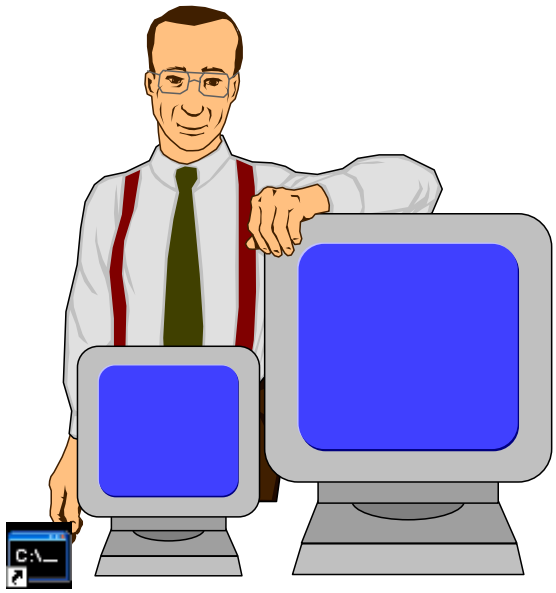
# ADMINISTRATORS RESPONSIBILITIES

- Allocating system storage and planning future storage.
- Creating primary database storage structures (Tablespaces).
- Creating primary objects (Tables, Views and indexes).
- Modifying the database structures .
- Enrolling Users and maintaining developers.
- Controlling and monitoring user access to the database.
- Monitoring and optimizing the performance.
- Backing up and restoring the database.




Command Prompt.lnk

A database Administrator

# Starting Up the Database

Load SQLDBA program

SQLDBA> CONNECT INTERNAL;

Then one can startup an instance and database in a variety of way:-

- Start the instance without mounting a database.
- Start the instance and mount the database, but leave it closed
- Start the instance, mount and open the database in
  - unrestricted mode (accessible to all users).
  - RESTRICTED mode (accessible to DBAs only).

## Starting an Instance

The process of starting an instance includes the allocation of an SGA, a shared area of memory used for database information, and creation of the background processes. Instance startup is performed in anticipation of a database being mounted by the instance. If an instance has been only started, no database is associated with these memory structures and processes.

SQLDBA> STARTUP NOMOUNT

This process will create Oracle's internal processes and memory structure, but will not locate the database files. This is needed when one needs to create a new database.

To startup an instance and mount the database:-

SQLDBA> STARTUP NOMOUNT;

SQLDBA> ALTER DATABASE MOUNT

The database need to be mounted but not opened in the following cases:-

- •Renaming data files
- •Adding, dropping or renaming redo log files.
- •Enabling and disabling redo log archiving recovery.

Starting up the instance and the database

SQLDBA> STARTUP ;

or

*SQLDBA> STARTUP NOMOUNT;*
*SQLDBA> ALTER DATABASE MOUNT;*
*SQLDBA> ALTER DATABASE OPEN.*

Instance recovery takes place at this stage. After the procedure is over, the Oracle database will be available to any valid user to connect to database and perform normal data access.

Starting up the instance and the database in the RESTRICTED mode:-

*SQLDBA> STARTUP RESTRICT;*

The restricted mode is available to carry out administrative tasks like:

- Performing structure maintenance, such as rebuilding indexes.
- Perform Export and Import of the database data.
- Temporarily preventing typical users from using data

The users who can connect to the database while it is operating in the restricted mode are those who are given the RESTRICTED SESSION system privilege (To be Introduced later).

One can also place the instance in the restricted mode after initially starting it up in normal mode as follows:-

*SQLDBA> ALTER SYSTEM ENABLE RESTRICTED SESSION;*

*Note*: At this point, all previously connected users will remian connected even if they do not have RESTRICTED SESSION privilege. However, no new connections are permitted for such users

Then back into normal mode by :

*SQLDBA> ALTER SYSTEM DISABLE RESTRICTED SESSION;*

Whenever the instance is started Oracle reads a parameter file called INIT.ORA or INITA.ora (where 'A' is SID). This parameter file is located in the ('dbs') directory of ORACLE home directory. If you change any parameter in the INIT.ORA, the effect of this parameter will NOT take effect immediatley, rather, it will take effect after the next startup operation. Therefore, we normally shutdown and startup the database after modifying the INIT.ORA file.

# Shutting Down The Database

*SQLDBA> CONNECT INTERNAL;*
*SQLDBA> SHUTDOWN;*

Normal database shutdown involves the following

New connections are not allowed.

ORACLE waits for all users to log out.

The next startup will not require instance recovery.

*Note*: if you issue the shutdown command and you feel that this command hangs, it most probably means that ORACLE is waiting for users to log out.

## Immediate Shutdown

*SQLDBA> SHUTDOWN IMMEDIATE;*

One may need the shutdown process to be effective immediately, for example, if power failure is going to occur. In This case the shutdown will proceed as follows:-

•All current SQL statement are terminated immediately.

•Any uncommitted transactions are rolled back.

•All active users are disconnected with their transactions rolled back.

•The next startup might require the automatic instance recovery.

## Aborting the Instance

*SQLDBA> SHUTDOWN ABORT;*

This is a critical shutdown option and should not be used unless all other types of shutdown failed. This type of shutdown will be carried out as follows:-

•Current SQL statements are terminated.
•Uncommitted transactions are NOT rolled back;
•All users are disconnected.
•The next startup will need the automatic recovery.

Alternatively one can shutdown as follows;

*SQLDBA> CONNECT INTERNAL;*

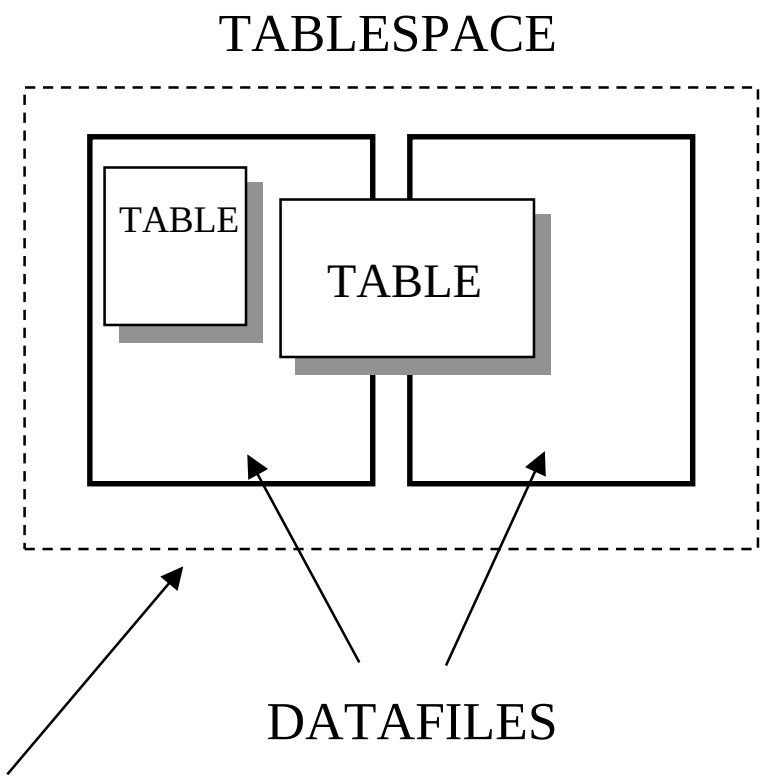
*SQLDBA> ALTER DATABASE CLOSE;*

*SQLDBA>ALTER DATABASE DISMOUNT;*

*SQLDBA>SHUTDOWN ABORT;*

# Tablespaces

Each **TABLESPACE** in an ORACLE database is comprised of one or more operating system files called data files. A tablespace's data files physically store the associated database data on disk. Database's data is collectively stored in the data files that constitute each tablespace of the database. For example, the simplest ORACLE database would have one tablespace, with one data file. A more complicated database might have three tablespaces, each comprised of two data files (for a total of six data files).




*Logical tablespace structure composed of two physical data files*

Therefore a database is divided into one or more of these logical storage units (called tablespaces). A database administrator can use tablespaces to do the following:

- Control disk space allocation for database data.
- Assign specific space quotas for database users.
- Control availability of data by taking individual tablespaces online or offline.

A database administrator can create new tablespaces, add and remove data files from tablespaces, set and alter default segment storage settings for segments created in a tablespace, and drop tablespaces.

## The SYSTEM Tablespace

Every ORACLE database contains a tablespace named SYSTEM, which is automatically created when the database is created. The SYSTEM tablespace always contains the data dictionary tables for the entire database as well as PL/SQL programming units such as Stored Procedures, Functions and Database Triggers.

A small database might need only the SYSTEM tablespace; however, it is recommended that at least one additional tablespace be created to store user data separate from data dictionary information. This allows you more flexibility in various database administration operations and can reduce contention among dictionary objects and schema objects for the same data files.

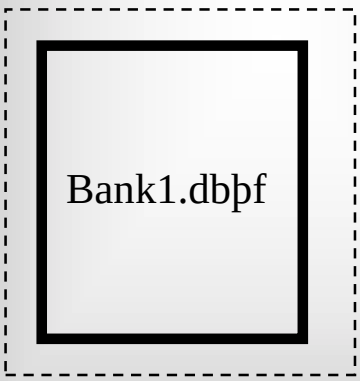
The SYSTEM tablespace must always be kept online, it cannot be taken offline or dropped (Offline concept will be introduced later).

# Allocating More Space to the Database

To enlarge the database, one could expand an existing tablespace or create a new tablespace. Let us consider creating a tablespace and then expanding it by adding datafiles.

*SQLDBA> CREATE TABLESPACE BANKING*
*DATAFILE 'BANK1.DBF' SIZE 20M;*

TABLESPACE BANKING




*BANK.DBF FILE size 20M*

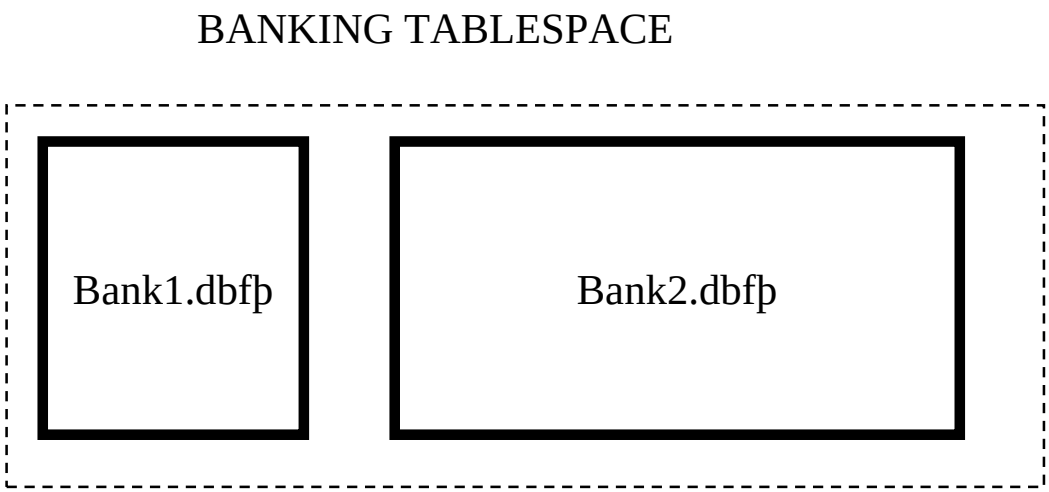
Users can use this tablespace until it is full, in which case, the system issues a system error message:-

*Cannot allocate space of size x in tablespace BANKING .*

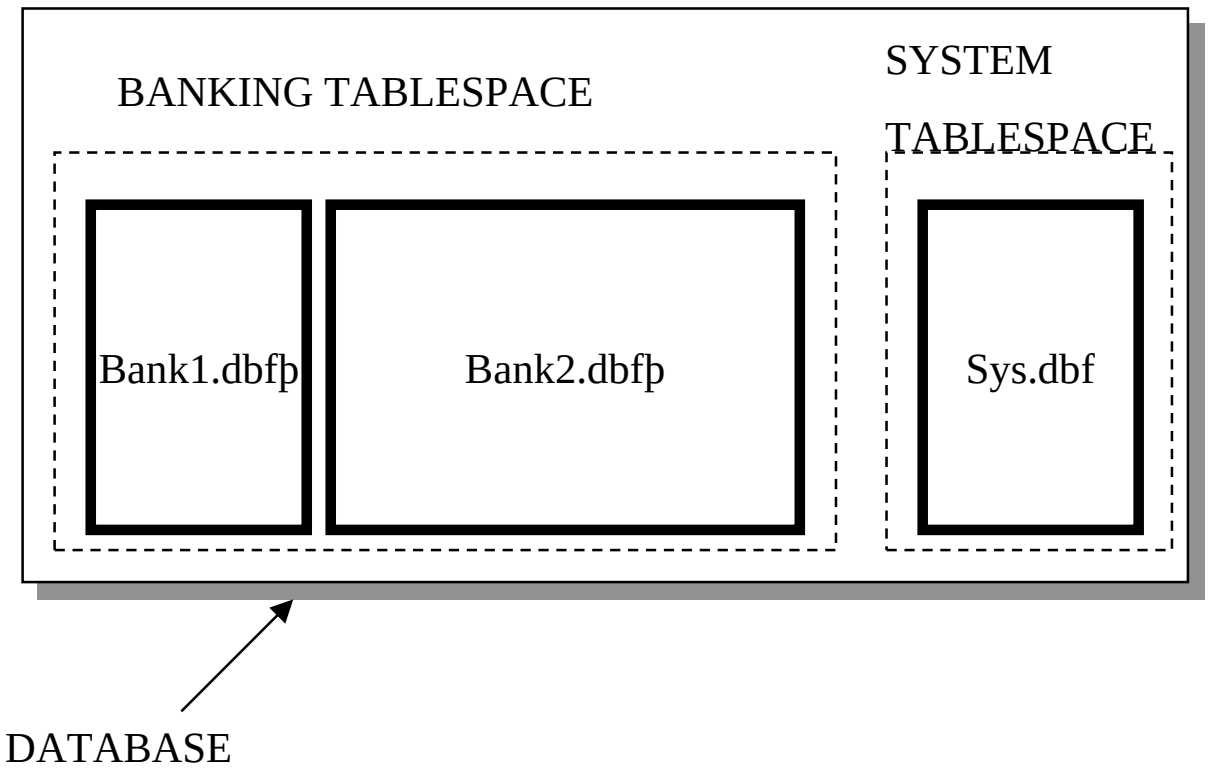
The DBA needs to extend this tablespace by adding to it another

datafile (Or using the RESIZE option to enlarge the same datafile. This option is only available in ORACLE7 release 7.2)

*SQLDBA> ALTER TABLESPACE BANKING ADD DATAFILE 'BANK2.DBF' SIZE 50M;*




Now take a look at the entire database



# Online and Offline Tablespaces

A database administrator can bring any tablespace in an ORACLE database *online* (made available) or *offline* (made unavailable) whenever the database is open. **The only exception is that the SYSTEM tablespace must always be online because the data dictionary must always be available to ORACLE**.

*SQLDBA> ALTER TABLESPACE BANKING OFFLINE*;

A tablespace is normally online so that the data contained within it is available to database users.

However, the database administrator might wish to take a tablespace offline for any of the following reasons:

- To make a portions of the database unavailable, while allowing normal access to the remainder of the database.
- To perform an offline tablespace backup (although a tablespace can be backed up while online and in use) .
- To make an application and its group of tables temporarily unavailable while updating or maintaining the application.

When a tablespace is offline, ORACLE does not permit any subsequent SQL statements to reference objects contained in the tablespace.

To drop a tablespace all together use

*SQL>DROP TABLESPACE BANKING including contents;*

You are not advised to use the *including contents* option. Rather, it is recommended that you drop all the objects in the database yourself. This way you make sure that you do not drop an object that you need. It is also recommended that you set the tablespace *offline* before dropping it.

*SQLDBA> ALTER TABLESPACE BANKING OFFLINE;*
*SQLDBA>DROP TABLESPACE BANKING;*

Now, you could recreate the tablespace again to ensure a storage space free of fragmentation

*SQLDBA> CREATE TABLESPACE BANKING DATAFILE*
*'BANK1.DBF' REUSE;*

The *REUSE* is used here because the database file does not get deleted from the file system when the tablespace associated with it is dropped. It is recommended to use the *REUSE* option because it helps to keep your file system structure less fragmented. This is because if one keeps creating and deleting files from the file system, the file system itself becomes very fragmented. To ensure that the database files associated with your tablespace are not highly fragmented, it is recommended that these files are created after formatting the hard disk; this ensures that the datafiles are contiguous. Then one could keep reusing them instead of deleting and recreating them.

## New Option

In Release 7.2 it is now possible to automatically extend the datafile of a tablespace when needed. This is done when creating or adding a datafile to the tablespace

*SQL>CREATE TABLESPACE tabspace_3 DATAFILE*
*'/u/oracle/dbs/new_data.dbf ' SIZE 500K REUSE*
***AUTOEXTEND ON** NEXT 500K MAXIMUM 10M;*

# EXTENTS

The basic building unit of any segment in the database is the *extent.* When space is allocated for data in a tablespace, it is allocated in *extents*. Each extent is, however, made out of *BLOCKS*. Oracle block size is typically 2 KB depending on the operating system being used. An important restriction of an extent is that it should be comprised of continuos (contiguous) blocks and not fragmented blocks.

When the segment is first created an ***INITIAL*** extent is allocated. The default value of the initial extent is 5 Blocks, or 10K for a 2KB block size. When this allocated extent becomes full, a ***NEXT*** Extent is allocated. The default value of the next extent is also 5 blocks. When this next extent becomes full, another extent is allocated, but this time the size of this new extent is equal to the size of the last extent plus a percentage of this extent. The value of this percentage is given by the parameter ***PCTINCREASE.*** The default value of pctincrease is 50% meaning that each new extent will be 1.5 larger than the previous one. Oracle keeps allocating extents in this manner until ***MAXEXTENT*** parameter value is reach. The maxextent default value is 99.

For optimum performance, one should try to reduce the number of extents per segment as much as possible. More extents means more fragmentation for the database.

Once an extent is allocated for a segment , let us say a table segment, it remains reserved for that table until the table is dropped. Therefore, deleting records even until an extent is free does not return space to the tablespace

## Changing the storage parameters of Segments:

1) By the create statement. Example,

*SQL> CREATE TABLE table_name ( ... )*
*storage (INITIAL 1M NEXT 500K PCTINCREASE 0*
*MAXEXTENTS 20 MINEXTENTS 1 FREELISTS 2 );*

*SQL> CREATE INDEX ind1 ON emp (deptno)*
*STORAGE(INITIAL 1500K );*

Note that all parameters not specified in the create statement will take on default values.

2) By adding new default storage parameters for the tablespace.

*SQL>ALTER TABLESPACE BANKING DEFAULT*
*STORAGE (INITIAL 500K NEXT 100K PCTINCREASE 0);*

Now, all segments created in the BANKING tablespace will have the storage parameters of the tablespace unless they are assigned specific values in the Create statements as shown above.

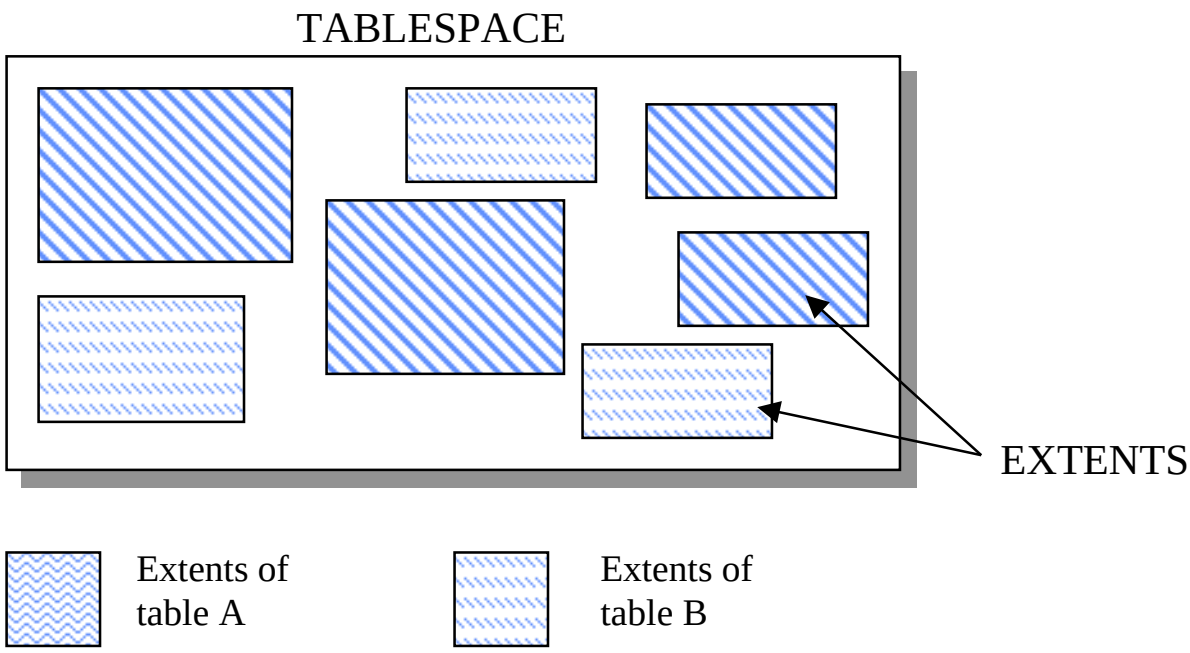
3)By using Alter Table (Index) command

*SQL> ALTER TABLE emp STORAGE (PCTINCREASE 10 MAXETNETS 40);*

You may have noticed two new parameters namely, MINEXTENTS and FREELISTS. The first parameter specifies the number of extents that ORACLE will pre-allocate when the table is created and its default value is 1. The second parameter is important when a large number of inserts have to be carried out in parallel. The freelist itself is a map of free blocks in the segment. If the segment is under heavy Insert activity it is better to duplicate this list so that other user can use it concurrently. The default is also one. Please note the following enhancement in Release 7.1

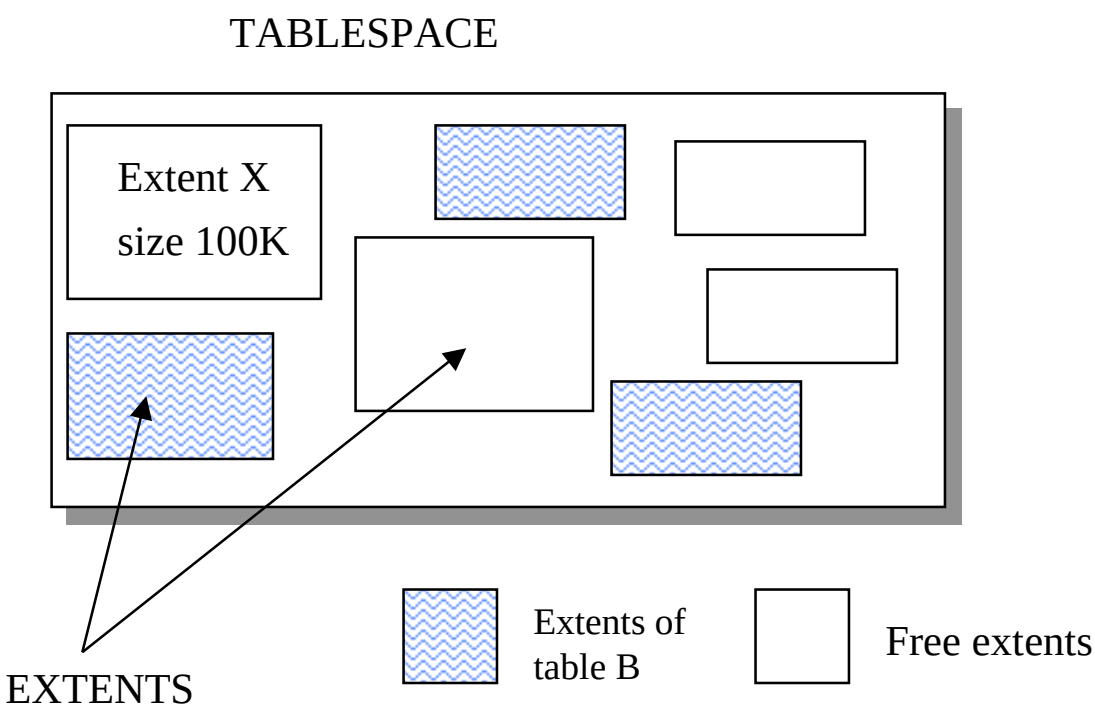
*SQL>ALTER TABLE emp ALLOCATE EXTENT SIZE (200K).*
This command will let you add a new extent to your table at any time.

If tables ,indexes and other segments are not assigned the proper storage parameters, one could end up with excessive fragmentation. One should keep fragmentations as low as possible to achieve more efficient storage utilization and better performance. Please take a look below at a tablespace with high rate of fragmentation.




Please note that table A consists of 4 separate extents, and table B consists of 3 separate extents; theses extents are scattered across the tablespace leaving non uniform gaps of free space. Note that even though it looks like at least 25% of the tablespace is still free, but it is hardly usable. This so because an extent should be , by definition, made up of contiguous space. In the figure shown above, one could only allocate extents of small sizes, leading to a high degree of space wastage. Additionally, the larger the number of extents in a tablespace, the worse the performance for that tablespace.

Dropping table A for example will not solve the problem, this so because dropping the table will not change the structure of the already existing extents. What it will do is that it will leave the extents free for future usage when extents of its size are required. ORACLE Version 7 added enhancement for extent defragmentation where it is now possible for adjacent free extents to be combined into one larger extent.




Looking at the figure above, if a user requires an extent of size 150K, the tablespace will fail to provide such a contiguous amount, even though the total space available is far more that 150K. This user will receive a message form ORACLE RDBMS informing him that the system failed to allocate extent of size 150K in the tablespace. This problem can be solved by enlarging the tablespace, or better yet, by backing up all data and recreate the tablespace after dropping it.

The tablespace can be brought online:-

*SQLDBA > ALTER TABLESPACE BANKING ONLINE;*

NOTE:

After a data file is initially created, the allocated disk space does not contain any data; however, the space is *reserved* to hold only the data for future segments of the associated tablespace. As a segment (such as the data segment for a table) is created and grows in a tablespace, ORACLE uses the free space in the data files to allocate space for the segment.

The data in the segments of objects (data segments, index segments, rollback segments, and so on) in a tablespace are physically stored in one or more of the data files that constitute the tablespace. Also Note, that an object can "span" one or more data files.

## DROPPING TABLESPACES

If your tablespace becomes much too fragmented you may consider dropping the tablespace and recreating it.

The command needed to drop the tablespace is :-

*SQLDBA>DROP TABLESPACE BANKING* ***INCLUDING CONTENTS [Cascade Constraints];***

**Note:** Be careful with including contents option as it might drop all database objects. It is better that you drop each object manually

in this way you know what you are dropping and you avoid dropping other objects that might be residing on this tablespace without you knowing about them. Also note that the CASCADE CONSTRAINT is important to drop any Referential Constraint that might exist between particular tables; because if these constraints remian enforced then tables that contain foreign keys will not be dropped if their master is not dropped first.

## REALLOCATING OR RENAMING TABLESPACES

You can rename a data file or even move it over to a different directory or a different hard disk:

*SQL>ALTER TABLESPACE RENAME DATAFILE '/u/oracle/dbs/bank1.dbf' TO '/usr/oracle1/data/bank1.dbf';*

Note that this command does not create new files. the new file must already exist in the specified directory. Therefore, it is your responsibility to move the file to the new destination before issuing the above command. Also make sure that you specify full file name complete with the path for both the new and the old file.

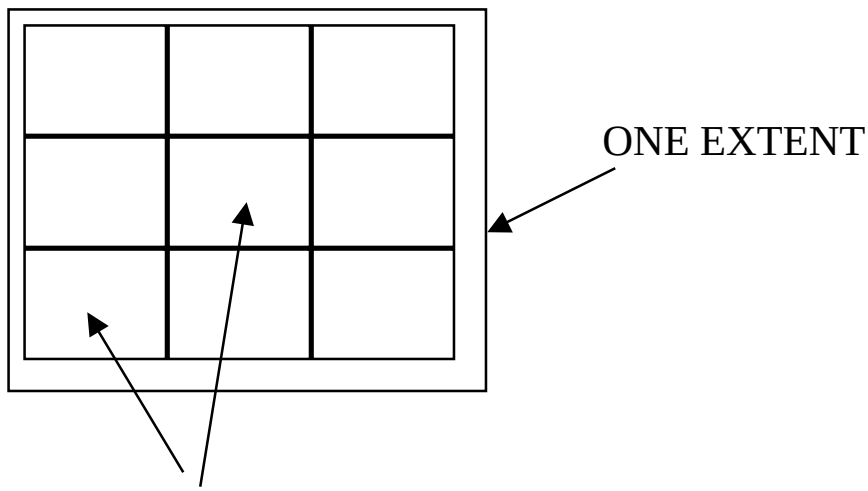
## AUTOMATIC DATAFILE EXTENSION

A new feature in Release 7.2 allows you to have an automatic extention of a tablespace when its datafile is full

*SQL> ALTER TABLESPACE BANKING ADD DATAFILE 'x1.dbf' SIZE 10M AUTOEXTEND ON NEXT 5M;*

# Data Blocks

Data Blocks are the atomic structure of extents. A collection of data blocks make up an extent. The extent size is usually indicated in data blocks unit. The data block size is defined by a parameter called *db_block_size*. This parameter is part of ORACLE initialization file (INIT.ORA). Remember that once the database is created, the db_block_size is read from INIT.ORA. After the database is built, you cannot change the block size. On most systems the ORACLE data block size is 2K.



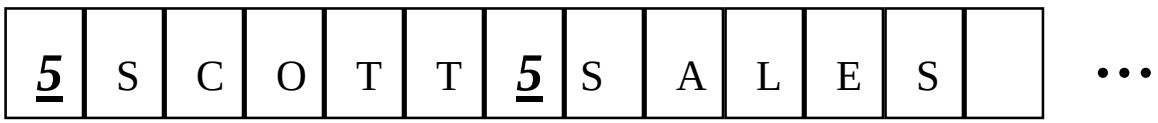

Data RECORDS are stored in data blocks. One block can accommodate one or more table rows depending on the record size. To achieve maximum performance it is recommended that records do not span more that one block.

Record that span more than one block are call *CHAINED RECORDS.* One of the Database Administrators (DBA) jobs is to make sure that the number of chained records is minimized.

The DBA can use the *ANALYZE* Command to scan Tables looking for chained records.

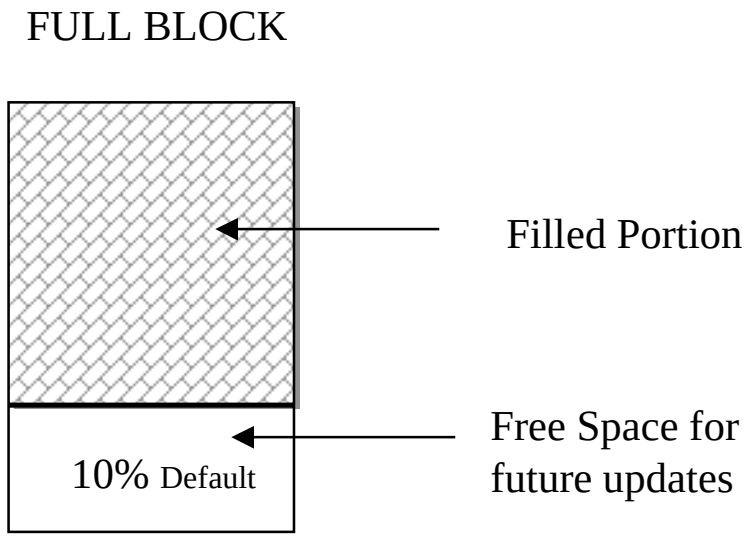
Records are made up of a collection of fields. When the fields have a variable length nature, each field is preceded by a number indicating its length. This ensures a fast way of locating the next field.

EXAMPLE:- Assume that a row contains the field SCOTT and SALES; each field is 5 characters long. The following figure shows how theses fields are stored



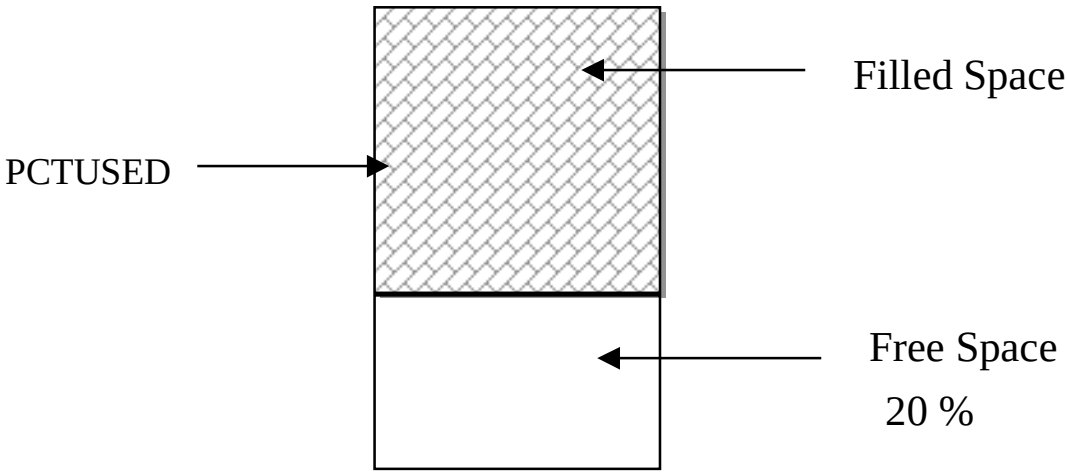

What happens when SCOTT is updated to MILLER, which is 6 characters long? Of course, if it directly replaces SCOTT, it will overwrite the length byte of the next field. If you think that shifting SALES (and whatever comes after it) solves the problem, you are right. However this is a costly operation in term of CPU time. What ORACLE RDBMS does to cater for this problem is that during normal data insertion it leaves a specific percentage of the block space free to be used for future updates; this is called *PCTFREE*. In case of an update that changes the size of a field, the RDBMS engine would move the record to an empty space in the same block where it would be represented correctly with only the fields of this record shifted rather than shifting the entire block and blocks following it. Consequently, it is highly recommended that

the DBA assigns enough free space for tables that are likely to be updated heavily. The default *PCTFREE* is 10%



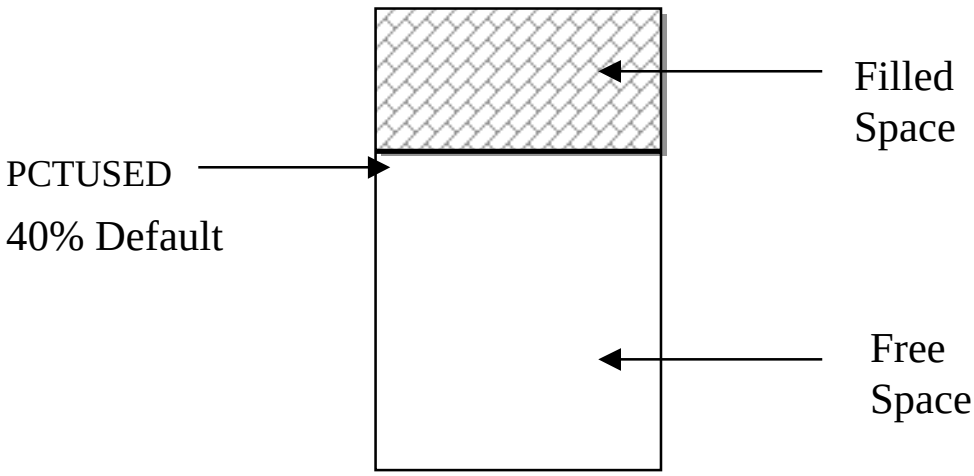

When a block is marked to be full, The RDBMS engine will not consider this blocks for future inserts of records belonging to the same table. Assume that records are deleted from this block and the free space becomes higher than the *PCTFREE* Consider for example that now only 80% of the block is used and 20% is free. The RDBMS will still consider this block as full until the used percentage of the block reaches a level indicated by the parameter *PCTUSED*. Even though this configuration might waste space, but it is more efficient in terms of Performance. If the RDBMS is going to change that status of each block every time one record is added or deleted in one block, so much of the CPU time will be wasted in managing these block. Imagine what happens when you have 500 Users making hundreds of updates, deletes and inserts per second.

FULL BLOCK




Note that in the figure above, the block is still labeled FULL, because the used space did not drop to the PCTUSED Level

FREE BLOCK




Now the block is labeled free and ready for new inserts

## **Assigning *PCTUSED* and *PCTFREE (For the Blocks)***

Those two parameters are assigned to The Object (Tables or Indexes) at the create time of theses objects

```
SQL> CREATE TABLE SUDATEL ( ... )
                STORAGE ( INITIAL ...          )
                PCTFREE  5     PCTUSED 60;

SQL> CREATE INDEX IND1 ON SUDATEL (..)
            STORAGE (INITIAL  NEXT    ....          )
            PCTFREE 5    PCTUSED 50;

SQL> ALTER TABLE TOT_PROD PCTFREE 10
```

When altering the data block space usage parameters (PCTFREE and PCTUSED) of a table, note that new settings apply to all data blocks used by the table, including blocks already allocated and subsequently allocated for the table. However, the blocks already allocated for the table are not immediately reorganized when space usage parameters are altered, but as necessary after the change.

To summarize, you need to estimate your PCTFREE and PCTUSED carefully, If your system undergoes high rate of update you must choose a high PCTFREE. If you also choose your PCTUSED high, you might gain space, but you will lose in terms of performance. If you think carefully, you will realize that logically speaking PCTFREE+PCTUSED should not exceed 99.

## Assigning Default tablespaces

When a user creates a segment, let us say a table, it is allocated in the default tablespace. When a user is created, he is normally assigned a default tablespace. Examine the following command.

*SQLDBA> CREATE USER team7 IDENTIFIED BY Jedco QUOTA UNLIMITED ON BANKING DEFAULT TABLESPACE BANKING;*

QUOTA UNLIMITED means that this user is granted unlimited access on this tablespace. This is the way to give access privilege to tablespaces. If the DEFAULT TABLESPACE BANKING clause is missing from the above statement, then the default tablespace of that user is SYSTEM tablespace.

Suppose that the user would like to create a table on a tablespace other that his default tablespace. In this case, the create statement of the table should look like the following:

*SQL> CREATE TABLE tablename ( ... ) TABLESPACE XYZ;*

provided that the owner of the table has access privilege on XYZ

## Physical Format of a RECORD

The last unit in our study of the physical structure is the database record itself. This is made up of two components; the first component is the Row Header and the second component is the Row Body

The **row header** contains the following information

-The number of columns in the row (1 Byte).

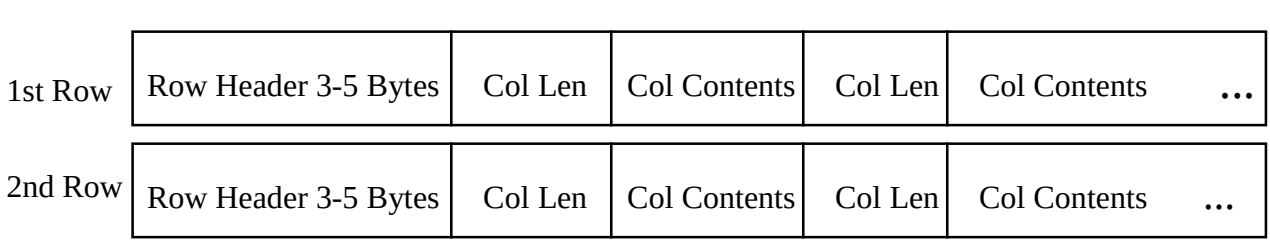
-Chaining information, where the row was split among two blocks or moved to another block because the free area in the original block is too small or the row itself is larger than the block and connot be accomodated in one block.

-Cluster key information, where the row belongs to a clustered table.

The row header accupies at least 3 bytes and at most 5 bytes (for clustered or chained row). Remember that this information (row header) exists for each row in the block.

The **row body** contains the actual data as defined by the CREATE TABLE command. Each physical column is preceded by its own column header which is used to record the exact length of the column. The following table will show the header length for various Data Types

| Data Type | Header Column Length |
|---|---|
| Number | 1 |
| Char | 1 |
| Date | 1 |
| Varchar2 | 3 |
| Long | 3 |
| Long raw | 3 |
| Raw | 3 |



# Calculating the Capacity of each Block

It is important to know how many rows of a given table can be stored in a datablock and how many blocks must be made available for that table.

First of all we must calculate the amount of space that ORACLE reserves for its internal use in each blocks.

1- PCTFREE is left unused for future updates of the data in the block.
2- 57 Bytes are reserved as general block information in the block header
3- 23 bytes are used for each transaction entry. It assumed that we are using one transaction entry per block (INITRANS=1). This parameter can be specified by the user during the CREATE TABLE statement. much like PCTFREE.
4- 2 bytes per record in the block's row directory.
5- 4 bytes for table directory in a non-clustered table.

Therefore, the number of bytes reserved for block header is equal to84 bytes + 2 times the actual number of records in that block. Please remeber that this calculation is based on the fact that the value of INITRANS=1

The space available for data entry is now equal to the Block size (Typically 2048 bytes) - block header(84+2*no_of_rec) - Space reserved by PCTFREE (Default = 10%).

Available storage area=(2048-(84*x)) * (100-pctfree)/100 , where x is the number of records

We have already seen that the physical record size is equal to :

Row Header size + Column len header + actual data length in all columns.

The total number of records that one block can take is

X = available storage area/ physical record size

substituting from previous page

$$X = \frac{(2048-84)*(100-(pctfree/100))}{data+col\ headers+record\ header+2*(100-(pctfree/100))}$$

Once you know the number of records needed per block, you can multiply by the total number of records expected for your table and then calculate your initial extent. If a sample of data already exists, you can estimate the actual size of a variable length field by using the built in SQL function VSIZE, example

*SQL> SELECT AVG(VSIZE(ENAME)) FROM DEPT;*

As we have seen ORACLE dynamically allocates additional extents for the data segment of a table, as required. However, you might want to allocate an additional extent for a table explicitly. To do that execute the statement (Available starting 7.1)

SQL> *ALTER TABLE EMP ALLOCATE EXTENT (SIZE 30K)*

*We have seen this before.*

# Creating the Database

*SQLDBA> CONNECT INTERNAL;*
*SQLDBA> STARTUP NOMOUNT;*
*SQLDBA>CREATE DATABASE CONTROL FILE*
*REUSE LOGFILE '?/dbs/log1.dbf' SIZE 500K,*
*'?/dbs/log2.dbf' SIZE 500K*
*DATAFILE '?/dbs/sysA.dbf' SIZE 20M*
*MAXDATAFILE 50*
*CHARACTERSET "AR8ISO8859P6";*

This command will take a few minutes, after which the database will be created and will consist of one datafile for SYSTEM tablespace, two logfiles and one control file. No database can exist without at least these components. You can have more than two log files if you wish. The log file is very important for the correct operation of the database. If any one of the two log files is corrupted or lost, the database will be useless. ORACLE understands the danger of losing such files, and therefore provided a means to replicate or mirror these files. You can, for example keep a replicated mirror image of each log file in a separate disk to protect your log files. The collection of the log files and their replica is called GROUP. Therefore if you want use the replication feature of log files, you must have at least two GROUPS. To create groups for a new database, use the following syntax:

*SQLDBA> CREATE DATABASE CONTROLFILE REUSE*
*LOGFILE GROUP 1 (‘C:\ LOG1A.LOG’,’D:\LOG1B.LOG’)*
*SIZE 500k,*
*GROUP 2 (‘C:\ LOG2A.LOG’,’D:\LOG2B.LOG’)*

....

If one member of a group fails, the other member can provide the information needed to ensure the continuos operation of your database.

## ROLLBACK SEGMENT

When the database is created successfully, the system tablespace is automatically created. This tablespace contains all data dictionary tables and views. In addition it contains a ***ROLLBACK SEGMENT***. The rollback segment is used by ORACLE to save your data before it is modified to enable you to cancel the changes you are making. When a transaction is committed, the rollback segment is released(because its data is no longer needed). When a transaction is rolled back, the changes are undone by writing the rollback entries back to their original place, thus replacing the modified data.

The initial rollback segment is created automatically by database installation. However, the database administrator can add more rollback segments if he wishes. The rollback segment is physically treated as any other database object (e.g. table). It allocates extents and extend the extents until it reaches the maximum extents.

The rollback segment can be created by using the following syntax.

*SQLDBA> CREATE ROLLBACK SEGMENT my_roll*
*STORAGE (INITIAL 200K NEXT 500K MAXEXTENTS 30)*
*TABLESPACE roll_tablespace;*

**Please note the following points:-**

•The rollback segment can take storage parameters.

•The rollback segment can be assigned to specific tablespace

•If storage parameter are not specified, it uses the default values of the tablespace.

•The rollback segment storage parameter does NOT have PCTINCREASE parameter and it is assumed to be 0 (Only in V7).

•The rollback segment will have a minimum of two extents.

•The rollback segment can be OFFLINE or ONLINE.

•When the rollback segment is created it is ONLINE by default.

•IF no tablespace is specified, the segment will be placed in the SYSTEM tablespace. (This not recommended).

•Private rollback segments must be specified in INIT.ORA before they become available for general use:

rollback_segments=(my_roll,...) -- in INIT.ORA

The rollback segment size is constrained (Like any other segment) by the size of the tablespace it belongs to or by its maximum extents parameter. The rollback segment must be able to hold the largest transaction that your application requires. Note that if all users are using the same rollback segment, it should be able to handle all of their transactions together. In fact, the rollback segment can rapidly grow..

If you have a large number of users in your environment you should consider creating many rollback segments to reduce contention on the rollback segment.

When a transaction is ended, the extents holding its rollback data are released for use by other transactions. When the number of active transactions drops, the rollback shrinks to its ***optimal*** size. This optimal size a new parameter introduced in ORACLE 7. It is specified in the storage parameter of the rollback segment Only (Not valid for other segments like tables)

*SQL>ALTER ROLLBACK my_roll*
*STORAGE (OPTIMAL 100K).*

**Note:** If you do not specify a value for OPTIMAL, it defaults to NULL (Rollback extents are never deallocated).

Let us see who the OPTIMAL size works:-

1. A transaction continues writing rollback information from one extent to the next.

2. When an extent is filled and the transaction is not over, then ORACLE must continue writing to the next extent, ORACLE compares the current size of the rollback segment to its optimal size.

3. If the rollback segment is larger that the optimal size and the next extent is inactive:

a) ORACLE deallocates inactive extents until the total size of allocated extents approx... equals the optimal size.
b) ORACLE releases the oldest extents first.

The rollback segment can exists in two states, ONLINE and OFFLINE. When the rollback is ONLINE then it can receive transactions immediately. When it is OFFLINE,it prevents transaction from using its extents.

To find out whether your rollback segment is ONLINE or OFFLINE, you can use the data dictionary view DBA_ROLLBACK_SEGS, and examine STATUS field:

| STATUS | |
|---|---|
| IN_USE | For ONLINE rollback segment |
| AVAILABLE | For OFFLINE rollback segment. |

The command needed to set rollback segment ONLINE or OFFLINE is:-

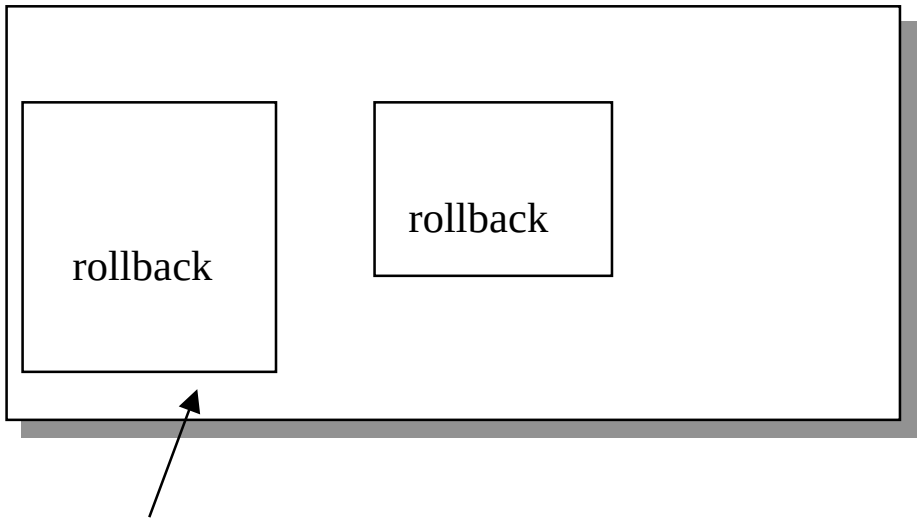
*SQLDBA> ALTER ROLLBACK SEGMENT my_roll OFFLINE;*

If the rollback segment becomes very fragmented, the Database administrator might consider dropping the rollback segment and recreating it. To drop a rollback segment, it must be OFFLINE.

*SQLDBA> DROP ROLLBACK SEGMENT my_rollback;*

Note: DBA’s usually create a tablespace to contain the rollback

This way, you can manage your rollback segments and the fragmentations they cause whitout affecting other structures of the database.




TABLESPACE CONTAINING TWO ROLLBACK SEGMENTS

When your system contains a mix of short and long rollback segments, you can assign your long transactions to the larger rollback segment. You can do this by the following command:

*SQL> SET TRANSACTION USE ROLLBACK SEGMENT my_roll;*

This statement must be specified prior the execution of the first statement of a transaction, and it is terminated with COMMIT / ROLLBACK . Note that you cannot assign users to rollback, rather you assign transactions to rollback.

Please note that the user is not required to have access privilege on the rollback segment nor on the tablespace that contains the segment. The reason for this is that it is ORACLE RDBMS that uses the rollback, and it is the user choice. The last type of segment we will address is the ***TEMPORARY SEGMENT.***

## TEMPORARY SEGMENT

Some commands need a temporary space to store intermediate values while those commands are being executed. For that purpose ORACLE creates what is called by TEMPORARY SEGMENT. This segment is allocated as needed by creating INITIAL extents and NEXT extents. When the temporary values are no longer needed. ORACLE automatically drops the segment. The default location of the temporary segment is the SYSTEM tablespace. Since this segment can be created and droped several times during a users session. It is highly recommended to create a separate tablespace especially allocated for this segment only. The segment is CREATED automatically, and DROPPED automatically by ORACLE, and therefore there is no Create Statement for temporary segment. The storage parameters for this segment are, therefore, taken from the tablespace storage parameter it belongs to. Let us assume that the DBA created a tablespace to accommodate temporary segments and he called it TEMP. We can then make sure that users use the TEMP tablespace for their temporary activities by executing the following command:

SQL> *ALTER USER TEAM1 TEMPORARY TABLESPACE TEMP;*

If this command is not executed, ORACLE will utilize the SYSTEM tablespace to store temporary activities, something a DBA should try to avoid.

The following is a partial list of commands that may require temporary segments:

•SELECT with ORDER BY clause.

•SELECT with GROUP BY clause.

•INDEX creation command.

•SET command (UNION, INTERSECT, MINUS)

ORACLE's parameter file (INIT.ORA) contains a parameter that can cache temporary segments in memory. This parameter is called ***SORT_AREA_SIZE***. If you make its value large, the performance of the system will be better. A typical value for this parameter is 128K.

# DATA DICTIONARY INFORMATION

Oracle Data Dictionary stores all information about the current database. You can query the data dictionary views in order to see who your database is organized, who the extents are allocated etc..

This section will show the important data dictionary views and who one could you then.

## TABLESPACES:

To find out information about your tablespace examine the following Table

*SQL> DESC USER_TABLESPACES;*

| *Name* | *Null?* | *Type* |
|---|---|---|
| *TABLESPACE_NAME* | *NOT NULL* | *VARCHAR2(30)* |
| *INITIAL_EXTENT* | | *NUMBER* |
| *NEXT_EXTENT* | | *NUMBER* |
| *MIN_EXTENTS* | *NOT NULL* | *NUMBER* |
| *MAX_EXTENTS* | *NOT NULL* | *NUMBER* |
| *PCT_INCREASE* | *NOT NULL* | *NUMBER* |
| *STATUS* | | *VARCHAR2(9)* |

## DATAFILES:

SQL> DESC SYS.DBA_DATA_FILES

| Name | Null? | Type |
|---|---|---|
| FILE_NAME | | VARCHAR2(257) |
| FILE_ID | | NUMBER |
| TABLESPACE_NAME | | VARCHAR2(30) |
| BYTES | | NUMBER |
| BLOCKS | | NUMBER |
| STATUS | | VARCHAR2(9) |

| FILE_NAME | FILE_ID | TABLESPACE_NAME | BYTES | BLOCK | S TATUS |
|---|---|---|---|---|---|
| SYS:ORANW\ guser.ora | 2 | USER_DATA | 3145728 | 1536 | AVAILABLE |
| SYS:ORANW\\wrbs.ora AVAILABLE | 3 | ROLLBACK_DATA | 5242880 | 2560 | |
| SYS:ORANW\wtemp.ora | 4 | TEMPORARY_DATA | 2097152 | 1024 | AVAILABLE |
| SYS:ORANW\wsys.ora | 1 | SYSTEM | 10485760 | 5120 | AVAILABLE |

## ROLLBACK SEGMENTS:

*SQL> DESC SYS.DBA_ROLLBACK_SEGS*

| Name | Null? | Type |
|---|---|---|
| SEGMENT_NAME | NOT NULL | VARCHAR2(30) |
| OWNER | | VARCHAR2(6) |
| TABLESPACE_NAME | NOT NULL | VARCHAR2(30) |
| SEGMENT_ID | NOT NULL | NUMBER |
| FILE_ID | NOT NULL | NUMBER |
| BLOCK_ID | NOT NULL | NUMBER |
| INITIAL_EXTENT | | NUMBER |
| NEXT_EXTENT | | NUMBER |
| MIN_EXTENTS | | NUMBER |
| MAX_EXTENTS | | NUMBER |
| PCT_INCREASE | | NUMBER |
| STATUS | | VARCHAR2(16) |
| INSTANCE_NUM | | VARCHAR2(40) |

*SQL> SELECT SEGMENT_NAME, TABLESPACE_NAME, OWNER, STATUS,INITIAL_EXTNET FROM DBA_ROLLBACK_SEGS;*

| *SEGMENT_NAME* | *TABLESPACE_NAME* | *OWNER* | *STATUS* | *INITIAL* |
|---|---|---|---|---|
| *SYSTEM* | *SYSTEM* | *SYS* | *ONLINE* | *51200* |
| *B_TEMP* | *SYSTEM* | *PUBLIC* | *ONLINE* | *10240* |
| *RB1* | *ROLLBACK_DATA* | *PUBLIC* | *ONLINE* | *512000* |

To find out how much freespace is left in my tablespaces examine the following data dictionary table:-

SQL> DESC USER_FREE_SPACE

| Name | Null? | Type |
|---|---|---|
| TABLESPACE_NAME | NOT NULL | VARCHAR2(30) |
| FILE_ID | NOT NULL | NUMBER |
| BLOCK_ID | NOT NULL | NUMBER |
| BYTES | | NUMBER |
| BLOCKS | NOT NULL | NUMBER |

SQL> *SELECT * FROM USER_FREE_SPACE;*

| TABLESPACE_NAME | FILE_ID | BLOCK_ID | BYTES | BLOCKS |
|---|---|---|---|---|
| SYSTEM | 1 | 4580 | 1107968 | 541 |
| USER_DATA | 2 | 1517 | 40960 | 20 |
| USER_DATA | 5 | 192 | 2754560 | 1345 |
| ROLLBACK_DATA | 3 | 827 | 3551232 | 1734 |
| TEMPORARY_DATA | 4 | 2 | 2095104 | 1023 |

Note: File_id refers to the physical file that contains this tablespace. Refere to table dba_data_files to get the name of the physical file.

If you want to know the storage parameters of your tables you can examime the USER_TABLES data dictionary View:

*SQL>SELECT TABLE_NAME, PCT_USED,PCT_FREE, INITIAL_EXTENT, NEXT_EXTENT,PCT_INCREASE FROM USER_TABLES;*

| *TABLE_NAME* | *PCT_USED* | *PCT_FREE* | *INITIAL_EXTENT* | *NEXT_EXTENT* | *PCT_INCREASE* |
|---|---|---|---|---|---|
| *DEPT* | *40* | *10* | *10240* | *10240* | *50* |
| *EMP* | *40* | *10* | *10240* | *10240* | *50* |

After you have used your tables for a while, you want to know the total number of extents your tables have allocated. The USER_SEGMENTS will help you:ý

*SQL>SELECT SEGMENT_NAME, SEGMENT_TYPE, BYTES, EXTENTS,INITIAL_EXTENT FROM USER_SEGMENTS;*

| *SEGMENT_NAME* | *SEGMENT_TY* | *BYTES* | *EXTENTS* | *INITIAL_EXTENT* |
|---|---|---|---|---|
| *DEPT* | *TABLE* | *10240* | *1* | *10240* |
| *EMP* | *TABLE* | *47104* | *3* | *10240* |

If you examine the previous list you will see that EMP table has 3 extents. you can examine the properties of each extent by examining the data dictionary view USER_EXTENTS:

*SQL>SELECT * FROM USER_EXTENTS WHERE SEGMENT_NAME = 'EMP';*

| SEGMENT_NAME | Segment_ty | Tablespace_name | EXTENT_ID | BYTES | BLOCKS |
|---|---|---|---|---|---|
| EMP | TABLE | USER_DATA | 0 | 10240 | 5 |
| EMP | TABLE | USER_DATA | 1 | 6144 | 3 |
| EMP | TABLE | USER_DATA | 2 | 30720 | 15 |

For the DBA you can try DBA_SEGMENTS, DBA_EXTENTS, DBA_USERS, DBA_TABLES and examine the kind of information it can provide you with