

# Security Concepts in Oracle Multitenant

ORACLE WHITE PAPER | JANUARY 2015





## Disclaimer

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.



Table of Contents	Disclaimer
	1
Introduction	2
New Concepts	3
Auditing in a Multitenant Database	13
Working with common users	14
Common Role	21
Common Profile	22
Common user challenges	24
Conclusion	25



## Introduction

Oracle Multitenant is an innovative database virtualization solution that helps customers to reduce their IT costs by simplifying consolidation, provisioning, and management of their database systems. Efficient use of database background processes and shared memory areas lead to greater consolidation densities and thus economies of scale.

With this new architecture, multiple applications can be deployed on fully functional pluggable databases that run inside the same container database. An existing database can adopt this multitenant architecture with no changes and no changes are needed in the other application tiers as well. Oracle Multitenant delivers the following compelling benefits:

- I. Reduction of Capital Expenditure by consolidating more applications to run in same or reduced hardware infrastructure
- II. Reduction of Operational Expenditure by allowing management of many databases as one. Consolidation also provides the opportunity to standardize procedure and services.
- III. Maximize Agility through rapid provisioning, out of the box support of snapshot cloning and portability through pluggability.
- IV. Easy to adopt as no application changes are needed.

With all the benefits described above, Oracle Multitenant is the obvious choice for consolidation of databases. Multitenancy is also at the core of cloud computing. In essence it allows the tenants of a service to cost-effectively share a common infrastructure reliably and securely. However be it an on-premise private cloud solution or a public cloud hosted somewhere outside the company's firewall there are various security considerations that today's IT organizations need to evaluate before taking the first step towards cloud. A multitenant cloud database should be architected appropriately to partition data and safeguard it against both external and internal threats. If properly designed a multitenant service can be even more secure than a traditional on-premise system because here the control can rest completely with the provider.

This technical whitepaper discusses the new security concepts introduced in Oracle Multitenant.

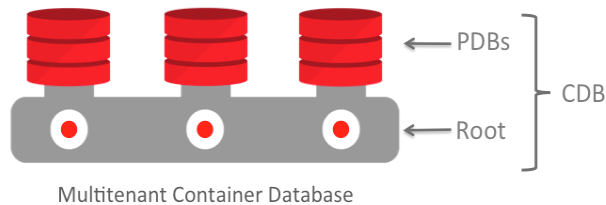
## New Concepts

Before moving into the new security concepts discussion let us take a step back and revisit some of the pluggable database concepts. The multitenant architecture enables customers to have one or many pluggable databases run on a root container. A pluggable database (PDB) is a fully functional self-contained Oracle database that runs your application without any changes. A container database (referred as CDB henceforth) is the collection of all pluggable databases, the root container and a special seed pluggable database (PDB\$SEED).

*Root Pluggable Database:* The root stores Oracle supplied metadata and common user information. We will discuss common users in the next section. The root container is also referred as CDB\$ROOT.

*Seed PDB:* The seed PDB is the system-supplied template used to provision new PDBs. This seed PDB is referred as PDB\$SEED internally.

*User Created PDB:* This is the user created entity that works and behaves exactly like a self-contained Oracle database. (Read [PDB/non-CDB compatibility guarantee](#))



With the introduction of container databases we have two types of database users. *Local Users* those are defined within a pluggable database and *Common users* that exist in all pluggable databases including the root container.

### Local User

Local users are the successors for customer created users in a non-CDB. They can only be created inside a PDB. In most cases it is expected that the PDB admin will be a local user and it is possible for a local user to be granted a common role that is created globally within the container database. A local user can administer a PDB and even have SYSDBA privilege contained within the bounds of the PDB. However, SYSDBA privilege does not enable the user to automatically get to another PDB in the same container database or drop oracle supplied objects\* in CDB\$ROOT that are available in the PDB through metadata links. This privilege allows the user to startup and shutdown the pluggable database whenever necessary.

\* Oracle supplied objects: Data Dictionary Objects that are created to store the database metadata and various PL/SQL packages that are created as part of the database install

**1. Create a local dba user that has SYSDBA privilege in the WIKIAPP pluggable database. Login to the CDB as a SYSDBA privileged user.**

```
SQL> show pdbs
```

CON_ID	CON_NAME	OPEN	MODE	RESTRICTED
2	PDB\$SEED	READ	ONLY	NO
3	WIKIAPP	READ	WRITE	NO

```

      4 PDB1          READ WRITE    NO
SQL> alter session set container = wikiapp;
Session altered.
SQL> create user localdba identified by <password>;
User created.
SQL> grant sysdba, create session to localdba;
Grant succeeded.
SQL> conn localdba/<password>@//localhost/wikiapp as sysdba
Connected.
SQL> show pdbs

```

CON_ID	CON_NAME	OPEN	MODE	RESTRICTED
3	WIKIAPP	READ	WRITE	NO

```
SQL> show con_name
```

```

CON_NAME
-----
WIKIAPP

```

**Note: the other pluggable database PDB1 is not visible to the user localdba.**

### 2. Restart the wikiapp pdb as the user localdba

```
SQL> alter pluggable database wikiapp close;
```

Pluggable database altered.

```
SQL> alter pluggable database wikiapp open;
```

Pluggable database altered.

### 3. Switch to another container

```
SQL> alter session set container = pdb1;
```

```

ERROR:
ORA-01031: insufficient privileges

```

This operation results in an error as this user is local to the

pluggable database.

#### 4. Drop an oracle-supplied pl/sql package

```
drop package sys.dbms_sql;
```

```
drop package sys.dbms_sql
```

```
*
```

```
ERROR at line 1:
```

```
ORA-65040: operation not allowed from within a pluggable database
```

As mentioned earlier the attempt to drop any oracle supplied package also results in an error.

#### Common User

A common user as the name suggests exists in all pluggable databases created in a container database. Common users are created in the root container and the user credentials are shared across all pluggable databases. They may perform operations in both the root and in individual containers.

#### Creating a common user

```
SQL> create user c##app identified by password container=all;
```

```
User created.
```

**Note:** All user created common users are identified by a common user prefix. The oracle supplied default is c## and the setting of the initialization parameter *common\_user\_prefix* controls the value of this prefix. It is not advised to set this value to null. A local user cannot be created with the same *common\_user\_prefix*.

```
SQL> show parameter common
```

NAME	TYPE	VALUE
common_user_prefix	string	C##

Out of the box Oracle supplied accounts such as SYS, SYSTEM and DBSNMP etc. are common users. Common users can have different privileges in different PDBs and thus can be prevented from logging into a particular PDB by simply not granting the ability to create session in that container or not granting the set container privilege.

Oracle does not recommend that you change the privileges of the Oracle-supplied common users. You may grant user-created common users different privileges locally in each PDB. As part of security best practices it is also advised that passwords of Oracle provided accounts like "SYSTEM" are not distributed and instead customers should create common accounts that serve the specific administrative purposes.

By default except for SYS and SYSTEM all common user accounts that are pre-created in the seed pluggable database (PDB\$SEED) are expired and locked. Unless instructed by Oracle support you should not enable these accounts in the seed database. In fact any change to the seed database is not supported. This way, every time a pluggable database (PDB) is created from the seed database the Oracle supplied common user accounts are all available in the new PDB.

**A COMMON USER CAN BE GRANTED PRIVILEGES LOCALLY IN A PDB (OR ROOT) AND THEREFORE DIFFERENTLY IN EACH PDB.**

**A COMMON USER CAN, ALTERNATIVELY, BE GRANTED A SYSTEM PRIVILEGE COMMONLY – THE GRANT IS MADE IN ROOT AND EVERY PDB, PRESENT AND FUTURE.**

**A LOCAL USER WITH DBA PRIVILEGES IN A PDB CANNOT CREATE OBJECTS IN A COMMON USER SCHEMA.**

**BEST PRACTICE: DO NOT CREATE OBJECTS IN COMMON USER'S SCHEMA.**

### List of Oracle Supplied Accounts in PDB\$Seed

```
SQL> show con_name
```

```
CON_NAME
```

```
-----  
PDB$SEED
```

```
SQL> Select username, account_status, common from dba_users  
Order by 2 desc;
```

USERNAME	ACCOUNT_STATUS	COMMON
SYSTEM	OPEN	YES
SYS	OPEN	YES
ORDSYS	EXPIRED & LOCKED	YES
DBSNMP	EXPIRED & LOCKED	YES
...		
...		
...		
FLows_FILES	EXPIRED & LOCKED	YES
DVF	EXPIRED & LOCKED	YES

```
35 rows selected.
```

**Note: If you choose to create the sample schemas at the time of database installation. The users (e.g. HR, OE etc.) are only created as local users within a PDB. These users are not created in PDB\$SEED.**



The newly added column ORACLE\_MAINTAINED in the dba\_users view can be used to distinguish between oracle created common users and customer created ones.

```
SQL> select count(*),oracle_maintained from dba_users
  2  group by oracle_maintained;
```

COUNT(*)	ORACLE_MAINTAINED
35	Y
1	N <- customer created common user

### Set Container privilege

For various administration tasks often a common user needs to move from one container to another while connected to the same session. The set container privilege allows the user to move from one container to another without reestablishing a connection.

**In this example below the common user with set container privilege first switches from CDB\$ROOT into a PDB named US and then moves to the PDB named APAC.**

```
SQL> show con_name
```

```
CON_NAME
-----
CDB$ROOT
```

```
SQL> alter session set container = US;
```

Session altered.

```
SQL> show con_name
```

```
CON_NAME
-----
US
```

```
SQL> alter session set container = APAC;
```

Session altered.

The set container privilege should be granted with caution. In fact there are few out-of-the-box roles that have the *set container* privilege granted by default. These roles should also be granted to common users after proper review.

```
SQL> select role, privilege, common from role_sys_privs where
privilege like '%CONTAINER%';
```

ROLE	PRIVILEGE	COMMON
CDB_DBA	SET CONTAINER	YES
CONNECT	SET CONTAINER	YES
DBA	SET CONTAINER	YES
EM_EXPRESS_ALL	SET CONTAINER	YES

As evident from the list above, the CONNECT role is commonly available in all PDBs and it now has the set container privilege. When the CONNECT role is granted to a local user inside a PDB the user does not automatically inherit the privilege to move to another container. The set container privilege is only applicable to common users. In the example below we see when a local user with the connect role attempts to move to another container it results in an error.

**Step 1: Create a local user in a pluggable database and grant the CONNECT role**

```
SQL> show con_name
```

```
CON_NAME
-----
WIKIAPP
```

```
SQL> create user patrick identified by password ;
```

User created.

```
SQL> grant connect to patrick;
```

Grant succeeded.

**Step 2: Login as the local user and try to set container to another pluggable database**

```
sqlplus patrick/password@//localhost/wikiapp
```

...

```
patrick@WIKIAPP> alter session set container = PDB1;
```

ERROR:

**ORA-01031: insufficient privileges**

The set container privilege has certain restrictions in PL/SQL. Although developers may think that they can use “*execute immediate alter session set container*” to switch between two containers inside a pl/sql procedure, function or package it is blocked from execution from within a PDB.

**In this example below we will use set container to switch to a PDB from CDB\$ROOT and then try to return back to the root container.**

```
SQL> show con_name
```

```
CON_NAME
```

```
-----  
CDB$ROOT
```

```
SQL> !cat set_con.sql
```

```
begin  
execute immediate 'alter session set container = wikiapp';  
--  
execute immediate 'alter session set container = cdb$root';  
--  
end;  
/
```

```
SQL> @set_con2.sql
```

```
begin  
*  
ERROR at line 1:  
ORA-01031: insufficient privileges  
ORA-06512: at line 5
```

```
SQL> show con_name
```

```
CON_NAME
```

```
-----  
WIKIAPP
```

**It is evident from the example above the first set container statement succeeded but the second one was blocked as insufficient privileges.**

The use of alter session set container is not the recommended method in PL/SQL. When executing from CDB\$ROOT It is advised to use dbms\_sql with the container parameter so that the context of execution is never changed from the issuing container (i.e. cdb\$root) even if there is an untrapped exception.

**Code snippet with dbms\_sql : How to execute a query in another container**

**-- Do not use set container in pl/sql**

```
Procedure Execute Stmt_In_PDB1(Stmt in varchar2) is  
Cur integer := DBMS_Sql.Open_Cursor(Security_Level=>2);
```

```

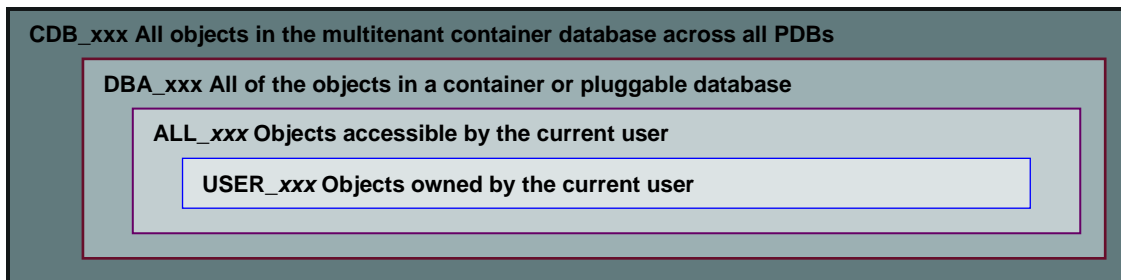
Dummy integer;
begin Dbms_Sql.Parse(
  c=>Cur,
  Statement=>Stmt,
  Container=>'PDB1',
  Language_Flag=>DBMS_Sql.Native);
Dummy := DBMS_Sql.Execute(Cur);
DBMS_Sql.Close_Cursor(Cur);
end Execute Stmt_In_PDB1;

```

### Data Dictionary and Performance (V\$) views

A new family of data dictionary views was introduced in 12c with names such as CDB\_xxx. Each DBA\_xxx view has a CDB\_xxx view counterpart with an extra column, Con\_ID that shows from which container the listed facts are aggregated. You may query the CDB\_xxx views from the root or from any PDB. The CDB\_xxx views are useful when queried from the root because the results from a particular CDB\_xxx view are the union of the results from the DBA\_xxx view counterpart over the root and all the currently open PDBs (but see *container\_data* clause below on how you may restrict the visibility of the data in these views.)

When a CDB\_xxx view is queried from a PDB, it shows the same information as the DBA\_xxx view counterpart. For example if you connect to the root and query CDB\_USERS, you get the list of users, common and local, of each pluggable database. If you query DBA\_USERS, you get the list of common users since in CDB\$ROOT only common users exist. If you connect to a PDB, and query CDB\_USERS or DBA\_USERS, you get the same list of users, common and local for that PDB.



Example: Leverage the CDB\_USERS view to find the number of local users in each PDB

List the count of local users in each pluggable database in a CDB from root.

```

select p.pdb_name,count(u.username) from cdb_users u, cdb_pdbs p
where u.con_id = p.pdb_id
and u.common='NO'
group by p.pdb_name;

SQL> /

```

PDB_NAME	COUNT
APAC	2
EMEA	2
US	3

Like the Data dictionary views the performance views (v\$) will show only information pertaining to the PDB when queried inside the PDB but will show information for the whole CDB when queried from root.

#### From the root container

```
SQL> select con_id, name, value from v$sysstat where name =
'logical read bytes from cache';
```

CON_ID	NAME	VALUE
0	logical read bytes from cache	349531979776

```
SQL> alter session set container = pdb1;
```

#### Now the same stat from inside a PDB

```
SQL> select con_id, name, value from v$sysstat where name =
'logical read bytes from cache';
```

CON_ID	NAME	VALUE
3	logical read bytes from cache	3171426304

**The value of the sysstat is completely different in root and inside the PDB.**

#### Container\_data

The *container\_data* setting for the common user determines what rows are aggregated and made visible from any container\_data object (CDB% or v\$ view). In the example below the common user c##scott is created and granted dba across all containers. But we can use the container\_data clause to restrict access to data from only a specific set of pluggable databases in the Container database.

#### 1. Create user c##scott and grant dba commonly across all containers.

```
SQL> show pdbs
```

CON_ID	CON_NAME	OPEN MODE	RESTRICTED
2	PDB\$SEED	READ ONLY	NO
3	WIKIAPP	READ WRITE	NO

```

4      PDB1      READ WRITE NO
5      NCDB1     READ WRITE NO
6      NCDB2     READ WRITE NO

```

```
SQL> create user c##scott identified by password container =all;
```

User created.

```
SQL> grant dba,create session to c##scott container=all;
```

Grant succeeded.

## 2. Log in as c##scott and query the number of data files in each of the pluggable databases

```
SQL> conn c##scott/password@cdb3
```

Connected.

```
C##SCOTT@CDB> select count(*),con_id from cdb_data_files group by
con_id;
```

COUNT(*)	CON_ID
4	1

We see that only data from the root container (CDB\$ROOT) is visible. This is because the `container_data` attribute is defaulted to CDB\$ROOT when the attribute has not been set for the view or the default `container_data` attribute for the user is not set.

## 3. As user SYS alter the common user (c##scott) and specify the `container_data` clause to allow access to all pluggable database in the container database

```
SQL> alter user c##scott set container_data = all for cdb_data_files
container=current;
```

User altered.

## 4. Execute the same query as c##scott again

```
C##SCOTT@CDB> select count(*),con_id from cdb_data_files group by
con_id;
```

COUNT(*)	CON_ID
4	1
3	3

```
4      4
4      5
4      6
```

Here data is displayed for all pluggable databases in that CDB

**5. As user SYS alter c##scott and specify the container\_data clause to allow access to only specific pluggable databases in the container database**

```
SYS@CDB> alter user c##scott set container_data =
(cdb$root,wikiapp,pdb1) for cdb_data_files container=current;
```

User altered.

**6. Re-execute the same data file count query as c##scott**

```
C##SCOTT@CDB> select count(*),con_id from cdb_data_files group by
con_id;
```

COUNT (*)	CON_ID
3	3
4	1
4	4

**Data is only displayed for the PDBs on which access was granted.**

## Auditing in a Multitenant Database

Before Oracle Database 12c there were separate audit trails for individual components: SYS.AUD\$ for the database audit trail, SYS.FGA\_LOG\$ for fine-grained auditing, DVSYS.AUDIT\_TRAIL\$ for Oracle Database Vault, Oracle Label Security, and so on. In Oracle Database 12c, these audit trails are all unified into one, viewable from the UNIFIED\_AUDIT\_TRAIL data dictionary view for single-instance installations or Oracle Database Real Application Clusters environments. In a Multitenant environment, each PDB, including the root, has its own unified audit trail.

With Oracle Multitenant enabled the audit policy can be set either locally or commonly across multiple containers. The local audit policy, as the name suggests exists either in the root or inside a PDB. A local or a common user with the audit\_admin role can enable local policies that will only be enabled inside that particular PDB.

In the following example we see a local audit policy (container = current) enabled for a common user c##tpch.

```
create audit policy table_privs privileges create any table,
drop any table container = current;

audit policy table_privs by c##tpch;
```

At the same time common audit policies are available to all PDBs in the multitenant environment. Only common users who have been granted the AUDIT\_ADMIN role can create and maintain common audit policies. However common audit policies can only be enabled on common users from the root container.

```
SQL> show con_name

CON_NAME
-----
CDB$ROOT

SQL> create audit policy dict_updates actions update on
sys.user$, delete on sys.user$, update on sys.link$, delete on
sys.link$ container = all;

Audit policy created.

SQL> audit policy dict_updates by c##tpch;

Audit succeeded.
```

For more information on unified auditing in an Oracle Multitenant environment please refer to the documentation [here](#).

## Working with common users

Common Users are a new concept introduced with Oracle Multitenant. In this section we discuss various scenarios that you will encounter with common users while working with Oracle Multitenant. In the first example we have a situation where the common user c##app has “create any table” privilege commonly. But you want to control its ability to log into a specific PDB

### CREATE A NEW COMMON USER AND CONTROL ITS ABILITY TO LOG INTO SPECIFIC PLUGGABLE DATABASES

**Step 1: Create a common user c##app that will have the privilege to create any table in all PDBs.**

```
SQL> create user c##app identified by <mypassword>
container=all;

User created.

SQL> grant create any table to c##app container=all;

Grant succeeded.
```

Note the container clause at the end of both statements. The container clause indicates the context where the user is to be created or the privilege be granted. ‘All’ signifies all existing and future pluggable databases and current indicates



the container where the command is executed.

### Step 2: List the PDBs in my container database

```
SQL> show pdbs
```

CON_ID	CON_NAME	OPEN MODE	RESTRICTED
2	PDB\$SEED	READ ONLY	NO
3	SPARK	READ WRITE	NO
4	ORDERS	READ WRITE	NO
5	PDB_000	READ WRITE	NO
6	JPM	READ WRITE	NO

**Step 3: This common user will have ability to log into any all PDBs except SPARK. We will use the catcon.pl script (installed by default in \$ORACLE\_HOME/rdbms/admin) to grant create session on all containers except SPARK. Usage and syntax for catcon.pl is documented [here](#)**

```
perl $ORACLE_HOME/rdbms/admin/catcon.pl -C 'SPARK' -e -b  
grant_session -- --x'grant create session to c##app  
container=current'
```

**Step 4: Let us check whether common user c##app can connect to ORDERS and SPARK**

```
SQL> conn c##app/<mypassword> @//localhost/orders.us.oracle.com  
Connected.
```

**Connecting to the spark PDB**

```
SQL> conn c##app/<mypassword> @//localhost/spark.us.oracle.com  
ERROR:  
ORA-01045: user C##APP lacks CREATE SESSION privilege; logon  
denied
```

As expected c##app was able to log into orders PDB but was denied logging into spark. Hence although create any table was granted to the common user the user cannot create any table in the spark PDB unless it has been granted create session or set container privilege.

**Step 5: From CDB\$ROOT let us examine CDB\_SYS\_PRIVS and check the list of PDBs where the CREATE SESSION privilege was assigned**

```
SQL> select a.grantee, a.privilege, a.common, b.name from  
cdb_sys_privs a,  
(select con_id,name from v$pdb) b
```

```
where a.con_id = b.con_id
and a.grantee = 'C##APP'
and a.privilege = 'CREATE SESSION';
```

GRANTEE	PRIVILEGE	COM	NAME
C##APP	CREATE SESSION	NO	ORDERS
C##APP	CREATE SESSION	NO	PDB_000
C##APP	CREATE SESSION	NO	JPM

**The PDB SPARK is not listed in the output.**

Note: Since the create session grant was executed locally via catcon.pl in each PDB, we will have to manually execute the same grant for any new PDB that is created or plugged into this CDB. On the other hand privileges granted commonly get automatically available in all future PDBs.

#### CREATE A COMMON USER THAT IS EXTERNALLY AUTHENTICATED BY OS

Although a user local to a PDB cannot be externally OS authenticated (*PDB local users may use other types of external authentication like SSL, Kerberos etc.*), a common user can be configured for external authentication. External authentication of users is not an Oracle recommended security best practice. For existing applications that use external authentication oracle database 12c will continue to work.

*Note: The method described below requires a database restart. The behavior is different in windows platform.*

**Step 1: Set the initialization parameter `os_authent_prefix` to match the common user prefix string that is set to `c##` by default. Thus for operating system user `scott` we need to create a database user `c##scott`. The alternative approach would be to modify the initialization parameter `common_user_prefix` to `ops$`.**

```
SQL> alter system set os_authent_prefix='c##' scope=spfile;
```

**Step 2: Restart the database. When the database is started and all the pluggable databases are opened, create the common user.**

```
SQL> create user c##scott identified externally container=all;
```

User created.

```
SQL> grant dba, set container to c##scott container=all;
```

**Step 3: List the PDBs in the container database**

```
SQL> show pdbs
```

CON_ID	CON_NAME	OPEN MODE	RESTRICTED
2	PDB\$SEED	READ ONLY	NO
3	SPARK	READ WRITE	NO
4	ORDERS	READ WRITE	NO
5	PDB_000	READ WRITE	NO
6	JPM	READ WRITE	NO

**Step 4: We have created the common user. Now let us test the login.**

```
[scott@slc07fhs ~]$ sqlplus /
SQL*Plus: Release 12.1.0.2.0 Production on Mon Aug 25 11:13:14
2014
Copyright (c) 1982, 2013, Oracle. All rights reserved.
Connected to:
Oracle Database 12c Enterprise Edition Release 12.1.0.2.0 -
64bit Production
With the Partitioning, OLAP, Advanced Analytics and Real
Application Testing options
SQL> show user
USER is "C##SCOTT"
SQL> select sys_context('userenv','authentication_method') as
Auth_Method, sys_context('userenv','authenticated_identity') as
Auth_Identity from dual;

AUTH_METHOD          AUTH_IDENTITY
-----
OS                    scott
SQL> alter session set container=orders;
Session altered.
SQL> show con_name
CON_NAME
-----
ORDERS
SQL> alter session set container=JPM;
Session altered.
SQL> show con_name
CON_NAME
```

-----  
JPM

### ADOPTING A NON-CDB AS A PDB

One of the most common questions that we get asked is, what happens to the Oracle supplied administrative accounts passwords when a non-CDB is plugged into a CDB? The Oracle supplied accounts are merged with the existing common user accounts and the password of the existing common user accounts takes precedence. It is not an Oracle recommended security best practice to grant an Oracle supplied administrative account a custom created role. However if the administrative account (e.g. SYSTEM) did have a custom role assigned to it in the non-CDB it is still carried forward but is only available locally at the PDB level. The following block explains the scenario

**Step 1: In the non-CDB the Oracle supplied account SYSTEM had a custom role app\_role assigned to it.**

```
SQL> select grantee,granted_role from dba_role_privs where grantee='SYSTEM';
```

GRANTEE	GRANTED_ROLE
SYSTEM	APP_ROLE
SYSTEM	AQ_ADMINISTRATOR_ROLE
SYSTEM	DBA

**Step 2: After plugging the non-CDB into a CDB and executing noncdb\_to\_pdb.sql , the system account will be merged with the existing system account. In the root container we can only see the default roles.**

```
SQL> show con_name
```

```
CON_NAME
```

```
-----  
CDB$ROOT
```

```
SQL> select grantee,granted_role from dba_role_privs where grantee='SYSTEM';
```

GRANTEE	GRANTED_ROLE
SYSTEM	AQ_ADMINISTRATOR_ROLE
SYSTEM	DBA

```
SQL> alter session set container=exnoncdb;
```

```
Session altered.
```

**Step 3: After logging into the ex-non-CDB pluggable database we can see the custom role that the PDB inherited.**

```
SQL> select grantee, granted_role from dba_role_privs where grantee='SYSTEM';
```

GRANTEE	GRANTED_ROLE
SYSTEM	APP_ROLE
SYSTEM	AQ_ADMINISTRATOR_ROLE
SYSTEM	DBA

```
SQL> show pdbs
```

CON_ID	CON_NAME	OPEN MODE	RESTRICTED
7	EXNONCDB	READ WRITE	NO

```
SQL> select * from dba_role_privs where grantee='SYSTEM';
```

GRANTEE	GRANTED_ROLE
SYSTEM	APP_ROLE
SYSTEM	AQ_ADMINISTRATOR_ROLE
SYSTEM	DBA

### PLUGGING A PDB INTO A DIFFERENT CDB

A PDB when plugged in a CDB may bring with itself a common user that does not exist in the current CDB. In these cases the common user account in the PDB is locked but the objects that may have been referenced in queries, packages, procedures etc. continue to work. The common user can be made functional again simply by creating a common user of the same name in the new CDB. If the common user already exists in the new CDB then the password of the existing account takes precedence and the users are merged. However any additional roles that the common user may have brought along are available locally at the PDB level. As a best practice be sure to study the impact on common users before moving a PDB from one CDB to another. This is not captured in the PDB compatibility checks *DBMS\_PDB.check\_plug\_compatibility*.

In the following example we show how a common user moved with a PDB can be resurrected.

#### Step 1: We create a common user c##DBA in the source container.

```
SQL> create user c##dba identified by ,<mypassword> container=all;
```

User created.

```
SQL> grant dba,connect session to c##dba;
```

Grant succeeded.

#### Step 2: After unplugging and plugging the PDB to another CDB we check the status of the user in the PDB.

```
SQL> select username, account_status from dba_users where
username='C##DBA';
```

```
USERNAME      ACCOUNT_STATUS
```

```
-----
C##DBA        LOCKED
```

**Step 3: We can resurrect this user by creating a common user of the same name in root.**

```
SQL> create user c##dba identified by <mypassword1>
container=all;
```

User created.

*Note: You will receive ORA-01920 if the target PDB is open. The PDB needs to be closed before the user can be created. Also we are using a different password for this user on the target PDB.*

```
SQL> alter session set container=pdb1;
```

Session altered.

```
SQL> select username, account_status from dba_users where
username='C##DBA';
```

```
USERNAME      ACCOUNT_STATUS
```

```
-----
C##DBA        OPEN
```

## Common Role

A common role exists in the root and in every existing and future pluggable database. Common roles are very useful for cross-pluggable database operations. As you would expect a common role can also be granted to a local user but the scope of the grant is limited only in that PDB.

From a security perspective common roles should be used with caution. The widely used DBA role is now a common role out-of-the-box. In a CDB it is very easy to create a common user and grant the common DBA role to create a container-wide database administrator. |

### THINGS TO REMEMBER

**A COMMON USER WITH A COMMON ROLE CAN BE GRANTED A ROLE THAT IS LOCAL TO A PDB.**

**IN THE CREATE ROLE STATEMENT, THE CONTAINER=ALL CLAUSE SPECIFIES THAT THE ROLE IS COMMON.**

**ONLY A COMMON USER WHOSE CURRENT CONTAINER IS ROOT CAN MODIFY A COMMON ROLE.**

```
Granting the DBA common role to a common user c##dba_scott
SQL> create user c##dba_scott identified by <mypassword>
container=all;
User created.
SQL> grant dba, set container to c##dba_scott container=all;
Grant succeeded.
```

The common role 'DBA' in the example above already includes the privilege to create session in all containers. So it was not needed to grant it explicitly. It is important to understand that any role or privilege granted commonly cannot be revoked locally. So if you want to restrict the common user from logging into a specific PDB you will have to create a new custom common role that is similar to the DBA role but does not include the create session privilege. Once that is done the role can be granted commonly with exceptions using *catcon.pl* as explained in one of the examples above.

It is worth mentioning that in a multitenant environment all grants to the PUBLIC role inside a PDB are granted with an implicit container=CURRENT clause. This allows the grants to be restricted only in the context of that PDB. Just like other common roles any customer created grants to PUBLIC at CDB\$ROOT cannot be altered from within the PDB.

## Common Profile

A profile is a set of limits on database resources and password access to the database. Like common users a common profile can also be created to apply the same password restrictions and resource limits to all database users across all PDBs. Remember a common profile (or a common role) used locally is not carried with the PDB when it is unplugged and plugged into another container.

In the following example we show how a common profile can be created to limit a local user from creating more than 5 sessions.

### Step 1: Create a profile with the limit sessions\_per\_user set to 5 and assign it to a common user

```
SQL> create profile c##common_profile1 limit SESSIONS_PER_USER
5 CONTAINER=ALL;
```

Profile created.

```
SQL> alter user c##dc_sysdba profile C##COMMON_PROFILE1;
```

User altered.

### Step 2: Ensure initialization parameter resource\_limit is set to TRUE

```
SQL> show parameter resource_limit
```

NAME	TYPE	VALUE
resource_limit	boolean	TRUE

### Step 3: Connect to the desired container

```
SQL> alter session set container=orders;
```

Session altered.

### Step 4: Assign profile to the local application user

```
SQL> alter user scotty profile c##common_profile1;
```

User altered.

```
SQL>
```

The local user scotty cannot create more than 5 sessions in the



orders PDB as it was assigned the c##common\_profile1 profile. A common user can be assigned a local\_profile although it was assigned a common profile in CDB\$ROOT.

**Step 5: Create and assign a local profile to a common user and make the user honor local PDB limits**

```
SQL> create profile local_profile limit SESSIONS_PER_USER 3;
Profile created.
```

```
SQL> alter user c##dc_sysdba profile local_profile;

User altered.
```

Now if we query cdb\_users from cdb\$root then we can see c##dc\_sysdba has a different profile in this PDB

```
SQL> select con_id, profile from cdb_users where username =
'C##DC_SYSDBA';
```

CON_ID	PROFILE
4	C##COMMON_PROFILE1
1	C##COMMON_PROFILE1
7	C##COMMON_PROFILE1
6	C##COMMON_PROFILE1
3	LOCAL_PROFILE
5	C##COMMON_PROFILE1

Note: A local DBA in a PDB cannot enforce password related restrictions on a common user using a local profile.



## Common user challenges

Common users in Oracle Multitenant provide a lot of flexibility to manage many pluggable databases as a single unit. However common users can inadvertently gain access to any existing or newly created pluggable databases when granted a role or privilege with implicit create session commonly. When a pluggable database is plugged in a container, common users automatically gain access to that new PDB if they were granted create session privilege commonly. This can be a violation of a security policy and should be prevented with proper security review.

As a best practice, common users should be minimally privileged with standardized access across the CDB. Avoid granting create session commonly as it cannot be revoked locally. Audit common user role and privileges to prevent inadvertent security violations.

### THINGS TO REMEMBER

**IF A ROLE OR PRIVILEGE IS GRANTED  
COMMONLY IT CANNOT BE REVOKED  
LOCALLY**







## Conclusion

Oracle Multitenant has been built from the ground up to enable the next generation of consolidation. With true in-database virtualization, Oracle Multitenant helps customers to reduce their IT costs by simplifying consolidation, provisioning, and management of their database systems. However as we have seen in this paper you will have to craft a security strategy for the different Multitenant use cases and prevent inadvertent or malicious security breaches of your database systems. In this paper we summarized the new security concepts introduced in Oracle Multitenant specifically around the use of common users, roles and profiles.



CONNECT WITH US

-  [blogs.oracle.com/oracle](http://blogs.oracle.com/oracle)
-  [facebook.com/oracle](http://facebook.com/oracle)
-  [twitter.com/oracle](http://twitter.com/oracle)
-  [oracle.com](http://oracle.com)

**Oracle Corporation, World Headquarters**

500 Oracle Parkway  
Redwood Shores, CA 94065, USA

**Worldwide Inquiries**

Phone: +1.650.506.7000  
Fax: +1.650.506.7200

**Hardware and Software, Engineered to Work Together**

Copyright © 2014, Oracle and/or its affiliates. All rights reserved. This document is provided for information purposes only, and the contents hereof are subject to change without notice. This document is not warranted to be error-free, nor subject to any other warranties or conditions, whether expressed orally or implied in law, including implied warranties and conditions of merchantability or fitness for a particular purpose. We specifically disclaim any liability with respect to this document, and no contractual obligations are formed either directly or indirectly by this document. This document may not be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without our prior written permission.

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Xeon are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Opteron, the AMD logo, and the AMD Opteron logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group. 0115

Security Concepts in Oracle Multitenant  
January 2015  
Author:  
Contributing Authors: