ORACLE DATABASE 12c

Many in One

Create many databases in one database instance with the Oracle Database 12c multitenant architecture.

John, the lead database architect at Acme Bank, has some important visitors today: the chief information officer and her senior IT leaders.

Acme has several divisions, all of which use a third-party application called MortEngage to manage the mortgage loan process. Over the past several years, all of these divisions have deployed and maintained separate installations of the product in their independent databases. The company understands the value of consolidating multiple databases and machines, and as part of its current consolidation project, the CIO wants to put all the separate installations into a single database running on one powerful machine. All the different instances of the application would be stored as schemas in the same database, and that would eliminate a lot of overhead. There would be one Oracle Database instance instead of hundreds, there would be only one set of Oracle Database metadata, fewer DBAs would be needed to manage the one database, and so on. The idea is great, but unfortunately, as the CIO has learned, the application needs a specific schema name—MORTENGAGE—in the database and it is hard-coded in the application and cannot be changed. Obviously, as the DBAs correctly informed her, it is not possible to create two different schemas with the same name in a database. Therefore, the only way to run multiple installations of the application is to create the required schema in multiple separate databases.

Consolidation? *Impossible* was the general verdict of Acme's DBAs.

But the smart CIO isn't ready to give up just yet. She reaches out to John in search of a solution, and she isn't disappointed. Indeed it is possible to consolidate the database—she learns from a smiling John—with the new multitenant architecture in Oracle Database 12c. In the rest of this article, you will see how John provides the solution.

ORACLE MULTITENANT

The problem, John tells the CIO and the senior IT leaders, has to do with the namespace. Each Oracle Database user is uniquely named, so if the application needs a database user named MORTENGAGE, only one instance of that application can run against that database. Each additional deployment of the same application would need to connect as the MORTENGAGE user on a different database.

But that changes in Oracle Database 12c, John explains. Instead of creating multiple databases, one can create multiple pluggable databases in a multitenant container database. The database instance—a set of memory areas, such as the buffer cache and shared pool and processes such as pmon and smon—is associated with the multitenant container database; the individual pluggable databases do not have their own database instances. The Oracle Database instance processes exist only for the multitenant container database—not the pluggable databases—saving a lot of resources on the host server.

To illustrate the concept, John points the CIO and the IT leaders in his office to Figure 1 and shows the various databases; the memory, CPU, and storage they consume; and the savings after they have been consolidated as pluggable databases in a single multitenant container database. In Figure 1, the red databases are database instances—three before consolidation and one multitenant container database after consolidation. The green databases—after consolidation—are pluggable databases.

The CIO chews on the information a bit and muses, "So, John, you are saying there is just one *actual* database, and therefore there is just one set each of memory areas such as SGA and background processes such as smon, regardless of the number of pluggable databases. Well, if there is just one actual database, how can there be multiple users with the same name—MORTENGAGE—in the database?"

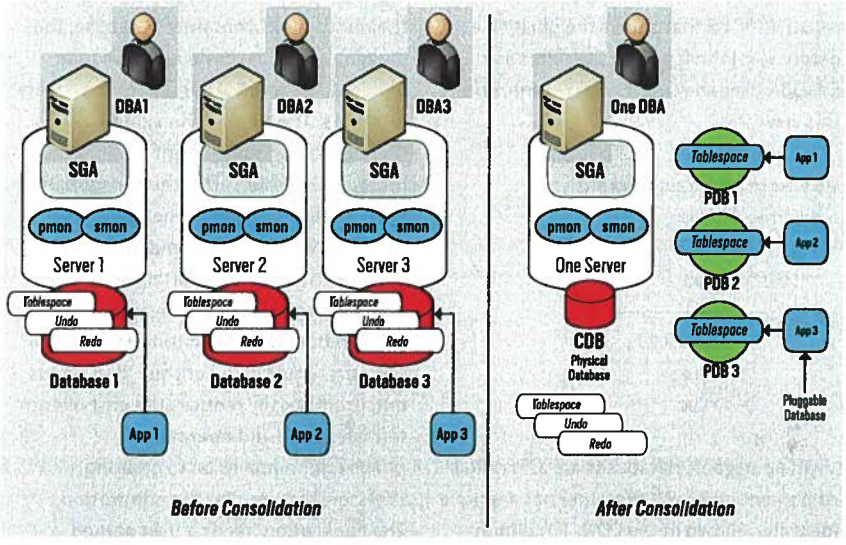

Figure 1: From multiple database instances to pluggable databases in a multitenant container database

I-HUA CHEN

This is where the beauty of the multitenant architecture in Oracle Database 12c comes in, John explains. To a user, the pluggable databases behave just like regular databases. In fact, a typical user may not even know the difference. If 50 instances of the application need to run, John continues, the Acme DBAs create 50 pluggable databases in a single multitenant container database. Each pluggable database will have one MORTENGAGE user and will support one installation of the application. The audience, now visibly enthused, urges John to demonstrate how it all works.

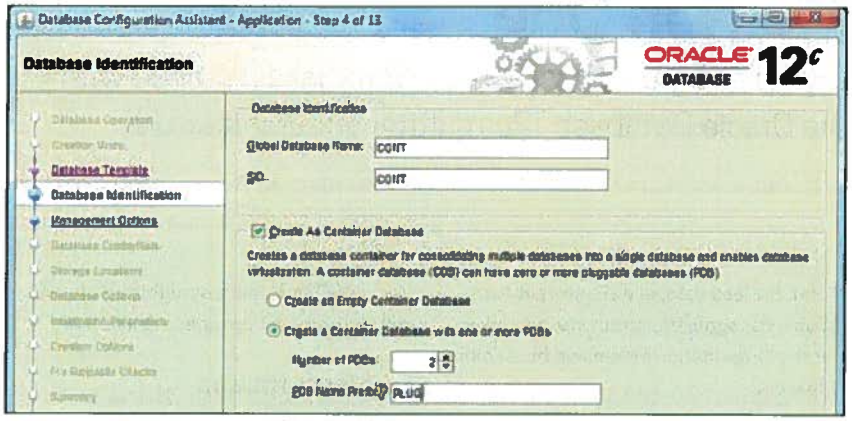

Figure 2: Oracle Database Configuration Assistant screen for creating pluggable databases

INSTALLATION

To create the databases, John kicks off the Oracle Database Configuration Assistant that came with Oracle Database 12c. After a few clicks, he comes to the Database Identification screen, shown in Figure 2. John selects Create a Container Database with one or more PDBs as shown and chooses 2 as the number of pluggable databases. He enters CONT as the multitenant container database name (in the Global Database Name field) and PLUG as the pluggable database name prefix (in the PDB Name Prefix field). This will create a multitenant container database named CONT and two pluggable databases named PLUG1 and PLUG2.

After the multitenant container database (CDB) is created, John wants to confirm that two pluggable databases were created. Oracle Database 12c introduces a new view called V$PDBS that shows the pluggable databases. John logs into SQL*Plus as a SYSDBA user and selects two columns from this view:

```
SQL> select con_id, name
  2  from v$pdbs;

    CON_ID  NAME
----------  ------------------------------
         2  PDB$SEED
         3  PLUG1
         4  PLUG2
```

The pluggable databases are also called *containers*, and each container has a unique identifier, shown in the CON_ID column in the output. John examines the output from the query and confirms that two containers—with CON_IDs 3 and 4—were indeed created as expected. By default, Oracle Database 12c creates a container called PDB$SEED, which also shows up in the output. This container can't be used by applications, John adds, but it can be used to create other containers by cloning.

The pluggable databases do not have their own background processes and shared memory areas. They do, however, take up some space in the multitenant container database's Oracle metadata, redo logfile, controlfile, and some tablespaces such as undo. Each of the pluggable databases has its own SYSTEM, SYSAUX, TEMP, and USERS tablespaces. There is a common location for the Automatic Diagnostic Repository feature of Oracle Database for the multitenant container database; the pluggable databases do not have independent Automatic Diagnostic Repository locations. Therefore, John explains, if there were 50 independent databases as mentioned earlier, after the consolidation into 1 multitenant container database, the DBAs would need to manage only the multitenant container database. There is just 1 instance and 1 pmon process instead of 50, reducing the amount of CPU and memory required. All of this, John points out, dramatically reduces the cost of both infrastructure and operation.

Next, John moves on to creating the users required for the application. The application needs a user named MORTENGAGE. John creates that user in each of the pluggable databases. To create the user in the PLUG1 pluggable database, he first sets the CONTAINER session parameter to the pluggable database name and then creates the user.

```
SQL> alter session set container =
plug1;

Session altered.

SQL> create user mortengage identified
by plug1pass;

User created.
```

To create the same username in the other pluggable database, he issues the following commands:

```
SQL> alter session set container =
plug2;

Session altered.

SQL> create user mortengage identified
by plug2pass;

User created.
```

After John issues the commands, he confirms that the users exist by checking a view—new in Oracle Database 12c—called CDB_USERS:

```
SQL> select con_id, username, common
  2  from cdb_users;
```

```
SQL> connect mortengage/plug2@plug2
SQL> show parameter optimizer_use_sql_
plan_baselines

NAME                          TYPE        VALUE
----------------------------- ----------- -----
optimizer_use_sql...          boolean     FALSE
```

The value of the parameter is FALSE, as expected. Then, connecting to PLUG1, John confirms that the value is TRUE.

```
SQL> connect mortengage/plug4@plug1
SQL> show parameter optimizer_use_sql_
plan_baselines

NAME                          TYPE        VALUE
----------------------------- ----------- -----
optimizer_use_sql...          boolean     TRUE
```

ADMINISTRATION

Although the CIO is growing in confidence about the multitenant architecture, Jill, the DBA manager, appears skeptical. "Well," she questions, "if this is actually a single database, how do the DBAs manage the pluggable databases independently? For example, how does a DBA shut down one pluggable database but not the other?"

That is a genuine concern, John agrees, but he assures her that the pluggable databases can still be managed separately. To demonstrate, John first logs into the PLUG1 pluggable database as SYSDBA and shuts it down:

```
SQL> conn sys/oracle@plug1 as sysdba
Connected.
SQL> shutdown immediate
Pluggable database closed.
```

After it is shut down, John checks the status of the pluggable databases:

```
SQL> conn / as sysdba
Connected.
SQL> select con_id, name, open_mode
  2  from v$pdbs;

CON_ID  NAME       OPEN_MODE
------  ---------  ----------
     2  PDB$SEED   READ ONLY
     3  PLUG1      MOUNTED
     4  PLUG2      READ WRITE
```

Code Listing 2: Alert log

```
2012-12-21 16:24:31.874000 -05:00
ALTER PLUGGABLE DATABASE CLOSE IMMEDIATE
ALTER SYSTEM: Flushing buffer cache inst=0 container=3 local
2012-12-21 16:24:32.923000 -05:00
Pluggable Database PLUG1 closed
Completed: ALTER PLUGGABLE DATABASE CLOSE IMMEDIATE
2012-12-21 16:24:38.095000 -05:00
ALTER PLUGGABLE DATABASE  OPEN READ WRITE
2012-12-21 16:24:45.659000 -05:00
Opening PDB 3 with no Resource Manager plan active
Pluggable Database PLUG1 opened read write
Completed: ALTER PLUGGABLE DATABASE  OPEN READ WRITE
... output truncated ...
```

The PLUG1 pluggable database is now shown as MOUNTED, John confirms. The other pluggable databases have not been affected.

Similarly, to start the pluggable database, John issues the following commands:

```
SQL> conn sys/oracle@plug1 as sysdba
Connected.
SQL> startup
Pluggable Database opened.
```

Because the database instance belongs to the multitenant container database and is shared between pluggable databases, the instance itself is not shut down when John shuts the pluggable database down. And given that the alert log is for a database instance, it is for the multitenant container database, and John displays the last part of it, as shown in Listing 2. From the lines in the output, John confirms that the PLUG1 pluggable database was closed and later reopened in read/write mode.

Jill still isn't convinced that this consolidation would be a cakewalk for her team. "We have a ton of scripts that use views with the prefix DBA_, such as DBA_USERS, to get a listing of all users," she explains. "Do we have to change all those scripts to use the CDB_ prefixed views? That's a lot of changes."

Not at all, assures John. The CDB_ prefixed views, newly introduced in Oracle Database 12c, show the data across *all* the pluggable databases inside a container database. However, when the DBA is connected to a single pluggable database, the DBA_ prefixed views show the metadata of that specific pluggable database only. None of the scripts referencing the DBA_ prefixed views needs to be changed.

In addition, Oracle Enterprise Manager 12c is also aware of the multitenant architecture, and the Acme DBAs can use the tool to manage the multitenant container database and the pluggable databases. Jill couldn't be happier.

CLONING

The beauty of the Oracle Database 12c multitenant architecture doesn't stop at just being able to run multiple pluggable databases within a multitenant container database, John adds. It is also possible to create another pluggable database as a copy of an existing one quickly—or clone the pluggable database. John demonstrates the procedure of cloning the PLUG2 pluggable database as a new pluggable database named PLUG3:

1. Connect to the multitenant container database as a SYSDBA user.

```
SQL> conn / as sysdba
```

2. Close the PLUG2 pluggable database.

```
SQL> alter pluggable database plug2
close;
```

3. Open the PLUG2 pluggable database in read-only mode, because that is the status it should be in when it is cloned.

```
SQL> alter pluggable database plug2
open read only;
```

4. Create the PLUG3 pluggable database as

```
CON_ID  USERNAME        COMMON
------  --------------  ------
     3  MORTENGAGE      NO
     4  MORTENGAGE      NO
     1  SYSTEM          YES
     2  SYSTEM          YES
     3  SYSTEM          YES
     4  SYSTEM          YES
```

John draws everyone's attention to this output. There are two users named MORTENGAGE, but they are in two different pluggable databases—containers—distinguished by CON_IDs 3 and 4. Because they are distinct in their respective pluggable databases, they are not visible across all the pluggable databases. They are called *local* or noncommon users, indicated by the NO value in the COMMON column in the output. In contrast, John points out, the SYSTEM user is visible in all the containers. However, unlike the MORTENGAGE user, the SYSTEM user is the same user in all the pluggable databases in a multitenant container database. SYSTEM is known as a *common* user, and the SYSTEM user's COMMON column value is YES.

CONNECTION

"I see that there is a MORTENGAGE user in each of the pluggable databases," offers one DBA, "but how does an application connect to a specific pluggable database?"

"Exactly as it used to connect in the past," replies John. "By using the appropriate TNS connect string." He puts the entries in the TNSNAMES.ORA file, located in the network\admin directory under Oracle Home on the client machines where the applications run; Listing 1 shows the TNSNAMES.ORA entries.

The service names in each connect string specify the pluggable database to connect to. Each pluggable database, John explains, has a unique service name that is the same as the pluggable database name. So the PLUG1 pluggable database has the default service name PLUG1, which cannot be defined in any other pluggable database in a multitenant container database. The applications connect to the database as they always did. For a simple demo, John connects to the PLUG1 pluggable database by using SQL*Plus:

Code Listing 1: TNS entries for pluggable databases

```
PLUG1 =
  (DESCRIPTION =
    (ADDRESS_LIST =
      (ADDRESS = (PROTOCOL = TCP)(HOST = prohost1)(PORT = 1521))
    )
    (CONNECT_DATA =
      (SERVICE_NAME = PLUG1)
    )
  )

PLUG2 =
  (DESCRIPTION =
    (ADDRESS_LIST =
      (ADDRESS = (PROTOCOL = TCP)(HOST = prohost1)(PORT = 1521))
    )
    (CONNECT_DATA =
      (SERVICE_NAME = PLUG2)
    )
  )
```

```
sqlplus mortenagage/plug1pass@plug1
```

The application running against the PLUG2 pluggable database uses the plug2 connect string, so nothing changes from the perspective of the applications or the typical users. Instead of connecting to separate independent databases, applications and application users now connect to multiple pluggable databases—containers—in a single multitenant container database. To the applications, the containers are independent databases. This is music to the ears of the CIO.

To identify which pluggable database a user is connected to, John demonstrates a new user environment variable called CON_NAME in the SYS_CONTEXT function:

```
SQL> select sys_context('userenv',
'con_name')
  2  from dual;

SYS_CONTEXT('USERENV','CON_NAME')
--------------------------------
PLUG1
```

At this point, May, the lead developer responsible for application deployment and maintenance, expresses a concern. Different installations of the MortEngage application require different settings in the database to improve performance, she informs everyone. For example, in one database, the OPTIMIZER_USE_SQL_PLAN_BASELINES database parameter is set to TRUE to take advantage of the baselines, whereas in other databases, the parameter is set to FALSE. Now that the multitenant container database is the same for all these pluggable databases, her concern is that all the pluggable databases will have the same value for the parameter and that therefore some application installations may have serious issues.

It's a valid concern, John concedes, but he announces that fortunately it is possible to set different values for different pluggable databases. He demonstrates this by setting the value of the parameter in the PLUG2 pluggable database to FALSE.

```
$ sqlplus sys/oracle@plug2 as sysdba

SQL> alter system set optimizer_use_sql_
plan_baselines = false scope=memory;
```

Then he sets the value of the same parameter to TRUE in the PLUG1 pluggable database.

```
$ sqlplus sys/oracle@plug1 as sysdba

SQL> alter system set optimizer_use_sql_
plan_baselines = true scope=memory;
```

John then logs in to the different pluggable databases as the MORTENGAGE user and checks for the value of the parameter. First, connecting to PLUG2, he checks for the value:

a new metadata file with information from the pluggable database. This metadata file is in XML format, and John names it pluginfo_plug4.xml. This file is created in the Oracle Home, under the database directory (in Windows) or dbs (in UNIX).

```
SQL> alter pluggable database plug4
  2  unplug into 'pluginfo_plug4.xml';

Pluggable database altered.
```

The Oracle-hosted online version of this article at bit.ly/158fW4g includes the remaining steps required to clone the pluggable database as well as questions and answers on backups in the multitenant container database architecture.

CONCLUSION

Pluggable databases running in the multitenant architecture of Oracle Database 12c offer the simplicity and familiarity of traditional databases while providing the flexibility to run multiple pluggable databases within one multitenant container database. The multitenant architecture enables many schemas with the same name to be created without the need to create many disparate databases. Because there is just one multitenant container database, there is just one database instance, eliminating the Oracle Database background process and memory areas such as SGA for separate databases. And running pluggable databases in the multitenant architecture of Oracle Database 12c requires no changes to applications.

Acme's IT leaders are all nods and smiles, and there are no more questions about Oracle Database 12c, multitenant container architecture, pluggable databases, provisioning, cloning, or backups. The meeting is adjourned. ◀

Arup Nanda (arup@proligence.com) has been an Oracle DBA since 1993, handling all aspects of database administration, from performance tuning to security and disaster recovery. He was *Oracle Magazine*'s DBA of the Year in 2003 and received an Oracle Excellence Award for Technologist of the Year in 2012.

NEXT STEPS

READ online-only article content
bit.ly/158fW4g

LEARN more about Oracle Database 12c
oracle.com/database

Oracle Multitenant concepts
Oracle Database Concepts 12c Release 1 (12.1)
bit.ly/13S7CZv

DOWNLOAD Oracle Database 12c
bit.ly/epBiUG

a copy of PLUG2. Because cloning creates new datafiles, John needs to indicate that the new datafile names should include "PLUG3" wherever "PLUG2" appears. The FILE_NAME_CONVERT clause takes care of that:

```
SQL> create pluggable database
plug3 from plug2 file_name_convert =
('PLUG2','PLUG3');
```

The command succeeds, with the message "Pluggable database created."

5. Open the newly created pluggable database.

```
SQL> alter pluggable database plug3
open;
```

Now the PLUG3 pluggable database is ready for business.

6. As a final step, John closes the PLUG2 pluggable database (which is now in read-only mode) and reopens it in read/write mode.

Jill, the DBA manager, sees a lot of potential for this feature. The DBAs are often asked by the application team to clone QA and test databases for their testing and to drop them after the testing is completed. This activity not only demands a considerable effort from the DBAs but it also requires significant CPU and memory on the server to run the new database instances. With the multitenant architecture and cloning, John continues, Jill can immediately spin up another database for testing without consuming any additional CPU or memory. When the testing is completed, she can drop the newly created pluggable database by issuing the following SQL:

```
drop pluggable database plug3 including
datafiles;
```

Jill also occasionally has to clone databases from another server. She asks whether the Oracle Database 12c multitenant architecture supports that. The cloning doesn't have to be within the same database, John answers. It is possible to clone a pluggable database from another multitenant container database as well, or "plug" a pluggable database from a remote multitenant container database into this multitenant container database. John demonstrates the technique with the following steps:

1. Close the pluggable database to be cloned in the source multitenant container database.

```
SQL> alter pluggable database plug4
close;

Pluggable database altered.
```

2. "Unplug" the pluggable database: Create