Managing a Shared Database Environment

By Yong Huang
Edited by Arup Nanda

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 costs, simplify database management, eliminate database links and minimize per-database resource usage similar to how companies save on overhead when merging. This article explains factors to carefully consider in such a shared database environment. You will also learn what to consider when using one database for many applications.

Resource Naming Conflict and Identification

This is just one database, so there's only one namespace for certain types of objects such as public synonyms, public database links, roles, users, etc. Just as a popular Internet domain name is worth tens of thousands of dollars, a good name with a databasewide namespace is a precious resource. For instance, a database hosting multiple applications for a hospital may 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 taken by another application. The DBAs, the application team and the third-party vendors will need to work together to resolve the issue. You have to make a decision: Either mandate that such highly generic public synonyms can be exclusively used by the most important application, or ban all public synonyms altogether (except Oracle's own). Almost all public synonyms can be recreated as private synonyms in the schemas that need this synonym, removing the need for public synonyms altogether. The same solution may not apply to the other types of objects within a namespace with a database scope, such as users or roles. Fortunately, in reality, rarely do we find a vendor whose application creates a user with a name so generic or popular that it causes a name conflict, as is the case of public synonyms.

As a result of databasewide namespace restrictions, it may be a challenge to easily identify the usernames in a shared database environment. For example, an application named Autosys creates a role UJOADMIN, which is not representative of the application name. If you ever perform a cleanup of userids, perhaps to drop users no longer employed or to decommission certain applications no longer used, it's difficult to identify which application uses such a role. Use of such names not representative of the application for which they are used leads to a higher chance of mistakes and additional identification work. Again, a close coordination among the DBAs, the application team and the vendors is needed to identify the users conclusively.

Unfortunately, there's no comment field in the views DBA_USERS or DBA_ROLES, which would have been very useful. Some applications use two or more schemas. You may find an application whose schema names don't have anything to tie it back to that application. But this is only a minor challenge compared to a legacy client-server application whose users are actual database users instead of users in the mid-tier. You need to develop an effective method to identify these users as needed. One approach is to always name the users with a short prefix or suffix identifying the application. Watch out for 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, which is 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, several of these resource names within the database scope may not be named intuitively. Even if they are named correctly, they may still need some manual effort to make them clearer. Nothing can completely replace documentation that clearly describes their usage, function and cross-schema relationship.

Client Connection Management

You should create a TNS connect identifier unique to an application — even though connect identifiers of all the applications point to the same physical database. Every application connected to a shared database environment is subject to change, whether it is being migrated to another database for various reasons, being decommissioned, etc. When you decide to move this application to a different database for any reason (e.g., to upgrade/downgrade the database version or to move the data closer to other data in a different database to avoid database links), you only need to change the hostname in the connect string in the tnsnames.ora file and not the configuration of the application. You can get the most advantage if the TNS entries are centrally located. For example, everyone reads one tnsnames.ora located in a network share or queries an OID (Oracle LDAP) server to resolve the connect identifier. With this arrangement, the data migration is completely transparent to the application support teams.

Although changes for one specific application are infrequent, it could be a challenge to keep track of these changes when you have hundreds of connect identifiers. You will need to determine which database is accessed by which string from the regular tnsnames.ora. To make it simple, or do it programmatically, you could change the format of the tnsnames.ora. Fortunately, each stanza in tnsnames.ora can actually be written on one single line. Doing so 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 available on **http://yong321.freeshell.org/oranotes/Ldap2Tnsnames.txt**. The result is a usable tnsnames.ora file in the grep-friendly format of each entry on one line.

Some users, particularly those using Java, may require the use of 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. DBAs may advise the users

continued on page 16

on how to change it to using a simple connect identifier. For example, if the application that 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 (in case of LDAP). If the application only supports the JDBC thin driver and you have an OID to centrally provide name resolution, the connection string can be written as:

```
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 the database hostname, port and SID. In one case, we identified the XML file generated by the GUI and manually changed the XML file to use our in-house OID. The vendor never approved or disapproved this change, but the application team is happy with this manual change.

Security

Many software vendors are small companies with little experience with an enterprise environment. The software assumes that the database is dedicated to their application and requests privileges more than actually 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. System privileges in the form of <action> ANY <object>, such as `SELECT ANY TABLE`, `CREATE ANY SEQUENCE` and `ANALYZE ANY`, should be removed from all roles except `DBA`, `EXP|IMP_FULL_DATABASE` and a few others. Users or vendors must be advised that these ANY privileges allow them to take action in any schema, which they may find, to their surprise, is not their intention.

The system privilege `SELECT ANY TABLE` is often abused for a different reason. It's rather 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 often 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. You should advise users to grant SELECT privilege on each of B's tables to A and remind them to do the same when B creates new tables in the future.

You may want to grant `SELECT_CATALOG_ROLE` to certain power users or developers in order to allow them to tune or troubleshoot their application. This may be acceptable if the other schema does not have any sensitive data or confidential programming logic in the form of PL/SQL code including trigger code or in view definitions. You may consider wrapping the PL/SQL code, but it's not a good defense because unwrapping tools are freely available on the Internet. Alternatively, these power users should only be granted SELECT privilege on certain views, such as V$SESSION and V$LOCK, on an as-needed basis.

You should grant privileges WITH GRANT OPTION sparingly and after careful consideration 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 those not using this application, may 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 corrected by Oracle; the UNLIMITED TABLESPACE privilege is a part of it. Developers or vendors find this privilege a convenience to obviate the need to give users UNLIMITED QUOTA on the tablespaces they use. But, in a shared environment, this privilege should be reserved for DBAs only, as it permits the grantee to create a segment in any tablespace. Incidentally, in Oracle Database 11*g*R2, if the grantee has both RESOURCE role and certain tablespace quotas, revoking RESOURCE will revoke all the quotas as well as UNLIMITED TABLESPACE privilege at the same time. So remember to give the quotas back after the revoke.

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;
/
```

Note: This workaround is just a suggestion. Please ensure this is an agreed process in your environment before usage. Also note that a mistake in this trigger may lock out logins for that user.

In an emergency related to a spike in execution of non-bind SQL in the shared pool resulting in fragmentation and ORA-4031 errors, run a harmless DDL (such as granting the SELECT privilege to a user that already has that privilege) on the table or view referenced by the many literal SQLs to selectively invalidate cursors, which is less damaging than flushing the entire shared pool. Then, request the application server admin to reconnect 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 redesign in a larger scope is needed.

A logon trigger is a simple solution to meet the requirement of those vendors who insist on certain database parameter settings and who have no knowledge of the coexisting schemas in the database. In our experience, the majority of the required parameters can be set with ALTER SESSION.

The other portion of the SGA, the buffer cache, is another story. Breaking down the contents of the buffer cache into usage by each user is more challenging than that of shared pool. Theoretically, you could join V$BH (or X$BH) and `DBA_EXTENTS` and group by OWNER of the extents. But such a query will affect

the performance of the database. Instead, you may want to break down the buffer cache usage by tablespace:

```
select name, count(*)
from v$tablespace a, v$bh b
where a.ts# = b.ts#
group by name
order by 2;
```

It serves the same purpose from the business point of view as long as each application has its own tablespace, which of course is not always true.

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. The less the buffer gets, the less the use of buffer cache. In addition, with the assurance from My Oracle Support, consider setting `_SERIAL_DIRECT_READ` to TRUE to bypass buffer cache for full table scans. (Oracle wisely changed its default value to the new value AUTO beginning with 11.2.0.2.) Use parallel executions if needed. While you can simply set the parallel DEGREE clause on a table or index, be warned that all SQL accessing that table or index will attempt to use parallel query slaves if available. Alter table NOCACHE to shorten the time the blocks stay in cache after a full table scan. Configure a recycle pool and assign infrequently used, fairly large tables or indexes to it. These are all measures to help reduce disproportionate or inefficient use 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 are other ways to limit usage of other types of resources, such as CPU, parallel degree, execution time, undo, etc. You can choose to implement Oracle Resource Manager so that when the user's limit is reached, the session is usually terminated or queued for execution. We, however, found this solution not optimal and chose instead to regularly monitor sessions and advise the application teams or help tune the applications.

You can manage performance management with database services, particularly in a RAC database. Unlike connect identifiers, it's better to have database services not to have a one-to-one mapping to the applications. It's common practice to have applications with 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 the same service. Applications with behaviors leaning toward OLTP can use a service that runs only on certain instances of the RAC database. Applications with a DSS slant 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 applications they serve.

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

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 single schema exceeds 100 GB. A conventional database upgrade probably requires an hour or more of downtime depending on factors such as the amount of PL/SQL code, size of data dictionary, etc. The migration of the data to a different server with zero or close to zero downtime requires setting up, at least temporarily, some sort of replication mechanism such as Data Guard or GoldenGate. But we know the database has many small applications, so there is another way to migrate the data: export and import schema or schemas of one application at a time. The advantage of this approach is not just the savings in overall downtime but its simplicity with near zero downtime. There's no need to set up a standby or even the need for transportable tablespaces. Unless the application is truly 24/7, this approach achieves practically zero downtime because the users' downtime can be exploited. The DBAs 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 on versions, OS or the hardware platform. After all the user schemas are migrated, the old database can be decommissioned. Oracle is never literally upgraded but always newly installed, so occasional bugs or restrictions associated with the version upgrade can be avoided. There may even be some unknown advantage in not using the same datafile images for more than just a few years.

Make sure you bundle schemas that should be migrated together. Schemas used by an application are usually — but not always — named intuitively, and multiple applications may have shared schemas that are not obvious to the DBAs. This knowledge is usually available with the application team and can be further supplemented by checking the data dictionary for cross-schema references. For instance, you can query against `DBA_DEPENDENCIES`, `DBA_SOURCE`, etc., to understand the inter-schema dependencies. Similarly, one schema may even be used by more than one application. We had such an incident that we resolved by searching through the history of the application.

Tablespace

Tablespace belongs to a namespace with databasewide scope. You should make sure that a vendor's installation script should not use a tablespace name so generic that it may collide with existing or future tablespace names. Fortunately, this is rarely an issue in reality. If the application does not specify a tablespace, you may choose to use a generic tablespace to host multiple small applications. This practice has its own advantages and disadvantages. A large number of tablespaces demand more attention from the DBAs for their maintenance. They also have a lingering effect: Even after the tablespaces are dropped, the entries remain in the SYS.TS$ table and may cause performance problems in recursive SQLs involving this table. On the other hand, the downside of using a generic tablespace is that a corruption in or recovery of the tablespace may affect more than one application. Additionally, although a minor annoyance, it's impossible to break down buffer cache usage into applications as discussed previously.

Certain applications require significantly more temporary space than others for sorting, hashing or other uses involving temporary tablespaces. You should create dedicated temporary tablespaces for these users. 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 DBAs may find it convenient to use bigfile tablespaces, but be aware of one drawback of bigfile tablespaces. If there is corruption too extensive for RMAN block media recovery to handle but not extensive enough to corrupt the entire datafile, the whole tablespace will still have to be brought offline during

continued on page 18

the recovery period. On the other hand, if a smallfile tablespace encounters such corruption, it may be limited to only one of many datafiles and the applications may run just fine if the queries don't need to visit the offlined file. This consideration between bigfile and smallfile tablespaces is more important in a shared database environment for the generic tablespace hosting multiple small applications.

Other Miscellaneous Considerations

Before accepting a vendor's software into a shared database environment, you must clarify two important requirements: the database version and the character set. Fortunately, in our experience, most vendors can accept the latest major release of the Oracle and AL32UTF8 character set. Therefore, you can create the most accommodating shared database with this character set and, as of this writing, one minor release of Oracle Database 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 the standby database is temporarily opened for read-write (after you set a restore point in order to flashback after the drill). The users connect to this new (or standby) database to verify their data. Don't forget to organize all application teams to test at the same time. It would be counterproductive if one team requested to check the data in January and another team insisted they check only in February.

Conclusion

Oracle is a powerful but expensive database software. Many small- to medium-sized applications can store their data inside one physical database. Peaceful coexistence of the schemas requires not only planning by the DBAs, but also cooperation and sometimes compromise between different application 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 more 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.

Acknowledgement: Many ideas in this article are contributed by my co-workers at MD Anderson Cancer Center.

About the Author

Yong Huang is a DBA at MD Anderson Cancer Center in Houston. Before joining MD Anderson, he worked as an Oracle DBA/consultant at Schlumberger, Unocal Oil, Nationwide Insurance, Electronic Arts and eBay.