Uncommon Sense About Common Objects

By Yong Huang

In this article, we will take a deep dive into a set of objects that a DBA commonly uses. The information present in this article is either missing or inadequately described in the documentation when it refers to these commonly used objects. As a result, there is some confusion and misunderstanding in the Oracle community. In this article, we hope to provide the DBAs and developers an opportunity to revisit their understanding of these objects and use this information during root-causing of issues.

The DBA_OBJECTS view

Almost every object in an Oracle database has an entry in DBA_OBJECTS view. We query this view when we want to find out details about an object such as whether a given object is a table or index, when it was created, when a DDL statement last modified it, etc. This article examines this view and a few other related ones, focusing on characteristics that may surprise us. Please note that the "Object" in this article is not related to "object-oriented" database or programming, but rather to the database object such as a table or view.

The columns OWNER and OBJECT_NAME of DBA_OBJECTS are self-explanatory. The first surprise is this: Although the OBJECT_NAME is documented to be a VARCHAR2 column of length 30 characters (even in the Oracle Database 11*g* Documentation), since at least Oracle8*i*, this column has been of data type VARCHAR2 with a length of 128! Note, however, that only certain types of objects, DB links for example, are allowed to have names of longer than 30 characters. Another point to note is that the SUBOBJECT_NAME column is mainly for listing table or index partitions and subpartitions.

The OBJECT_ID and DATA_OBJECT_ID columns hold two different types of information. The OBJECT_ID column is a general data dictionary identifier assigned to every object in the database (a notable exception is DB link). However, the DATA_OBJECT_ID column, whose underlying column DATAOBJ# from the OBJ$ table is "data layer object number," can be considered as the ID for a segment, because only a segment can actually hold data. For example, if DATA_OBJECT_ID is null, it is implied that it must refer to an object not associated with a physical segment, such as view, sequence or domain index whose data exists in its secondary objects, or is a partitioned table or index whose data is in the individual partitions.

Normally, OBJECT_ID and DATA_OBJECT_ID, if the latter is not null, are equal to each other. However, they can differ after certain operations such as noted here.

- TRUNCATE TABLE (or ALTER TABLE TRUNCATE PARTITION), unless the table or partition is empty
- ALTER TABLE MOVE (or ALTER TABLE MOVE PARTITION)
- ALTER INDEX REBUILD (or ALTER INDEX REBUILD PARTITION)
- ALTER TABLE EXCHANGE PARTITION

An ALTER TABLE MOVE command, even without the use of the TABLESPACE clause or specifying the same tablespace, physically moves the table, based on DBA_EXTENTS.FILE_ID and BLOCK_ID. As a result, because the location of the rows have changed, the ROWID values become invalid and this requires the indexes for that table be set to UNUSABLE, unless the table or partition was empty before the move. ALTER TABLE MOVE LOB also changes the lob segment's DATA_OBJECT_ID. However, TRUNCATE TABLE doesn't move the table.

In this case, what value will be assigned to the OBJECT_ID or DATA_OBJECT_ID columns? To keep this simple, assume that the object was not dropped since it was last created. Except in the case of partition-table exchange, when a new DATA_OBJECT_ID is assigned, these columns are set to

```
SELECT MAX(DATA_OBJECT_ID)+1 FROM DBA_OBJECTS
```

If you just create a new table, it will be assigned an OBJECT_ID and DATA_OBJECT_ID of

```
SELECT GREATEST(MAX(OBJECT_ID), MAX(DATA_OBJECT_ID)) FROM
DBA_OBJECTS
```

If the table has one index, TRUNCATE will increment DATA_OBJECT_ID by 2 because its index takes the number one below it. If the table does not have an index, TRUNCATE should increment DATA_OBJECT_ID by 1 only. However, these increments will be larger (than the 1's and 2's noted above) if any other object was dropped since the object was last created. Those dropped objects, regardless whether they were purged from the recyclebin, already used some IDs. As a result, these IDs can't be reused, much like a sequence that has already been used. Note: Because exchange of a partition with a table swaps their DATA_OBJECT_ID's, we cannot assume that DATA_OBJECT_ID's always increment.

If the table or its partition is already empty, truncating it does not increment DATA_OBJECT_ID. However, moving an empty table or rebuilding an empty index or its partition still increments DATA_OBJECT_ID, consistent with relocation of their data blocks. In case of an online index rebuild, the DATA_OBJECT_ID could increment by 3 because of "transient" objects created during the rebuild, i.e., the journal table and IOT TOP index.

Because an object has two IDs, when should we use which ID? You should use the DATA_OBJECT_ID when you call the DBMS_ROWID package or query the V$BH or X$BH views. However, you should use the OBJECT_ID in most other cases. Some views such as the V$SEGSTAT, V$SEGMENT_STATISTICS and V$LOGMNR_CONTENTS views, refer to both IDs, so there won't be confusion. However, pay special attention to the DBMS_ROWID package or X$BH fixed table, because their object number or OBJ column does not have letter "D." So it's easy to incorrectly assume they match OBJECT_ID.

Another less obvious mistake some DBAs make is this: The lock checking script blindly assumes V$LOCK.ID1 matches an object's OBJECT_ID, without

continued on page 15

qualifying lock type. In fact, only a few types of locks, most notably TM, have the object's ID recorded in V$LOCK.ID1. Check V$LOCK_TYPE (10*g* and up only) for the meaning of ID1.

Another source of confusion and error is related to the timestamp columns in the DBA_OBJECTS view. These three timestamp-related columns are: CREATED, LAST_DDL_TIME and TIMESTAMP. (Although TIMESTAMP is actually VARCHAR2(19), the documentation says VARCHAR2(20) and is formatted from the DATE type by the definition of ALL_OBJECTS view.) The LAST_DDL_TIME records when a DDL statement last ran against this object. A DDL is a SQL that modifies data dictionary and commits your transaction implicitly ("implicit" means it's not associated with *user commits* statistic). However, note that not all DDLs start with the word "ALTER" (and not all ALTER SQLs are DDLs!). We know that if many SQLs using a particular object, table, view, etc., start to hard parse at the same moment, it is possible that a DDL statement, such as GRANT, ran on the object not long ago. Conversely, if you intentionally need to invalidate your SQLs (and thus force a hard parse) but don't want to indiscriminately flush the whole shared pool, you can run a harmless DDL on the object, e.g., GRANT SELECT ... TO DBA. (In Oracle 10.2.0.4 and up, you can have even finer granularity to flush an individual cursor.)

One surprise with LAST_DDL_TIME is that not all DDLs update this timestamp. It's understandable that it will not be updated by a failed DDL, which nevertheless commits the transaction. But some successfully executed DDLs do not update this column either. The following list, based on tests in Oracle 10*g*R2 and 11*g*R1, contains a few commands that do not update the timestamp (This list may not be complete.).

- ALTER INDEX SHRINK SPACE (This command, however, will update the *table*'s LAST_DDL_TIME!)
- ALTER TABLE SHRINK SPACE
- ALTER TRIGGER COMPILE
- ANALYZE
- DBMS_STATS.GATHER|DELETE|LOCK_TABLE_STATS (consider it as DDL because it implicitly commits and touches data dictionary)

To find the time these DDLs last ran, you probably have to drill into redo logfiles using LogMiner or other means. (You may find a timestamp in X$KGLOB.KGLNATIM, but that's the parent cursor's creation time, not child cursor's last active time.)

Perhaps related to this is the fact that the first three DDLs also don't invalidate cursors that reference the table. Of this list, note that invalidation of cursors when DBMS_STATS.GATHER_TABLE_STATS runs can be controlled by the NO_INVALIDATE option. (An interesting point to note is this: By default, NO_INVALIDATE was set to FALSE in Oracle 9*i*, i.e. cursors were invalidated when DBMS_STATS gathers table level statistics. In Oracle Database 10*g*, this default was changed to AUTO, allowing the cursor to be invalidated at a later time. Hence, make sure you use the right value in Oracle Database 10g, as required.)

The TIMESTAMP column in the DBA_OBJECTS view is less well-known. It is internally known as the "specification timestamp" column (see $ORACLE_HOME/rdbms/admin/sql.bsq or dcore.bsq in 11*g*). If you try a DDL that does *not* change the table specification, such as GRANT, the LAST_DDL_TIME will change to the current time but the TIMESTAMP column remains unchanged. For a view of other dependent objects, a DDL such as ALTER COMPILE would do the same as well. However, change in TIMESTAMP does not necessarily mean the table specification changes to something different. For example, if you "ALTER TABLE MODIFY" a column to exactly the same current type, TIMESTAMP will still be updated. Also, a change in the TIMESTAMP value does correspond to invalidation of other objects depending on this object. For instance, a view will become invalid if the table used in this view has a new TIMESTAMP.

The STATUS column of DBA_OBJECTS, obviously, is only meaningful for the types of objects that can ever become invalid. Objects containing physical segments by nature will never become invalid. Although an index can become unusable (visible in DBA_INDEXES.STATUS), its object status is still valid. Beginning with Oracle Database 10*g*, a synonym can become invalid, signifying a translation error. This is a great improvement, because it allows DBAs invalid object monitoring script to catch the error before it causes problems. STATUS takes values of 'VALID' and 'INVALID', but 'INVALID' is actually an umbrella value that covers a few others. These values can be seen in the "$ORACLE_HOME/rdbms/admin/sql.bsq" script, including real invalids that can't compile (denoted by a value of 3 in OBJ$.STATUS), and invalids that may be only temporarily so (denoted by a value of 5 in OBJ$.STATUS), etc. For instance, in case of materialized views, OBJ$.STATUS of 3 may correspond to 'NEEDS_COMPILE' under DBA_MVIEWS.COMPILE_STATE, while 5 to 'COMPILATION_ERROR'. However, STATUS of 'INVALID', specifically COMPILE_STATE of 'COMPILATION_ERROR' for materialized views, may be a false alarm due to numerous bugs such as 8320150, and 6402311.

An object generally becomes invalid when its underlying or referenced object changes its structure or specification. However, that's not an entirely accurate statement. It is better to say the underlined object has undergone certain DDL. As discussed above, only those DDLs that change the base object's TIMESTAMP invalidates the dependent object(s). A GRANT command is not one of them, and the five "softer" DDLs (noted in the bullets above) that don't even invalidate cursors will also not invalidate dependent objects.

Normally an invalid object such as a view automatically compiles itself to a valid state when it is subsequently used. However, sometimes the object requires explicit compilation to become valid. A common case involves an invalid trigger that throws ORA-4020, a library cache deadlock (not to be confused with ORA-60, an enqueue deadlock). It's always good practice to manually compile an invalid object unless you know for sure it can be auto-compiled when it's used later, because sometimes compilation involves code change, or compilation of other objects that require human intervention one way or another. Synonym compilation is special in that it always changes STATUS to valid even if the target object cannot be accessed. Therefore, a better way to compile a synonym is simply to describe it once. If it describes the target successfully, it is validated, and if it fails, it remains invalid.

Compilation is independent of whether an object is enabled or disabled (assuming the object – such as triggers – can be enabled or disabled). Assuming that recompiling an invalid trigger to valid status automatically enables a disabled trigger could have dangerous consequences – it does not!

The TEMPORARY column in the DBA_OBJECTS view indicates whether this object is a global temporary table, its index or a temporary LOB segment. This is actually redundant in a sense, because you can derive this information from the TEMPORARY column of DBA_TABLES and DBA_INDEXES.

The GENERATED column indicates whether the name is system generated. If both OBJECT_NAME and SUBOBJECT_NAME have values, GENERATED is about SUBOBJECT_NAME. However, this is not completely reliable. If part of the name is system-generated and part of the name is specified by you, this column could be either 'Y' or 'N'. For example, although names of the secondary indexes created as a result of a new CTXSYS.CONTEXT text index are considered system-generated, none of the secondary objects created as a

continued on page 16

Uncommon Sense About Common Objects *continued from page 15*

result of a new CTXSYS.CTXCAT index are. In fact, even a completely generated name may be marked as 'N' – the name of a journal table used during online index rebuild is not considered generated, although the interim IOT TOP index is. The query below will leave a completely system-generated type named like 'SYSTP%==', and yet DBA_OBJECTS does not consider it as generated.

```
CREATE TYPE MYTYPE AS TABLE OF VARCHAR2(4000);
SELECT CAST(COLLECT(MYCOL) AS MYTYPE) FROM MYTABLE;
```

It is possible that this may be just a side effect of Bug 4033868 where the generated type is not supposed to persist in the first place. (Warning: Sometimes the type won't be cleaned by SMON, which repeatedly throws ORA-21779, due to Bugs 8623930, 8674868. So if you test the query above, drop the type manually.)

The SECONDARY column of DBA_OBJECTS is about secondary objects created when you create a domain index, such as an Oracle text index. The undocumented DBA_SECONDARY_OBJECTS view reveals what the "primary" objects are for these secondary objects. However, that view only lists the DR$ tables for the text index, and the information is not as complete as in DBA_OBJECTS.

So far we're reviewed all documented columns of DBA_OBJECTS. Some columns of OBJ$ are not exposed, such as different types of invalid status, and other attributes of an object. See the description of the internal table OBJ$ in the sql.bsq script for details.

Other "Object"- related Views

Apart from object-oriented Oracle, the word "object" is used in other places in Oracle. The DBA_OBJECT_SIZE view has PL/SQL code size, separated into source, parsed and executable code size. You can measure code size yourself, which is the number of bytes of your code including the words such as "PROCEDURE MYPROC AS ...". It only counts one character at each line break even on Windows.

V$OBJECT_USAGE is one of the very few V$ views that, in turn, is based on other data dictionary tables, instead of internal X$ fixed tables. (As a result, it has no GV$ versions.) The name implies a more ambitious, broader-scale, plan to record usage of multiple types of objects. So far, it is only used to monitor index usage after an "ALTER INDEX MONITORING" command has been run on that index. If you do use index monitoring, unless the columns the index is built on are all null, remember to *not* collect stats with DBMS_STATS on the index, or on the table or schema with AUTO_CASCADE. This resets your index usage monitoring and you would have to start over again. (See Bugs 4615996 and 4432354.) Based on its current definition and limitation of performing-only index monitoring, a more appropriate name for V$OBJECT_USAGE would be USER_INDEX_USAGE (but not ALL_ or DBA_INDEX_USAGE, because the view is defined to have a limit on owner to userenv('SCHEMAID')).

V$SESSION has four columns recording what object, file, block and row on which a session is waiting. However, these columns don't always have the correct numbers. Instead, previous entries may hang around without proper cleanup. The object column, ROW_WAIT_OBJ#, could have a value of "-1," or a number totally irrelevant to your work. Thus far, I haven't found a bug report associated with this. (See my summary: http://yong321.freeshell.org/oranotes/RowWaitObj%23NotUpdated.txt)

The V$OBJECT_DEPENDENCY view is for the library cache what DBA_DEPENDENCIES is for the data dictionary. The column names for V$OBJECT_DEPENDENCY suggest a dependency flow from children back to parent, because FROM_xxx indicate cursors or PL/SQL objects and TO_xxx refer to the objects that the PL/SQL objects or cursors use or depend on. I don't know how useful this view is, because you can always read the cursor or PL/SQL code to see what tables or views it references, or check V$SQL (combined with DBA_SOURCE) to see what code uses a given table. A more frequently asked question is this: Given a SQL statement, how do we find the PL/SQL that calls this SQL? From 10*g* on, the PROGRAM_ID and PROGRAM_LINE# columns of V$SQL point to the stored PL/SQL object and the line in the code. 11*g* V$SESSION provides PLSQL_xxx columns, which may also help under certain circumstances. However, if it's a PL/SQL anonymous block, or your database version is older than 10*g*, or the cursor has partially aged out, only X$KGLRD retains this dependency information. (See www.ixora.com.au/q+a/0110/31164749.htm.) Keep in mind that queries on V$OBJECT_DEPENDENCY could potentially hold the library cache and shared pool latch for long periods when run on large, fragmented shared pools. So use this with caution!

Conclusion

In this article, we have taken an uncommon look at some common attributes of Oracle objects through exploration of a few data dictionary views, primarily DBA_OBJECTS. I hope it helped you understand Oracle's various types of objects better, write DBA scripts more efficiently, and helps in troubleshooting vexing issues.

About the Author

Yong Huang is a DBA at M. D. Anderson Cancer Center in Houston, Texas. Before joining M. D. Anderson, Huang worked as an Oracle DBA or consultant at Schlumberger, Unocal Oil, Nationwide Insurance, Electronic Arts, and eBay. Other than Oracle, his interests include UNIX and Windows. Find more information at **http://yong321.freeshell.org/computer.html**.