Multitenancy:
heavily underestimated and rarely used

Mag. Dr. Thomas Petrik
Sphinx IT Consulting
Wien

Keywords

Oracle Database, Multitenancy, Pluggable Database, Container, Recovery, Snapshots, Standard Edition, Enterprise Edition

Introduktion

Multitenancy has been introduced with 12c but still the feature has not been largely adopted by the customers. This is probably due to the fact that it has never been clear to the DBAs that there is no need to buy a Multitenancy Option in order to create a container with just a single pluggable database. Or if they knew they have not seen the benefit of changing their paradigms and modifying all their administration and monitoring scripts. However, earlier this year the use of non-CDBs has officially been deprecated and, therefore, DBAs are forced to move to this new technology medium-term. Nevertheless, multitenancy turned out to be a great feature not only for consolidation purposes but for rapid deployment environments as well as in the context of restore and recovery.

Architecture

During the "create database" a so called container database (CDB) is created. The CDB with the container ID (con_id) 0 has a root container (cdb$root) with con_id=1 which holds the base data dictionary. All instance processes (SMON, PMON, LGWR, etc.) run within the CDB context. Finally one or more pluggable databases (PDBs) are created with con_id >2. The pdb$seed (con_id=2) is a special pluggable database being automatically created during the create database command and simply serves as a template for new pluggable databases. Each pluggable database has its own admin user (e.g. PDBADMIN) which may be used as an autonomous application DBA depending on the roles granted. Furthermore, each container registers with the listener with its own service name being the same as the PDB name.

Accordingly, the root container is identified by ORACLE_HOME and ORACLE_SID and can be reached by an internal connect (/ as sysdba) while the PDBs can only be reached by the service name. This is certainly the most important change in the basic concept compared to non-CDBs because not only the DBA must change his good old administrative scripts but also the one or the other application change might be due. Still the use of SIDs especially in JDBC thin connects is widely spread. However, Oracle has provided a workaround for this problem: in the listener.ora file the parameter USE_SID_AS_SERVICE_listener may be set to ON for specific listeners.

Basic infrastructure services like RAC, Dataguard and RMAN backup are handled in the context of the CDB. There is a basic data dictionary (system, sysaux) for the root container and there are separate data dictionaries on top for each PDB which, of course, must match in their structure and patch status. Redo logs, archivelogs and undo space are provided by cdb$root while the TEMP tablespace is individual for each container. The same is true for the default user tablespace.

Creation of a PDB

The creation of a PDB from the seed template is straight forward. First of all the CDB must be prepared by using the "enable pluggable database" clause within the create database statement and as a parameter in the initial init.ora file. Then a PDB can be created as follows:

```
create pluggable database DWH00E
  admin user pdbadmin identified by "…"roles=(connect,dba)
  file_name_convert=NONE;
```

The create statement implicitly defines the admin user for the new PDB. file_name_convert may be NONE in case Oracle Managed Files (OMF) are in use which is strongly recommended for the sake of simplicity.

The thus created PDB is in mounted state and must be opened explicitly:

```
alter pluggable database all open;
```

Notice that all PDBs are only mounted by default after a startup of the CDB (except for the seed database which is always opened read only). In order to accomplish an auto-open the open state may be saved:

```
alter pluggable database … save state;
```

This is a 12.1.0.2 feature. In version 12.1.0.1 a startup database trigger in the CDB has to do the work.

While v$database still shows the status and properties of the CDB (con_id=0) the new views v$containers and v$pdbs supply information about all containers and the PDBs, respectively.

Common users and container views

A new class of data dictionary views has been added: CDB_* views simply show a union all over all DBA_* views of each container with the con_id added as an additional column. The pdb$seed is excluded from these views by default which can be changed by setting the parameter exclude_seed_cdb_view to false.

The multitenancy architecture distinguishes between local and common users. Common users exist in all containers but may be granted different privileges. Users like SYS, SYSTEM, etc. are system-defined common users while user-defined users must be prefixed by C##. On the contrary local users are specific within each PDB. The root container cannot have any local user.

Oracle supports user-defined container views as well by providing the containers function starting with 12.1.0.2. In 12.1.0.1 the cdb$view function provided the same functionality but was undocumented. In any case the functions traverse through all open (read write or read only) containers and the object selected must exist in all these containers. Also the selecting common user must have select privilege on this object and the object must be located in the common users schema.

In addition a common user may be granted a privilege on an object for a subset of containers:

```
create user c##mon identified by sx123;

grant connect to c##mon;

grant select on v_$session to c##mon container=all;

alter user c##mon set container_data = (cdb$root,tst00e) for v_$session
  container=current;
```

Clone histories for rapid deployment

Having licensed the multitenancy option one can easily create a clone history of a PDB in order to save specific development stages. The clones are separate PDBs in mounted, read only or read write state. Stepping back to a previous development stage is straight forward by cloning that PDB over the current working PDB.

Creating a new clone:

```
alter pluggable database pdb1 close immediate;
alter pluggable database pdb1 open read only;
create pluggable database pdb1_1 from pdb1;
alter pluggable database pdb1_1 open read only;
alter pluggable database pdb1 close;
alter pluggable database pdb1 open;
```

Restoring a previous development stage:

```
alter pluggable database pdb1 close immediate;
drop pluggable database pdb1 including datafiles;
create pluggable database pdb1 from pdb1_3;
alter pluggable database pdb1 open;
```

A similar procedure may be used without the multitenancy option (e.g. in case of a standard edition). Instead of online PDBs the clones exist as sets of DB files on the file system with corresponding xml-files describing the structure of the DB:

```
alter pluggable database pdb1 close immediate;
alter pluggable database tst00e unplug into 'pdb1_1.xml';
drop pluggable database pdb1 keepfiles;
create pluggable database pdb1 as clone using 'pdb1_1.xml';
alter pluggable database pdb1 open;
```

The use of the "as clone" clause is key for this procedure.

PDB subset recovery

Currently there is no "flashback pluggable database". However, a PDB can be restored and recovered in place:

```
RMAN> restore pluggable database … until time 'sysdate-1/24';
RMAN> recover pluggable database … until time 'sysdate-1/24';
RMAN> alter pluggable database tst02e open resetlogs;
```

The PDB files are restored in place while the recover command builds up an auxiliary instance for the CDB in the flash recovery area.

As an alternative the PDB (and – implicitly – the CDB) can be cloned in a separate cloning environment. The thus cloned PDB can be migrated into the original CDB over a database link. This feature does not work in 12.1.0.1 (bug 15931910).

```
duplicate target database to … pluggable database pdb1 until time …
```

After having opened the clone PDB read only execute on the primary CDB:

```
create database link clone_link connect to pdbadmin
  identified by … using '…';

create pluggable database pdb1 from pdb1@clone_link;
```

In addition the "no data" clause can be used to clone metadata (i.e. empty table structure) only.

Patching a CDB

In the world of containers executing an SQL script from rdbms/admin or elsewhere against a DB is obviously not enough. Some patches must be executed in the root container only, others have to be applied to all containers or just to a subset of PDBs. catcon.pl is a specialized but not very well documented tool for these tasks. One of the undocumented options is "-n" which specifies the number of containers which should be patched concurrently. catcon.pl also patches the pdb$seed making use of the undocumented parameter _oracle_script which can also be seen in the create database scripts created by the dbca tool.

Conclusion

A 12c database (EE and SE) should be built in the new CDB-structure only. In the view of further developments it makes no sense to continue with the old non-CDB structure. In addition, the benefit is huge for the DBA especially in very dynamic development and test environments although it must be admitted that the initial overhead is remarkably high. The much higher packing density per machine, the new recovery methods on application level in a highly consolidated environment as well as the concept of individual application DBAs make this feature an essential component especially in midsize and enterprise landscapes.

Contact:

Mag. Dr. Thomas Petrik

Sphinx IT Consulting

Aspernbrückengasse 2

A-1020 Wien

Phone:	+43 664 155 8304
Fax:	+43 (1) 599 31-99
E-Mail	Thomas.Petrik@sphinx.at
Internet:	www.sphinx.at