ORACLE DATABASE 12c

Oracle Multitenant: Isolation

In Oracle Database 12*c* Release 2 (12.2)

ORACLE WHITE PAPER | MARCH 2017

ORACLE

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.

Oracle Database 12*c* Release 2. Available now.

Oracle Database 12*c* Release 2 (12.2), the latest generation of the world's most popular database, is now available in the Oracle Cloud and for download from Oracle Technology Network (OTN).

Table of Contents

Isolation

Economies of scale through consolidation are of limited use if that consolidation comes at the expense of isolation. In Oracle Database 12*c* Release 2 (12.2), we introduce some very sophisticated capabilities in this area that can ensure great isolation between PDBs and avoid what is colloquially referred to as “the noisy neighbor problem”. Importantly, this is configurable so that the level of isolation can be tailored appropriately for the use case.

In 12.2 we build on what was already a powerful suite of isolation capabilities to deliver a comprehensive model, which can simply be configured to deliver precisely the appropriate level of isolation for a particular use case.

These capabilities will be explored in more detail shortly, but it is important to understand why a “one size fits all” approach to isolation is not appropriate for the Database Cloud, and why the true requirement is configurable isolation. When considering this topic, it’s helpful to consider a familiar real-world analogy: residential security. At first thought, one might think that the more security the better, but in reality security is a trade-off with convenience. “Maximum Security” is a phrase associated more with a prison than with a home. A home might be more secure with bars on the windows, surrounded by a high wall topped with barbed wire, armed sentries and triple locks on a steel door, but it probably wouldn’t be very nice to live there. On the other hand, leaving all the doors and windows unlocked, while making it easy for the kids to come and go, is likely to result in loss of property. One tries to find the appropriate balance, and that balance will be different in different circumstances. In a dense city environment one is likely to take more precautions than in a suburb where the neighbors are better known. In small towns people sometimes don’t bother to lock at all. Everybody knows what everybody is doing. Security in business hotels is interesting to consider. There’s typically 24-hour security, with cameras in all common areas, security guards and sophisticated keys providing access to the guest rooms. Isn’t it interesting to think how alarming is the prospect that a guest in another room might have access to yours, yet we typically learn to have very little concern that the hotel staff have access to the room literally at countless times during the day without our knowledge (except that perhaps the beds are made and the bathroom is cleaned) and in general even when we’re in the room at night. Somehow in this situation it becomes perfectly acceptable to delegate security to the hotel management.

Similar considerations apply to the Database Cloud in different use cases.

In Database as a Service (DBaaS) on a Public Cloud, it's reasonable to assume that "adjacent" tenants may be competitors. This is a particularly challenging use case because each tenant wants both powerful administrative capabilities within his own PDB, but also that this PDB is fully isolated from all PDBAs in adjacent PDBs. A good residential analogy here is condominium ownership. One wants full sovereignty over one's own space. Everything on the inside of the front door is one's responsibility.

DBaaS on a Private Cloud is a very productive configuration for development teams. Each developer needs to be isolated from the others to the extent that one developer's test does not interfere with another's, but it's typically a collaborative environment in which there is an expectation that everybody will respect everybody else's environments. A good analogy here is sharing a large house with friends. Everybody has a key to the same front door and the individual bedroom doors are usually left unlocked. There are some common areas and common equipment but there's a reasonable expectation of privacy in one's own bedroom.

Software as a Service (SaaS) may be compared to staying in a hotel. For the price of your room you delegate all maintenance and security to the hotel management and within reason expect them to respect the sovereignty of the contents of your room even though they have access more or less at any time. (Perhaps in this case we'd use the in-room safe to secure anything sensitive from the housekeepers.) Everyone understands that there are other guests in the hotel and there is a well-founded expectation that no guest from another room will have access to yours.

In general, when considering the topic of PDB isolation – one that is especially important in a highly consolidated environment such as a Database Cloud – we need to consider all of the potential risks of sharing. These fall into several categories:

- Contention for shared computing resources
- System access
- File system access
- Network access
- Common User or Common Object Access
- Administrative Features

System and file system access can be managed by certain PDB-level parameters while the rest can be secured by lockdown profiles.

Managing System Access

The Oracle OS user is usually a highly privileged user, and using it for operating system interactions is not recommended. Using the same OS user for operating system interactions from different PDBs might compromise data belonging to a given PDB. In a multitenant environment, instead of using the oracle OS user, you can designate a specific user account in a PDB for OS interactions by using `PDB_OS_CREDENTIAL` initialization parameter. When the database accesses an external procedure with the `extproc` agent, `PDB_OS_CREDENTIAL` determines the identity of the OS user employed when interacting with the operating system from a PDB. For better security, you should set a unique operating system user for each PDB in a multitenant environment. Using an OS user identified by `PDB_OS_CREDENTIAL` can ensure that OS interactions are performed as a less powerful user and provides the ability to protect data belonging to one PDB from being accessed by users connected to another PDB.

If you do not set a specific user to be the operating system user for the PDB, then by default the PDB uses the `oracle` operating system user. For the root, you can use the `oracle` OS user when you interact with the operating system.

You can use the `DBMS_CREDENTIAL.CREATE_CREDENTIAL` procedure to set an operating system user for a PDB as follows:

1. Connect to `CDB$ROOT` as a user who has the `EXECUTE` privilege for the `DBMS_CREDENTIAL` PL/SQL package and `ALTER SYSTEM` privilege.

   For example:
   ```
   sqlplus c##sec_admin
   Enter password: password
   ```

2. Run the `DBMS_CREDENTIAL.CREATE_CREDENTIAL` procedure to create an Oracle credential for the operating system user.

   For example, to set the credential for a user named `os_admin`:
   ```
   BEGIN
     DBMS_CREDENTIAL.CREATE_CREDENTIAL (
          credential_name => 'PDB1_OS_USER',
          username => 'os_admin',
          password => 'password');
   END;
   /
   ```

3. Connect to the PDB for which the operating system user will be used.
   For example:

   ```
   CONNECT c##sec_admin@hrpdb
   Enter password: password
   ```

   To find the available PDBs, query the DBA_PDBS data dictionary view. To check the current PDB, run the `show con_name` command.

4. Set the `PDB_OS_CREDENTIAL` initialization parameter for the user whose credential was set in Step 2.
   For example:
   ```
   ALTER SYSTEM SET PDB_OS_CREDENTIAL = PDB1_OS_USER SCOPE = SPFILE;
   ```
   The `PDB_OS_CREDENTIAL` parameter is a static parameter, so you must set it using the `SCOPE = SPFILE` clause.

5. Restart the PDB.
   ```
   ALTER PLUGGABLE DATABASE PDB1 CLOSE IMMEDIATE;
   ALTER PLUGGABLE DATABASE PDB1 OPEN;
   ```

Managing File Access

We have covered how tenants can interact with the OS while making sure other tenants' data is not at risk. Another important principle in tenant isolation is securing file access starting from the PDB creation, for example in DBaaS in a public cloud environment. One of the main goals here is to provide unique paths for data files and directory objects of each tenant at the PDB creation so that there are not any shared directories across PDBs and they cannot access to each other's files.

CREATE_FILE_DEST Clause

Starting with Oracle Database 12*c* Release 1 (12.1.0.2), the `CREATE_FILE_DEST` clause of the `CREATE PLUGGABLE DATABASE` statement specifies the default Oracle Managed Files (OMF) file system directory or Oracle ASM disk group for the PDB's files. This clause enables OMF for the new PDB, independent of any OMF default path specified in the root for the CDB. The PDB's data files and temporary files are restricted to the specified directory and its subdirectories. However, there are a couple of pre-checks that you need to follow to successfully use this clause. If a file system directory is specified as the default location in this clause, then the directory must exist. Also, the user who runs the `CREATE PLUGGABLE DATABASE` statement must have the appropriate privileges to create files in the specified directory. Alternatively, you can specify the name of a directory object that exists in the CDB root (`CDB$ROOT`). The directory object points to the file system directory used by `CREATE_FILE_DEST`. If the default OMF location is set for the CDB in the root, the value of the `CREATE_FILE_DEST` overrides the root's setting. Additionally, specifying `CREATE_FILE_DEST=NONE` disables OMF for the PDB. If you omit this clause and root uses OMF, the PDB inherits the default path for OMF from the root.

For example, to use `/u01/app/oracle/pdb1/` as the default OMF directory, the syntax is

```
CREATE PLUGGABLE DATABASE PDB1 ADMIN USER ADMIN IDENTIFIED BY PASSWORD
CREATE_FILE_DEST = '/u01/app/oracle/pdb1/';
```

Another example can be using a directory object that already exists in `CDB$ROOT`. Assuming there is a directory object called `pdb_dir` in the root that points to `/u02/oracle/pdb/`, to create a PDB and set its OMF directory to `/u02/oracle/pdb/`, the syntax is

```
CREATE PLUGGABLE DATABASE PDB2 ADMIN USER ADMIN IDENTIFIED BY PASSWORD
CREATE_FILE_DEST = pdb_dir;
```

PATH_PREFIX Clause

The `PATH_PREFIX` clause of the `CREATE PLUGGABLE DATABASE` statement enables you to restrict all directory object paths associated with the PDB to a specified directory or its subdirectories. The `PATH_PREFIX` clause does not apply to data files, temporary files, or files created by OMF. It only applies to user-created directory objects. Additionally, the Oracle XML repository for the PDB, files created with the `CREATE_PFILE` statement, and the export directory for Oracle wallets are all restricted to the specified `PATH_PREFIX` directory of their corresponding PDB. In this clause, you can either specify an absolute path that is used as a prefix for all file paths associated with the PDB or the name of a directory object that exists in the root. (The directory object points to the absolute path to be used for `PATH_PREFIX`). In order not to impose any restrictions for the file paths, you can specify `NONE`, which has the same effect as omitting the entire clause. Similar to `CREATE_FILE_DEST` clause, `PATH_PREFIX` clause has its own restrictions. For example, after a PDB is created, you cannot change the value of `PATH_PREFIX`. In addition, the value of `PATH_PREFIX` is always added as a prefix to all local directory objects in the PDB. Therefore, it's important to update local directory objects accordingly so that the prefix doesn't affect their functionality.

For instance, to use `/u01/app/oracle/pdb2/` as a prefix for all file paths associated with the PDB2, the syntax is

```
CREATE PLUGGABLE DATABASE PDB2 ADMIN USER ADMIN IDENTIFIED BY PASSWORD
PATH_PREFIX='/u01/app/oracle/pdb2/';
```

Be sure to specify the path name carefully so that an issue does not occur when file names are appended to it. For example, on UNIX systems, path name must end with a forward slash (/).

About Lockdown Profiles

In the Multitenant architecture, economies of scale are achieved by sharing the key infrastructure and memory components. However, these are not the only resources that tenants share. Besides sharing the host environment, PDBs also share the OS, network, and common objects. Considering how certain privileges might let database users to perform cross-PDB operations, there is a possibility that PDBs can be exposed to some vulnerabilities. Especially in any private or public cloud environment, tenant isolation is a key requirement for security. We have recently explored how Multitenant can help you manage OS and file system interactions. The remaining areas such as network access, common object access, and administrative feature access can be controlled by lockdown profiles. Lockdown profile is new capability introduced with 12.2.

A lockdown profile is a mechanism to restrict certain operations or functionalities in a PDB. This new Multitenant feature is managed by a CDB administrator and can be used to restrict user access in a particular PDB. A lockdown profile can prevent PDB users from:

- Executing certain SQL statements, such as `ALTER SYSTEM` and `ALTER SESSION`,
- Running procedures that access the network (e.g. `UTL_SMTP`, `UTL_HTTP`),
- Accessing a common user's objects,
- Interacting with the OS (In addition to the capabilities covered by `PDB_OS_CREDENTIAL`),
- Making unrestricted cross-PDB connections in a CDB,
- Taking AWR snapshots,
- Using JAVA partially or as a whole,
- Using certain database options such as Advanced Queueing and Partitioning.

Creating a Lockdown Profile

In order to create a lockdown profile, you must have the `CREATE LOCKDOWN PROFILE` system privilege and be connected to a CDB root. Once you create the lockdown profile, you can add restrictions that you want to enforce to the profile. A lockdown profile is capable of enforcing more than one restriction at the same time. For example, a lockdown profile can disable both the network access and the use of `ALTER SYSTEM` statement at the same time in a PDB.

The following example demonstrates how to create a lockdown profile called `sec_profile` that restricts all the privileges associated with the `SET` clause of `ALTER SYSTEM` statement except for changing the value of `CURSOR_SHARING` parameter. Additionally, this lockdown profile disables the use of XDB protocols (FTP, HTTP, HTTPS) by the PDB.

1. Connect to the CDB root as a user who has the `CREATE LOCKDOWN PROFILE` system privilege.

   ```
   CONNECT c##cdb_admin
   Enter password: password
   ```

2. Create a lockdown profile called `sec_profile`.

   ```
   CREATE LOCKDOWN PROFILE sec_profile;
   ```

3. Use ALTER LOCKDOWN PROFILE statement to add restrictions to the profile.

```
ALTER LOCKDOWN PROFILE sec_profile DISABLE STATEMENT = ('ALTER SYSTEM') CLAUSE
= ('SET') OPTION ALL EXCEPT = ('CURSOR_SHARING');
ALTER LOCKDOWN PROFILE sec_profile DISABLE FEATURE = ('XDB_PROTOCOLS');
```

This is one of the typical use cases of lockdown profiles in which you can limit the administrative functionality enabled by the grant of a privilege. Grants alone are "all or nothing". If you are granted a privilege, you can do everything that comes along with that privilege. For instance, an autonomous PDB administrator who has the ALTER SYSTEM privilege can use that privilege to its full potential while the whole purpose of the grant may have merely been to administer certain parameters. The example above shows how you can restrict the scope of the ALTER SYSTEM privilege and only allow specific operations thanks to a lockdown profile. Lockdown profiles are complementary to grants and they give you the ability to take away the capabilities that otherwise come bundled with a privilege.

Enabling a Lockdown Profile

Lockdown profiles prevent users from accessing certain features or performing the operations that are disabled by the profile. However, creating a lockdown profile and adding some restrictions to it are not sufficient to enforce those restrictions for a PDB. A lockdown profile must be assigned to a PDB in order for it to take effect. This operation can be done by setting the value of PDB_LOCKDOWN initialization parameter to the profile name. After the parameter is set for the first time or changed from one profile to another, the new lockdown profile takes effect immediately. A lockdown profile can be assigned to individual PDBs, or to all PDBs in a CDB or an application container, as follows:

- If you set PDB_LOCKDOWN parameter while connected to a CDB root, then the lockdown profile applies to all PDBs in the CDB. It doesn't apply to the CDB root. For example, if we have a CDB named CDB1, the syntax is:

```
CONNECT sys/password@localhost/CDB1 AS SYSDBA
ALTER SYSTEM SET PDB_LOCKDOWN = sec_profile;
```

- If you set PDB_LOCKDOWN parameter while connected to an application root, then the lockdown profile applies to the application root and all application PDBs in the application container. For example, if we have an application root called APP_ROOT, the syntax is:

```
CONNECT sys/password@localhost/APP_ROOT AS SYSDBA
ALTER SYSTEM SET PDB_LOCKDOWN = sec_profile;
```

- If you set PDB_LOCKDOWN parameter while connected to a PDB, then the lockdown profile applies only to that PDB and overrides the lockdown profile that is enforced by the CDB or the application container, if one exists. For example, if we have a PDB called PDB3, the syntax is:

```
CONNECT sys/password@localhost/PDB3 AS SYSDBA
ALTER SYSTEM SET PDB_LOCKDOWN = sec_profile;
```

As the third bullet point states, a lockdown profile that is set in an individual PDB has the higher precedence and it overrides any other lockdown profile that is set in a CDB or application root. This capability gives you flexibility to increase or decrease the restrictions on individual PDBs on demand while still having the ease of managing many as one through the CDB-level or application root-level lockdown profiles.

You can enable a lockdown profile for all PDBs in a CDB as follows:

1. Connect to the CDB root as user who has common ALTER SYSTEM or common SYSDBA privilege.

```
CONNECT c##cdb_admin
Enter password: password
```

2. Run the ALTER SYSTEM SET PDB_LOCKDOWN statement.

```
ALTER SYSTEM SET PDB_LOCKDOWN = sec_profile;
```

The details of the available lockdown profiles, such as profile name, rules, and rule types, are available in the DBA_LOCKDOWN_PROFILES data dictionary view.

Dropping a Lockdown Profile

Similar to creating and enabling lockdown profiles, dropping a lockdown profile is a single-command operation as well. Especially in a cloud environment where restrictions on PDBs might change dynamically, you should be able to easily create, modify, or drop lockdown profiles. In order to drop a lockdown profile, you must be connected to CDB root and must have the DROP LOCKDOWN PROFILE system privilege, either granted commonly or granted locally in the CDB root. If the lockdown profile you want to drop is assigned to the PDB_LOCKDOWN initialization parameter (in other words, if it is currently in use), then the effect of the lockdown profile will be disabled immediately when you drop it. However, the value of the PDB_LOCKDOWN parameter will remain as the name of the dropped profile. You can find the list of existing lockdown profiles by querying DBA_LOCKDOWN_PROFILES in CDB root.

You can drop a lockdown profile as follows:

1. Connect to the CDB root as user who has the DROP LOCKDOWN PROFILE system privilege.

```
CONNECT c##cdb_admin
Enter password: password
```

2. Run the DROP LOCKDOWN PROFILE statement.

```
DROP LOCKDOWN PROFILE sec_profile;
```

Using Lockdown Profiles to Restrict Operations in PDBs

Lockdown profiles can play a crucial role in both on-premises and cloud deployments of Oracle Multitenant 12.2. Database consolidation is an area that requires operational isolation as well as proper resource allocation among tenants. However, providing isolation and allocating resources for independent tenants are not always sufficient, especially when you are dealing with vast number of tenants in private or public cloud environments. In addition to providing initial isolation and resource allocation, maintaining these two throughout the life span of a CDB is another key principle. In other words, you need to make sure your tenants are not violating any resource usage limitations or they do not have more privileges than they are supposed to have. If you remember our SaaS analogy earlier, it's not so different than managing an enormous hotel with thousands of rooms. When a new guest arrives, they get their room key that also has access to certain common areas. Each room is completely isolated from each other. In other words, you don't give your guests the keys to the other rooms. Moreover, if a guest is a VIP customer or has a special membership, they might have access to VIP lounges or they might be eligible for free room service. In other words, different guests might have access to different resources while these policies are strictly protected. This is indeed very similar to administering a CDB in the cloud. In our hotel example, a guest's access rights determined by the hotel management are embedded into their room key. In a CDB, this is achieved by lockdown profiles that are applied on tenant PDBs. For illustration, Oracle Exadata Express Cloud Service is a cloud platform that actively benefits from lockdown profiles. Several resource manager parameters such as CPU_COUNT, SESSIONS, and SGA_TARGET are set and restricted by lockdown profiles with varying values based on the service level in Exadata Express Service.

Lockdown profiles can be used to restrict database features, options, SQL statements, and clauses of SQL statements.

Disabling Features with Lockdown Profiles

The `FEATURE` clause of `ALTER LOCKDOWN PROFILE` statement lets you disable or enable operations associated with certain database features. You can specify one or more feature names in the `FEATURE` clause or just feature bundle name to disable or enable user operations for all features in that bundle. For example, `COMMON_USER_LOCAL_SCHEMA_ACCESS` and `LOCAL_USER_COMMON_SCHEMA_ACCESS` are two feature names that belong to the feature bundle `COMMON_SCHEMA_ACCESS`. Table 1 presents all supported features with their corresponding feature bundles and operation descriptions.

TABLE 1 - LOCKDOWN PROFILE FEATURES

Feature Bundle	Feature	Operations
AWR_ACCESS	AWR_ACCESS	The PDB taking automatic and manual Automatic Workload Repository (AWR) snapshots
COMMON_SCHEMA_ACCESS	COMMON_USER_LOCAL_SCHEMA_ACCESS	A common user invoking an invoker's rights code unit or accessing a BEQUEATH CURRENT_USER view owned by any local user in the PDB
COMMON_SCHEMA_ACCESS	LOCAL_USER_COMMON_SCHEMA_ACCESS	• A local user with an ANY system privilege (for example, CREATE ANY TABLE) creating or accessing objects in a common user's schema for which the privilege applies. Note: Disabling the LOCAL_USER_COMMON_SCHEMA_ACCESS feature does not prevent a local user with the SYSDBA privilege or specific object privileges from creating or accessing objects in a common user's schema. • A local user with the BECOME USER system privilege becoming a common user • A local user altering a common user by issuing an ALTER USER statement • A local user using a common user for proxy connections
COMMON_SCHEMA_ACCESS	SECURITY_POLICIES	Creation of certain security policies by a local user on a common object, including: • Data Redaction • Fine Grained Auditing (FGA) • Real Application Security (RAS) • Virtual Private Database (VPD)
CONNECTIONS	COMMON_USER_CONNECT	A common user connecting to the PDB directly. If this feature is disabled, then in order to connect to the PDB, a common user must first connect to the CDB root and then switch to the desired PDB using the ALTER SESSION SET CONTAINER statement.

CONNECTIONS	LOCAL_SYSOPER_RESTRICTED_MODE_CONNECT	A local user with the SYSOPER privilege connecting to a PDB that is open in RESTRICTED mode
CTX_LOGGING	CTX_LOGGING	Logging in Oracle Text PL/SQL procedures such as CTX_OUTPUT.START_LOG and CTX_OUTPUT.START_QUERY_LOG
JAVA	JAVA	Java as a whole. If this feature is disabled, then all options and features of the database that depend on Java will be disabled.
JAVA_RUNTIME	JAVA_RUNTIME	Operations through Java that require java.lang.RuntimePermission
NETWORK_ACCESS	AQ_PROTOCOLS	Using HTTP, SMTP, and OCI notification features through Oracle Streams Advanced Queuing (AQ)
NETWORK_ACCESS	CTX_PROTOCOLS	Operations that access the Oracle Text datastore types FILE_DATASTORE and URL_DATASTORE Printing tokens as part of CTX logging with events EVENT_INDEX_PRINT_TOKEN and EVENT_OPT_PRINT_TOKEN
NETWORK_ACCESS	DBMS_DEBUG_JDWP	Using the DBMS_DEBUG_JDWP PL/SQL package
NETWORK_ACCESS	UTL_HTTP	Using the UTL_HTTP PL/SQL package
NETWORK_ACCESS	UTL_INADDR	Using the UTL_INADDR PL/SQL package
NETWORK_ACCESS	UTL_SMTP	Using the UTL_SMTP PL/SQL package
NETWORK_ACCESS	UTL_TCP	Using the UTL_TCP PL/SQL package
NETWORK_ACCESS	XDB_PROTOCOLS	Using HTTP, FTP, and other network protocols through XDB
OS_ACCESS	DROP_TABLESPACE_KEEP_DATAFILES	Dropping a tablespace in the PDB without specifying the INCLUDING CONTENTS AND DATAFILES clause in DROP TABLESPACE statement
OS_ACCESS	EXTERNAL_GLOBAL_AUTHENTICATION	Creating external and global users in the PDB Creating external and global roles in the PDB
OS_ACCESS	EXTERNAL_FILE_ACCESS	Using external files or directory objects or directory objects in the PDB when PATH_PREFIX is not set
OS_ACCESS	EXTERNAL_PROCEDURES	Using external procedure agent extproc in the PDB
OS_ACCESS	FILE_TRANSFER	Using the DBMS_FILE_TRANSFER package
OS_ACCESS	JAVA_OS_ACCESS	Using java.io.FilePermission from Java
OS_ACCESS	LOB_FILE_ACCESS	Using BFILE and CFILE data types
OS_ACCESS	TRACE_VIEW_ACCESS	Using the following trace views: • [G]V$DIAG_OPT_TRACE_RECORDS • [G]V$DIAG_SQL_TRACE_RECORDS • [G]V$DIAG_TRACE_FILE_CONTENTS

		• V$DIAG_SESS_OPT_TRACE_RECORDS • V$DIAG_SESS_SQL_TRACE_RECORDS
OS_ACCESS	UTL_FILE	Using UTL_FILE. If this feature is disabled, then the database blocks use of the UTL_FILE.FOPEN function.

The following example disables all features in the feature bundle NETWORK_ACCESS:

```
ALTER LOCKDOWN PROFILE sec_profile DISABLE FEATURE = ('NETWORK_ACCESS');
```

The following example disables all features except the `COMMON_USER_LOCAL_SCHEMA_ACCESS` and `LOCAL_USER_COMMON_SCHEMA_ACCESS` features:

```
ALTER LOCKDOWN PROFILE sec_profile DISABLE FEATURE ALL EXCEPT =
('COMMON_USER_LOCAL_SCHEMA_ACCESS', 'LOCAL_USER_COMMON_SCHEMA_ACCESS');
```

Disabling Database Options with Lockdown Profiles

The `OPTION` clause of `ALTER LOCKDOWN PROFILE` statement lets you disable or enable certain database options. Similar to limiting database features, supported database options can be disabled altogether by specifying the `ALL` clause or partially by using the `ALL EXCEPT`. The latter disables all supported database options except for the ones specified in the SQL statement. Currently, Oracle Database Advanced Queueing and Oracle Partitioning are the options that can be turned on or off by lockdown profiles.

The following example disables user operations associated with the Oracle Partitioning option:

```
ALTER LOCKDOWN PROFILE sec_profile DISABLE OPTION = ('PARTITIONING');
```

The following example enables user operations associated with the Oracle Database Advanced Queueing option:

```
ALTER LOCKDOWN PROFILE sec_profile ENABLE OPTION = ('DATABASE QUEUEING');
```

Disabling SQL Statements and Statement Clauses with Lockdown Profiles

Lockdown profiles can also be used as means of limiting the scope of certain SQL statements. This is a powerful capability as it can restrict mission critical operations being performed by any user who has the right privilege. For this purpose, you can disable any combination of `ALTER DATABASE`, `ALTER PLUGGABLE DATABASE`, `ALTER SESSION`, and `ALTER SYSTEM` statements by a lockdown profile. However, if you don't want to disable these statements completely, it is also possible to disable only certain clauses of these statements. In order to do this, specifying enough keywords to unambiguously identify a single clause is required. For instance, specifying `ARCHIVE` to disable `ARCHIVE LOG` clause of `ALTER SYSTEM` statement is sufficient while specifying `FLUSH` to disable `FLUSH SHARED_POOL` clause of `ALTER SYSTEM` statement is not adequate because several `ALTER SYSTEM` statements begin with this keyword such as `ALTER SYSTEM FLUSH GLOBAL CONTEXT`.

Additionally, being specific to the `SET` clause of `ALTER SYSTEM` or `ALTER SESSION` statements, you can set default, min or max values for the option that you specify. The `VALUE` clause lets you set default value for a clause option and goes into effect for any PDB to which the profile applies after you close and reopen the PDB. However, if the lockdown profile that you are enabling does not contain the `VALUE` clause for any of its rules then you don't have to close and reopen the PDB since the profile takes effect immediately as mentioned in previous sections. The goal of this clause is to simultaneously set a default value for an option and restrict users from setting or modifying the

value. The MINVALUE and MAXVALUE clauses, on the other hand, restrict you from setting a value less than or greater than the value of the option clause respectively. MINVALUE and MAXVALUE settings take effect immediately when the lockdown profile is assigned to a PDB without requiring you to close and reopen the PDB.

The following examples better illustrate what the syntax looks like and how these functionalities work.

» Disable ALTER DATABASE statement:

```
ALTER LOCKDOWN PROFILE sec_profile DISABLE STATEMENT = (‘ALTER DATABASE’);
```

» Disable ALTER SYSTEM SUSPEND and ALTER SYSTEM RESUME statements:

```
ALTER LOCKDOWN PROFILE sec_profile
  DISABLE STATEMENT = ('ALTER SYSTEM')
  CLAUSE = ('SUSPEND', 'RESUME');
```

» Disable COMMIT_WAIT and CURSOR_SHARING parameters of ALTER SESSION statement:

```
ALTER LOCKDOWN PROFILE sec_profile
  DISABLE STATEMENT = ('ALTER SESSION’)
  CLAUSE = (‘SET’)
  OPTION = (‘COMMIT_WAIT’, ‘CURSOR_SHARING’);
```

» Disable PDB_FILE_NAME_CONVERT parameter of ALTER SESSION statement. It also sets the default value for PDB_FILE_NAME_CONVERT to 'cdb1_pdb0', 'cdb1_pdb1'. This default value will take effect the next time the PDB is closed and reopened.

```
ALTER LOCKDOWN PROFILE sec_profile
  DISABLE STATEMENT = ('ALTER SYSTEM')
  CLAUSE = (‘SET’)
  OPTION = (‘PDB_FILE_NAME_CONVERT’)
  VALUE = ('cdb1_pdb0', 'cdb1_pdb1');
```

» Disable CPU_COUNT parameter of ALTER SESSION statement for values less than 2 or greater than 6:

```
ALTER LOCKDOWN PROFILE sec_profile
  DISABLE STATEMENT = ('ALTER SYSTEM')
  CLAUSE = (‘SET’)
  OPTION = (‘CPU_COUNT’)
  MINVALUE = ‘2’
  MAXVALUE = ‘6’;
```

Conclusion

Oracle Multitenant 12.2 delivers economies of scale through consolidation while providing great isolation across tenants. Considering each PDB might have different resource and operational requirements, the ability to provide custom isolation levels for independent PDBs comes forward as one of main principles in 12.2. While being a highly capable product in 12.1, significant enhancements are delivered in the area of tenant isolation with 12.2.

Unlike first generation cloud architectures, in which individual databases are hosted in dedicated VMs and therefore the relationship between number of databases and cost is linear, Multitenant delivers true economies of scale to the Database Cloud. The greater the scale, the greater the economies, and therefore the key requirement for this potential to be fulfilled is to eliminate any barriers to consolidation. Tenant isolation is a major component of this. In a highly consolidated environment, tenants share not only the key infrastructure and memory components but also the network, OS, file system, and common objects of the database. Use `PDB_OS_CREDENTIAL` parameter to designate a less powerful OS user to reduce the potential risk of OS interactions. Furthermore, `CREATE_FILE_DEST` and `PATH_PREFIX` clauses enable CDB administrators to set PDB-specific paths for PDBs' data files and directory objects respectively. In addition to these two capabilities, starting with 12.2, we offer another powerful feature called lockdown profiles. A lockdown profile is fundamentally a security mechanism to limit certain operations in a PDB. It can restrict the scope of powerful privileges such as `ALTER SYSTEM` or `ALTER SESSION`. Moreover, lockdown profiles can make CDB management significantly easier by disabling access to certain resources and administrative features. All these solutions come hand in hand when it comes to tenant isolation and help us deliver a world-class database cloud architecture.

Oracle Corporation, World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065, USA

Worldwide Inquiries
Phone: +1.650.506.7000
Fax: +1.650.506.7200

ORACLE

CONNECT WITH US

blogs.oracle.com/multitenant
facebook.com/oracle
twitter.com/OraclePDB
oracle.com/goto/multitenant

Integrated Cloud Applications & Platform Services

Oracle Multitenant: Isolation In Oracle Database 12*c* Release 2 (12.2)
March 2017
Author: Can Tuzla
Contributing Author: John P. McHugh, Prashanth Shanthaveerappa, Patrick Wheeler