**Managing Shared Database Environment**

In many shops, an Oracle database is not dedicated to a single application. Multiple small, mostly unrelated, applications store data in the same database, to save license cost, simplify database management, eliminate database links, and minimize per-database resource usage in the same fashion as merger of companies saving on administrative overhead. This shared database environment has some unique characteristics, which will be discussed in this article.

**I. Resource naming conflict and identification**

Because this is one database, there's only one namespace for certain types of objects, such as public synonyms, public database links, roles, users. Just as a popular Internet domain name is worth tens of thousands of dollars, a good name with a database-wide namespace is a precious resource. For instance, a database hosting multiple applications for a hospital will find it difficult to accept a vendor's application that requires a public synonym called PATIENT pointing to a table in the schema of this application, because such a common name in this environment may be contended for by another application. The DBA's, the application team, and the vendors (if the app is not built in-house) will cooperate to resolve the issue. Such high profile public synonyms should be exclusively used by the most important application, or all public synonyms should be banned altogether (except Oracle's own). In fact, almost all public synonyms can be recreated as private synonyms in the schemas that need this synonym. The same solution may not apply to the other types of objects with a database scope namespace, such as users or roles. Fortunately, in practice, rarely do we find a vendor whose application creates a user with a name so general or popular as to cause a name conflict as in the case of public synonyms.

Also as a result of database-wide namespace restriction, easy identification of the names in a shared database environment may be a challenge. For instance, an application named Autosys creates a role UJOADMIN. For any reason that the database needs a cleanup or user identification, perhaps to drop users no longer employed, or to decommission certain applications no longer used, finding the application that uses such role needs additional time. Proliferation of the names not reminiscent of the application they're used in leads to a higher chance for mistakes and more work time. Again, cooperation between the DBA's, the application team, and developers or vendors is needed.

There's no comment field in DBA\_USERS or DBA\_ROLES, and no DBA\_USER|ROLE\_COMMENTS view, which could be more useful and more often used than the comment views for tables or columns. Some applications use two or a few more schemas. Among these, occasionally you'll find an application whose schema names don't spell similarly. But this is only a minor challenge when compared to a legacy client-server application whose users are actual database accounts instead of users in the mid-tier. An efficient method to tag these users is needed. One way is to name the users always with a short prefix or suffix specific to the application. But be aware of users of a different application that happen to use the same prefix or suffix. Another way is to assign all these users to an application specific profile, visible in DBA\_USERS, not for the purpose of managing their resource usage although you can, but for identification or grouping of these users.

In spite of these efforts, multitude of these database scope resource names not spelled similarly or even spelled similarly may still need manual, human, work, i.e., writing documents. Nothing can completely replace words that clearly describe their usage, function, and cross-schema relationship.

**II. Client connection management**

It is beneficial to create each TNS connect identifier unique to the application, even though all these identifiers point to the same physical database. Every application connected to a shared database environment is subject to change, such as migration to another database for various reasons or being decommissioned. When the application team, the DBA's, or the system admins decide to move this data to a different database, for a higher or lower database version or for proximity to the data of another application to avoid database links, or for any other reason, you only need to change the hostname for the connect identifier in the tnsnames.ora file, not the configuration of the application because the identifier remains the same. The greatest advantage is realized if the TNS entries are centrally managed by the DBA's; for example, everyone reads one tnsnames.ora located in a network share, or queries an OID (Oracle LDAP) server, to resolve the connection string. With this arrangement, the data migration is completely transparent in the sense that the application team does not need to change anything.

Even though this change for one specific application rarely happens, it is a challenge to keep track of these changes if you have hundreds of connect identifiers. To "query" which database is accessed by which identifiers, a little programming is needed to read the regular tnsnames.ora. Fortunately, each stanza in tnsnames.ora can actually be written on one single line. This undocumented feature allows a simple grep command, possibly piped to awk or perl, to easily query anything from the file. If the connect identifiers are stored in OID, you can dump all the entries into a file using a simple script at

http://yong321.freeshell.org/oranotes/Ldap2Tnsnames.txt

The result is actually a proper, usable, tnsnames.ora file in the grep-friendly format of each entry on one line.

Some users, particularly those using Java, insist on using the hardcoded hostname, port, and SID for their applications, which will create problems in otherwise transparent data migration, in addition to a separate headache during rolling maintenance work on a RAC database. DBA's may advise the users on how to change it to using a simple connect identifier. For example, if the application which uses JDBC supports OCI driver, the connection string is simply

db\_url = "jdbc:oracle:oci:@*myapp*";

along with appropriate sqlnet.ora, and either tnsnames.ora or ldap.ora. If the application only supports JDBC thin driver and you have an OID to centrally provide name resolution, the string is

db\_url = "jdbc:oracle:thin:@ldap://*oidhostname*:389/*myapp*,cn=OracleContext,dc=*mycompany*,dc=*com*";

Some application software only provides a configuration GUI that takes database hostname, port, and SID. In one case in our shop, we identified the XML file generated by the GUI and manually changed the XML file to using our in-house OID. The vendor never approved or disapproved this change, but the application team is happy with this manual change.

**III. Security**

Many software vendors are small companies that do not have sufficient experience in an enterprise environment. The software assumes a database dedicated to their application and requests for unnecessarily high privileges than needed, sometimes even a DBA role, just to make programming easy. Applications that come into a shared database environment must go through a thorough check on their security requirement. Apart from DBA, EXP|IMP\_FULL\_DATABASE and a few other roles, system privileges in the form of <action> ANY <object>, such as SELECT ANY TABLE, CREATE ANY SEQUENCE, ANALYZE ANY, should be removed. Users or vendors must be advised that these ANY privileges allow them to take action in any schema, which is, to their surprise, not their intention.

SELECT ANY TABLE is often abused for a different reason. It's unfortunate that for user A to query any of user B's tables or views, Oracle provides no simple role or privilege just to do that. Developers or vendors resort to this system privilege as a quick fix, even though they honestly do not have interest in peeking at data in schemas other than the few used by their own application. They should be advised to grant SELECT privilege on each of B's tables to A and be warned to do the same when B creates new tables in the future.

In order to allow certain power users or developers to tune or troubleshoot their application, SELECT\_CATALOG\_ROLE may be granted. This is acceptable if no sensitive data or confidential programming logic in other schemas exists as PL/SQL code including trigger code, or in view definitions. (Wrapping PL/SQL code is not a good defense because unwrapping tools are freely available on the Internet. Also be aware that V$SQLSTATS exposes unmasked text after "alter user|grant … identified by" in 11.2.0.2 and up.) Otherwise, these power users should only be granted SELECT privilege on certain views, such as V$SESSION, V$LOCK, on an as-needed basis.

WITH GRANT OPTION should be used sparingly and carefully because this option is inheritable. The "downstream" grant is normally used by a power user or application admin to grant an application-specific role to individual database users of the application using a client-server model. If the power user further grants WITH GRANT OPTION, it's possible that some end users, even some not using this application, will one day be found to have a role this application team has never expected.

Granting the commonly used RESOURCE role has a side effect never stopped by Oracle: UNLIMITED TABLESPACE privilege is also granted, even in the case RESOURCE is granted not manually but by an import, including automatic user creation by a data pump import. Developers or vendors find UNLIMITED TABLESPACE a convenience to obviate the need to give users UNLIMITED QUOTA on the tablespaces they actually use. But in a shared environment, this privilege should be reserved for DBA's only, as it permits the grantee to create a segment in any tablespace. (Incidentally, in 11*g*R2, if the grantee has both RESOURCE role and certain tablespace quotas, revoking UNLIMITED TABLESPACE or RESOURCE will also revoke all the quotas at the same time. So remember to give the quotas back after the revoke.)

**IV. Performance management**

Oracle provides no facility to limit a user's usage of the shared pool. In a shared database environment, you may need to monitor per-schema SQL area usage with

select parsing\_schema\_name, sum(sharable\_mem)

from v$sqlarea

group by parsing\_schema\_name

order by 2;

Add "having sum(sharable\_mem) > ..." as needed. It's not uncommon to see an application that does not use bind variables taking a big chunk of the shared pool with thousands of literal SQLs. Although an ultimate solution is a rewrite of the code, a schema-level logon trigger to set CURSOR\_SHARING to FORCE may be created to correct most of the literal SQLs:

create trigger appuser.logon\_set\_cursorsharing

after logon on appuser.schema

begin

execute immediate 'alter session set cursor\_sharing=force';

end;

/

In emergency, run a harmless DDL on the table or view referenced by the many literal SQLs, which is less damaging than flushing the shared pool. Then request the application server admin to re-connect to the database for the logon trigger to take effect. Of course, not all unjustified high usage of shared pool is due to literal SQLs; for example, SQLs may be written differently simply due to a large number of possible permutations of column names, and a re-design in a larger scope is needed.

A logon trigger is a powerful solution to meet the requirement of those vendors who insist on certain database parameter settings but have no knowledge of the co-existing schemas in the database. In our experience, the majority of the required parameters can be set with ALTER SESSION. Requirement of the rest is usually already met or can be explained away.

Another precious memory resource, buffer cache, can be broken down into each user's usage by a join to V$BH (or X$BH):

select owner, round(count(\*)\*8/1024) mb

from v$bh a, dba\_objects b

where a.objd = b.data\_object\_id

group by owner

order by 2;

That assumes 8k block size throughout the entire instance. Alternatively, if each application schema has its own tablespace, which of course is not always true, we can also break down into tablespaces:

select name, round(count(\*)\*8/1024) mb

from v$tablespace a, v$bh b

where a.ts# = b.ts#

group by name

order by 2;

It runs faster and serves about the same purpose from the business point of view.

As in the case of shared pool, Oracle provides no facility to limit a user's usage of the buffer cache. SQL tuning is the best overall solution. Less buffer gets, less unnecessary use of buffer cache. In addition, with the blessing of Oracle support, consider setting \_SERIAL\_DIRECT\_READ to TRUE to bypass buffer cache for full table scans. (Oracle wisely changes its default value to the new value AUTO beginning with 11.2.0.2.) Use parallel executions if appropriate. (Don't forget the simple method of setting the table or index parallel degree.) Make sure CACHE attribute of tables is not set without an explanation, as some developers might do. Configure a recycle pool and assign infrequently used, fairly large tables or indexes to it. These are all measures to help use less of the precious buffer cache. They can be implemented with either a logon trigger (when setting the parameter), or a change to the segment property, in the end creating a more friendly shared database environment in which no user uses the cache excessively.

There're ways to limit usage of other types of resources, CPU, parallel degree, execution time, undo, etc. The implementation of Oracle resource manager is such that when the user's limit is reached, the session is usually terminated or queued for execution. We found this solution not optimal, and chose to regularly monitor sessions and advise the application teams or help tune the applications.

Performance management is assisted with database services, particularly in a RAC database. Unlike connect identifiers, database services are best not to have a one-to-one mapping to the applications. Common practice is that applications having similar performance characteristics share one service. In addition, in a shared database environment, if some applications are somewhat related to each other, they're best assigned to the same service. Applications that bear the feature of OLTP can use a service that only runs on certain instances of the RAC database, and applications that more resemble DSS can use a service that only runs on a different set of instances. The two sets of instances can have different parameter settings to match the behavior of the "local" applications.

Lastly, database jobs in a shared database are better scheduled for different times if they use a fair amount of resource. If business requires two jobs run at the same time, and if the database is RAC, consider separating them on different nodes. If the jobs are submitted from application servers, the DBAs may have to coordinate with various teams to find the best scheduling windows.

**V. Database upgrade or migration**

A shared database may have a new option for data migration or database upgrade. Suppose the database contains 1 TB data with many schemas used by a number of applications, but no schema exceeds 100 GB. A conventional database upgrade probably requires an hour or more of downtime, depending on the amount of PL/SQL code, size of data dictionary, etc. Migration of the data to a different server with zero or close to zero downtime requires a temporary data guard, GoldenGate, or similar setup. But since we know the database has multiple small applications, we're offered another way to migrate the data: export and import schema or schemas for one application at a time. The advantage of this approach is not in saving in absolute downtime, but its simplicity with practically (to be understood literally) zero downtime. It's simple because there's no need to set up a standby, not even the need to work on transportable tablespaces. Unless the application is truly 24x7, there's practically zero downtime because your downtime may not be the users' downtime; the DBA's can work with one application team at a time to negotiate a zero-usage window, in which only a small amount of data needs to be exported and imported into the target database.

The same approach can be used to "upgrade" a database, or even OS or hardware, because the target database has no restriction in version, OS, or hardware. After all the user schemas are migrated, the old database can be decommissioned. Since Oracle is never literally upgraded but always newly-installed, occasional bugs or restrictions associated with the version upgrade are avoided. There may even be unknown advantage in not using the same datafile images for more than just a few years.

Take caution in bundling schemas that should be migrated together. Schemas used by one application are usually but not always named similarly, and multiple applications may have inter-dependency not obvious to the DBA's. This knowledge is passed from the application team, and can be supplemented by checking data dictionary for cross-schema reference, i.e., queries against DBA\_DEPENDENCIES, DBA\_SOURCE, DBA\_SYS|ROLE|TAB\_PRIVS, etc. On the other hand, one schema may even be used by more than one application. An incident that happened to us was resolved by some application history search, which revealed this "convenient", quiet, re-use of the same schema for a different application unknown to most people.

**VI. Tablespace**

Tablespace has a database scope namespace. A vendor's installation script should not use a name so generic that it may collide with existing or future names. Fortunately, this is rarely an issue. If the application does not specify a tablespace, you may choose to use a generic tablespace to host multiple small applications. There's pro and con to this practice. It avoids proliferation of tablespaces, too many of which requires more attention from the DBA's, and a very large number of tablespaces, even after they once existed and were later dropped (but still in SYS.TS$ table), cause performance problems in various recursive SQLs. The downside of using a generic tablespace, however, is that corruption or recovery of this tablespace may affect more than one application, and as a minor annoyance, it's impossible to break down buffer cache usage into applications as discussed above.

Certain applications require significantly more temporary space than others for sorting, hashing, or other uses. Dedicated temporary tablespaces for these users may be created. On the other hand, Oracle provides no user-specific undo tablespace. In RAC, these users may be bound through service to a dedicated instance where the undo tablespace for that instance is more or less dedicated to their own use.

Some DBA's find convenience in using bigfile tablespaces. But one drawback is that if there's corruption more extensive than RMAN block media recovery normally handles but not extensive enough to corrupt the single datafile everywhere, the entire tablespace will still have to be brought offline during the recovery period. If on the other hand a smallfile tablespace encounters such corruption, it may be isolated to only one of many datafiles and the applications may run just fine if the SQLs don't happen to visit the offlined file. This effect is more evident in a shared database environment when you evaluate the choice between bigfile and smallfile for the generic tablespace hosting multiple small applications.

**VII. Other trivia**

Before accepting a vendor's software into a shared database environment, two requirements must be cleared: database version and character set. Fortunately, in our experience, most vendors can accept the latest major release of Oracle and AL32UTF8 character set. Therefore the most accommodating shared database is best created with this character set and, as of this writing, one minor release of 11*g*R2 database version.

Some companies have annual or more frequent data recovery drills, to test the validity of the backed-up data. The backup is restored into a new, temporary database, or you open the standby database temporarily for read-write (after you set a restore point in order to flashback after the drill). The users connect to this DR database to verify their data. Don't forget to organize all application teams to test at the same time. It would be extremely counter-productive if one team requested to check the DR in January and another team insisted they check in February only.

**Conclusions**

Oracle is powerful yet expensive relational database software. Multiple small to medium size applications can store their data in one physical database. Peaceful coexistence of the schemas requires not only planning by the DBA's, but also cooperation and sometimes compromise between different teams. If one or a few applications are dramatically different in database setting from the others, they may be grouped and allowed to run on one or two instances of the RAC database. But if a dedicated but low resource usage database must be created, one server running multiple databases can be considered before budgeting for a separate server, to save license cost and relieve some administrative workload.

There's always a limit to which a database can be shared. Within the limit, a shared database environment may be the ideal choice for the business.

**Acknowledgement**: Many ideas in this article are contributed by my coworkers at M. D. Anderson Cancer Center.

**Author**

Yong Huang is a DBA at M. D. Anderson Cancer Center in Houston, Texas. Before joining M. D. Anderson, Yong worked as an Oracle DBA or consultant at Schlumberger, Unocal Oil, Nationwide Insurance, Electronic Arts, and eBay. More information is at <http://yong321.freeshell.org/computer.html>