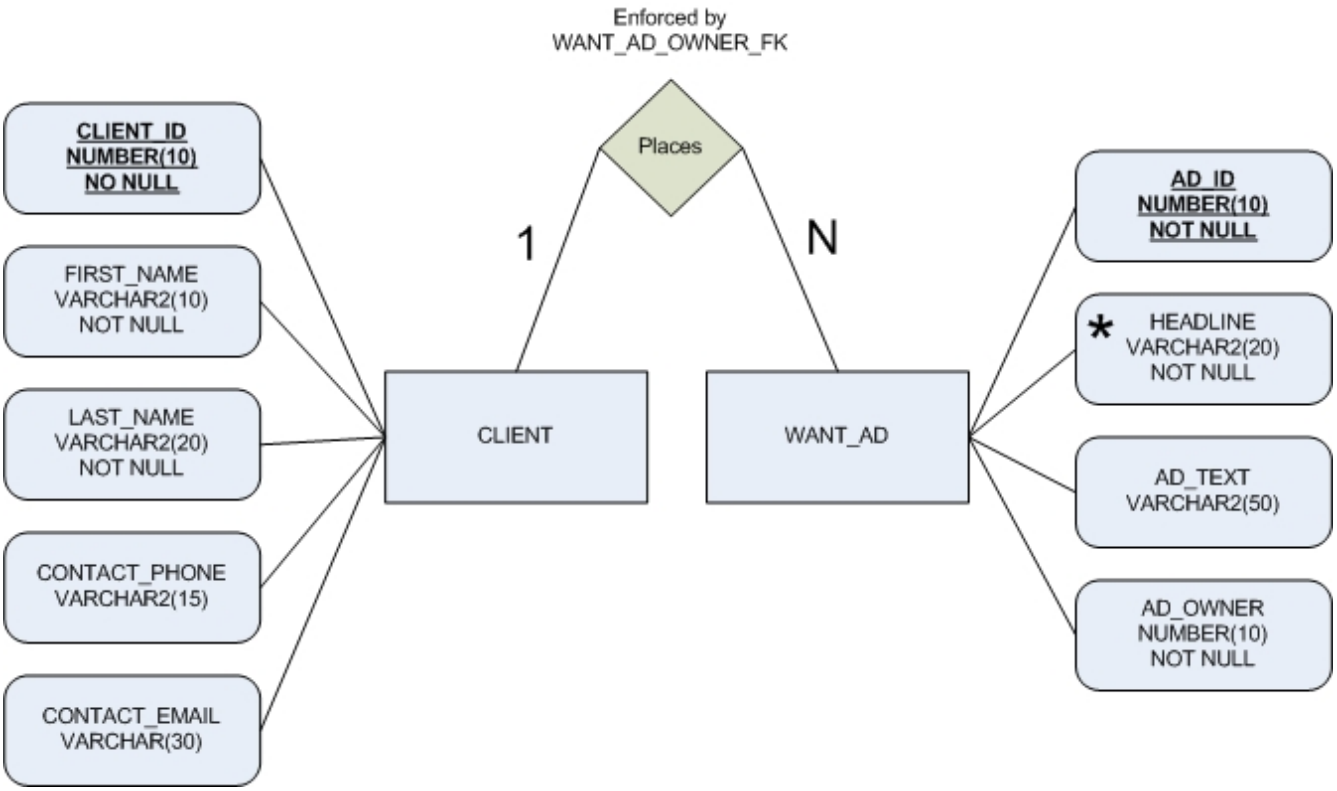
60-425
Oracle Database Design & Administration
Project Write-Up

Jay Lamont
Chris Lorenz

April 19th 2010

Case Study 1



Case Study 2

Since we were unable to create the database manually, we used the DBCA to create a script to create the database. It is a large set of files, so please see the files contained in the complete project submission.

Case Study 3

Using the SQL statement:

```
select a.name, b.phyrds, b.miniotim, b.maxiowtm
from v$datafile a, v$filestat b where a.file# = b.file#;
```

We found:

a) SYSTEM01.DBF has the most physicial read activity (4321).

b) Minimum IO time - all files share minimum value 0.
Maximum time for single write process - SYSAUX01.DBF with a time of 18.

c) The most recent checkpoint number is 629516.

d) The database was created March 15th, 2010.

Case Study 4

Using the following SQL statement we can determine the tablespace name, extent management and segment management mode.

```
select tablespace_name, extent_management, segment_space_management
from dba_tablespaces;
```

```
TABLESPACE_NAME                EXTENT_MAN  SEGMEN
------------------------------ ----------  ------
SYSTEM                         LOCAL       MANUAL
UNDOTBS1                       LOCAL       MANUAL
SYSAUX                         LOCAL       AUTO
TEMP                           LOCAL       MANUAL
USERS                          LOCAL       AUTO
EXAMPLE                        LOCAL       AUTO
```

As we can see, the SYSTEM tablespace has its extent management set to LOCAL and its segment space management set to MANUAL.

Next we can create three tablespace (t_accounting, t_sales, t_training) using the following three SQL statements.

```
CREATE TABLESPACE t_accounting
DATAFILE 'C:\oracle\product\10.2.0\oradata\orcl\t_accounting.dbf'
SIZE 5M
EXTENT MANAGEMENT LOCAL AUTOALLOCATE
SEGMENT SPACE MANAGEMENT AUTO;

CREATE TABLESPACE t_sales
DATAFILE 'C:\oracle\product\10.2.0\oradata\orcl\t_sales.dbf'
SIZE 5M
EXTENT MANAGEMENT LOCAL AUTOALLOCATE
SEGMENT SPACE MANAGEMENT AUTO;

CREATE TABLESPACE t_training
DATAFILE 'C:\oracle\product\10.2.0\oradata\orcl\t_training.dbf'
SIZE 5M
EXTENT MANAGEMENT LOCAL AUTOALLOCATE
SEGMENT SPACE MANAGEMENT AUTO;
```

The datafiles are all contained in 'C:\oracle\product\10.2.0\oradata\orcl\'.

We created the view MIS_data_status which returns the name of the tablespace/datafile/log file, a varchar2 representing which of the three the row represents, the file_id# (or for the log files, the group #), the number of bytes each occupies on the disk, and the number of blocks each occupies (null

for the redo logs). Thus, if the MIS manager ever wants to view this information, he can just use the view, rather than typing a very long SQL query statement.

```
create view MIS_data_stats as
select * from (
(select tablespace_name, 'TABLESPACE', file_id, bytes, blocks from dba_free_space)
UNION
(select name, 'DATAFILE', file#, bytes, blocks from v$datafile)
UNION
(select a.member, 'REDO LOG', a.group#, b.bytes, NULL
        from v$logfile a, v$log b
        where a.group# = b.group#)
);
```

Case Study 5

We thought that rather than altering the constraints for each table that we would create the appropriate tables with the appropriate constraints. This forces us to check data, when migrating it, that has already been entered against these constraints resulting in more correct data.

1. Employees:
   a) All employees must have a first and last name and job title.
      CONSTRAINT Descriptions: flj_chk makes sure none of these columns are null
   b) Valid job titles are: EDITOR, CHIEF EDITOR, WRITER, FREELANCE WRITER, MANAGER, DELIVERY PERSON, OFFICE ASSISTANT, PRESIDENT, PROGRAMMER, DBA.
      CONSTRAINT NAME: job_tit_chk

```
CREATE TABLE employee
(
employee_id NUMBER (10) PRIMARY KEY,
job_title VARCHAR2(45),
first_name VARCHAR2(40),
last_name VARCHAR2(40),
phone_number VARCHAR2(20),
CONSTRAINT flj_chk
        CHECK (job_title IS NOT NULL AND first_name IS NOT NULL
        AND last_name IS NOT NULL),
CONSTRAINT job_tit_chk
        CHECK (job_title IN (
                'EDITOR', 'CHIEF EDITOR', 'WRITER', 'FREELANCE WRITER',
                'MANAGER', 'DELIVERY PERSON', 'OFFICE ASSISTANT',
                'PRESIDENT', 'PROGRAMMER', 'DBA'
                ))
);
```

2. Editor Revenue:
   a) Each editor has one row in the EDITOR_REVENUE table that shows the total annual revenue for the current year. The primary key for this table is the editor's employee id
      CONSTRAINT NAME: prim_eid, fk_in_editor_id

```
CREATE TABLE editor_revenue
(
employee_id NUMBER (10),
ann_rev NUMBER (10),
CONSTRAINT prim_eid
        PRIMARY KEY (employee_id),
CONSTRAINT fk_in_editor_id
        FOREIGN KEY (employee_id) REFERENCES employee(employee_id)
);
```

We also felt that we should only be allowed to add employees who are editors or chief editors since they are the only employees relevant for this relation, thus we added the following trigger.

```
CREATE OR REPLACE TRIGGER editor_trig
BEFORE INSERT
ON editor_revenue
DECLARE
   e_invalid_input EXCEPTION;
BEGIN
      SELECT employee_id
      FROM employee
      WHERE employee_id = :NEW.employee_id
      AND (job_title = 'EDITOR' OR job_title = 'CHIEF EDITOR');
      IF SQL%NOTFOUND
         RAISE e_invalid_input;
      END IF
      COMMIT;
      EXCEPTION
         WHEN e_invalid_input THEN
            DBMS_OUTPUT.PUT_LINE ('Person is not an editor');
END;
/
```

3. Customers:
   a) Each customer has a unique id that is the primary key.
      CONSTRAINT NAME: prim_cid
   b) If a customer has a business name, he must also have a business type.
      CONSTRAINT NAME: bname_chk.
   c) Valid business types are: Corporation, Partnership, Sole Proprietor, Non-profit, and Other. The default is Other.
      CONSTRAINT NAME: type_chk
   d) Valid discount percentages range from 4.0 to 15.8.
      CONSTRAINT NAME: disc_chk

```
CREATE TABLE customer
(
c_id NUMBER (10),
b_name VARCHAR2(30),
b_type VARCHAR2(20) DEFAULT 'Other',
discount NUMBER(2,1),
CONSTRAINT prim_cid
        PRIMARY KEY (c_id),
CONSTRAINT disc_chk
        CHECK (discount BETWEEN 4.0 AND 15.8),
CONSTRAINT type_chk
        CHECK (b_type IN ('Corporation', 'Partnership', 'Sole Proprietor', 'Non-profit', 'Other')),
CONSTRAINT bname_chk
        CHECK (( b_name IS NOT NULL AND b_type IS NOT NULL) OR (b_name IS
NULL))
);
```

4. Classified section:
   a) Each section has a unique section id that is its primary key.
      CONSTRAINT NAME: pk_sno

   b) A section must have a non-null section title.
      CONSTRAINT NAME: sec_title_chk

   c) The priority rate per word must be between $.50 and $5.75 but can also be null.
      CONSTRAINT NAME: rate_chk

```
CREATE TABLE classified_section
(
section_no NUMBER NOT NULL,
section_title VARCHAR2(50),
base_rate_per_word NUMBER(4,3),
CONSTRAINT pk_sno PRIMARY KEY (section_no),
CONSTRAINT sec_title_chk CHECK(section_title IS NOT NULL),
CONSTRAINT rate_chk
        CHECK ((base_rate_per_word BETWEEN 0.50 AND 5.75)
        OR base_rate_per_word IS NULL)
);
```

1. Classified Ad:
   a) Each ad has a unique sequence-generated number as its identifier.
      CONSTRAINT NAME: ad_id, drawn from sequence class_ad_seq
   b) Each ad must be categorized in a valid classified section. This constraint is deferrable.
      CONSTRAINT NAME: fk_sec_id
   c) Each ad must have a valid intake editor. If the editor's employee record is deleted, all corresponding ads have the intake editor field set to null.
      CONSTRAINT NAME: fk_in_editor_id_2

   d) Valid priority codes are: H for high priority, and L for low priority.

CONSTRAINT NAME: pricode_chk

e) Each ad must be connected to a customer in the CUSTOMER table. If the customer's record is removed, the ad remains but the customer_id becomes null.
CONSTRAINT NAME: fk_cust_id

```
CREATE SEQUENCE class_ad_seq
INCREMENT BY 1
START WITH 1;

CREATE TABLE classified_ad
(
ad_id NUMBER (5),
section_id NUMBER,
in_editor_id NUMBER(10) DEFAULT NULL,
pri_code VARCHAR(1),
cust_id NUMBER(10) DEFAULT NULL,
placed_date DATE,
CONSTRAINT pk_adid PRIMARY KEY (ad_id),
CONSTRAINT fk_sec_id
        FOREIGN KEY (section_id) REFERENCES classified_section(section_no),
CONSTRAINT fk_in_editor_id_2
        FOREIGN KEY (in_editor_id) REFERENCES employee(employee_id),
CONSTRAINT pricode_chk
        CHECK (pri_code IN ('H', 'L')),
CONSTRAINT fk_cust_id
        FOREIGN KEY (cust_id) REFERENCES customer(c_id)
);
```

Case Study 6

First, we create a tablespace named “USER_AUTO” with the appropriate sizes to accommodate future expansion of the table on the tablespace. What we concluded was that the initial size needed to handle the rows is approximately 120000 bytes, each additional month would require an additional 17500 bytes therefore next should be set to that.

```
CREATE TABLESPACE USER_AUTO
        DATAFILE 'C:\oracle\product\10.2.0\oradata\orcl\user_auto.dbf' SIZE 120K
        AUTOEXTEND ON NEXT 18K MAXSIZE UNLIMITED
        EXTENT MANAGEMENT LOCAL;
```

We also need to create a partitioned index, where two partitions are stored on the USERS tablespace and the remaining partition on the USER_AUTO tablespace. Since we have 12 years to address, we decided to split them into three equal parts with the last partition on USER_AUTO since it will be the tablespace that will increase in size.

```
CREATE INDEX ad_date_index ON classified_ad(placed_date)
GLOBAL PARTITION BY RANGE (placed_date)
(
        PARTITION p1 VALUES LESS THAN (TO_DATE('03/18/2002', 'MM/DD/YYYY'))
        TABLESPACE USERS,
        PARTITION p2 VALUES LESS THAN (TO_DATE('03/18/2006', 'MM/DD/YYYY'))
        TABLESPACE USERS,
        PARTITION p3 VALUES LESS THAN (MAXVALUE) TABLESPACE USER_AUTO
);
```

To be able to hold the first and last name of an individual we chose a length of 81 (1 extra for space) for the VARCHAR2 which will hold the full name.

```
ALTER TABLE EMPLOYEE ADD name VARCHAR2(81);
```

After creating the column, we need to concatenate the first and last name of each employee and put it into the name column, this was accomplished with:

```
UPDATE EMPLOYEE SET name = first_name || ' ' || last_name;
```

Finally, we can drop the first and last name tables.

```
ALTER TABLE EMPLOYEE DROP COLUMN first_name;
ALTER TABLE EMPLYOEE DROP COLUMN last_name;
```

Case Study 7

First, we need to create a manager profile, which will allow for the reuse of a password after 300 unique passwords, each password is useable for 30 days before expiring and an idle time of 10 minutes before disconnection.

```
CREATE PROFILE MANAGERS LIMIT
PASSWORD_REUSE_MAX 300
PASSWORD_LIFE_TIME 30
IDLE_TIME 10;
```

Next, we need to create the writers profile, which will can have a single password for 30 days before expiring, an idle time of only 15 minutes before being disconnected, up to 1000 logical reads per session and has a private SGA allocation of 256K.

```
CREATE PROFILE WRITERS LIMIT
PASSWORD_LIFE_TIME 30
IDLE_TIME 15
LOGICAL_READS_PER_SESSION 1000
PRIVATE_SGA 256K;
```

Finally, we need to create the editors profile, which will have a password life time of 30 days, an idle time before disconnect of 45 minutes and a total connect time limit of 8 hours.

```
CREATE PROFILE EDITORS LIMIT
PASSWORD_LIFE_TIME 30
IDLE_TIME 45
CONNECT_TIME 480;
```

To easily assign a bulk amount of users this profile and grant them all the “CREATE SESSION” privilege, the easiest way is to spool the following SQL statements to a file.

```
SELECT 'CREATE USER ' ||
first_name || SUBSTR(last_name,1,3) || ' IDENTIFIED BY "TEMPPASS" ' ||
'PASSWORD EXPIRE PROFILE ' || job_title || 'S;' FROM employee
WHERE job_title IN ('WRITER', 'EDITOR', 'MANAGER');

SELECT 'GRANT CREATE SESSION  TO ' || first_name || SUBSTR(last_name,1,3) || ';'
FROM employee;
```

These will then be output into a file which can be run (after some minor cleanup) to grant the users the appropriate profile and privilege. This will take into account their current job title, meaning no one will get permissions they should not get.

Case Study 8

“An administrative assistant who queries the database and reports on data result sets” should likely only have read-only permissions thus the only permissions they need are the following.

```
GRANT CREATE SESSION TO AdmAst;
GRANT SELECT ANY TABLE TO AdmAst;
```

“A systems analyst who suggests and implements system enhancements, such as database views” should likely only need to create/alter things such as indexes, sequences, triggers, procedures and views. Thus, the following would be appropriate permissions.

```
GRANT CREATE SESSION TO AdmAst;
GRANT CREATE ANY VIEW,
        CREATE ANY INDEX,
        CREATE ANY SEQUENCE,
        CREATE ANY TRIGGER,
        CREATE ANY PROCEDURE,
        CREATE ANY SYNONYM,
        ALTER ANY VIEW,
        ALTER ANY INDEX,
        ALTER ANY SEQUENCE,
        ALTER ANY TRIGGER,
        ALTER ANY PROCEDURE,
        ALTER ANY SYNONYM
TO SysAnst;
```

“An applications developer who redesigns certain business processes” should likely be allowed to do everything, assuming they are doing so on a database that is NOT currently being used by the business (i.e. a test server) until the redesign is done.

```
GRANT all_sys_privs to AppDev;
```

Case Study 9

Administrative and Support Staff should only need one session per user since they should only be querying the database and not doing anything that warrants more. As well, they should be permitted a modest connect and idle time so they can remain productive.

```
CREATE PROFILE AdmSup LIMIT
SESSION_PER_USER 1
CONNECT_TIME 1800
IDLE_TIME 300;

GRANT SELECT ANY TABLE TO AdmSup;
```

Supervisory Staff should also only need one session per user since they are simply supervising and possibly updating tables. They should also be given a slightly larger connect time and idle time.

```
CREATE PROFILE super LIMIT
SESSION_PER_USER 1
CONNECT_TIME 3600
IDLE_TIME 600;

GRANT SELECT ANY TABLE, INSERT ANY TABLE, UPDATE ANY TABLE,
DELETE ANY TABLE TO Super;
```

Similar to Supervisors, Client Account Managers should only require one session and since they are dealing with clients, should also be given a slightly larger connect and idle time.

```
CREATE PROFILE Mgr LIMIT
SESSION_PER_USER 1
CONNECT_TIME 3600
IDLE_TIME 600;

GRANT SELECT ANY TABLE, INSERT ANY TABLE, UPDATE ANY TABLE,
DELETE ANY TABLE to Mgr;
```

If information related to privileges is needed then there are two data dictionary views that are important to know about, DBA_COL_PRIVS and DBA_TAB_PRIVS. DBA_COL_PRIVS and DBA_TAB_PRIVS views give information about privileges granted to other users, this includes the username of the grantee and grantor, as well as the owner, the object the privileges are being granted for, whether they've been done with the GRANT OPTION or HEIRARCHY OPTION. This means we would be able to view whether a user has been granted select privileges on tables they do not own.

Case Study 10

Phase I involves many inserts, updates, and deletes on new_enrollment and new_student. In this scenario, we need to balance PCTFREE and PCTUSED carefully so that both inserts and updates are optimized. First, we want to allocated a decent percentage to PCTFREE, so that rows can be updated without the need to be moved to a new block, which would reduce performance. However, too large a PCTFREE would leave too little room for our inserts, which would cause many new blocks to be created, reducing performance and poorly utilizing disk space. Since we are also experienceing a high number of inserts, PCTUSED should not be set too small. This will allow space in blocks freed up by deletes to be used more quickly for inserts, rather than just updates. Several FREELISTS should be allocated to this table, so that the insert statements are not bottlenecked by competition for access to the free lists.

For the other tables, PCTFREE should be set rather low and PCTUSED rather high. Because there are not many insert or updates performence tuning here is not as important. PCTFREE should be small because we want disk space to be used as efficiently as possible. Since not many updates will occur it does not make sense to allocate a large amount of free space to be dedicated for updates - doing so would be a waste of space. PCTUSED should be set high so that in case the PCTFREE threshold is reached, the free space from any deletes can quickly be used for inserts once again. Overall, the performance of both inserts and updates on these tables will be poor. The benefit is that disk space is used a very efficient manner.

In general, a balance must be struck between the performance of inserts and the performance of updates in our tables. Setting PCTFREE and PCTUSED to benefit one operation will seem to generally negatively impact the other.

Phase I:

```
CREATE TABLE new_course(
        CODE VARCHAR(10) PRIMARY KEY,
        NAME VARCHAR(50)
)  TABLESPACE USERS
PCTFREE 10
PCTUSED 85;

CREATE TABLE new_enrollment(
        ID NUMBER PRIMARY KEY,
        FNAME VARCHAR(20),
        LNAME VARCHAR(20),
        ADDRESS VARCHAR(200),
        PHONE VARCHAR(15)
)TABLESPACE USERS
PCTFREE 25
PCTUSED 45;
```

```
CREATE TABLE new_instructor(
        ID NUMBER PRIMARY KEY,
        FNAME VARCHAR(20),
        LNAME VARCHAR(20)
)TABLESPACE USERS
PCTFREE 10
PCTUSED 85;

CREATE TABLE new_section(
        CODE VARCHAR(10),
        SECT_NUM NUMBER,
        TIMES VARCHAR(20),
        INSTRUCTOR_ID NUMBER,
        CONSTRAINT prim_sect PRIMARY KEY (CODE, SECT_NUM),
        CONSTRAINT fk_inst FOREIGN KEY (INSTRUCTOR_ID)
                REFERENCES new_instructor(ID)
)TABLESPACE USERS
PCTFREE 10
PCTUSED 85;

CREATE TABLE NEW_ZIPCODE(
        ZIP VARCHAR(30) PRIMARY KEY
)TABLESPACE USERS
PCTFREE 10
PCTUSED 85;


CREATE TABLE new_student(
        ID NUMBER PRIMARY KEY,
        FNAME VARCHAR(20),
        LNAME VARCHAR(20),
        ADDRESS VARCHAR(200),
        PHONE VARCHAR(15)
)TABLESPACE USERS
PCTFREE 25
PCTUSED 45;
```

Phase II involves a large number of inserts but not many deletes or updates on new_student and new_enrollment. The performance of the other tables remains largely unchanged as they do not experience any more activity as they did in Phase I. In this case, we should set our PCTFREE to a rather low number. This means that a very small amount of space in each block will be reserved for updates, which makes sense because not many updates are occuring. A large PCTFREE would cause our insert statements' performance to drop, because it would need to switch blocks more often once it hits the maximum percentage for row insertion allowed. If our number of updates were to increase in Phase II, their performance would be negatively affected because a very small amount of space in each block is reserved for updates. Therefore, a lot of row migration would occur as rows are moved to blocks that have enough free space to accomodate the update. We would want to assign a high number of freelists for our two tables in this example, so that competition for the freelists is reduced when inserts occur, which will improve performance.

```
CREATE TABLE new_course(
        CODE VARCHAR(10) PRIMARY KEY,
        NAME VARCHAR(50)
)  TABLESPACE USERS
PCTFREE 10
PCTUSED 85;

CREATE TABLE new_enrollment(
        ID NUMBER PRIMARY KEY,
        FNAME VARCHAR(20),
        LNAME VARCHAR(20),
        ADDRESS VARCHAR(200),
        PHONE VARCHAR(15)
)TABLESPACE USERS
PCTFREE 5
PCTUSED 90;

CREATE TABLE new_instructor(
        ID NUMBER PRIMARY KEY,
        FNAME VARCHAR(20),
        LNAME VARCHAR(20)
)TABLESPACE USERS
PCTFREE 10
PCTUSED 85;

CREATE TABLE new_section(
        CODE VARCHAR(10),
        SECT_NUM NUMBER,
        TIMES VARCHAR(20),
        INSTRUCTOR_ID NUMBER,
        CONSTRAINT prim_sect PRIMARY KEY (CODE, SECT_NUM),
        CONSTRAINT fk_inst FOREIGN KEY (INSTRUCTOR_ID)
                REFERENCES new_instructor(ID)
)TABLESPACE USERS
PCTFREE 10
PCTUSED 85;

CREATE TABLE NEW_ZIPCODE(
        ZIP VARCHAR(30) PRIMARY KEY
)TABLESPACE USERS
PCTFREE 10
PCTUSED 85;
```

```
CREATE TABLE new_student(
        ID NUMBER PRIMARY KEY,
        FNAME VARCHAR(20),
        LNAME VARCHAR(20),
        ADDRESS VARCHAR(200),
        PHONE VARCHAR(15)
)TABLESPACE USERS
PCTFREE 5
PCTUSED 90;
```

Row migration will occur primarily when you perform a large amount of updates on rows in a table. If your PCTFREE for the table is set too low, there may not be adequate space in the blocks to accomodate updates to the rows. If an update to a row requires more free space than is available in the block (possibly because PCTFREE is set too low) the entire row will move to a new block where there is enough space to accomodate it and the update. Row migration can affect the performance of select operations because the database may need to follow pointers across multiple blocks until it finally finds the data it is looking for. Inserts will not result in row migration because the row will initially be inserted in a block (or blocks) where there is adequate space. Deletes will also not result in row migration because you are freeing up space in a block rather than taking up more. In fact, a large number of deletes coupled with a large number of updates could actually mitigate the amount of row migration that will take place.

Row chaining will occur if your block size for the table is too small compared to the amount of data in your rows. By setting the block size large enough, you can prevent row chaining from taking place. Chaining will negatively affect performance for inserts, because the data will need to be written across multiple blocks, rather than to a single block. Select performance will also be diminished because data will need to be read from multiple blocks, rather than from a single block. Row chaining will not generally

Case Study 11

There are two methods of creating a consistent backup of a database, the first involves taking the tablespaces you want to backup offline, making a copy of the DBF files and then bringing them back online. This is consistent because no changes can be done while the tablespaces are offline.

```
ALTER TABLESPACE SYSTEM OFFLINE NORMAL;
ALTER TABLESPACE SYSAUX OFFLINE NORMAL;
ALTER TABLESPACE USERS OFFLINE NORMAL;
ALTER TABLESPACE TEMP OFFLINE NORMAL;
ALTER TABLESPACE UNDO OFFLINE NORMAL;

ocopy C:\oracle\product\10.2.0\oradata\orcl\*.DBF C:\backup\

ALTER TABLESPACE SYSTEM ONLINE;
ALTER TABLESPACE SYSAUX ONLINE;
ALTER TABLESPACE USERS ONLINE;
ALTER TABLESPACE TEMP ONLINE;
ALTER TABLESPACE UNDO ONLINE;
ALTER SYSTEM ARCHIVE LOG CURRENT;
```

The second method involves shutting down the entire database and using the Export function to make a full backup of the entire database. This is consistent because it also guarantees that no changes are made when the backup is being made.

```
SHUTDOWN NORMAL;
EXP system/manager FULL=y INCTYPE=cumulative FILE=C:\total.dmp
STARTUP OPEN;
```

There are also two methods to make an inconsistent backup. The first involves putting each tablespace into backup mode, which unlike the similar consistent backup method, does allow changes to be made while the backup is being created making it inconsistent (point-in-time).

```
ALTER TABLESPACE SYSTEM BEGIN BACKUP;
ALTER TABLESPACE SYSAUX BEGIN BACKUP;
ALTER TABLESPACE USERS BEGIN BACKUP;
ALTER TABLESPACE TEMP BEGIN BACKUP;
ALTER TABLESPACE UNDO BEGIN BACKUP;

ocopy C:\oracle\product\10.2.0\oradata\orcl\*.DBF C:\backup\

ALTER TABLESPACE SYSTEM END BACKUP;
ALTER TABLESPACE SYSAUX END BACKUP;
ALTER TABLESPACE USERS END BACKUP;
ALTER TABLESPACE TEMP END BACKUP;
ALTER TABLESPACE UNDO END BACKUP;
```

The second method is exactly the same as the similar consistent method, except that you do not shutdown the database, you just make a full point-in-time backup.

```
Exit SQLPLUS

EXP system/manager FULL=y INCTYPE=cumulative FILE=C:\total.dmp
```

Case Study 13

First, let's assume that we're doing the following backup of the tablespace sample. Since we are simulating a failure, the database must be running, thus we need to start to do an inconsistent backup.

```
ALTER TABLESPACE SAMPLE BEGIN BACKUP;

ocopy C:\oracle\product\10.2.0\oradata\orcl\SAMPLE.DBF C:\backup\SAMPLE.DBF
```

In order to simulate a failure, we used:

```
SHUTDOWN IMMEDATE;
or
SHUTDOWN ABORT;
```

To start the restoration, we first need to copy latest back-up to the appropriate oradata folder. This is NOT the backup we just made, since that is not complete or readable backup (corrupt!). Then we need to start the database again.

STARTUP OPEN;

Since we are restoring the datafile SAMPLE.DBF, we need to take that datafile offline so that we can up restore it properly.

ALTER DATABASE DATAFILE
'C:\oracle\product\10.2.0\oradata\orcl\SAMPLE.DBF' OFFLINE

Now we can attempt the recovery.

ALTER DATABASE OPEN;

RECOVER DATAFILE 'C:\oracle\product\10.2.0\oradata\orcl\SAMPLE.DBF'

Once the datafile has been recovered, we can now bring the datafile online again.

ALTER DATABASE DATAFILE
'C:\oracle\product\10.2.0\oradata\orcl\SAMPLE.DBF' ONLINE

Now the database can be used again.

Case Study 13

a)
```
CREATE TABLE ME
(
        col1 NUMBER,
        col2 VARCHAR2(100)
) TABLESPACE USERS;
```

b)
```
CREATE OR REPLACE PROCEDURE insert_me
IS
        counter NUMBER := 0;
        blar VARCHAR2(100) := 'omgwtfbbq';
BEGIN
        WHILE counter != 100 LOOP
                INSERT INTO me VALUES (counter, blar || counter );
                counter := counter + 1;
        END LOOP;
END;
/
```

c)
```
SELECT * FROM ME WHERE col1 > (SELECT COUNT(*) FROM ME) - 11;
```

d) EXP system/welcome FILE=C:\ME.DMP LOG=C:\ME.LOG
TABLES=ME ROWS=YES INDEXES=NO

e) DROP TABLE ME CASCADE CONSTRAINTS;

f) IMP system/welcome FILE=C:\ME.DMP FULL=yes