An Overview of the Mechanisms of Oracle RDBMS Transactions and Logs

JOHN A. THYWISSEN
The University of Texas at Austin

1. INTRODUCTION

We will examine how Oracle RDBMS transactions and logs work by breaking the concepts into three architectural "layers":

(1) Data layer: Tables & Indices & LOBs
(2) Transaction layer: Transactions & Undo entries
(3) Cache layer: Data blocks & Redo entries

(To simplify this discussion, we will ignore clustering, IOTs, partitioning, compression, some optimizations, and obsolete things.)

Note that at this point in the Oracle architecture, SQL has long since been decomposed into simple operations of the form "fetch block 321's row 99", "update block 1234's row 85 column 3 to 42", or "delete block 2222's row 183".

2. DATA LAYER: TABLES & INDICES & LOBS

Database objects are, of course, stored in data files. Some types of database objects, like tables and indexes, contain the "real" data of a database, and some types of objects, like users or sequences, are metadata only.

The stored form of each of these "real" data objects is a *segment*. Oracle segment types are: table, index, LOB, cluster, temporary, undo (and some internal types that we'll ignore). The metadata-only objects are kept in the database's *data dictionary*, which is simply a group of tables managed by the database system.

In a database, there are many segments and many data files used for storage. This many-to-many relationship is kept in the form of *tablespaces*. Each segment is created in one tablespace, and each datafile is part of one tablespace.

So, now you know how to find the set of data files an object might be found in. If the object is metadata-only, it will be in the data dictionary, which is in the system's tablespace's data files. Otherwise, for "real" objects, look at the tablespace for that object's segment to find the set of data files.

Files are partitioned into fixed-sized spans of bytes called *blocks*, typically 8 kB long. Note that Oracle blocks do not necessarily correspond to the underlying operating system's blocks.

The data dictionary lists, for each segment, the *segment header* block, which, in turn, stores a list of data blocks that belong to that segment[2]. A block is identified by a 32-bit *relative data block address* (DBA) which consists of a (relative) file number (within the tablespace) and a block number in that file. For example, DBA `0x01000014` indicates block 20 in file 4 in a given tablespace.

The data in a block is laid out according to the block's type. In particular, a *table data block* contains the number of rows, free space info, a *row directory*, and the rows. The row directory is simply an array of offsets into the block for the

---

[1] Author's Addresses: J. Thywissen, Department of Computer Science, The University of Texas at Austin, 2317 Speedway, Stop D9500, Austin, Texas 78712-1757, USA; email: jthywiss@cs.utexas.edu

[2] There are several layers of storage management complexity we're eliding here: extents, free space lists, bitmaps, etc. These are covered in the Oracle *Concepts* and administration manuals.

start of each row. Each row has a *flag byte*, *lock byte* (discussed later), column count, and the column data. The flag byte is used to indicate that the row is deleted, split among multiple blocks, etc. The column data is a sequence of variable-sized columns, each of which starts with its length, followed by its data.

By the way, the `ROWID` pseudocolumn in Oracle SQL is a combination of object number, relative file number, block number, and row number. This uniquely specifies the location of a row in the database at a particular instant.

The other block types—index blocks, LOB data blocks, undo blocks, etc.—have structures that are conceptually similar to table data blocks, but laid out to suit their purpose. For example, there are B-tree index block types that have interior ("branch") or leaf nodes in them.

To fetch a row, given the data block address (file number and data block number) and row number, read the indicated block from the file, look up the row's offset in the block's row directory, and then read the row header and column data at that offset. To insert a row into a table data block, go to the free space (as indicated in the block header), add a row header and column data, update the row directory to point to the row, and update the free space info.

3. TRANSACTION LAYER: TRANSACTIONS & UNDO ENTRIES

When SQL statements modify data, they do so as part of *transactions*. When a transaction starts, it is allocated a slot in a *transaction table*. A transaction table is part of an undo segment header, which also has a list of the undo blocks that are part of this undo segment. (Each Oracle database has one undo tablespace, with a few undo segments (typically 10-to-30-ish). Oracle tries to assign each active transaction to its own undo segment, but if there are many active transactions, sharing undo segments among multiple transactions may occur.)

Each transaction table slot contains the transaction's state (active or not), some flags, a sequence number (incremented every time the slot is reused), snapshot SCN, and the data block address of the last undo block that this transaction has used (discussed later). An Oracle *transaction ID* (XID) is composed of the undo segment number, the slot number, and the sequence number, and is written like `0x0004.016.00001a69`.

At certain points in a transaction, the Oracle instance's system change number (SCN), a global 48-bit integer stored in the SGA, is copied to the transaction table slot as that transaction's *snapshot SCN*. (SCNs are written like `0x0000.03ec99f5`, or in decimal 65837557.)

3.1. ITL & Lock Bytes

When a transaction starts to change a table row, it first marks the row as changed by an active transaction. Active transactions are tracked for table, index, and LOB blocks using an area in the block called the *interested transaction list* (ITL). An ITL has a small number of slots, each of which lists the transaction ID (XID), undo block address (UBA) of the last undo record that affected this block, a state (active/inactive), a commit SCN (covered later) and some other data.

Each row in a block has a *lock byte*. To mark a row as "in use" by a transaction, the transaction fills its data into an unused ITL slot, and sets the row's lock byte to that entry's index. A lock byte value of zero indicates that no ITL entry applies to this row.

3.2. Undo records

Every change to a table data block, LOB data block, or index block in a transaction creates an undo record, which is stored in an undo block. The undo record contains instructions to reverse the change made by the transaction—a delete undoes an insert, an update to an old value undoes an update to a new value, and so on.

Each undo record indicates: which block (AFN & DBA) it affects, the type of operation (for example, delete of a row), and details of the change, and a pointer to the previous undo record for this transaction. Since each undo record has a pointer to the previous one, they form a singly linked list (for each transaction). The details of the change are operation specific. For a row update, the details are: the row number in the block, the column number(s) changed, and the old data.

Again, remember, at this level, operations are not SQL statements, and tables and columns are not referred to by name.

Here is a sampling of undo operation types:
—insert a table row,
—delete a table row,
—update columns in a table row,
—insert index leaf node,
—delete index leaf node,
—initialize index root block after split,
—allocate new blocks,
—and so forth.

3.3. Rollback

When a transaction rolls back, the operations in the transaction's undo records are applied to reverse the effects of the transaction on the table data, indexes, etc. This is done by following the pointer in the transaction table to the last undo record written, and applying its operation, and then following its pointer to the previous undo record, applying its operation, and so on until the end of the transaction's undo list. Then, the transaction table slot for the transaction is marked as rolled-back.

The redo log (discussed later) is written to disk.

Locks are released, and enqueued transactions for those locks can retry.

3.4. Commit

When a transaction commits, the instance SCN in the SGA is incremented, the transaction table slot for the transaction has this *commit SCN* is recorded, and the transaction's slot is marked as committed.

The redo log (discussed later) is written to disk.

The ITL entry for this transaction in blocks where this transaction held a lock are marked as committed and with the commit SCN. (Oracle only does this if it can be done quickly. If this can't be done quickly, Oracle will skip it, leaving the "clean out" of blocks' lock bytes and ITLs to whenever the blocks are read next. We'll ignore delayed block clean out for the rest of this discussion.)

Locks are released, and enqueued transactions for those locks can retry.

3.5. Consistent Read

Before a transaction uses a table, index, or LOB block in memory (in the database buffer cache), the ITL in the block header is examined. The need for a read-consistent copy of the block is determined by the following condition: The are transactions in the ITL that are not the current transaction; *and* they either 1) are active (not committed), *or* 2) have a commit SCN is greater than the snapshot SCN of the current transaction.

If this check indicates a need, then a read-consistent copy of the block is constructed as follows:

(1) Clone the current block—make a copy of it in the database buffer cache.
(2) (A delayed block clean out would happen here, if needed.)
(3) Undo uncommitted transactions (other than the current transaction) that affect this block, i.e. all active transactions found in the block's ITL. This is similar to the process as used for rollback (described above), but only as applicable to this block, and without any globally visible effects, such as log write activity or lock releases.
(4) While the highest commit SCN in the ITL is greater than the current transaction's snapshot SCN, repeat the following: Pick the transaction with highest commit SCN and apply its undo to the block. It may be possible that the current transaction's snapshot SCN is too far back in history—that there is not enough undo data to go back that far. In this case, the operation is aborted with the error "`ORA-01555: snapshot too old`".

Note that applying undo records not only undoes the table row changes, but also undoes changes to the ITL, so after a transaction is undone, the block's ITL is guaranteed to show the previous transaction's entries in the ITL. This allows chaining backward in time through all the various transactions' changes to this block.

This "consistent read" copy of the block is kept in the database buffer cache, under the same address (DBA) as the current block, but marked as a consistent read copy at the snapshot SCN.

3.6. Update

When a SQL `UPDATE` statement executes, rows of the target table are searched according to the `WHERE` clause of the statement. This search is performed on read-consistent copies of the blocks of the table, as described above. When a row to be

updated is identified, the *current* copy of the block has the update applied to it. Note that there may be conflicts between the state of the row in the read-consistent copy and the current state of the row. These conflicts may cause the update to block and/or fail.

A change to a block's current copy invalidates any private read-consistent copies of that block.[3] Future uses of that block may require reconstruction of the private read-consistent copy.

3.7. Transaction Isolation Levels

Oracle supports two transaction isolation levels, *read committed* and *serializable*. The *read committed* level prohibits dirty reads, but permits unrepeatable reads and phantoms. The *serializable* level prohibits all three phenomena.

The consistent read process is used for all queries, at either transaction isolation level. The impact of the isolation level is simply when the snapshot SCN is reset: At the read committed level, the snapshot SCN is set to the current SCN at the start of each *query*. At the serializable level, the snapshot SCN is set to the current SCN at the start of each *transaction*.

4. CACHE LAYER: DATA BLOCKS & REDO ENTRIES

The foundational layer of Oracle data management is the Oracle kernel cache layer. It manages data blocks, their movement between memory and data file, logs changes in the redo log buffer, and writes the redo log buffer to the online redo log files.

Data blocks in memory are kept in the database buffer cache, and on disk are stored in data files. Redo log entries in memory are kept in the redo log buffer, and are stored on disk in the online redo log (file).

Note that "data block" means block in a data file, not necessarily storing table row data. For example, data files have blocks that list all the blocks in a segment (segment headers) and blocks that list free space in data files. These types of blocks have no user data in them at all, but are still "data blocks" as far as the cache layer is concerned. Other block types include table row blocks, LOB data blocks, index blocks, undo segment header blocks, and undo blocks.

Every block in a data file or in the buffer cache starts with a 20-byte common data block header that indicates its type, relative data block address (DBA), the SCN when it was last changed, a *sequence number* that indicates the number of changes to this block at this SCN, some status flags, and a checksum.

Before any change can happen to a data block, redo (and usually undo) entries must be generated. Any data block modification is first reified as a *change vector*. A change vector exactly specifies an operation performed on a data block, for example: operation code `11.5` (update row part) in block number `0x01000014`, row number `0x0142`, column number `0x14`, with the new data `'test'`. In addition to all the types of operations used in undo logs (discussed above), redo logs cover operations that are not recorded the transaction-rollback-focused undo logs. Some examples include: file space allocation, backups, dirty block writes to disk, block clean outs, and checkpoints. Also, redo logs may contain added entries that are useful to administrators when managing the database ("supplemental logging").

Each change vector to be performed on a block is first stored in the redo log buffer. Each change vector indicates its type, the block it applies to, the SCN and sequence number of the changed block, in addition to the operation code and the details for that specific operation. Change vectors generated at the same time are grouped into *redo records*, which have an SCN, a subSCN (which orders redo records at the same SCN), and a timestamp.

After the change vector for an operation is stored in the redo log, only then is the block in the database buffer cache changed. The block's last changed SCN and sequence number are updated, and the block is added to the list of *dirty* blocks. A dirty block has been changed in memory, but not written back to disk.

The redo log buffer is frequently written to the online redo log file. In particular, whenever a transaction commits or rolls back, the redo log buffer is written to disk. However, dirty blocks in the database buffer cache are written far less frequently. If the Oracle instance were to abort or

[3] This is conjecture. Details of this are not documented in the sources I've reviewed.

crash, the change vectors in the redo log file can be "replayed" to recover the state of the data blocks shortly before the time of the abort.

5. PUTTING IT ALL TOGETHER

Suppose a query results in the need to read every row in a table. The data dictionary is used to map the table name to an object ID, which is used to look up the file & block number of the segment header for the table. This lists all the table row data blocks, each of which is read in to the database buffer cache. Each table data block's ITL is examined, which may trigger the need to create a consistent read copy of the block as described above. Then the block's row directory is iterated through to find each row in the block, which has the column data for the row.

Now, suppose a query updates a row. First, the row is read, as described above. Even though the data appears to the query as the consistent read versions, the update actually changes the *current* data block. A redo record is generated containing the undo and redo change vectors for the update. Then the undo block has an undo record added to it[4]. Then, the current transaction's information is added to an ITL entry in the table data block. Then, the update table row's lock byte is set to the new ITL entry. Then, finally, the row is updated.

If this change were on a column that has an index, the index would be updated in a manner very similar to the table row, including generation of undo and redo entries, and use of the index block's ITL and lock bytes.

ACKNOWLEDGMENTS

This article was developed while the author was a Graduate Teaching Assistant for Dr. P. Cannata's *Data Management* course at The University of Texas at Austin.

REFERENCES

Oracle Corporation. 2013. *Oracle Database Concepts*. Available at: http://docs.oracle.com/

Oracle Corporation. 2013. *Oracle Database Administrator's Guide*. Available at: http://docs.oracle.com/

Oracle Corporation. 2013. *Oracle Database Reference*. Available at: http://docs.oracle.com/

LEWIS, Johnathan. 2011. *Oracle Core: Essential Internals for DBAs and Developers*. Apress: New York. ISBN 978-1-4302-3954-3.

LEWIS, Johnathan. Oracle Scratchpad [blog]. Available at: http://jonathanlewis.wordpress.com/

DYKE, Julian. *Oracle Internals* [Web site]. Available at: http://www.juliandyke.com/

ADAMS, Steve. *Oracle Internals Notes* [Web site]. Available at: http://www.ixora.com.au/notes/ and http://www.ixora.com.au/q+a/

LITCHFIELD, David. *Oracle Forensics* series of papers. 2007–2010. Available at: http://www.databasesecurity.com/oracle-forensics.htm and http://www.v3rity.com/research.php

[4] Assuming the undo block was already allocated, and assuming there is space in it for this new undo record.