6. Segments and Storage Structures.

Abstract: Segments are logical storage units that fit between a tablespace and an extent in the logical storage hierarchy. A segment has one or more extents, and it belongs to a tablespace. This lesson covers in detail segments, extents, and blocks. The lesson also discusses the types of segments and the type of information stored in these segments..

Contents

Objectives:

- Describe the logical structure of segments within the database
- Describe the segment types and their uses
- List the keywords that control block space usage
- Obtain information about storage structures from the data dictionary
- Describe the purpose of undo data
- Implement Automatic Undo Management

1. Data Blocks

A *data block* is the smallest logical unit of storage in Oracle. You define the block size with the DB_BLOCK_SIZE initialization parameter when you create the database, and the block size cannot be changed. The block size is a multiple of the operating system block size and is the unit of I/O used in the database. The format of the data block is the same, whether it is used to store a table, index, or cluster. A data block consists of the following:

Common and variable header The header portion contains information about the type of block and block address. The block type can be data, index, or undo. The common block header can take 24 bytes, and the variable (transaction) header occupies (24 × INITRANS) bytes. By default, the value of INITRANS for tables is 1 and for indexes is 2.

Table directory This portion of the block has information about tables that have rows in this block. The table directory occupies 4 bytes.

Row directory Contains information (such as the row address) about the actual rows in the block. The space allocated for the row directory is not reclaimed, even if you delete all rows in the block. The space is reused when new rows are added to the block. The row directory occupies (4 × number of rows) bytes.

Row data The actual rows are stored in this area.

Free space This space is available for new rows or for extending the existing rows through updates. Deletions and updates may cause fragmentation in the block; this free space is coalesced by the Oracle Server when deemed necessary.

NOTE: The space used for the common and variable header, table directory, and row directory in a block is collectively known as the block overhead. The overhead varies, but mostly it is between 84 and 107 bytes. If more rows are inserted into the block (row directory increases) or a large INITRANS is specified (header increases), this overhead size might be greater.

1.1. Block Storage Parameters

When you create objects such as tables or indexes, you can specify the block storage options. Choosing proper values for these storage parameters can save you a lot of space and provide better performance. The storage parameters affecting the block are as follows:

PCTFREE and PCTUSED These two space management parameters control the free space available for inserts and updates on the rows in the block. You can specify these parameters when you create an object.

INITRANS and MAXTRANS These two transaction entry parameters control the number of concurrent transactions that can modify or create data in the block. You can specify these parameters when you create an object. Based on these parameters, space is reserved in the block for transaction entries.

FREELIST Each segment has one or more *free lists* that list the available blocks for future inserts. The FREELIST parameter specifies the number of desired free lists for a segment. By default, one free list is allocated for each segment.

PCTFREE and PCTUSED

Before discussing these parameters, let's consider two important aspects of storing rows in a block: row chaining and row migration. If the table row length is bigger than a block, or if the table has LONG or LOB columns, it is difficult to fit one row entirely in one block. Oracle stores such rows in more than one block. This situation is unavoidable, and storing such rows in multiple blocks is known as *row chaining*.

In some cases, the row will fit into a block with other rows, but due to an update activity, the row length increases and no free space remains available to accommodate the modified row. Oracle then moves the entire row from its original block to a new block, leaving a pointer in the original block to refer to the new block. This process is known as *row migration*.

Both row migration and row chaining affect the performance of queries, because Oracle has to read more than one block to retrieve the row. You can avoid row migration if you plan the block's free space properly using the PCTFREE and PCTUSED parameters.

PCTFREE and PCTUSED are specified in percentages of the data block.

PCTFREE specifies what percentage of the block should be allocated as free space for future updates. If the table can undergo a lot of updates and the updates increase the size of the row, set a higher value for the PCTFREE parameter, so that even if the row length increases due to an update, the rows are not moved out of the block (no row migration). Whenever a new row is added to a block, Oracle determines whether the free space will fall below the PCTFREE threshold. If it does, the block is removed from the free list, and the row is stored in another block.

PCTUSED specifies when the block can be considered for adding new rows. After the block becomes full as determined by the PCTFREE parameter, Oracle considers adding new rows to the block only when the used space falls below the percent value set by PCTUSED. When the used space in a block falls below the PCTUSED threshold, the block is added to the free list.

To understand the usage of the PCTFREE and PCTUSED parameters, consider an example. The table EMP is created with a PCTFREE value of 10 and a PCTUSED value of 40. When you insert rows into the EMP table, Oracle adds rows to a block until it is 90 percent full (including row data and overhead), leaving 10 percent of the block free for future updates. During an update operation, Oracle uses the free space available if the row length increases. Once no free space is available, Oracle moves the row out of the block and provides a pointer to the new location (row migration). If you delete rows from the table (or update the rows such that the row length decreases), more free space will be available in the block. Oracle starts inserting new rows into the block only when the used space falls below PCTUSED, which is 40 percent. Therefore, when the row data and overhead is below 40 percent of the block, new rows are inserted into the block. Such inserts will continue until the block is 90 percent full. When the block has only PCTFREE (or less) percent free space available, it is removed from the free list. The block is added back to the free list only when the used space in the block falls below PCTUSED percent.

The default value of PCTFREE is **10**, and the default for PCTUSED is **40**. The sum of PCTFREE and PCTUSED cannot be more than 100.

If the rows in a table are subject to a lot of updates, and the updates increase the row length, set a higher PCTFREE.

If the table has a large number of inserts and deletes, and the updates do not cause the row length to increase, set the PCTFREE low and set the PCTUSED high.

A high value for PCTUSED will help to reuse the space freed by deletes faster. If the table row length is larger, or if the table rows are never updated, set the PCTFREE very low so that a row can fit into one block and you fill each block.

You can specify PCTFREE when you create a *table*, an *index*, or a *cluster*, and you can specify PCTUSED while creating tables and clusters, but not indexes.

INITRANS and MAXTRANS

These transaction entry settings reserve space for transactions in the block. Base these parameters on the maximum number of transactions that can touch a block at any given point in time. INITRANS reserves space in the block header for DML transaction entries. If you do not specify INITRANS, Oracle defaults the value to 1 for table data blocks and to 2 for index blocks and cluster blocks.

When multiple transactions access the data block, space is allocated in the block header for each transaction. When no pre-allocated space is available, Oracle allocates space from the free area of the block for the transaction entry. The space allocated from the free space thus becomes part of the block overhead and is never released.

The MAXTRANS parameter limits the number of transaction entries that can concurrently use data in a data block. Therefore, you can limit the amount of free space that can be allocated for transaction entries in a data block by using MAXTRANS. The default value is operating system specific, and the maximum value you can specify is 255.

Base the values for INITRANS and MAXTRANS on the number of transactions that can simultaneously update/insert/delete the rows in a block. If the row length is large or the number of users accessing the table is low, set INITRANS to a low value. Some tables, such as an application's control tables, are accessed frequently by the users, and chances are high that more than one user can access a block simultaneously to update, insert, or delete.

If a sufficient amount of transaction entry space is not reserved, Oracle dynamically allocates transaction entry space from the free space available in the block (this is an expensive operation, and the space allocated in this way cannot be reclaimed). When you set MAXTRANS, Oracle limits the number of transaction entries in a block.

You can specify INITRANS and MAXTRANS when you create a table, an index, or a cluster. Set a higher INITRANS value for tables and indexes that are queried most often by the application, such as application control tables.

1.2. Automatic Space Management

If a segment will not contain LOBs (Large OBject types containing large blocks of unstructured data, such as Binary Large OBjects [BLOBs] or Character Large OBjects [CLOBs]), there is an alternative to using PCTUSED and FREELISTS to manage data blocks: Automatic Space Management. In short, bitmaps are used instead of free lists to manage free and used space.

The advantages are many. You no longer have to guess at optimal values for PCTUSED and FREELISTS. The space is managed more efficiently, and the performance is greatly enhanced for many INSERT statements occurring concurrently on the same segment.

Other than the restrictions on segments containing LOBs, the only other major restriction is on the tablespace that contains the segments that will be automatically managed—the tablespace has to be locally managed, and the automatic management is defined for the entire tablespace and can't be enabled for individual segments.

The following is an example of a statement that creates a tablespace with automatic segment space management:

```
CREATE TABLESPACE APPL_DATA2
    DATAFILE '/disk4/oradata/DB01/appl_data02.dbf'
    SIZE 200M
    EXTENT MANAGEMENT LOCAL UNIFORM SIZE 512K
    SEGMENT SPACE MANAGEMENT AUTO;
```

REAL WORLD SCENARIO Limitations of OEM
You are a busy Oracle DBA, and you like to use OEM for most of your day-to-day tasks. You want to make your life even easier by creating new tablespaces whose segment space is automatically managed, so you bring up OEM and browse the tablespaces. Right-clicking Tablespaces, you select Create…, and you find out that there is no option for setting segment space management! In a mild panic, you dig through your documentation and manually construct a CREATE TABLESPACE statement to create the tablespace with the desired characteristics. OEM doesn't always cover every possible option when creating database objects, so it's important to keep your command-line SQL*Plus skills sharp. In this scenario, you could set all the basic options for the creation of the tablespace and then click the Show SQL button to at least give you the basics for running the command manually. In fact, for any database operation, it's a good idea to click the Show SQL button to make sure you know what's going on behind the scenes, as well as stay on top of your SQL DDL

syntax.

2. Extents

An *extent* is a logical storage unit that is made up of contiguous data blocks. An extent is first allocated when a segment is created, and subsequent extents are allocated when all the blocks in the segment are full. Oracle can manage the extent's allocated and free information through the data dictionary or locally by using the bitmaps on data files.

You have already seen the parameters that control the size of the extents. To refresh your memory, these are as follows:

INITIAL The first extent size for a segment, allocated when the segment (object) is first created.

NEXT The second extent size for a segment.

PCTINCREASE The size by which the extents should be increased based on the previously allocated extent size. This parameter affects the third extent onward in a segment.

MINEXTENTS The minimum number of extents to be allocated when creating the segment.

MAXEXTENTS The maximum number of extents that are allowed in a segment. You can set no extent limits by specifying UNLIMITED.

When the extents are managed locally, the storage parameters do not affect the size of the extents. For locally managed tablespaces, you can either have uniform extent sizes or variable extent sizes managed completely by Oracle.

Once an object (such as a table or an index) is created, its INITIAL and MINEXTENTS values cannot be changed. Changes to NEXT and PCTINCREASE take effect when the next extent is allocated for the object—already allocated extent sizes are not changed.

TIP: The header block of each segment contains a directory of the extents in that segment.

2.1. Allocating Extents

Oracle allocates an extent when an object is first created or when all the blocks in the segment are full. For example, when you create a table, contiguous blocks specified by INITIAL are allocated for the table. If the MINEXTENTS value is more than 1, that many extents are allocated at the time of creation. Even though the table has no data, space is allocated for the table. When all the blocks allocated for the table are completely filled, Oracle allocates

another extent. The size of this extent depends on the values of the NEXT and PCTINCREASE parameters.

New extents in locally managed tablespaces are allocated by searching the data file's bitmap for the amount of contiguous free space required. Oracle looks at each file's bitmap to find contiguous free space; Oracle returns an error if none of the files have enough free space.

In dictionary-managed tablespaces, Oracle allocates extents based on the following rules:

1. If the extent requested is more than 5 data blocks, Oracle adds one more block to reduce internal fragmentation. For example, if the number of blocks requested is 24, Oracle adds one more block and searches the tablespace where the segment belongs for a free extent with 25 blocks.
2. If an exact match fails, Oracle searches the contiguous free blocks again for a free extent larger than the required value. When it finds one, Oracle allocates the entire extent for the segment if the number of blocks above the required size is less than or equal to 5 blocks. Using our example, if the free contiguous blocks found is 28 blocks, Oracle allocates 28 blocks to the segment to eliminate fragmentation. If the number of blocks above the required size is more than 5 blocks, Oracle breaks the free extent into two and allocates the required space for the segment. The rest of the contiguous blocks are added to the free list. In our example, Oracle allocates 25 blocks to the segment as an extent and 15 blocks are marked as a free extent if the free extent size is 40 blocks.
3. If step 2 fails, Oracle coalesces the free space in the tablespace and repeats step 2.
4. If step 3 fails, Oracle checks to see if the files are defined as autoextensible; if so, Oracle tries to extend the file and repeats step 2. If Oracle cannot extend the file or cannot allocate an extent even after resizing the data file to its maximum size specified, Oracle issues an error and does not allocate an extent to the segment.

Extents are normally de-allocated when you drop an object. To free up the extents allocated to a table or a cluster, use the TRUNCATE <NAME> DROP STORAGE command to remove all rows. The TRUNCATE command can be used to remove all rows from a table or cluster. The DROP STORAGE clause is the default, and it removes all the extents higher than MINEXTENTS after removing all rows. The REUSE STORAGE clause does not de-allocate extents; it just removes all the rows from the table/cluster. Rows deleted using the TRUNCATE command cannot be rolled back. Deleting rows by using DELETE does not free up the extents. You can also manually de-allocate extents by using the command ALTER [TABLE/INDEX/CLUSTER] <NAME> DEALLOCATE UNUSED.

2.2. Querying Extent Information

You can query extent information from the data dictionary by using the following views.

DBA_EXTENTS

This view lists the extents allocated in the database for all segments. It shows the size, segment name, and tablespace name where it resides.

```
SQL> select owner, segment_type, tablespace_name, file_id, bytes
  2*  from dba_extents where owner='HR'
  3   ;

OWNER      SEGMENT_TYPE    TABLESPACE_NAME       FILE_ID     BYTES
---------- ----------------- ---------------------- ---------- --------
HR         TABLE           EXAMPLE                     5      65536
HR         TABLE           EXAMPLE                     5      65536
HR         TABLE           EXAMPLE                     5      65536
HR         TABLE           EXAMPLE                     5      65536
HR         TABLE           EXAMPLE                     5      65536
HR         TABLE           EXAMPLE                     5      65536
HR         INDEX           EXAMPLE                     5      65536
HR         INDEX           EXAMPLE                     5      65536
HR         INDEX           EXAMPLE                     5      65536
```

DBA_FREE_SPACE

This view lists information about the free extents in a tablespace.

```
SQL> select tablespace_name, max(bytes) largest,
  2  min(bytes) smallest, count(*) ext_count
  3  from dba_free_space
  4* group by tablespace_name
SQL> /

TABLESPACE_NAME      LARGEST    SMALLEST   EXT_COUNT
---------------------- ---------- ---------- ----------
CWMLITE              14680064   14680064          1
DRSYS                12845056   12845056          1
```

EXAMPLE	196608	196608	1
INDX	26148864	26148864	1
OEM_REPOSITORY	36634624	36634624	1
SYSTEM	85872640	85872640	1
TOOLS	4390912	4390912	1
UNDOTBS	207880192	65536	8
USERS	26083328	26083328	1

TIP: All free space in the operating system files must be represented in either DBA_FREE_SPACE or DBA_EXTENTS.

3. Segments

A *segment* is a logical storage unit that is made up of one or more extents. Every object in the database that requires space to store data is allocated a segment. The size of the segment is the total of the size of all extents in that segment. When you create a table, an index, a cluster, or a materialized view (snapshot), a segment is allocated for the object (for partitioned tables and indexes, a segment is allocated for each partition). A segment can belong to only one tablespace, but may spread across multiple data files belonging to the tablespace.

There are many types of segments:

Table This is the most common type of segment in a database. All data in a table segment must reside in the same tablespace, unlike partitioned tables. Tables that have LOB or VARRAY columns do not store these columns in the same segment.

Table Partition To support large enterprise databases that need high levels of availability and performance, a table may be split into partitions, stored in separate tablespaces. The partitions may be accessed by a distinct key range (range partitioning), by a hashing algorithm (hash partitioning), or by both. Each part of the table that resides in a different tablespace is considered a segment.

Cluster As the name implies, a cluster segment is a single segment that is composed of one or more tables. The data is stored in key order, and all tables within the cluster have the same storage characteristics. Typically, tables stored in a cluster are joined; for example, an order table and a line-item table.

Nested Table If a table has columns that are tables themselves (nested tables), each column is stored in its own segment. Each segment may have its own storage parameters.

Index All index entries for a table index are stored in the same segment.

Index Organized Table (**IOT**) An IOT segment is essentially a table and an index combined into a single segment, stored in index order. Access to an IOT is very fast because a query accessing a particular row need only traverse one segment to find the results.

Index Partition An index partition segment is similar to a table partition segment in that the index segments are usually stored in separate tablespaces to enhance availability, performance, and scalability.

Temporary In a nutshell, *temporary segments* hold overflow information from sort operations that don't fit in memory. User-initiated sort operations are usually the result of DML operations such as CREATE INDEX, SELECT ... GROUP BY, or SELECT DISTINCT. These segments are allocated in the temporary tablespace assigned to the user that runs these statements. Since the activity from these operations causes frequent allocation and de-allocation, it is recommended that a separate tablespace be allocated just for temporary segments. Having a separate tablespace prevents fragmentation on the SYSTEM or other application tablespaces. Entries made to the temporary segment blocks are not recorded in the redo log files.

LOB For LOBs in a table that are larger than about 4 KB, space is allocated in a LOB segment, separate from the segment containing the elements of the rest of the table. The only piece of information remaining in the table for a LOB is a pointer to the segment containing the LOB itself.

Undo Transactions that change rows in a table also store information in *undo segments*, specifically, the information that would be needed to restore the row to its original state in case of a rollback or an instance failure. For automatic undo management, all user undo information must reside in an undo tablespace.

Bootstrap A special system segment that is used to initialize the data dictionary upon instance startup. It cannot be queried, needs no maintenance, and is basically transparent to all users and administrators of the database.

3.1. Segment Storage Parameters

Storage parameters for segments generally take preference over storage parameters specified at the tablespace or database level. If no segment-level storage parameters are specified, the default tablespace parameters are used; if these do not exist, the database server defaults are used.

NOTE: Changes to storage parameters at any of these three levels will only affect new extents and will not affect any extents in existing segments.

3.2. Querying Segment Information

You can obtain segment information from the data dictionary by using the following views.

DBA_SEGMENTS

This view shows the segments created in the database, their size, tablespace, type, storage parameters, and so on. Notice that the LOB segment types are listed as LOBINDEX for index and LOBSEGMENT for data.

```
SQL> select tablespace_name, segment_type, count(*)
  2     seg_cnt from dba_segments
  3     where owner != 'SYS'
  4     group by tablespace_name, segment_type;

TABLESPACE_NAME      SEGMENT_TYPE         SEG_CNT
-------------------- -------------------- -----------
CWMLITE              INDEX                     67
CWMLITE              TABLE                     28
DRSYS                INDEX                     76
DRSYS                LOBINDEX                   2
DRSYS                LOBSEGMENT                 2
DRSYS                TABLE                     43
EXAMPLE              INDEX                    132
EXAMPLE              INDEX PARTITION           84
EXAMPLE              LOBINDEX                  23
EXAMPLE              LOBSEGMENT                23
EXAMPLE              NESTED TABLE               3
EXAMPLE              TABLE                     61
EXAMPLE              TABLE PARTITION           24
SYSTEM               INDEX                    181
SYSTEM               INDEX PARTITION           17
SYSTEM               LOBINDEX                  22
```

```
SYSTEM          LOBSEGMENT          22
SYSTEM          TABLE               150
SYSTEM          TABLE PARTITION     19
TOOLS           INDEX               63
TOOLS           TABLE               29
USERS           TABLE               1
```

V$SORT_SEGMENT

This view contains information about every sort segment in a given instance. The view is updated only when the tablespace is of the TEMPORARY type. It shows the number of active users, sort segment size, extents used, extents not used, and so on.

```
SQL> select tablespace_name, extent_size, current_users,
  2  total_blocks, used_blocks, free_blocks, max_blocks
  3  from v$sort_segment;

TABLESPACE_NAME        EXTENT_SIZE      CURRENT_USERS   TOTAL_BLOCKS
       USED_BLOCKS             FREE_BLOCKS       MAX_BLOCKS
------------------------ ---------------------- -------------------- ----------------------
       -------------------- ------------------ ----------
TEMP                   8                    0                 1552
       0                       1552              1552
```

4. Managing Undo Segments

Undo segments record old values of data that were changed by a transaction. Undo segments provide read consistency and the ability to undo changes, as well as assist in crash recovery. Information in an undo segment consists of several entries called undo entries. Before updating or deleting rows, Oracle stores the row as it existed before the operation (known as the before-image data) in an undo segment. An *undo entry* consists of the before-image data along with the block ID and data file number. The undo entries that belong to a transaction are all linked together, so that the transaction can be rolled back, if necessary. The data block header is also updated with the undo segment information to identify where to find the undo information. This information provides a read-consistent view of the data at a given point in time. The changes to data in a serial transaction are stored in a single undo segment. When the transaction is complete (either by a COMMIT or by a ROLLBACK), Oracle finds a new undo segment for the session.

When a user performs an update or a delete operation, the before-image data is saved in the undo segments; then the blocks corresponding to the data are modified. For inserts, the undo entries include the ROWID of the row inserted, because to undo an insert operation, the rows inserted must be deleted. If the transaction modifies an index, the old index keys also will be stored in the undo segments. The undo segments are freed when the transaction ends, but the undo information is not destroyed immediately. The undo segments are used to provide a read-consistent view of relevant data for queries in other sessions that started before the transaction is committed.

Oracle records changes to the original data block and undo segment block in the redo log. This second recording of the undo information is important for transactions that are not yet committed or rolled back at the time of a system crash. If a system crash occurs, Oracle automatically restores the undo segment information, including the undo entries for active transactions, as part of instance or media recovery. Once the recovery is complete, Oracle performs the actual rollbacks of transactions that had been neither committed nor rolled back at the time of the system crash.

4.1. Creating Undo Segments

When you create the database, Oracle creates the SYSTEM undo segment in the SYSTEM tablespace. Every database should have an undo tablespace for non-SYSTEM undo segments, other than the SYSTEM undo segment, if the database contains more than one tablespace. Oracle uses the SYSTEM undo segment primarily for transactions involving objects in the SYSTEM tablespace. For changes involving objects in the non-SYSTEM tablespace, use undo segments in an undo tablespace.

Although multiple undo tablespaces can exist in a database, only one can be active at any given time. The currently active undo tablespace must be large enough to handle the workload for all concurrent transactions.

Two initialization parameters control the use of automatic undo management in the database: UNDO_MANAGEMENT and UNDO_TABLESPACE.

The parameter UNDO_MANAGEMENT can be set to AUTO or MANUAL and cannot be dynamically altered after the database is started.

The parameter UNDO_TABLESPACE specifies the tablespace to be used for undo segments, and unlike the UNDO_MANAGEMENT parameter, it can be changed dynamically while the instance is running. If no undo tablespace is specified at system startup, the Oracle Server will automatically create one called SYS_UNDOTBS, with a system-assigned data file name in the directory $ORACLE_HOME/dbs.

Maintaining Undo Segments

After you create the database, you can create additional undo tablespaces, as in the following example:

```
CREATE UNDO TABLESPACE SYS_UNDOTBS_NIGHT
        DATAFILE 'undo2.dbf' SIZE 15M;
```

Figure 1 shows how to create an undo tablespace using OEM.

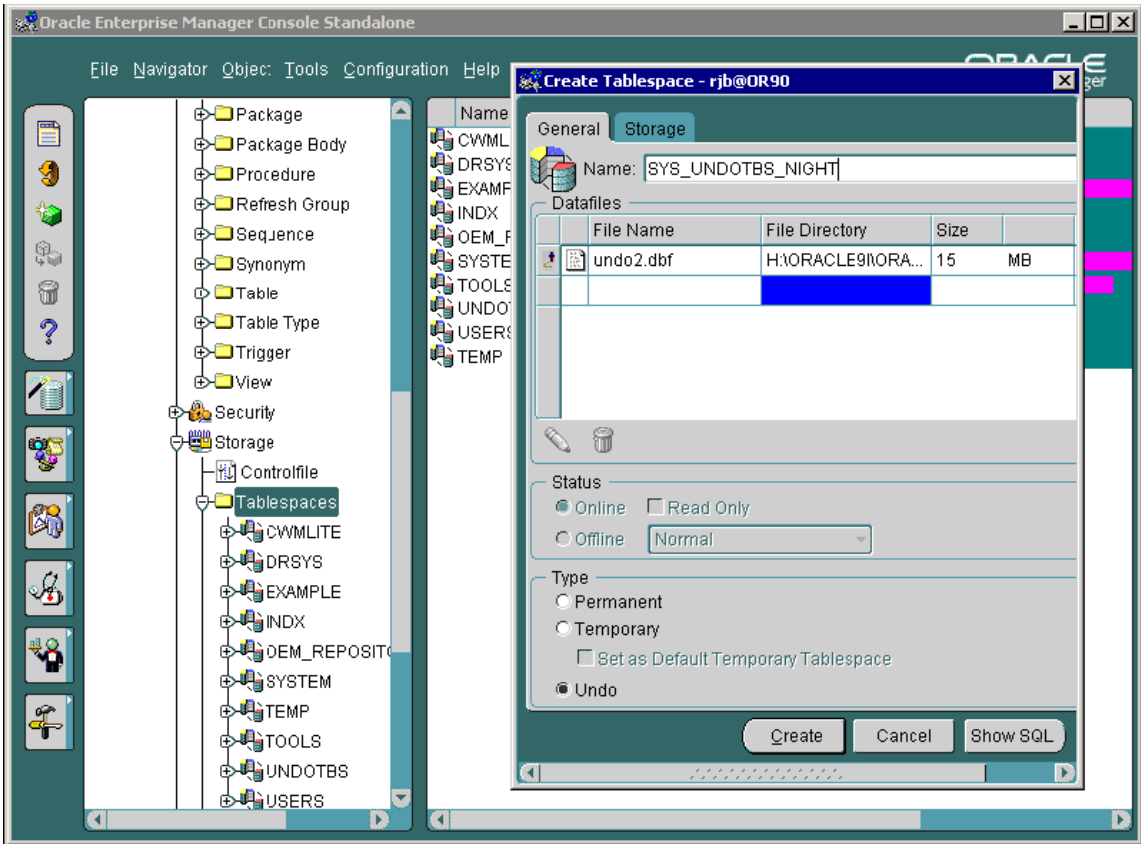

Fig. 1. Using OEM to create an undo tablespace.

Most of the other clauses that apply to regular tablespaces also apply to undo tablespaces, such as ADD DATAFILE, ONLINE/OFFLINE, BEGIN BACKUP, END BACKUP, and RENAME.

Switching undo tablespaces is also very straightforward:

```
ALTER SYSTEM SET UNDO_TABLESPACE = SYS_UNDOTBS_NIGHT;
```

An undo tablespace can be dropped like any other tablespace, but not until all transactions within the tablespace are complete. First, specify a new undo tablespace as in the previous example. To see if the tablespace has any pending transactions, run the following query:

```
SQL> select rn.name, rs.status
  2  from v$rollname rn, v$rollstat rs
  3  where rn.name in
  4  (select segment_name from dba_segments
  5  where tablespace_name = 'UNDOTBS')
  6  and rn.usn = rs.usn;

NAME                           STATUS
------------------------------ ---------------
_SYSSMU2$                      PENDING OFFLINE
_SYSSMU8$                      PENDING OFFLINE
```

If lines with a status of PENDING OFFLINE are returned from this query, the undo tablespace cannot be dropped.

An undo tablespace may need to be enlarged to support long-running queries against the database that need *consistent reads*. These queries need the original values of the rows of a table, even though another transaction may have changed and already committed rows in the same table; undo segments provide the mechanism to save the original values of the rows and provide this read-consistency. The amount of time that undo data is retained for consistent reads is controlled with the initialization parameter UNDO_RETENTION, specified in seconds.

To control system resources and prevent individual users or groups of users from using too much undo space, you can use Resource Manager to place limits on a resource group. Specify the Resource Manager parameter UNDO_POOL. The default value for this parameter is UNLIMITED. The following is an example of specifying UNDO_POOL:

```
EXEC DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE
        (PLAN => 'QPS',
        GROUP_OR_SUBPLAN => 'DirectMarketing',
        COMMENT => 'Restrict undo space usage',
        SWITCH_TIME => 3, SWITCH_ESTIMATE => TRUE,
        CPU_P1 => 60, UNDO_POOL => 450);
```

In this example, the resource group DirectMarketing under the resource plan QPS will be limited to a total of 450KB of undo information.

Snapshot Too Old Error

An ORA-1555 snapshot too old error occurs when Oracle cannot produce a read-consistent view of the data. This error usually happens when a transaction commits after a long-running query has started, and the undo information is overwritten or the undo extents are de-allocated. Here's an example. User SCOTT has updated the EMP table and has not committed the changes. The old values of the rows updated by SCOTT are written to the undo segment. When user JAKE queries the EMP table, Oracle uses the undo segment to produce a read-consistent view of the table. If JAKE initiated a long query, Oracle fetches the blocks in multiple iterations. User SCOTT can commit his transaction, and the undo segment is marked committed. If another transaction overwrites the same undo segment, JAKE's transaction will not be able to get the view of the EMP table when the transaction started.

This produces a *snapshot too old* error. You can reduce the chances of generating a Snapshot too old error by estimating the amount of undo activity during peak usage periods and adjusting the size of the undo tablespace.

4.2. Querying Undo Information

You can query the following dictionary views to obtain information about the undo segments and transactions.

DBA_ROLLBACK_SEGS

This view provides information about all undo segments (online or offline), their status, tablespace name, sizes, and so on. Note that some of the following code has been reformatted to fit on our page. The table will appear as a single, long table on your screen.

```
SQL> select segment_name, owner, tablespace_name, initial_extent ini,
  2    next_extent next, min_extents min, status stat
  3*   from dba_rollback_segs
SQL> /

SEGMENT_NAME  OWNER     TABLESPACE_NAME  INI  NEXT  MIN  STAT
------------------    ---------    ----------------------   -----  -----  ------  ------
SYSTEM        SYS       SYSTEM           53248 53248 2  ONLINE
_SYSSMU1$     PUBLIC    UNDOTBS          131072       2  ONLINE
_SYSSMU2$     PUBLIC    UNDOTBS          131072       2  ONLINE
_SYSSMU3$     PUBLIC    UNDOTBS          131072       2  ONLINE
_SYSSMU4$     PUBLIC    UNDOTBS          131072       2  OFFLINE
```

V$ROLLNAME

This view lists all online undo segments. The USN is the undo segment number, which can be used to join with the V$ROLLSTAT view.

```
SQL> select * from v$rollname;

       USN NAME
---------- ------------------------
         0 SYSTEM
         1 _SYSSMU1$
         2 _SYSSMU2$
         3 _SYSSMU3$
         4 _SYSSMU4$
```

V$ROLLSTAT

This view lists the undo statistics. You can join this view with the V$ROLLNAME view to get the undo segment name. The view shows the segment size, OPTIMAL value, number of shrinks since instance start-up, number of active transactions, extents, status, and so on.

```
SQL> select * from v$rollstat
  2  where usn = 1

USN   EXTENTS    RSSIZE     WRITES     XACTS      GETS WAITS
      OPTSIZE HWMSIZE
---   -------    -------    ------     -----      ---- -----
      -------    -------
1     8          4186112    152556     0          1008 0
      4194304    4186112
SHRINKS    WRAPS      EXTENDS    AVESHRINK AVEACTIVE STATUS
      CUREXT     CURBLK
-------    -----      -------    --------- --------- ------
      ------     ------
0          0          0          0         0         ONLINE
      3          40
```

V$UNDOSTAT

This view collects 10-minute snapshots that reflect the performance of the undo tablespace to aid in adjusting the undo tablespace size to support changing system load requirements.

```
SQL> select begin_time, end_time, undoblks, maxquerylen
  2*  from v$undostat
SQL> /

BEGIN_TIME       END_TIME         UNDOBLKS  MAXQUERYLEN
---------------- ------------ ------------- -----------
19-OCT-01 8:05:01 19-OCT-01 8:15:01         1           0
19-OCT-01 7:55:01 19-OCT-01 8:05:01         1           0
19-OCT-01 7:45:01 19-OCT-01 7:55:01         0           0
19-OCT-01 7:35:01 19-OCT-01 7:45:01         0           2
19-OCT-01 7:25:01 19-OCT-01 7:35:01         0           1
19-OCT-01 7:15:01 19-OCT-01 7:25:01        15           1
```

5. Summary

This practical lesson discussed the logical storage structures in detail. A data block is the smallest logical storage unit in Oracle. The data block overhead is the space used to store the block information and row information. The overhead consists of a common and variable header, a table directory, and a row directory. The rows are stored in the row data area, and the free space is the space available to accommodate new rows or the space available for the existing rows to expand.

The free space can be managed by two parameters: PCTFREE and PCTUSED. PCTFREE determines the amount of free space that should be maintained in a block for future row expansion due to updates. When the used space in a block reaches the PCTFREE threshold, the block is removed from the free list. The block is added back to the free list when the used space drops below PCTUSED. The INITRANS and MAXTRANS parameters specify the concurrent transactions that can access a block. INITRANS reserves transaction space for the specified number of transactions, and MAXTRANS specifies the maximum number of concurrent transactions for the block.

Extents are logical storage units consisting of contiguous blocks. Sizes of extents are specified by the INITIAL, NEXT, and PCTINCREASE parameters. The minimum value of INITIAL should be two blocks. A segment consists of one or more extents, and there are four types of segments. Data segments store the table rows. Index segments store index keys. (The data and index segments used to store LOB or VARRAY data types are known as LOB segment and

LOB index segments, respectively.) Temporary segments are used for sort operations, and undo segments are used to store undo information.

When the database is created, Oracle creates a SYSTEM undo segment. You should create an undo tablespace for the non-system undo segments. Undo tablespaces are similar to other tablespaces in that they can be modified, added, dropped, backed-up, and switched. You can control the amount of undo space used by a user or consumer group with the Resource Manager directive UNDO_POOL.

DBA_EXTENTS and DBA_SEGMENTS are views that can be queried to get information on extents and segments. Undo segment information can be queried from the DBA_ROLLBACK_SEGS, V$ROLLNAME, V$ROLLSTAT, and V$UNDOSTAT views.

References

[1] Oracle9i DBA Fundamentals I.