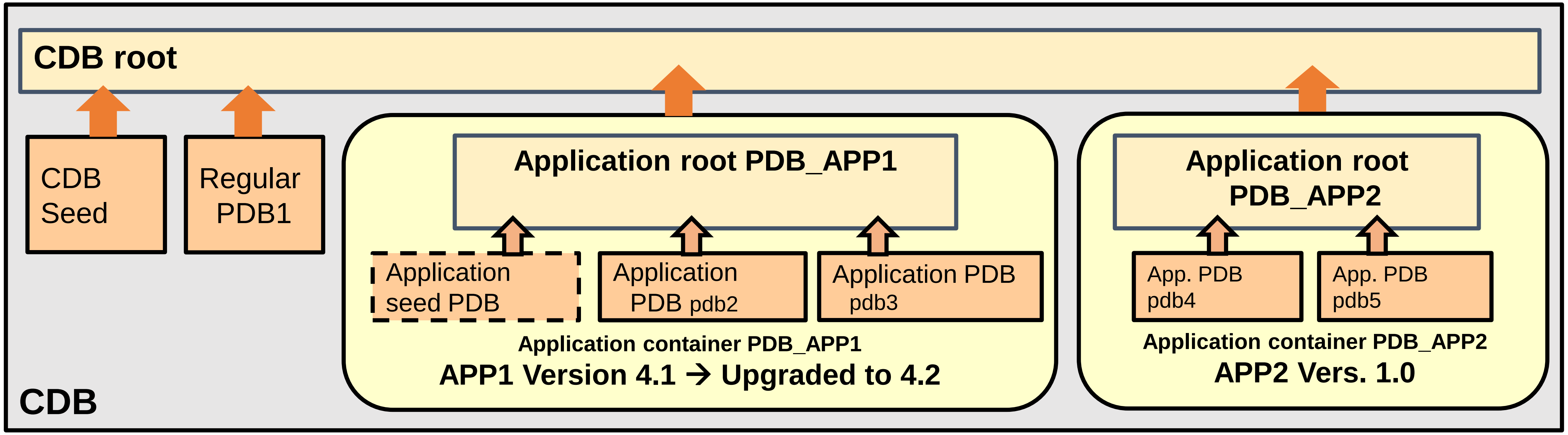
```
SQL> CONNECT / AS SYSDBA
SQL> CREATE PLUGGABLE DATABASE pdb_app
            AS APPLICATION CONTAINER …;
```

4. Install the application.
5. Optionally, create the application seed for the application PDBs in the application root.
6. Create the **PDB2** PDB as an application PDB within the **PDB_APP1** application root.
7. Create other application PDBs if required.
8. Synchronize all application PDBs with the application installed if step 5 was not completed.

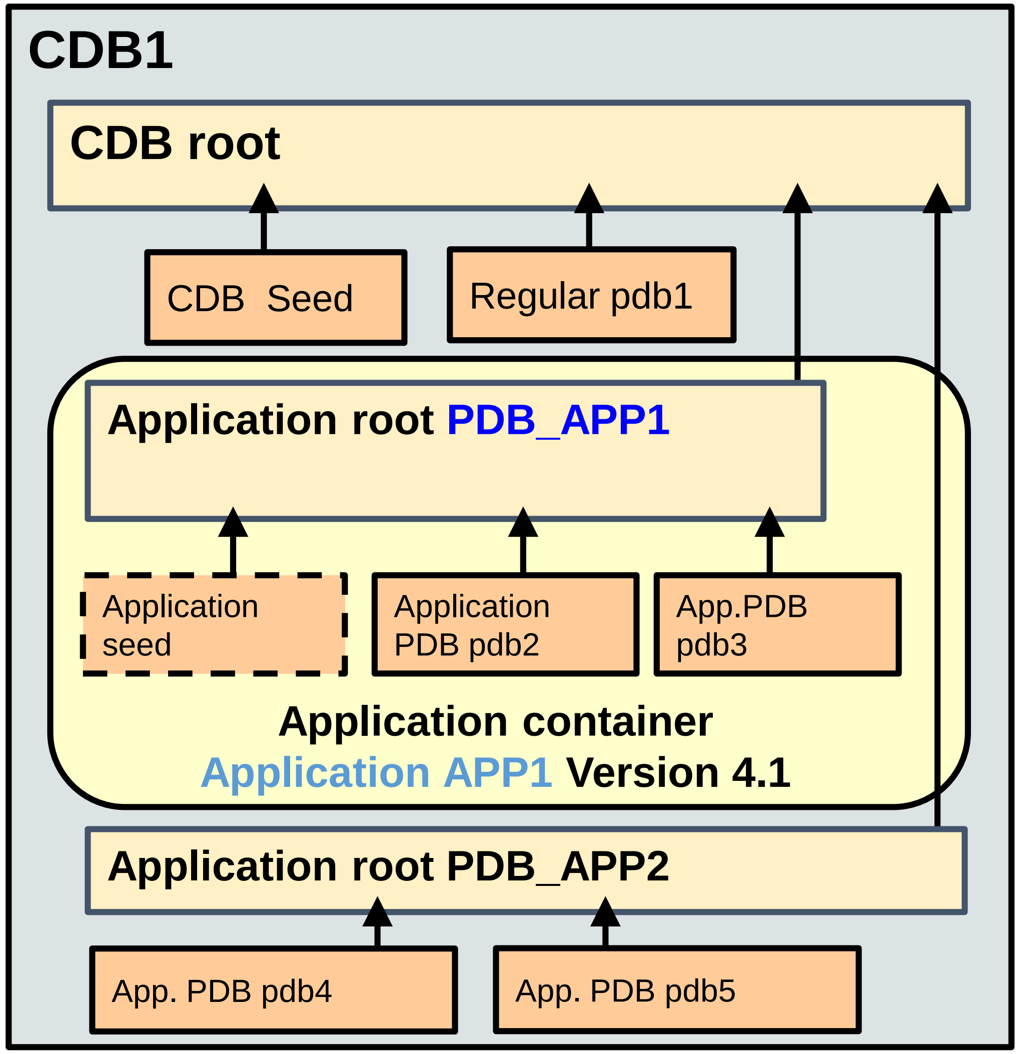
Application Name and Version

- An application container can be tagged with:
  - An application name
  - An application version
- An application can be patched, upgraded, or uninstalled.

DBA_APPLICATIONS
DBA_APP_VERSIONS
DBA_APP_PATCHES
DBA_APP_ERRORS
DBA_APP_STATEMENTS


CDB root
CDB Seed
Regular PDB1
Application root PDB_APP1
Application seed PDB
Application PDB pdb2
Application PDB pdb3
Application container PDB_APP1
APP1 Version 4.1 → Upgraded to 4.2
Application root PDB_APP2
App. PDB pdb4
App. PDB pdb5
Application container PDB_APP2
APP2 Vers. 1.0
CDB

Installing Applications



1. Connect to the **PDB_APP1** application root.
2. Assign an application name and version to the new **APP1** application that is being installed.

```
SQL> ALTER PLUGGABLE DATABASE APPLICATION app1
          BEGIN INSTALL '4.1';
```

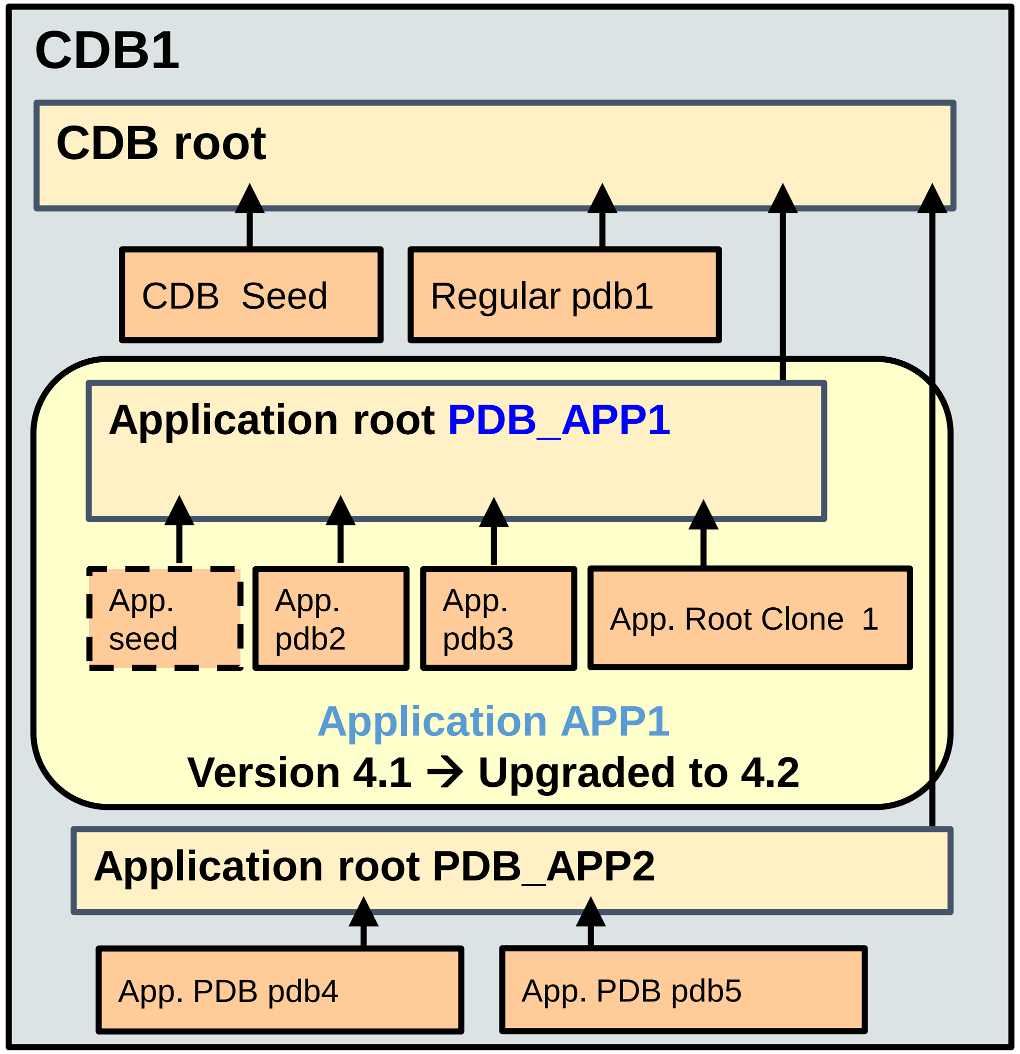
3. Execute the user-defined scripts.

```
SQL> @scripts
```

```
SQL> ALTER PLUGGABLE DATABASE APPLICATION app1
          END INSTALL '4.1';
```

```
SQL> CONNECT sys@pdb2
SQL> ALTER PLUGGABLE DATABASE APPLICATION app1
          SYNC;
```

Patching and Upgrading Applications



1. Connect to the **PDB_APP1** application root of the **APP1** application.
2. Check the current version of the **APP1** application before starting the upgrade.
3. Start the application upgrade to a higher version.

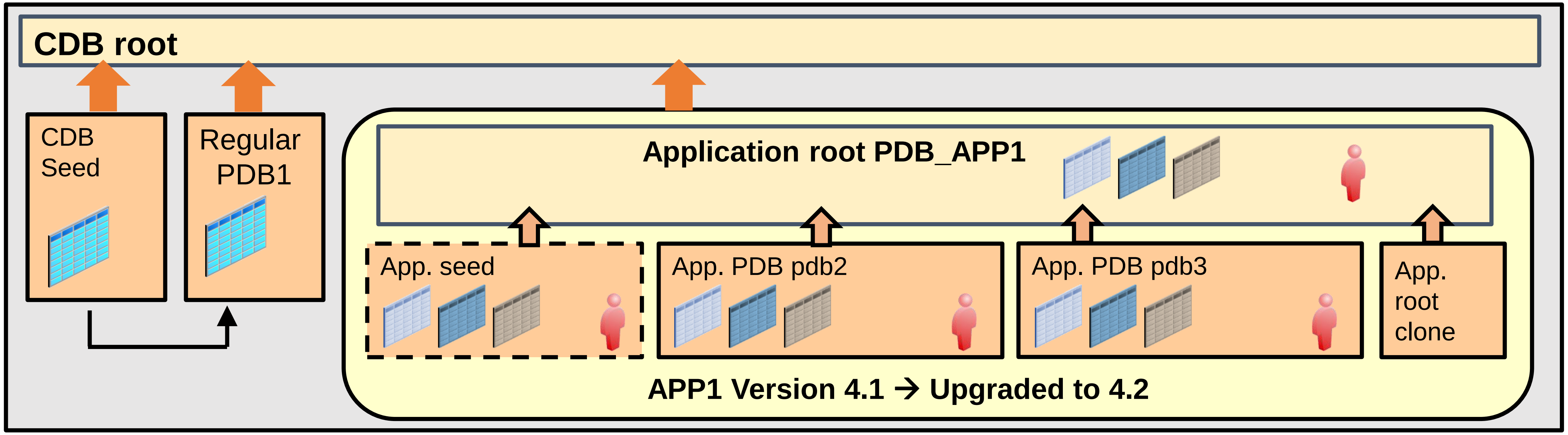
```
SQL> ALTER PLUGGABLE DATABASE APPLICATION app1
           BEGIN UPGRADE '4.1' TO '4.2';
```

```
SQL> @scripts
SQL> ALTER PLUGGABLE DATABASE APPLICATION app1
           END UPGRADE TO '4.2';
```

```
SQL> CONNECT sys@pdb2
SQL> ALTER PLUGGABLE DATABASE APPLICATION app1
           SYNC;
```

Application Common Objects

- The application root holds the common objects:
  - Users, roles, granted privileges, profiles, tables, views, and so on
- Synchronization of application PDBs with the application root is required.
- If an application is patched or upgraded, resynchronization of application PDBs is required.



Use Cases for Application Containers

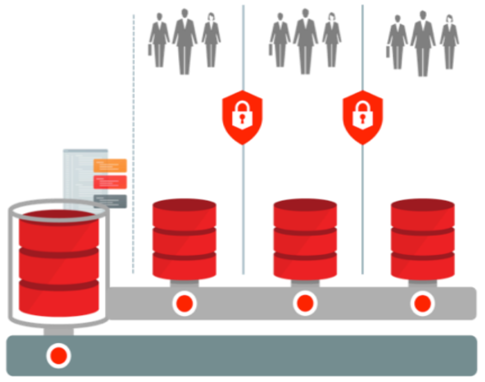
- Pure SaaS
  - Each customer’s data resides in an individual PDB.
  - All PDB-level operations are applicable on individual customer data.
  - Customer data can be securely managed.
  - Thousands of tenants can be handled.

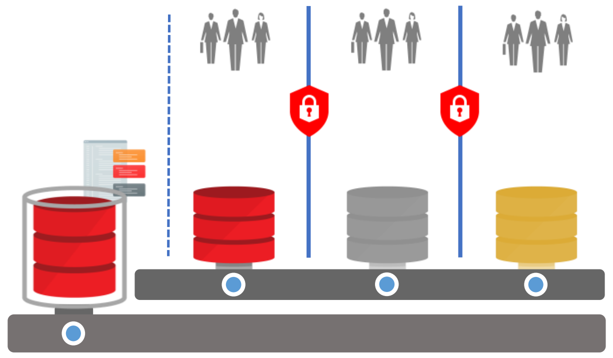
Hybrid SaaS

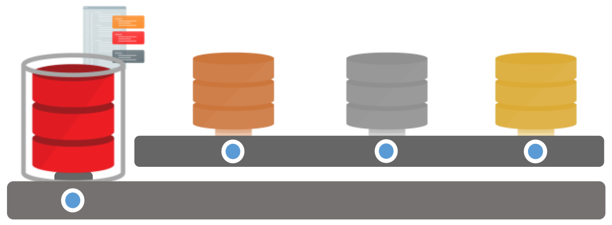
- Large customers reside in individual PDBs.
- Smaller customers share a PDB.
- It is suitable for applications with a high density of customers.
- Similar types of customers can be grouped in a PDB.
- Hundreds of thousands of tenants can be handled.

Logical DW

- Customers may address data sovereignty issues: *Country or region data will be segregated into a separate PDB.*
- There is efficient execution of ETLs for every region without impacting each other.
- The best execution plans are based on actual data distribution.

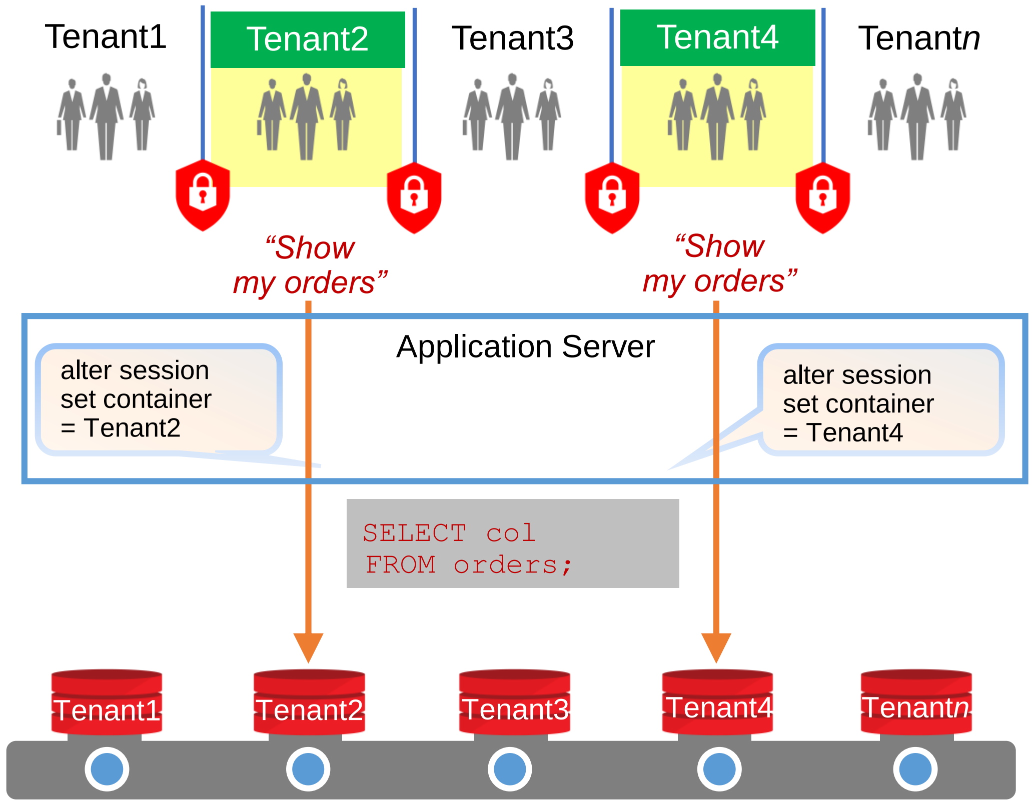






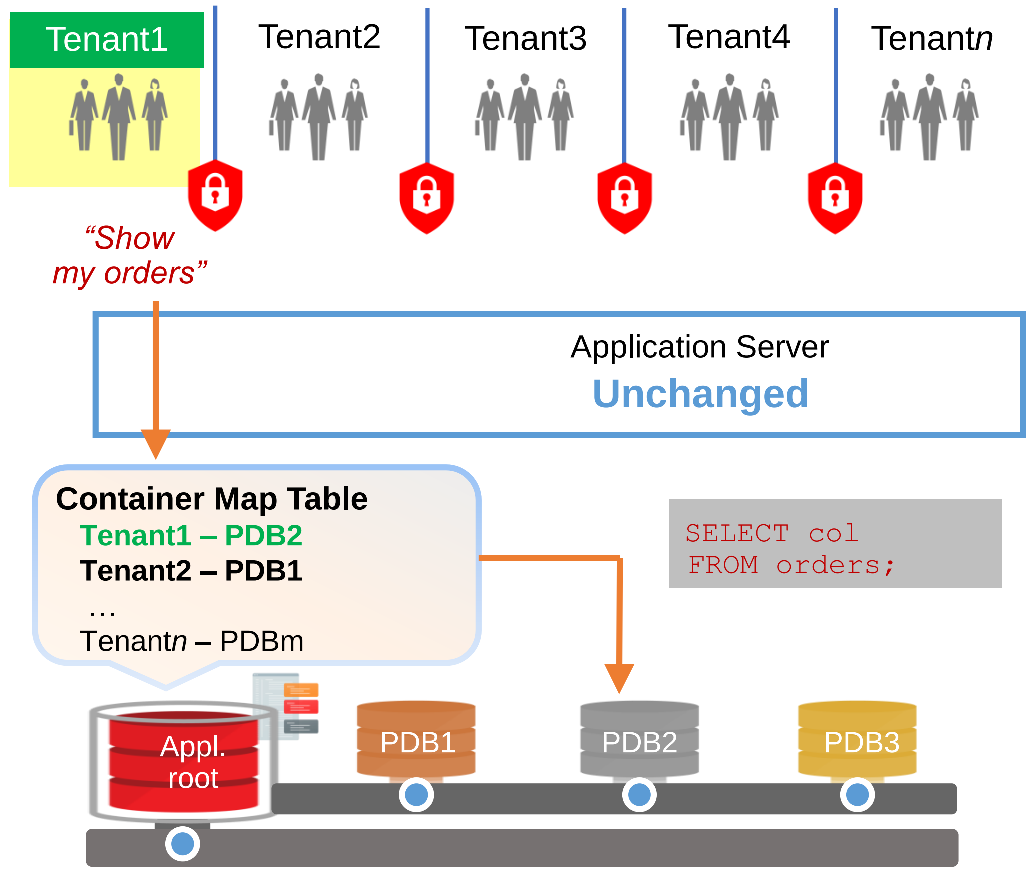
Use Case: Pure PDB-Based Versus Hybrid Model

- Pure PDB-based tenancy



```
SELECT col
FROM orders;
```

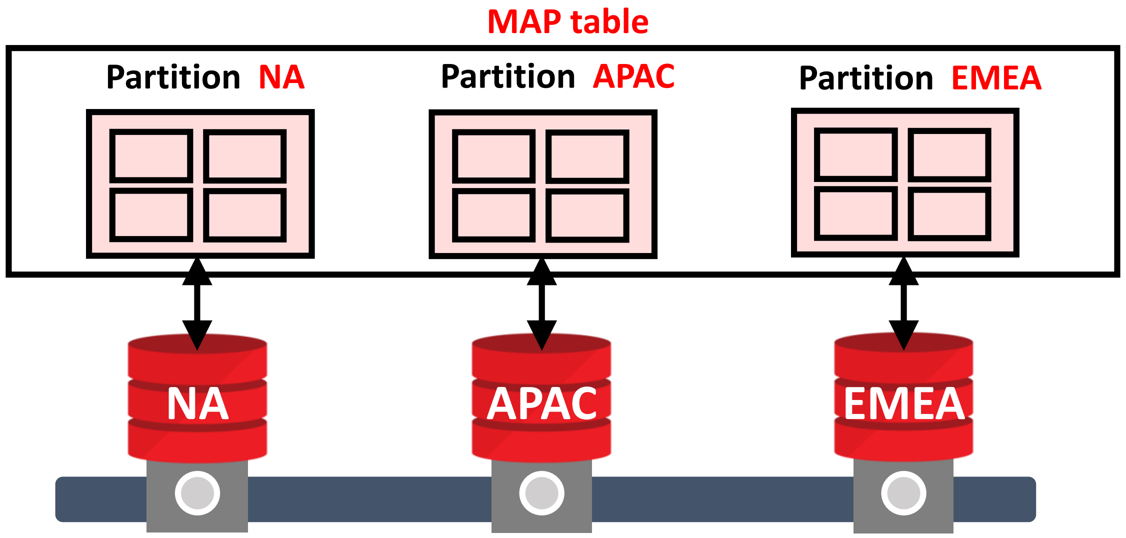
- Hybrid model: Container Map



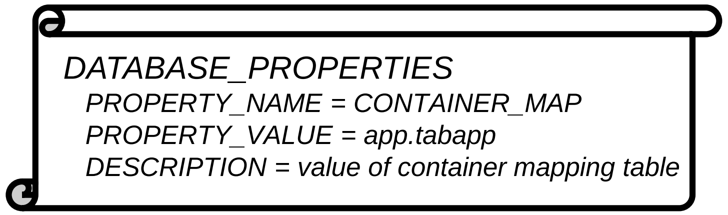
```
SELECT col
FROM orders;
```

Container Map

- 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

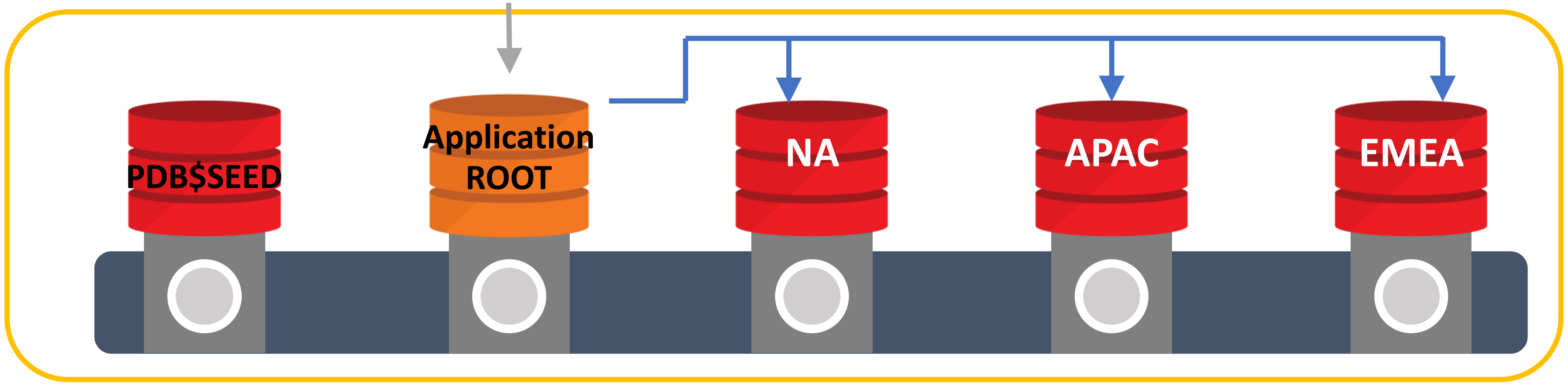
Container Map: Example

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

CREATE TABLE app1.app_map ( columns …, region  VARCHAR2(20))
PARTITION BY LIST (region)
 (PARTITION NA VALUES ('AMERICA', '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



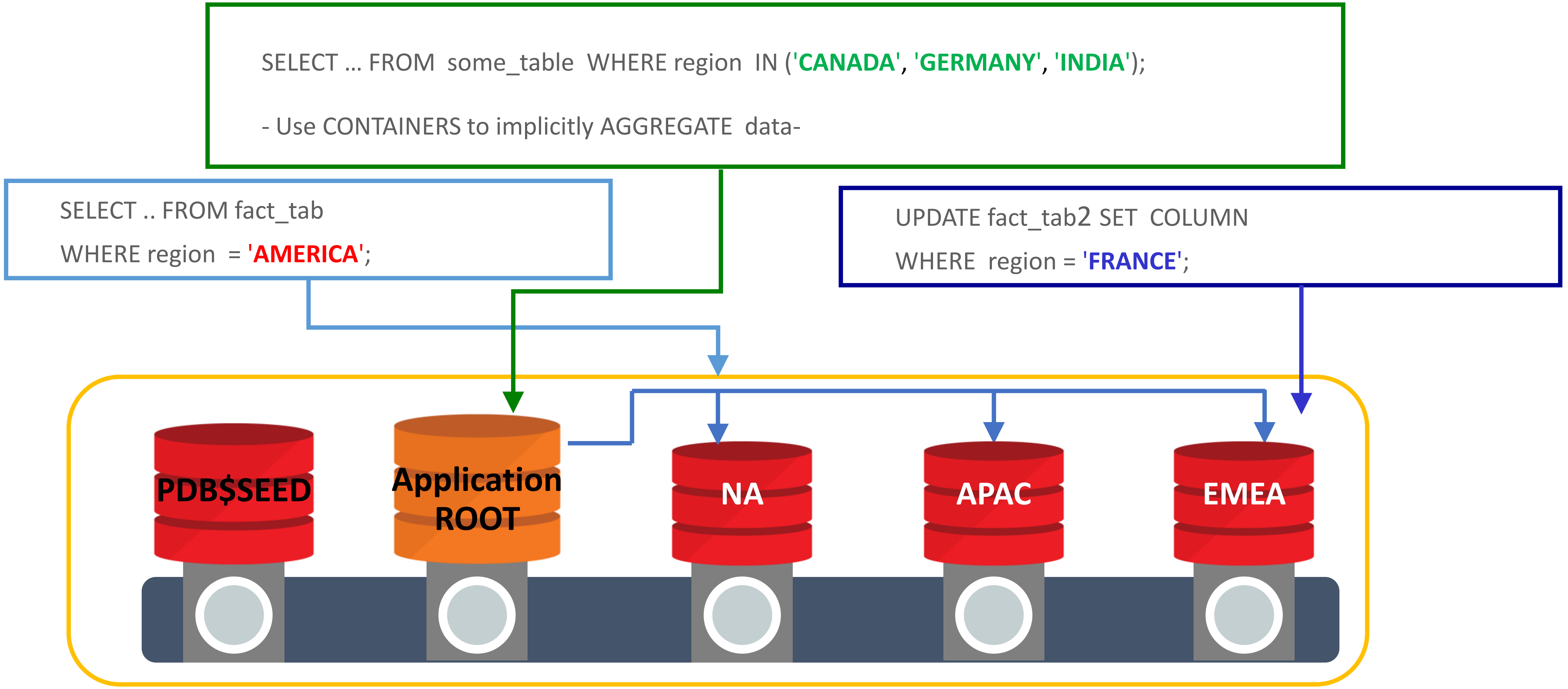
Query Routed Appropriately

SELECT ... FROM some_table WHERE region IN ('**CANADA**', '**GERMANY**', '**INDIA**');

- Use CONTAINERS to implicitly AGGREGATE data-

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

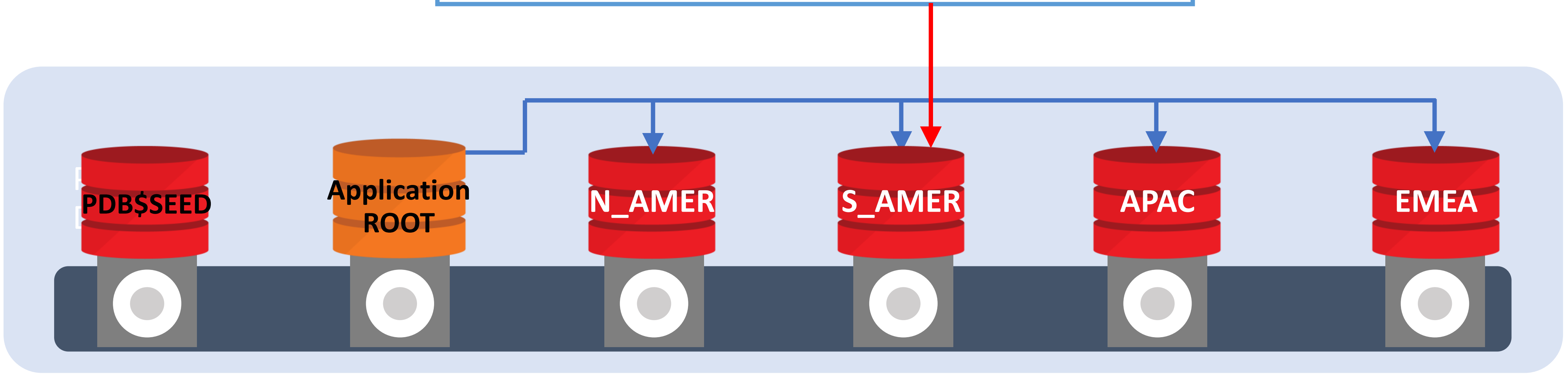
UPDATE fact_tab2 SET COLUMN
WHERE region = '**FRANCE**';



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';
```

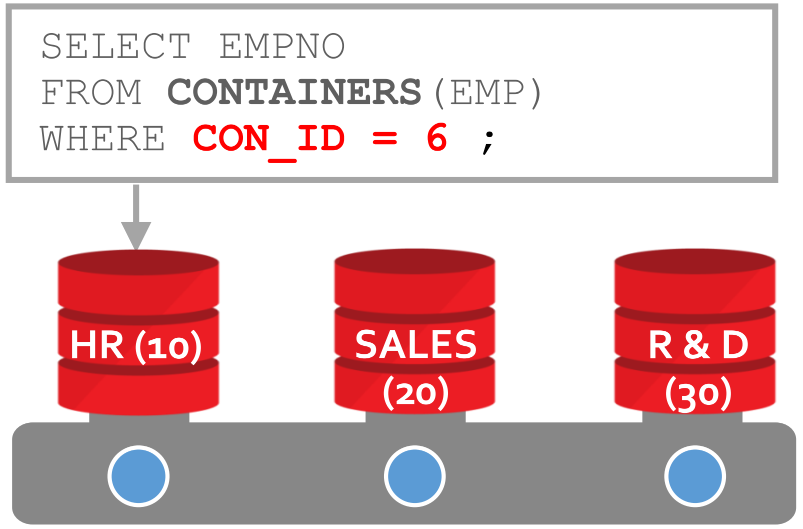


```
CREATE PLUGGABLE DATABASE s_amer_peru …
       CONTAINER_MAP UPDATE (SPLIT PARTITION s_amer
                     INTO (partition s_amer ('ARGENTINA'), partition s_amer_peru));
```

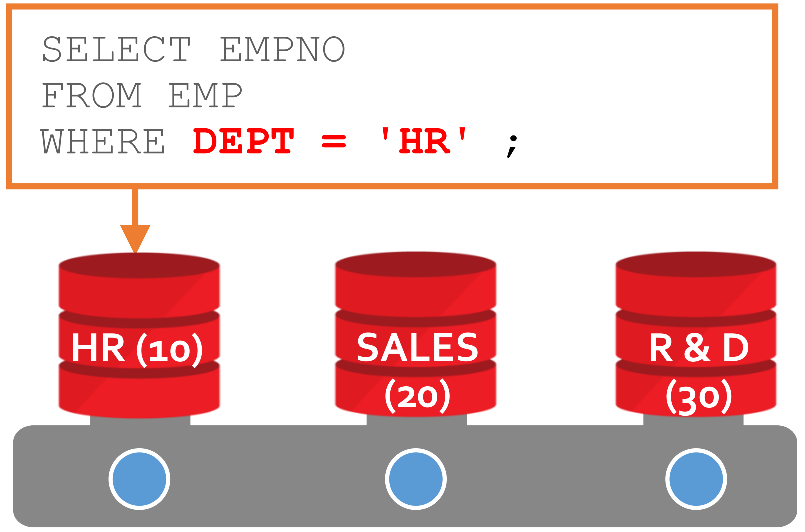
Container Map and Containers Default

`CONTAINERS_DEFAULT` allows you to wrap the `CONTAINERS()` clause around any table.

`CONTAINER_MAP`, when used in conjunction with `CONTAINERS_DEFAULT`, prunes the partitions (PDBs) based on the key passed to the query.



```
SELECT EMPNO
FROM CONTAINERS(EMP)
WHERE CON_ID = 6 ;
```

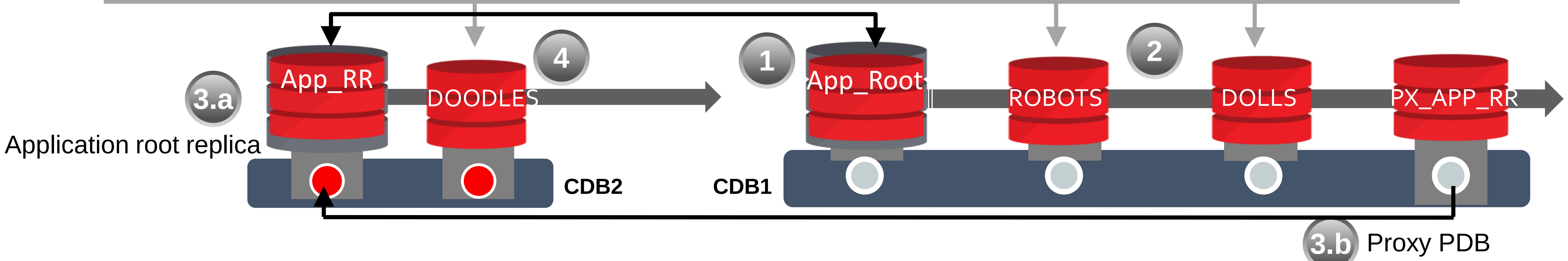


```
SELECT EMPNO
FROM EMP
WHERE DEPT = 'HR' ;
```

DBA_TABLES
CONTAINERS_DEFAULT= YES
CONTAINER_MAP = YES

Query Across CDBs Using Application Root Replica

```
SELECT sum(revenue), year, CDB$NAME, CON$NAME
FROM   CONTAINERS(sales_data)
WHERE  year =  2014  GROUP  BY year, CDB$NAME, CON$NAME;
```



➔Retrieves all rows from the shared table whose data is stored in all application PDBs in the application root and replicas in CDBs.

5

Revenue	Year	CDB$NAME	CON$NAME
15000000	2014	CDB1	ROBOTS
20000000	2014	CDB2	DOODLES
10000000	2014	CDB1	DOLLS

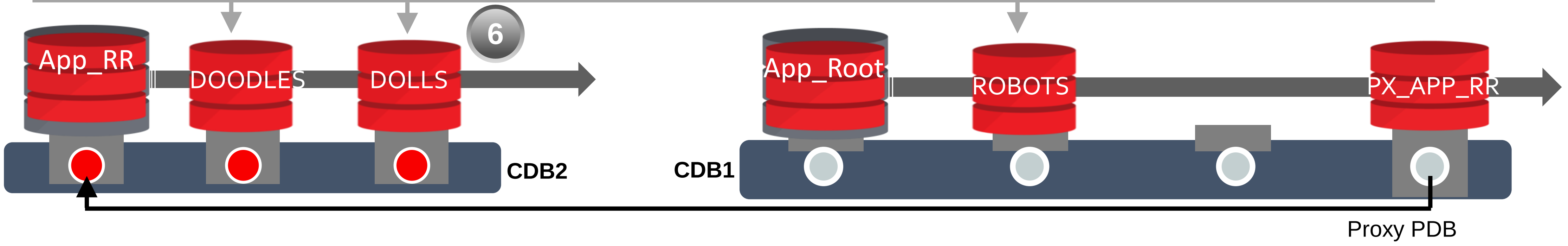
Durable Location Transparency

Load balance by relocating one of the application PDBs:

➔ The query still retrieves all the rows from the shared table in all the PDBs under the application roots in the CDBs.

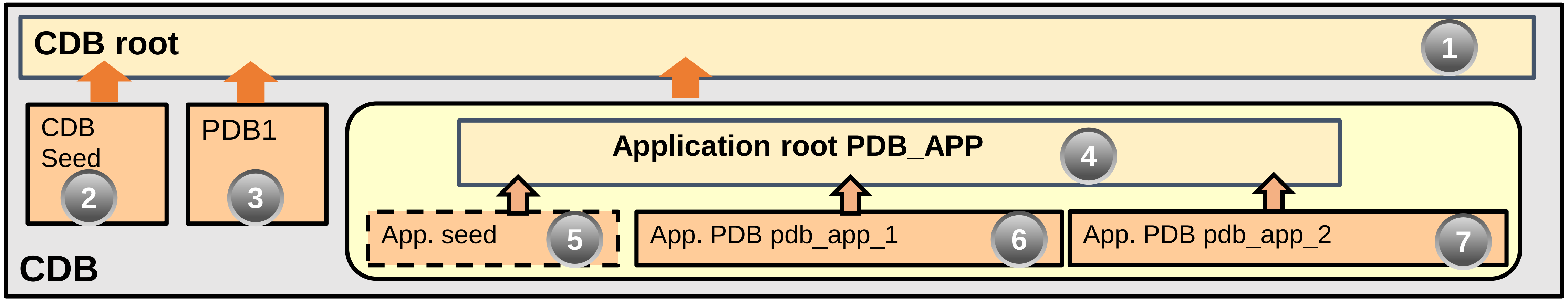
➔ The application code is unchanged.

```
SELECT sum(revenue), year, CDB$NAME, CON$NAME FROM CONTAINERS(sales_data) WHERE year =  2014  GROUP
BY year, CDB$NAME, CON$NAME;
```



Revenue	Year	CDB$NAME	CON$NAME
15000000	2014	CDB1	ROBOTS
20000000	2014	CDB2	DOODLES
10000000	2014	CDB2	DOLLS

Data Dictionary Views



```
SQL> SELECT name, con_id, application_root "APP_ROOT", application_seed "APP_Seed",
            application_pdb "APP_PDB", application_root_con_id "APP_ROOT_CONID"
     FROM   v$containers order by con_id;

NAME              CON_ID APP_ROOT  APP_Seed  APP_PDB   APP_ROOT_CONID
------------- ------ -------- --------  -------- --------------
CDB$ROOT               1  NO        NO        NO
PDB$SEED               2  NO        NO        NO
PDB1                   3  NO        NO        NO
PDB_APP                4  YES       NO        NO
PDB_APP$SEED           5  NO        YES       YES                  4
PDB_APP_1              6  NO        NO        YES                  4
PDB_APP_2              7  NO        NO        YES                  4
```

Terminology in Application Container Context

- Common versus Local:
  - Users
  - Privileges / Roles
  - Objects
  - Profiles
  - Auditing policies and FGA policies
  - Application context and VPD policies
  - Transparent sensitive data protection (TSDP) policies
  - Database Vault realms and common command rules
- **Note:** Any statement that can be issued in a CDB root can also be issued in an application root.

Commonality in Application Containers

- In an application root, statements to create common entities can be issued only as part of an application operation.

Application Operation	Common Entity

BEGIN INSTALL / END INSTALL
 	
BEGIN UPGRADE / END UPGRADE
 	
BEGIN PATCH / END PATCH
	Create, alter, or drop a common user.
	Create, alter, or drop a common role.
	Create, alter, or drop a common profile.
	Commonly grant privileges or roles to or revoke them from a common user or common role.
	Create, alter, and drop common objects.

Impacts

- Per PDB character set:
    - Enables storing multilingual data
    - Facilitates conversion of existing non-CDBs to PDBs
    - Facilitates fast and seamless unplug/plug of PDBs across CDBs that have different compatible character sets
    - Is the same for all PDBs in an application container
    - Is supported with the LogMiner data dictionary
- Common unified and FGA policies in application containers
- Database Vault common realms and command rules at CDB level
- Common objects in application PDBs supported by LogMiner

Summary

- In this lesson, you should have learned how to:
  - Describe application containers in CDBs
  - Explain the purpose of application root and application seed
  - Define application PDBs
  - Create application PDBs
  - Explain application installation on top of application containers
  - Install an application
  - Upgrade and patch applications
  - [illegible]

Practice 3: Overview

- 3-1: Installing an application in an application container
- 3-2: Upgrading an application in an application container
- 3-3: Querying data across application PDBs in CDB