School of Business Systems

Bus4580 and Bus4590

ORACLE DBA 1 & 2

Yu Ting Hu (Ivy)(12237043)

Master of Business System

Semester 1, 2002

Supervisor: Dr. David Taniar

Acknowledgement

I would like to thank Dr. David Taniar for his helpful and insightful comments during the development of this project.

Yu Ting Hu

CHAPTER 1 INTRODUCTION

Database management systems are changing and growing more complex with each new version and release. As companies grow, the database also expands and more problems tend to occur. It is the Database Administrator's (DBA) main duty to ensure the database is operated in an optimal state.

The DBA can be responsible for many different components of the database environment. The key responsibilities are performance tuning, database security, and backup and recovery. This includes supporting and organizing the database engine, its physical layout, and data security. Other tasks include installing the database binaries (program files), preparing and testing a backup scenario, and performing imports and exports of data. The DBA tasks may also include the installation of Oracle's networking and Web components, and installing Oracle front-end tools—such as Oracle Developer Forms—on client machines.

This project was intended for a self-study documentation to simulate the tasks performed by an ORACLE DBA. All the areas covered in the project were experimented and simulated with examples. The modules covered in this documentation include:

- **CHAPTER 2: SQL and PL/SQL**

In this module, most of the essential SQL statements and PL/SQL a DBA needs for monitoring and maintaining an ORACLE database are explored. Areas covered include:

  - Basic SQL statement
  - SQL functions
  - Multiple table queries
  - Subqueries
  - DML statement
  - Transaction control
  - Views
  - Sequences
  - PL/SQL
  - PL/SQL program units (procedures, functions, triggers, and packages)

- **CHAPTER 3: Database Architecture and Administration**

  - Software installation
  - Administrator Authentication Method
  - Managing an Oracle instance
  - Create a Database
  - Maintaining control files
  - Maintaining redo log files
  - Managing tablespaces and data files
  - Managing rollback segments
  - Managing tables
  - Managing indexs
  - Managing users
  - Managing privileges
  - Managing roles
  - Managing profiles

- **CHAPTER 4: Performance Tuning**

  - SQL tuning
  - Tuning the share pool
  - Tuning the database buffer cache
  - Tuning the redo log buffer
  - Tuning sort operation
  - Tuning rollback segments

Throughout this project, all the examples used are based on the following tables. All the scripts to create the tables are included in the Appendix A. and Appendix B.

Customers Table

CID	CNAME	CADD	CPCODE	CPHONE
50001	ColesMyer	123 Clayton road	3600	96782415
50002	Samsung	41 Glenhuntly road	3112	95712800
50003	Hilton	38 Mcmaster court	3897	97580322
50004	Evian	1 Flinder street	3112	96508016
50005	Monash	3500 Lancell road	5038	98133049

Employees Table

EID	EFNAME	ELNAME	EDOB	EADD	EPHONE	EPCODE	DID
520	Jim	Peterson	02-NOV-78	13/78 King Street	94037878	1352	101
521	Clair	Manson	30-MAR-74	2/8 Park Street	97750322	2012	101
522	Todd	Smith	17-APR-76	11 Kew Street	94219660	1352	103
523	Rebecca	Gwen	20-JAN-75	16/2 Melrose place	93315716	1032	102
524	Mareen	Joans	13-APR-79	133/6 Gleniris road	97305643	1033	101
525	Miller	Chang	03-JUL-71	23/40 Water street	97784106	2012	Null

Orders Table

ORDERID	ORD_DATE	CID	ITEMID	ORD_QTY	DELVER_DATE	EID
3331	13-APR-02	50005	90003	10	20-APR-02	521
3332	14-APR-02	50001	90006	47	20-APR-02	520
3333	15-APR-02	50003	90001	5	22-APR-02	520
3334	16-APR-02	50002	90003	15	22-APR-02	524
3335	16-APR-02	50004	90002	10	22-JUN-02	524

Items Table

ITEMID	ITEMDESC	PRICE	QOH
90001	HITACHI monitor 17inch	900	50
90002	HITACHI monitor 19inch	1500	12
90003	Sony 56k modem	90	68
90004	Microsoft keyboard	40	87
90005	Sony 52x CDROM drive	120	94
90006	TDK flopy disc x 12	8	112

Departments Table

DID	DEPART_NAME
101	Sales
102	Accounting
103	Marketing

CHAPTER 2 SQL and PL/SQL

1. SELECT STATEMENT

The SELECT statement is the most commonly used statement in SQL. It is normally used to retrieve information that is already stored in the database.

The basic syntax (Ault, 2001)

```
SELECT [DISTINCT] {*,column [alias],....}
FROM table;
```

1.1 Select all columns / all rows

Using asterisk (*) to indicate all columns and all rows in the SELECT statement.

```
SQL> SELECT * FROM CUSTOMERS;

       CID CNAME      CADD                    CPCODE     CPHONE
---------- ---------- -------------------- ---------- ----------
     50001 ColesMyer  123 Clayton road           3600   96782415
     50002 Samsung    41 Glenhuntly road         3112   95712800
     50003 Hilton     38 Mcmaster court          3897   97580322
     50004 Evian      1 Flinder street           3112   96508016
     50005 Monash     3500 Lancell road          5038   98133049


5 rows selected.
```

1.2 Select specific columns

Specifying the particular columns of information for retrieval in the SELECT statement.

```
SQL> select cid, cname
  2  from customers;

       CID CNAME
---------- ----------
     50001 ColesMyer
     50002 Samsung
     50003 Hilton
     50004 Evian
     50005 Monash

5 rows selected.
```

1.3 Limiting Rows

A WHERE clause is used to limit the number of rows processed. Condition(s) need to be specified within the clause.

For example, display order(s) that is processed by employee id 520.

```
SQL> select orderid, ord_date, cid
  2  from orders
  3  where eid = 520;

   ORDERID ORD_DATE         CID
---------- --------- ----------
      3332 14-APR-02      50001
      3333 15-APR-02      50003

2 rows selected.
```

The following operators can also be included in the WHERE clause to limit the number of rows selected.

- **BETWEEN operator**

For example, display information of the order(s) that items will be delivered between 18-APR-02 and 21-APR-02.

```
SQL> select orderid, cid, itemid, ord_qty
  2  from orders
  3  where delver_date BETWEEN '18-APR-02' and
  4  '21-APR-02';

   ORDERID        CID     ITEMID    ORD_QTY
---------- ---------- ---------- ----------
      3331      50005      90003         10
      3332      50001      90006         47
```

- **IN operator**

The following example uses IN operator to display employee information who belong to the sales department (101).

```
SQL> select eid, efname, elname,eadd, epcode, did
  2  from employees
  3  where did in (101);

Employee First    Last                                POST Dept
      Id Name     Name     Address                    CODE   ID
-------- -------- -------- ------------------------ ----- ----
     520 Jim      Peterson 13/78 King Street          1352  101
     521 Clair    Manson   2/8 Park Street           2012  101
     524 Mareen   Joans    133/6 Gleniris road       1033  101
```

- **LIKE operator**

The following example demonstrates using LIKE operator to retrieve customer information that company name contains ‘am’.

```
SQL> select cid, cname
  2  from customers
  3  where cname LIKE '%am%';

       CID CNAME
---------- ----------
     50002 Samsung
```

1.4 Sorting Rows

The SELECT statement may include the ORDER BY clause to sort the resulting rows in a specific order based on data in the columns. The rows can be returned in the ascending order of the columns specified or descending order.

For example, display the details of employees who belong to department 101, order by the employee id.

```
SQL> select eid, efname, elname, eadd ephone, did
  2  from employees
  3  where did = 101
  4  order by eid;

       EID EFNAME     ELNAME          EPHONE                         DID
---------- ---------- --------------- ------------------------ ----------
       520 Jim        Peterson        13/78 King Street               101
       521 Clair      Manson          2/8 Park Street                 101
       524 Mareen     Joans           133/6 Gleniris road             101

3 rows selected.
```

1.5 Arithmetic Expressions

Arithmetic expressions are used to modify the data the way it is displayed, perform calculations and to also look at the what-if scenarios. (Ault, 2001)

For example, use multiplication operator to display the total price for item 90003 when 12 of which is purchased.

```
SQL> select itemid, itemdesc, price * 12
  2  from items
  3  where itemid = 90003;

    ITEMID ITEMDESC                       PRICE*12
---------- ------------------------------ ----------
     90003 sony 56k modem                      1080

1 row selected.
```

1.6 Column Alias

When displaying the result of a query, SQL*PLUS normally uses the selected column name as the heading. In some cases, it might occur to be meaningless or hard to understand for the users. Column Alias is therefore used to change the column headings to make the information more understandable. This is done by including the AS key word before the alias name. If the alias contains space or special characters, it is enclosed in double quotation mark (“ “). (Ault, 2001)

Example: use the above example and display the headings with proper names.

```
SQL> select itemid as "Item Id", itemdesc as "Item Description", price * 12
as "Total price"
  2  from items
  3  where itemid = 90003;

   Item Id Item Description               Total price
---------- ------------------------------ -----------
     90003 sony 56k modem                        1080

1 row selected.
```

1.7 Concatenation Operator

The concatenation operator is used to join two character strings, which produces another character string. Two vertical bars (||) are used as the concatenation operator.

For example, display employee's first name and last name together with the heading 'Employee Name'.

```
SQL> select efname||elname As "Employee Name"
  2  from employees;

Employee Name
-------------------------
JimPeterson
ClairManson
ToddSmith
RebeccaGwen
MareenJoans
MillerChang

6 rows selected.
```

1.8 Duplicate Rows

The DISTINCT keyword followed by the SELECT keyword eliminates duplicate rows. This ensures that the resulting rows are unique.

Example: display all department id in the employees table.

```
SQL> select did from employees;

       DID
----------
       101
       101
       103
       102
       101
       102

6 rows selected.
```

In order to eliminate the duplicate rows in the result, using the DISTINCT keyword in the SELECT statement.

```
SQL> select distinct did from employees;

       DID
----------
       101
       102
       103

3 rows selected.
```

As the result shown, only the unique department id is displayed.

2. CREATING/FORMATING A REPORT

In order to produce a more presentable and readable report, various issues need to be considered and defined; for example, the width of the columns, proper headings and data formatting.

Example: produce an employee report. The following select statement is used to retrieve all the employee information.

```
SQL> select * from employees;
```

```
       EID EFNAME     ELNAME          EDOB      EADD
---------- ---------- --------------- --------- -------------------------
    EPHONE     EPCODE        DID
---------- ---------- ----------
       520 Jim        Peterson        02-NOV-78 13/78 King Street
  94037878       1352        101

       521 Clair      Manson          30-MAR-74 2/8 Park Street
  97750322       2012        101

       522 Todd       Smith           17-APR-76 11 Kew Street
  94219660       1352        103


       EID EFNAME     ELNAME          EDOB      EADD
---------- ---------- --------------- --------- -------------------------
    EPHONE     EPCODE        DID
---------- ---------- ----------
       523 Rebecca    Gwen            20-JAN-75 16/2 Melrose place
  93315716       1032        102

       524 Mareen     Joans           13-APR-79 133/6 Gleniris road
  97305643       1033        101

       525 Miller     Chang           03-JUL-71 23/40 Water street
  97784106       2012        102


6 rows selected.
```

It is apparent that this report is not very appealing to its intended reader. This is because it does not have any proper headings and settings to present all the columns in the same row. The presentation of this report can be improved through modification of the page size and column width.

2.1 Adjust page size

We need to adjust settings of page size and line size in order to place every column in one single row.

To find out the values of current settings, use SHOW command.

```
SQL> SHOW PAGESIZE LINE;
pagesize 14
linesize 80
```

To change the values of current settings, use SET command.

```
SQL> SET PAGES 55 LINES 90;
```

2.2 Column command

The COLUMN command is used to format the heading and data. The basic syntax is: (Ault, 2001)

```
COL[UMN] [{column|alias} [option ...]]
```

```
SQL> COLUMN eid HEADING "Employee|Id" FORMAT 999
SQL> COLUMN efname HEADING "First|Name" FORMAT A8
SQL> COLUMN elname HEADING "Last|Name" FORMAT A8
SQL> COLUMN edob HEADING "Date|of Birth"
SQL> COLUMN eadd HEADING "Address"
SQL> COLUMN ephone HEADING "Phone|Number" FORMAT 99999999
SQL> COLUMN EPCODE HEADING "POST|CODE" FORMAT 9999
SQL> COLUMN did HEADING "Dept|ID" FORMAT 999;
```

2.3 Title command

The TITLE command is used to display additional report heading.

```
SQL> TTITLE CENTER "Employee Information" SKIP 2;
```

2.4 The Report

```
SQL> select * from employees;
```

```
                              Employee Information

Employee First    Last     Date                                  Phone  POST Dept
      Id Name     Name     of Birth  Address                    Number  CODE   ID
-------- -------- -------- --------- ------------------------ --------- ----- ----
     520 Jim      Peterson 02-NOV-78 13/78 King Street          94037878  1352  101
     521 Clair    Manson   30-MAR-74 2/8 Park Street           97750322  2012  101
     522 Todd     Smith    17-APR-76 11 Kew Street             94219660  1352  103
     523 Rebecca  Gwen     20-JAN-75 16/2 Melrose place        93315716  1032  102
     524 Mareen   Joans    13-APR-79 133/6 Gleniris road       97305643  1033  101
     525 Miller   Chang    03-JUL-71 23/40 Water street        97784106  2012  102

6 rows selected.
```

3. ACCEPTING VALUES AT RUN TIME

3.1 Substitution Variable

An interactive SQL command allows the user to supply values at runtime. An ampersand (&) is used in the statement to identify the variable. This function further enhances the ability to reuse the SQL script.

Example:

The following statement is created to prompt the user for a department id at run time, and to create a report that contains employee id, employee name, address and phone number based on the department id entered.

```
SQL> select eid, efname, elname, eadd, ephone
  2  from employees
  3  where did = &Department_Id;
```

```
Enter value for department_id: 101
old   3: where did = &Department_Id
new   3: where did = 101

       EID EFNAME     ELNAME          EADD                          EPHONE
---------- ---------- --------------- ------------------------- ----------
       520 Jim        Peterson        13/78 King Street           94037878
       521 Clair      Manson          2/8 Park Street             97750322
       524 Mareen     Joans           133/6 Gleniris road         97305643
```

The following statement prompts the user for a delivery date at run time in order to create a report containing order details based on the delivery date specified.

```
SQL> Select orderid, itemid, ord_qty
  2  from orders
  3  where delver_date = '&Deliver_date';
```

```
Enter value for deliver_date: 20-apr-02
old   3: where delver_date = '&Deliver_date'
new   3: where delver_date = '20-apr-02'

   ORDERID     ITEMID    ORD_QTY
---------- ---------- ----------
      3331      90003         10
      3332      90006         47
```

The substitution variables can also be used on the column names and expressions. In the following example, display the order Id and any other column(s) with any run time specified conditions.

```
SQL> select orderid, &column_name
  2  from orders
  3  where &condition;
```

In order to look for orders processed by employee '524', the following values are entered at run time.

```
Enter value for column_name: itemid, ord_qty
old   1: select orderid, &column_name
new   1: select orderid, itemid, ord_qty
Enter value for condition: eid = 524
old   3: where &condition
new   3: where eid = 524

   ORDERID     ITEMID    ORD_QTY
---------- ---------- ----------
      3334      90003         15
      3335      90002         10
```

3.2 Define User Variables

Variables can be predefined using DEFINE and ACCEPT commands.

- **DEFINE command**

Example: Using the DEFINE command to provide a value to the variable 'Department_id'.

```
SQL> DEFINE Department_id = 101
```

This method allows the user to avoid the prompt for the value at run time. Wherever the variable occurs, it will be substitute with the predefined value.

```
SQL> select eid, efname, elname, eadd, ephone
  2  from employees
  3  where did = &Department_Id;
old   3: where did = &Department_Id
new   3: where did = 101

       EID EFNAME     ELNAME          EADD                          EPHONE
---------- ---------- --------------- ------------------------- ----------
       520 Jim        Peterson        13/78 King Street           94037878
       521 Clair      Manson          2/8 Park Street             97750322
       524 Mareen     Joans           133/6 Gleniris road         97305643
```

- **UNDEFINE command**

A variable remains defined until the user exits SQL*PLUS. However, it can be cleared during the session using UNDEFINE command.

```
SQL> DEFINE department_id
DEFINE DEPARTMENT_ID   = "101" (CHAR)
```

```
SQL> UNDEFINE department_id
```

Use the DEFINE command to confirm the undefined variable.

```
SQL> DEFINE department_id
SP2-0135: symbol department_id is UNDEFINED
```

- **ACCEPT and PROMPT command**

ACCEPT command is used to create a customized prompt during run time when accepting input from the user. PROMPT command is used to display text to the user. In order to demonstrate the use of the commands, a sql script 'accept.sql' is created.

accept.sql script

```
PROMPT This query displays the Customer details for
PROMPT the customer you supply
PROMPT ===========================================


ACCEPT customer PROMPT 'Please provide the customer name: '

select cid, cname, cadd, cphone
from customers
where cname = '&customer';
```

To run the script, enter the following command.

```
SQL> @ accept.sql
```

The screen then displays like the following and prompt the user to enter customer name with the customized message.

```
This query displays the Customer details for
the customer you supply
===========================================
Please provide the customer name: Hilton
```

Once the user inputs the customer name, SQL*PLUS will display the following information.

```
old   3: where cname = '&customer'
new   3: where cname = 'Hilton'

       CID CNAME      CADD                      CPHONE
---------- ---------- -------------------- ----------
     50003 Hilton     38 Mcmaster court      97580322
```

3.3 Passing Values Into a Script File

When the substitution variables are used in a script file, the substitution variable values can be submited when invoking the script. The values are assigned to the variables according to its position.

For example, order.sql is created to produce order reports by item id and order date.

order.sql script

```
select * from orders
where itemid = &1
and ord_date = '&2';
```

Execute the file by passing values in the command line. The first value after the script file name is substituted for &1 and the second value is substituted for &2.

```
SQL> start order.sql 90003 13-apr-02
```

The SQL*PLUS then displays the following report without prompting the user to input values.

```
old   2: where itemid = &1
new   2: where itemid = 90003
old   3: and ord_date = '&2'
new   3: and ord_date = '13-apr-02'

   ORDERID ORD_DATE          CID     ITEMID    ORD_QTY DELVER_DA        EID
---------- --------- ---------- ---------- ---------- --------- ----------
      3331 13-APR-02      50005      90003         10 20-APR-02        521
```

4. SQL FUNCTIONS

4.1 Single Row Functions

- **Single-Row Character Functions**

CONCAT

This function concatenates two strings, which serves the same function as the operator ||. For example, display employee's first and last name together as EmployeeName.

```
SQL> select concat(efname, elname) EmployeeName from employees;

EMPLOYEENAME
------------------------
JimPeterson
ClairManson
ToddSmith
RebeccaGwen
MareenJoans
MillerChang
```

LENGTH

This function returns the numeric length of the character specified. For example, display the length of the item description of item '90003'.

```
SQL> select LENGTH(itemdesc)
  2  from items
  3  where itemid = '90003';

LENGTH(ITEMDESC)
----------------
              14
```

LOWER

This function returns character strings in lowercase. In the example, the LOWER function returns the customer's name in lower case even though they were stored in the database with the first letter being capital.

```
SQL> select cid, LOWER(cname) Customer
  2  from customers;

       CID CUSTOMER
---------- ----------
     50001 colesmyer
     50002 samsung
     50003 hilton
     50004 evian
     50005 monash
```

SUBSTR

This function returns certain portion of a character. The portion of the character is specified using integers. In the example, the SUBSTR function returns a portion of

the item description for item 90003, which is 4 characters long and begins at the first position.

```
SQL> select SUBSTR(itemdesc,1,4)
  2  from items
  3  where itemid = '90003';

SUBS
----
sony
```

REPLACE

This function performs sub string search and replace it. For example, modify the brand for item '90002' from 'Hitachi' to 'Mitsubishi'.

```
SQL> Select REPLACE(itemdesc, 'HITACHI', 'MITSUBISHI')
  2  from items
  3  where itemid = '90002';

REPLACE(ITEMDESC,'HITACHI','MITSUBISHI')
----------------------------------------------------
MITSUBISHI monitor 19inch
```

UPPER

This function returns the character strings in upper case. In the example, the UPPER function returns department name in the form of capital letters even though they are stored in lower case within the database.

```
SQL> select did, UPPER(depart_name)
  2  from departments;

Dept
  ID UPPER(DEPA
---- ----------
 101 SALES
 102 ACCOUNTING
 103 MARKETING
```

- **Single-Row Numeric Functions**

The single-row numeric functions are used to manipulate numeric data and return numeric values.

ABS

This function returns the absolute value of the number specified. For example,

```
SQL> select ABS(-456) FROM dual;

 ABS(-456)
----------
       456
```

ROUND

```
SQL> SELECT ROUND(78.631,2), ROUND(78.631,0),
  2  ROUND(78.631,-1)
  3  FROM DUAL;

ROUND(78.631,2) ROUND(78.631,0) ROUND(78.631,-1)
--------------- --------------- ----------------
          78.63              79               80
```

TRUNC

```
SQL> SELECT TRUNC(78.631,2), TRUNC(78.631,0),
  2  TRUNC(78.631,-1)
  3  FROM DUAL;

TRUNC(78.631,2) TRUNC(78.631,0) TRUNC(78.631,-1)
--------------- --------------- ----------------
          78.63              78               70
```

POWER

```
SQL> select POWER(3, 8) FROM DUAL;

POWER(3,8)
----------
      6561
```

- **Single-Row Date Functions**

Single-row date functions are used to operate on date datatypes. It usually returns a date value.

ADD_MONTHS

This function adds or minuses a number of months to / from a date.

```
SQL> select orderid, delver_date, Add_months(delver_date,6) plus6mon,
add_months(delver_date, -3) mi
nus3mon
  2  from orders;

   ORDERID DELVER_DA PLUS6MON  MINUS3MON
---------- --------- --------- ---------
      3331 20-APR-02 20-OCT-02 20-JAN-02
      3332 20-APR-02 20-OCT-02 20-JAN-02
      3333 22-APR-02 22-OCT-02 22-JAN-02
      3334 22-APR-02 22-OCT-02 22-JAN-02
```

MONTHS_BETWEEN

This function returns the number of month(s) between two dates specified. For example, display the number of months between the order date of order '3335' and the date it was delivered.

```
SQL> SELECT MONTHS_BETWEEN(delver_date, ord_date)
  2  from orders
  3  where orderid = '3335';

MONTHS_BETWEEN(DELVER_DATE,ORD_DATE)
------------------------------------
                          2.19354839
```

NEXT_DAY

The function returns the next day following the date specified. For example, display the date 2 weeks after the next Wednesday of the ordering date for order number '3331'.

```
SQL> select NEXT_DAY(ord_date, 'Wednesday')+14 Delivery
  2  from orders
  3  where orderid = '3331';

DELIVERY
---------
01-MAY-02
```

LAST_DAY

This function returns the last day of the month of the specified date. In the example, the LAST_DAY function returns the last day of the month of the delivery date for order '3333'.

```
SQL> select delver_date, LAST_DAY(delver_date)LastDay from orders
  2  where orderid = '3333';

DELVER_DA LASTDAY
--------- ---------
22-APR-02 30-APR-02
```

ROUND

This function is used to round date or time.

Example 1: Round the month of the date specified.

```
SQL> select ord_date, ROUND(ord_date, 'MONTH') round_month
  2  from orders
  3  where orderid = '3333';

ORD_DATE  ROUND_MON
--------- ---------
15-APR-02 01-APR-02
```

Example 2: Round the year of the date specified.

```
SQL> select ord_date, ROUND(ord_date, 'YEAR') round_year
  2  from orders
  3  where orderid = '3333';

ORD_DATE  ROUND_YEA
--------- ---------
15-APR-02 01-JAN-02
```

TRUNC

This function truncates a given date/time.

Example 1: Truncate the month of the given date.

```
SQL> select ord_date, TRUNC(ord_date, 'MONTH') round_month
  2  from orders
  3  where orderid = '3333';

ORD_DATE  ROUND_MON
--------- ---------
15-APR-02 01-APR-02
```

Example 2: Truncate the year of the given date.

```
SQL> select ord_date, TRUNC(ord_date, 'YEAR') round_year
  2  from orders
  3  where orderid = '3333';

ORD_DATE  ROUND_YEA
--------- ---------
15-APR-02 01-JAN-02
```

SYSDATE

This function returns the current date.

```
SQL> select SYSDATE from dual;

SYSDATE
---------
05-MAY-02
```

- **Single-Row Conversion Functions**

TO_CHAR

This function converts and formats a date or numbers into character string.

- **TO_CHAR FUNCTION WITH DATE**

In the example, the TO_CHAR function converts the employees' birthday from its default format ('dd/mm/yyyy') to ('dd Month yyyy').

```
SQL> SELECT efname||' '||elname as "Employees",
  2  TO_CHAR(edob, 'fmDD Month YYYY') Birthday
  3  FROM employees;

Employees                 BIRTHDAY
------------------------- -----------------
Jim Peterson              2 November 1978
Clair Manson              30 March 1974
Todd Smith                17 April 1976
Rebecca Gwen              20 January 1975
Mareen Joans              13 April 1979
Miller Chang              3 July 1971
```

- **TO_DATE FUNCTION WITH NUMBERS**

In the example, the TO_CHAR function converts the item price and displays it with a floating dollar sign $.

```
SQL> select itemid, TO_CHAR(price, '$99,999') Price
  2  from items;

    ITEMID PRICE
---------- --------
     90001     $900
     90002   $1,500
     90003      $90
     90004      $40
     90005     $120
     90006       $8
```

- **TO_DATE**

This function is used to convert a character string to a specified date format.
For example, display all the orders that will be delivered on April 22, 2002. Even though this is not the default format, it can still be converted by specifying the format style.

```
SQL> SELECT orderid, delver_date
  2  FROM orders
  3  WHERE delver_date =
  4  TO_DATE('April 22, 2002','Month dd, YYYY');

   ORDERID DELVER_DA
---------- ---------
      3333 22-APR-02
      3334 22-APR-02
```

4.2 Group Functions

Group functions operate on sets of rows, which might be the whole table, or the table split into groups. The functions return a value based on a number of inputs. The exact number of inputs is not determined until the query is executed and all rows are fetched. This differs from single row functions, in which the number of inputs is known at parse time before the query is executed. (Morison and Morison, 2000)

Types of Group Functions

The types of group functions include:

- AVG
- COUNT
- MAX
- MIN
- STDDEV
- SUM
- VARIANCE

In this example, the query retrieves information about the average, highest, lowest and the sum of total amount for every order.

```
SQL> select AVG(ord_qty*price) AvgTotal, MAX(ord_qty*price) MaxTotal,
  2  MIN(ord_qty*price) MinTotal, SUM(ord_qty*price) SumTotal
  3  from orders, items
  4  where orders.itemid = items.itemid;

  AVGTOTAL   MAXTOTAL   MINTOTAL   SUMTOTAL
---------- ---------- ---------- ----------
    4425.2      15000        376      22126
```

MIN and MAX can also operate on the character datatype. In the following example, MIN function returns employee's last name that is the first in an alphabetised list of all employees. MAX function returns employee's last name that is the last in the alphabetised list.

```
SQL> select MIN(elname), MAX(elname)
  2  from employees;

MIN(ELNAME)     MAX(ELNAME)
--------------- ---------------
Chang           Smith
```

This example displays the number of orders processed by employee '524' using the COUNT function.

```
SQL> select count(*)
  2  from orders
  3  where eid = '524';

  COUNT(*)
----------
         2
```

The following example using STDDEV function to return the statistical standard deviation and VARIANCE function to return the statistical variance.

```
SQL> select STDDEV(cpcode),VARIANCE(cpcode)
  2  from customers;

STDDEV(CPCODE) VARIANCE(CPCODE)
-------------- ----------------
    793.272463         629281.2
```

- **GROUP BY Clause**

Example 1: display the total number of the orders handled by the employees using GROUP BY clause.

```
SQL> select eid, count(*)
  2  from orders
  3  GROUP BY eid;

       EID   COUNT(*)
---------- ----------
       520          2
       521          1
       524          2
```

Example 2: This is an example of illegal query. Without the GROUP BY clause, an error message would occur.

```
SQL> select eid, count(*)
  2  from orders;
select eid, count(*)
       *
ERROR at line 1:
ORA-00937: not a single-group group function
```

Example 3: This is an example of groups within groups where it queries information regarding the number of orders made for each customer handled by the employee.

```
SQL> select eid, cid, count(*)
  2  from orders
  3  GROUP BY eid, cid;

       EID        CID   COUNT(*)
---------- ---------- ----------
       520      50001          1
       520      50003          1
       521      50005          1
       524      50002          1
       524      50004          1
```

- **HAVING Clause**

The main function of HAVING clause is to limit grouped data to be displayed. It is normally used when rows are restricted based on the result of a group function.

Example 1: display the employees' id that handled more than $2000 worth of orders.

```
SQL> select eid
  2  from orders, items
  3  where orders.itemid = items.itemid
  4  group by eid
  5  HAVING SUM(ord_qty*price) > 2000;

       EID
----------
       520
       524
```

Example 2: The following is an illegal query. When using a group function to limit the output without the having clause, an error message occurs.

```
SQL> select eid
  2  from orders, items
  3  where orders.itemid = items.itemid and
  4  SUM(ord_qty*price) > 2000
  5  group by eid;
SUM(ord_qty*price) > 2000
*
ERROR at line 4:
ORA-00934: group function is not allowed here
```

5. MULTIPLE TABLE QUERIES

5.1 Equality Joins

An equality join, also known as an inner join or an equijoin, uses an equality operator (=) to link two different tables.

```
SQL> Select orderid, orders.cid, cname AS "Customers", (ord_qty*price) as
"Total"
  2   from orders, customers, items
  3   where orders.itemid = items.itemid and
  4  orders.cid = customers.cid;

   ORDERID        CID Customers       Total
---------- ---------- ---------- ----------
      3331      50005 Monash            900
      3332      50001 ColesMyer         376
      3333      50003 Hilton           4500
      3334      50002 Samsung          1350
      3335      50004 Evian           15000
```

5.2 Non-Equality Joins

The difference between Equality joins and non-equality joins is that the non-equality joins are joining tables with no columns correspond to each other. The relationship can be established using the BETWEEN key word.

For example, the following is performance table, which evaluates employees' sales performance. However, this table has no relationship with any other tables. In order to evaluate the sales rating for employee '521', BETWEEN keyword is used to link the performance table with other tables. To evaluate the performance, the total sale for employee '521' must fall in one of the pair of the low and high sales range.

Performance Table

Rating	Lowsale	Highsale
A	12001	20000
B	9001	12000
C	6001	9000
D	3001	6000
E	1001	3000
F	0	1000

```
SQL> SELECT o.EID, EFNAME ||' '||ELNAME AS "EMPLOYEES", (ord_qty*price) as
"Total_Sale", Rating
  2  from employees e, orders o, items i, Performance
  3  where e.eid = o.eid and i.itemid = o.itemid
  4  and (ord_qty*price) BETWEEN lowsale AND highsale
  5  and o.eid = '521';

       EID EMPLOYEES                 Total_Sale R
---------- ------------------------- ---------- -
       521 Clair Manson                     900 F
```

5.3 Outer Joins

When a row does not satisfy a join condition, the row will not appear in the query result. Outer joins operator (+) is therefore used in the join condition to return the missing row(s). (Ault, 2001)

For example, display all the employees' id, employee names, and department id and department name they belong to.

If using normal equality joins, 5 results are returned. It does not return those employees that have not been assigned a department id.

```
SQL> select eid, efname, elname, e.did, depart_name
  2  from employees e, departments d
  3  where e.did = d.did;

       EID EFNAME     ELNAME                DID DEPART_NAME
---------- ---------- --------------- ---------- ----------------
       520 Jim        Peterson               101 Sales
       521 Clair      Manson                 101 Sales
       522 Todd       Smith                  103 Marketing
       523 Rebecca    Gwen                   102 Accounting
       524 Mareen     Joans                  101 Sales

5 rows selected.
```

However, using outer joins return 6 results including employee '525' who do not have a department id.

```
SQL> select eid, efname, elname, e.did, depart_name
  2  from employees e, departments d
  3  where e.did = d.did (+);

       EID EFNAME     ELNAME                DID DEPART_NAME
---------- ---------- --------------- ---------- --------------------
       520 Jim        Peterson               101 Sales
       521 Clair      Manson                 101 Sales
       522 Todd       Smith                  103 Marketing
       523 Rebecca    Gwen                   102 Accounting
       524 Mareen     Joans                  101 Sales
       525 Miller     Chang

6 rows selected.
```

5.4 Set Operators

Set operators can be used to select data from multiple tables. It combines the results of two queries into one. (Ault, 2001) There are four set operators in Oracle:

- UNION
- UNION ALL
- INTERSECT
- MINUS

The following examples use 2 queries to illustrate the set operators. One of the queries retrieves employees whose order(s) were delivered on 20-APR-02 and the other returns employees whose order(s) were delivered on 22-APR-02.

When using UNION operator to combine the two queries, only unique values are returned. There are no repeating values.

```
SQL> select O.eid, efname, elname
  2  from orders o, employees e
  3  where o.eid = e.eid and delver_date = '20-APR-02'
  4  UNION
  5  select O.eid, efname, elname
  6  from orders o, employees e
  7  where o.eid = e.eid and delver_date = '22-APR-02';

       EID EFNAME     ELNAME
---------- ---------- ---------------
       520 Jim        Peterson
       521 Clair      Manson
       524 Mareen     Joans

3 rows selected.
```

When using UNION ALL operator to combine the two queries, all rows from both queries are returned.

```
SQL> select O.eid, efname, elname
  2  from orders o, employees e
  3  where o.eid = e.eid and delver_date = '20-APR-02'
  4  UNION ALL
  5  select O.eid, efname, elname
  6  from orders o, employees e
  7  where o.eid = e.eid and delver_date = '22-APR-02';

       EID EFNAME     ELNAME
---------- ---------- ---------------
       521 Clair      Manson
       520 Jim        Peterson
       520 Jim        Peterson
       524 Mareen     Joans

4 rows selected.
```

When INTERSECT operator is used, only values of employee(s) who have order(s) delivered on 20-APR-02 and 22-APR-02 are returned.

```
SQL> select O.eid, efname, elname
  2  from orders o, employees e
  3  where o.eid = e.eid and delver_date = '20-APR-02'
  4  INTERSECT
  5  select O.eid, efname, elname
  6  from orders o, employees e
  7  where o.eid = e.eid and delver_date = '22-APR-02';

       EID EFNAME     ELNAME
---------- ---------- ---------------
       520 Jim        Peterson

1 row selected.
```

When MINUS operator is used, only values of employee(s) whose order(s) were delivered on 20-APR-02 but not on 22-APR-02 are returned.

```
SQL> select O.eid, efname, elname
  2  from orders o, employees e
  3  where o.eid = e.eid and delver_date = '20-APR-02'
  4  MINUS
  5  select O.eid, efname, elname
  6  from orders o, employees e
  7  where o.eid = e.eid and delver_date = '22-APR-02';

       EID EFNAME     ELNAME
---------- ---------- ---------------
       521 Clair      Manson

1 row selected.
```

6. SUBQUERIES

6.1 Single – Row Subqueries

Single-row subquery only returns one row of result and uses the following single-row operators. (Ault, 2001)

Operator	Meaning
=	Equal to
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to
<>	Not equal to

Example: display the order(s) that has the highest total and the employee ID who processes that order. The subquery used in the example is to query the highest price in total from all the orders. The parent query is then executed with the results returned from the subquery.

```
SQL> select orderid, eid
  2  from orders o, items i
  3  where o.itemid = i.itemid and
  4  (ord_qty*price) = (select MAX(ord_qty*price) from
  5     orders o, items i where
  6     o.itemid = i.itemid);

   ORDERID        EID
---------- ----------
      3335        524
```

If the query returns more than one row, an error message would occur.
Example: display order(s) that is delivered after '20/04/2002'.

```
SQL> select orderid, eid
  2  from orders
  3  where delver_date >
  4  (select delver_date from orders
  5  where delver_date = TO_DATE('20/04/2002', 'DD/MM/YYYY'));
(select delver_date from orders
 *
ERROR at line 4:
ORA-01427: single-row subquery returns more than one row
```

6.2 Multiple – Row Subqueries

Multiple – Row subqueries, on the contrary, return one or more rows of results from the subqueries. The operator(s) can be used in a multiple – row subquery include:

- IN
- ANY
- ALL
- EXIST

From the previous example in single-row subqueries, if there is more than 1 order delivered after '20/04/2002', the query will fail. But if convert the query into a multiple-row subquery using **IN** operator, the results are returned.

```
SQL> select orderid, eid
  2  from orders
  3  where delver_date IN
  4  (select delver_date from orders
  5  where delver_date = TO_DATE('20/04/2002', 'DD/MM/YYYY'));

   ORDERID        EID
---------- ----------
      3331        521
      3332        520
```

- **ALL operator.**

Example: Display all the order(s) that has a total price below the average sale of every employee.

```
SQL> select orderid, o.itemid, ord_qty, eid
  2  from orders o, items i
  3  where o.itemid = i.itemid
  4  and (ord_qty*price) < ALL (select AVG(ord_qty*price)
  5     from orders o, items i
  6     where o.itemid = i.itemid
  7     group by eid);
```

The results returned from the subquery.

```
SQL> select AVG(ord_qty*price)
  2     from orders o, items i
  3     where o.itemid = i.itemid
  4     group by eid;

AVG(ORD_QTY*PRICE)
------------------
              2438
               900
              8175
```

The parent query compares all its returned values with the subquery values, and returns the following result.

```
   ORDERID     ITEMID    ORD_QTY        EID
---------- ---------- ---------- ----------
      3332      90006         47        520
```

- **ANY operator .**

Example: Display all the order(s) that have a total price less than that of the average total price.

```
SQL> select orderid, o.itemid, ord_qty
  2  from orders o, items i
  3  where o.itemid = i.itemid
  4  and (ord_qty*price) < ANY (select AVG(ord_qty*price)
  5     from orders o, items i
  6     where o.itemid = i.itemid group by o.itemid);

   ORDERID     ITEMID    ORD_QTY
---------- ---------- ----------
      3331      90003         10
      3332      90006         47
      3333      90001          5
      3334      90003         15
```

The difference between ANY operator and ALL operator is that the ANY operator is used to compare values to every value returned by the subquery. As long as the parent value (ie. Ord_qty*price) satisfies any of the subquery values, the result will then be returned. On the contrary, the ALL operator is also used to compare every value returned by the subquery. However, the parent value has to satisfy all the subquery values. The following uses the previous query but with an ALL operator instead, the result is different from using ANY operator. No row is returned.

```
SQL> select orderid, o.itemid, ord_qty
  2  from orders o, items i
  3  where o.itemid = i.itemid
  4  and (ord_qty*price) < ALL(select AVG(ord_qty*price)
  5     from orders o, items i
  6     where o.itemid = i.itemid group by o.itemid);

no rows selected
```

7. DATA MANIPULATION LANGUAGE (DML) STATEMENTS

7.1 INSERT statement

- **Insert a New Row to a Table**

For example, insert a new row to table orders.

```
SQL> INSERT INTO orders
  2  VALUES ('3336', to_date('13/05/2002', 'dd/mm/yyyy'), '50002', '90003',
'12',to_date('20/05/2002','dd/m
m/yyyy'), '521');

1 row created.
```

- **Inserting Values by using Substitution Variables**

Using substitution variables within the INSERT statement allows the user to add values interactively.

```
SQL> INSERT INTO orders VALUES
  2  ('3337', sysdate, '&cid', '&itemid', '&qty','&deliverydate', '&eid');

Once the INSERT statement is executed, SQL*PLUS will prompt the user for the
values of the variables: customer id, itemid, quantity, delivery date,
employee id.
Enter value for cid: 50003
Enter value for itemid: 90006
Enter value for qty: 38
Enter value for deliverydate: 23-May-02
Enter value for eid: 524
old    2: ('3337', sysdate, '&cid', '&itemid', '&qty','&deliverydate',
'&eid')
new   2: ('3337', sysdate, '50003', '90006', '38','23-May-02', '524')

1 row created.
```

Confirm the additional row in the table

```
SQL> select * from orders
  2  where orderid = '3337';

   ORDERID ORD_DATE         CID     ITEMID    ORD_QTY DELVER_DA        EID
---------- --------- ---------- ---------- ---------- --------- ----------
      3337 21-APR-02      50003      90006         38 23-MAY-02        524
```

- **Copying Rows from other Table**

Rows can be added to a table based on the values in the existing tables. In this case, the VALUE key word is omitted and a subquery is used.

Example: ORDERPRICE table is created to store prices for each order. It contains the following columns.

SALES_DPART TABLE	
Name	**Type**
ORDERID	NUMBER(4)
ITEMID	NUMBER(5)
ORD_QTY	NUMBER(5)
PRICE	NUMBER

Copy the selected rows from the existing orders table and items table to the ORDERPRICE table.

```
SQL> INSERT INTO orderprice
  2  SELECT orderid, o.itemid, ord_qty, (ord_qty*price)
  3  from orders o, items i
  4  where o.itemid = i.itemid;

7 rows created.
```

Confirm the newly inserted rows in the SALES_DPART table.

```
SQL> select * from orderprice;

   ORDERID     ITEMID    ORD_QTY      PRICE
---------- ---------- ---------- ----------
      3331      90003         10        900
      3332      90006         47        376
      3333      90001          5       4500
      3334      90003         15       1350
      3335      90002         10      15000
      3336      90004         12        480
      3337      90006         38        304

7 rows selected.
```

7.2 UPDATE statement

- **Updating Rows**

Example: updating the itemid for order 3336 in orders table.

```
SQL> UPDATE orders
  2  SET itemid = 90004
  3  WHERE orderid = 3336;

1 row updated.
```

Confirm the changes to order 3336.

```
SQL> select * from orders
  2  where orderid = 3336;

   ORDERID ORD_DATE         CID     ITEMID    ORD_QTY DELVER_DA        EID
---------- --------- ---------- ---------- ---------- --------- ----------
      3336 13-MAY-02      50002      90004         12 20-MAY-02        521
```

- **Updating All Rows**

Example: give a 5 % discount for every order.

```
SQL> UPDATE orderprice
  2  SET price = price * 0.95;

7 rows updated.
```

Confirm the changes on the price.

```
SQL> select * from orderprice;

   ORDERID     ITEMID    ORD_QTY      PRICE
---------- ---------- ---------- ----------
      3331      90003         10        855
      3332      90006         47      357.2
      3333      90001          5       4275
      3334      90003         15     1282.5
      3335      90002         10      14250
      3336      90004         12        456
      3337      90006         38      288.8

7 rows selected.
```

7.3 DELETE statement

- **Deleting Rows**

Example: delete order 3337 from ORDERPRICE table.

```
SQL> DELETE FROM orderprice
  2  where orderid = 3337;

1 row deleted.
```

- **Deleting All Rows**

Example: eliminate all rows in the ORDERPRICE table.

```
SQL> DELETE FROM ORDERPRICE;

7 rows deleted.
```

Confirm the deletion.

```
SQL> SELECT * FROM ORDERPRICE;

no rows selected
```

- **Integrity Constraint Error**

The integrity constraint error occurs when the table the user tries to delete contains a primary key that is used as a foreign key in the other table.

Example: delete all rows in the CUSTOMERS table.

```
SQL> DELETE FROM CUSTOMERS;

DELETE FROM CUSTOMERS
*
ERROR at line 1:
ORA-02292: integrity constraint (IVY.ORDERS_CID_FK) violated - child record
found
```

Because cid in the CUSTOMERS table is referenced in the orders table, the rows cannot be deleted. In order to delete rows from CUSTOMERS table, the child records in ORDERS table need to be deleted first. However, if the referential integrity constraint contains the ON DELETE CASCADE option, then the selected row and its children are deleted from their respective tables.

8. TRANSACTION CONTROL

8.1 COMMIT command

Example: create a new employee.

```
SQL> insert into ivy.employees values
  2  ('526', 'Jessica', 'Taylor', to_date('23/10/1972', 'dd/mm/yyyy'), '8
York street', '98070506',
'2012', '102');

1 row created.
```

After the COMMIT command is issued, the changes are written to the database and remain permanent.

```
SQL> COMMIT;

Commit complete.
```

8.2 ROLLBACK command

Example: assume all rows are deleted from the orders table accidentally. Correct the mistake and restore all the data.

```
SQL> DELETE FROM orders;

7 rows deleted.
```

```
SQL> select * from orders;

no rows selected
```

After ROLLBACK command is issued, the data deletion is undone. All rows remain in orders table.

```
SQL> ROLLBACK;

Rollback complete.
```

```
SQL> select * from orders;

   ORDERID ORD_DATE         CID     ITEMID    ORD_QTY DELVER_DA        EID
---------- --------- ---------- ---------- ---------- --------- ----------
      3331 13-APR-02      50005      90003         10 20-APR-02        521
      3332 14-APR-02      50001      90006         47 20-APR-02        520
      3333 15-APR-02      50003      90001          5 22-APR-02        520
      3334 16-APR-02      50002      90003         15 22-APR-02        524
      3335 16-APR-02      50004      90002         10 22-JUN-02        524
      3336 13-MAY-02      50002      90004         12 20-MAY-02        521
      3337 21-MAY-02      50003      90006         38 23-MAY-02        524

7 rows selected.
```

8.3 SAVEPOINT command

Example: Create a savepoine named ‘before_update’.

```
SQL> SAVEPOINT before_update;

Savepoint created.
```

Update item price by increasing 10 %.

```
SQL> UPDATE items
  2  SET price = price * 1.1;

6 rows updated.
```

Verify the changes on price.

```
SQL> select * from items;

    ITEMID ITEMDESC                          PRICE        QOH
---------- ------------------------------ ---------- ----------
     90001 HITACHI monitor 17inch              990         50
     90002 HITACHI monitor 19inch             1650         12
     90003 sony 56k modem                       99         68
     90004 Microsoft keyboard                   44         87
     90005 sony 52x CDROM drive                132         94
     90006 TDK flopy disc x 12                   9        112

6 rows selected.
```

Create another savepoint named ‘after_update’.

```
SQL> SAVEPOINT after_update;

Savepoint created.
```

Empty table ORDERS and verify the changes.

```
SQL> DELETE FROM orders;

7 rows deleted.


SQL> select * from orders;

no rows selected
```

Using ROLLBACK TO command to ‘after_update’ savepoint. By doing so, all the changes made after the savepoint was created are undone. Therefore, all rows remain in the ORDERS table.

```
SQL> ROLLBACK TO after_update;

Rollback complete.
```

```
SQL> select * from orders;

   ORDERID ORD_DATE         CID     ITEMID    ORD_QTY DELVER_DA        EID
---------- --------- ---------- ---------- ---------- --------- ----------
      3331 13-APR-02      50005      90003         10 20-APR-02        521
      3332 14-APR-02      50001      90006         47 20-APR-02        520
      3333 15-APR-02      50003      90001          5 22-APR-02        520
      3334 16-APR-02      50002      90003         15 22-APR-02        524
      3335 16-APR-02      50004      90002         10 22-JUN-02        524
      3336 13-MAY-02      50002      90004         12 20-MAY-02        521
      3337 21-MAY-02      50003      90006         38 23-MAY-02        524

7 rows selected.
```

Using ROLLBACK TO command to 'before_update' savepoint. By doing so, all the changes made after the savepoint was created are undone. Therefore, the 10% increase on price is not effective.

```
SQL> ROLLBACK to before_update;

Rollback complete.
```

```
SQL> select * from items;

    ITEMID ITEMDESC                          PRICE        QOH
---------- ------------------------------ ---------- ----------
     90001 HITACHI monitor 17inch              900         50
     90002 HITACHI monitor 19inch             1500         12
     90003 sony 56k modem                       90         68
     90004 Microsoft keyboard                   40         87
     90005 sony 52x CDROM drive                120         94
     90006 TDK flopy disc x 12                   8        112

6 rows selected.
```

9. VIEWS

9.1 Creating a View

A View is created by using the CREATE VIEW command.

Example: create a view named 'Invoice' that contains company id whose total order price is larger than $1000.

```
SQL> CREATE VIEW Invoice (company_Id, Total)
  2  AS SELECT o.cid, SUM(ord_qty*price)
  3  FROM customers c, orders o, items i
  4  where c.cid = o.cid and o.itemid = i.itemid
  5  having SUM(ord_qty*price) > 1000
  6  group by o.cid;

View created.
```

Verify the 'Invoice' view with the select statement. The 'cid' and 'SUM(ord_qty*price)' columns are renamed in the view definition.

```
SQL> select * from invoice;

COMPANY_ID      TOTAL
---------- ----------
     50002       1830
     50003       4804
     50004      15000
```

Error message**:** The message occurred when attempting to update one row within the view. This is due to the reason that the invoice view definition contains a GROUP BY clause and GROUP function. In this case, the view cannot be inserted into, updated in, or deleted from the base tables using the view.

```
SQL> Update Invoice
  2  set total = 500
  3  where company_id = 50002;
Update Invoice
       *
ERROR at line 1:
ORA-01732: data manipulation operation not legal on this view
```

9.2 With Check Option

When a view is created using the WITH CHECK OPTION, any DML statement performed on that view should satisfy the condition in the WHERE clause of the view.

Example: create a view 'Accounting_emp' which only contains information of the employees from the accounting department, using the WITH CHECK OPTION.

```
SQL> CREATE VIEW Accounting_EMP
  2  AS SELECT eid, efname, elname, edob
  3  FROM employees
  4  WHERE did = 102
  5  WITH CHECK OPTION CONSTRAINT Acc_emp_check;

View created.
```

If attempting to insert an employee who belongs to Marketing Department into the Accounting_emp view, the following message will be generated. This is because the employee 528 does not satisfy the condition.

```
SQL> INSERT INTO Accounting_emp
  2  SELECT eid, efname, elname, edob
  3  from employees
  4  where eid = 528;
INSERT INTO Accounting_emp
            *
ERROR at line 1:
ORA-01402: view WITH CHECK OPTION where-clause violation
```

9.3 With Read Option

When a view is created using the WITH READ ONLY option, any attempt to perform a DML statement will result in an error message.

Example: recreate the above example using the WITH READ ONLY option.

```
SQL> CREATE OR REPLACE VIEW Accounting_EMP
  2  AS SELECT eid, efname, elname, edob
  3  FROM employees
  4  WHERE did = 102
  5  WITH READ ONLY;

View created.
```

An error message is generated when attempting to delete one row from the accounting_emp view.

```
SQL> Delete from accounting_emp
  2  where eid = 523;
Delete from accounting_emp
            *
ERROR at line 1:
ORA-01752: cannot delete from view without exactly one key-preserved table
```

9.4 Dropping a View

View is dropped using DROP VIEW command.

Example: drop accounting_emp view.

```
SQL> DROP VIEW accounting_emp;

View dropped.
```

10. SEQUENCES

10.1 Creating a Sequence

Example: Create a sequence named 's_order_id' to be used for the orderid column of the ORDERS table. The sequence starts at 3338. Do not allow caching and do not allow the sequence to cycle.

```
SQL> CREATE SEQUENCE s_order_id
  2  INCREMENT BY 1
  3  START WITH 3338
  4  MAXVALUE 999999
  5  NOCACHE
  6  NOCYCLE;

Sequence created.
```

Create a sequence named 's_cid' to be used for the cid column of the CUSTOMERS table.

```
SQL> CREATE SEQUENCE s_cid
  2  INCREMENT BY 1
  3  START WITH 50005
  4  MAXVALUE 999999
  5  NOCACHE
  6  NOCYCLE;

Sequence created.
```

Create a sequence named 's_item_id' to be used for the itemid column of the ITEMS table.

```
SQL> CREATE SEQUENCE s_item_id
  2  INCREMENT BY 1
  3  START WITH 90007
  4  MAXVALUE 999999
  5  NOCACHE
  6  NOCYCLE;

Sequence created.
```

Create a sequence named 's_emp_id' to be used for the eid column of the EMPLOYEES table.

```
SQL> CREATE SEQUENCE s_emp_id
  2  INCREMENT BY 1
  3  START WITH 527
  4  MAXVALUE 999999
  5  NOCACHE
  6  NOCYCLE;

Sequence created.
```

10.2 Confirming a Sequence

The sequences created can be verified in the USER_SEQUENCES data dictionary table.

```
SQL> SELECT sequence_name, min_value, max_value, increment_by,
  2  last_number
  3  from USER_SEQUENCES;



SEQUENCE_NAME                   MIN_VALUE  MAX_VALUE INCREMENT_BY LAST_NUMBER
------------------------------ ---------- ---------- ------------ -----------
S_CID                                   1     999999            1       50005
S_EMP_ID                                1     999999            1         527
S_ITEM_ID                               1     999999            1       90007
S_ORDER_ID                              1     999999            1        3338

4 rows selected.
```

10.3 Using a Sequence

Example: insert a new employee with the next available eid sequence value using NEXTVL.

```
SQL> insert into employees values
  2     (s_emp_id.NEXTVAL,   'Nancy',   'Spenser',   to_date('04/03/1970',
'dd/mm/yyyy'), '103/2 Queen street
', '96603215', '3010', '103');

1 row created.
```

View the current value for the s_emp_id sequence with CURRVAL.

```
SQL> select s_emp_id.CURRVAL
  2  from sys.dual;

   CURRVAL
----------
       528
```

10.4 Modifying a Sequence

Sequences can be modified by using ALTER SEQUENCE command.

Example: modify sequence ‘s_emp_id” to have maximum eid value of 8888.

```
SQL> ALTER SEQUENCE s_emp_id
  2  Maxvalue 8888;

Sequence altered.
```

10.5 Removing a Sequence

Sequences are removed from the data dictionary by using DROP SEQUENCE command.

Example: remove ‘s_item_id’ sequence

```
SQL> DROP SEQUENCE s_item_id;

Sequence dropped.
```

11. PL / SQL

PL/SQL is a block – structured language. A PL/SQL program may contain one or more blocks. Each block may be divided into three sections. (Ault, 2001)

```
Declare
     <declarations section>
Begin
     <executable commands>
  Exception
  <exception handling>
End;
```

11.1 Declarations Section

All variables must be declared prior to use. This includes variables used in the executable and exception part of the block. The basic syntax for declaring variables and constants:

```
identifier [CONSTANT] datatype [NOT NULL]
           [:= | DEFAULT expr];
```

Example:

```
Declare
    V_delver_date  DATE;
    V_total_sale   NUMBER;
    V_orderid      NUMBER(4);
    V_itemname     VARCHAR2(20);
    V_order_date   v_delver_date%Type
```

The above example declares a variable to store the delivery date of an order (v_delver_date) with DATE datatype, v_total_sale variable to store the total sales figure with NUMBER datatype, V_ordered variable to store the order id with NUMBER data type and V_itemname variable to store item name() with VARCHAR2 datatype. The V_order_date stores the order date of an order and has the same datatype as v_delver_date.

11.2 Executable Commands Section

- **DML Statement in PL/SQL**

INSERT statement

Example: Insert a new row in the ORDERS table with the next available sequential orderid. The sequence ‘s_order_id’ created from the previous section is used.

```
SQL> DECLARE
  2    v_orderid number(4);
  3  BEGIN
  4    SELECT    s_order_id.NEXTVAL
  5    INTO    v_orderid
  6    FROM    dual;
  7      INSERT INTO orders(orderid, ord_date, cid, itemid, ord_qty,
delver_date, eid)
  8       VALUES(v_orderid, to_date('23-May-2002', 'dd-mm-yyyy'), 50002,
90004, 43, to_date('30-May-2002', 'dd-mm-yyyy'), 520);
  9  END;
 10  /

PL/SQL procedure successfully completed.
```

Verify the new inserted row in the ORDERS table.

```
SQL> select * from orders;

   ORDERID ORD_DATE          CID     ITEMID    ORD_QTY DELVER_DA        EID
---------- --------- ---------- ---------- ---------- --------- ----------
      3331 13-APR-02      50005      90003         10 20-APR-02        521
      3332 14-APR-02      50001      90006         47 20-APR-02        520
      3333 15-APR-02      50003      90001          5 22-APR-02        520
      3334 16-APR-02      50002      90003         15 22-APR-02        524
      3335 16-APR-02      50004      90002         10 22-JUN-02        524
      3336 13-MAY-02      50002      90004         12 20-MAY-02        521
      3337 21-MAY-02      50003      90006         38 23-MAY-02        524
      3338 23-MAY-02      50002      90004         43 30-MAY-02        520

8 rows selected.
```

UPDATE statement

Example: Increase the price for item 90003 by 10%.

```
SQL> DECLARE
  2    v_percentage  number := 0.1;
  3  BEGIN
  4    UPDATE Items
  5    SET  price = price * (1 + v_percentage)
  6    WHERE itemid = 90003;
  7  END;
  8  /

PL/SQL procedure successfully completed.
```

Verify the changes in the ITEMS table. The price has increased from 90 to 99.

```
SQL> select * from items
  2  where itemid = 90003;

    ITEMID ITEMDESC                         PRICE        QOH
---------- ------------------------------ ---------- ----------
     90003 sony 56k modem                         99         68
```

DELETE statement

Example: delete order 3338 from ORDERS table.

```
SQL> DECLARE
  2    v_orderid   orders.orderid%TYPE := 3338;
  3  BEGIN
  4    DELETE  FROM orders
  5    WHERE  orderid = v_orderid;
  6  END;
  7  /

PL/SQL procedure successfully completed.
```

Verify the deletion in the ORDERS table.

```
SQL> select * from orders
  2  where orderid = 3338;

no rows selected
```

11.3 Conditional Logic

- **IF statement**

Example: find out if order 3335 has been delivered to the customer.

In this case, 2 variables are declared. 'v_delver_date' holds the value of delve_date of order 3335 from the orders table whilst 'Today' variable has been assigned 'sysdate' value. If the first condition which is 'Today > v_delver_date is true, then the screen displays 'The order has been delivered'. If the second condition, which is 'Today < v_delver_date' is true, then the screen displays ' The order is not yet delivered'. If none of the condition is true, then the output is 'The order is delivered today'.

```
SQL> DECLARE
  2    v_delver_date   DATE;
  3    Today DATE;
  4  BEGIN
  5    Today := sysdate;
  6
  7    select delver_date
  8    INTO v_delver_date
  9    from orders
 10    where orderid = 3335;
 11
 12    If Today > v_delver_date THEN
 13   DBMS_OUTPUT.PUT_LINE('The order has been delivered.');
 14    ELSIF
 15       Today < v_delver_date THEN
 16   DBMS_OUTPUT.PUT_LINE('The order is not yet delivered.');
 17    ELSE
 18       DBMS_OUTPUT.PUT_LINE('The order is delivered today.');
 19    END IF;
 20  END;
 21  /
The order is not yet delivered.

PL/SQL procedure successfully completed.
```

11.4 Cursors

- **Implicit Cursor**

An implicit cursor is used when the query returns only one record.

Example: create an implicit cursor to return the total sale made figure made by employee 524.

2 variables are declared. ‘v_eid’ variable holds the value of employee id 524 whilst the ‘v_total’ variable holds the value of the total sales figure derived from the select statement. When the query is executed, the output is displayed with the sales figure.

```
SQL> SET SERVEROUTPUT ON;
SQL> DECLARE
  2     v_eid NUMBER(3);
  3     v_total number;
  4  BEGIN
  5     v_eid := 524;
  6
  7     SELECT SUM(ord_qty*price)
  8     INTO v_total
  9     FROM orders o, items i
 10     WHERE i.itemid = o.itemid
 11     and eid = v_eid;
 12
 13     DBMS_OUTPUT.PUT_LINE('The total sale made by employee '||v_eid||' is
        '||v_total ||' '||'dollars.');
 14  END;
 15  /
The total sale made by employee 524 is 16789 dollars.

PL/SQL procedure successfully completed.
```

Error message:
If the query returns more than one record, the following error message is generated. In the example, the select statement queries more than one record, as a result, the error message occurrs.

```
SQL> DECLARE
  2     v_eid NUMBER(3);
  3     v_efname VARCHAR(15);
  4     v_elname VARCHAR(15);
  5
  6  BEGIN
  7
  8     SELECT eid, efname, elname
  9     INTO v_eid, v_efname, v_elname
 10     FROM employees
 11     WHERE did = 103;
 12
 13  END;
 14  /
DECLARE
*
ERROR at line 1:
ORA-01422: exact fetch returns more than requested number of rows
ORA-06512: at line 8
```

- **Explicit cursors**

Cursor FOR Loops

Example: process an explicit cursor using a cursor FOR loop that returns and displays the employee name and the department they are working within.

```
SQL> DECLARE
  2    v_did number(2);
  3    CURSOR empcursor IS
  4      SELECT eid, efname, elname, e.did, depart_name
  5      FROM   employees e, departments d
  6      WHERE e.did = d.did;
  7    emprow empcursor%ROWTYPE;
  8
  9  BEGIN
 10    FOR emprow IN empcursor LOOP
 11       DBMS_OUTPUT.PUT_LINE('Employee '|| emprow.efname||' '||
          emprow.elname ||' '||'is working in the' ||emprow.depart_name
          ||'department.' );
 12    EXIT WHEN empcursor%NOTFOUND;
 13    END LOOP;
 14  END;
 15  /

Employee Jim Peterson is working in the Sales department.
Employee Clair Manson is working in the Sales department.
Employee Todd Smith is working in the Marketing department.
Employee Rebecca Gwen is working in the Accounting department.
Employee Mareen Joans is working in the Sales department.
Employee Miller Chang is working in the Accounting department.
Employee Jessica Taylor is working in the Accounting department.
Employee Nancy Spenser is working in the Marketing department.

PL/SQL procedure successfully completed.
```

Cursors With Parameters

Example: process an explicit cursor using a cursor with a parameter to display output of orders processed by the selected employee with the total amount of each order.

In the query, cursor ‘ordcursor’ is defined for the orders. A parameter ‘p_eid’ is declared to pass employee id value to the cursor. The ‘ordcursor’ uses the ‘p_eid’ parameter to select only the orders processed by the specified parameter value. As shown in the example, ‘524’ is placed in the parameter and therefore all the order id and the total amount of each order handled by employee ‘524’ are displayed.

```
SQL> DECLARE
  2    CURSOR ordcursor (p_eid NUMBER) IS
  3      SELECT orderid, (ord_qty*price)
  4      FROM orders o, items i
  5      WHERE o.itemid = i.itemid
  6      AND  eid = p_eid;
  7
  8    v_oid number(4);
  9    v_Total number;
 10
 11
 12  BEGIN
 13    OPEN ordcursor(524);
 14    LOOP
 15    FETCH ordcursor INTO v_oid, v_total;
 16    EXIT WHEN ordcursor%NOTFOUND;
 17
 18    DBMS_OUTPUT.PUT_LINE ('Order: '||v_oid ||' '||'Total: '||v_total);
 19
 20    END LOOP;
 21    CLOSE ordcursor;
 22
 23  END;
 24  /
Order: 3334 Total: 1485
Order: 3335 Total: 15000
Order: 3337 Total: 304

PL/SQL procedure successfully completed.
```

11.5 Exception Handling Section

- **Trapping Exceptions**

Trapping Predefined Errors

In the following example, an invalid search condition is entered to select delivery date from the orders table. This resulted a run time error, which is the Oracle error code ORA-01403, and the associated error message 'no data found'.

```
SQL> DECLARE
  2    v_orderid BINARY_INTEGER;
  3    v_delverdate date;
  4
  5  BEGIN
  6    v_orderid := 90020;
  7    SELECT delver_date
  8    INTO v_delverdate
  9    FROM orders
 10    WHERE orderid = v_orderid;
 11    DBMS_OUTPUT.PUT_LINE('The delivery date for order ID '||v_orderid||'
 12    is on'||' '||to_date (v_delverdate, 'dd-mm-yyyy'));
 13
 14  END;
 15  /
DECLARE
*
ERROR at line 1:
ORA-01403: no data found
ORA-06512: at line 7
```

Invalid orderid

When an exception handler is included in the query, the error messages can be customized to be more informative and user friendly.

```
SQL> DECLARE
  2    v_orderid BINARY_INTEGER;
  3    v_delverdate date;
  4
  5  BEGIN
  6    v_orderid := 90020;
  7    SELECT delver_date
  8    INTO v_delverdate
  9    FROM orders
 10    WHERE orderid = v_orderid;
 11    DBMS_OUTPUT.PUT_LINE('The delivery date for order ID '||v_orderid||'
is on'||
 12    ' '||to_date (v_delverdate, 'dd-mm-yyyy'));
 13
 14  EXCEPTION
 15    WHEN NO_DATA_FOUND THEN
 16    DBMS_OUTPUT.PUT_LINE('Order ID specified is not valid.');
 17    DBMS_OUTPUT.PUT_LINE('Please enter a valid Order ID value.');
 18  END;
 19  /
Order ID specified is not valid.
Please enter a valid Order ID value.

PL/SQL procedure successfully completed.
```

Trapping Non-predefined Errors

In the following example, a NULL value is entered in a NOT NULL field. This resulted a run time error to be displayed, which is the Oracle error code ORA-01400, and the associated error message 'cannot insert NULL into ("IVY"."ORDERS"."ORDERID")'.

```
SQL> DECLARE
  2    v_itemid binary_integer;
  3    v_cid  binary_integer;
  4    v_ord_date date;
  5    v_ord_qty binary_integer;
  6    v_delver_date date;
  7    v_eid         binary_integer;
  8  BEGIN
  9    v_itemid := 90003;
 10    v_cid := 50005;
 11    v_ord_date := to_date('23-May-2002', 'dd-mm-yyyy');
 12    v_ord_qty := 13;
 13    v_delver_date := to_date('30-May-2002', 'dd-mm-yyyy');
 14    v_eid := 521;
 15    INSERT INTO orders VALUES(null, v_ord_date, v_cid, v_itemid,
 16    v_ord_qty,   v_delver_date, v_eid);
 17  END;
 18  /
DECLARE
*
ERROR at line 1:
ORA-01400: cannot insert NULL into ("IVY"."ORDERS"."ORDERID")
ORA-06512: at line 15
```

Unlike exception handler for defined exception, undefined exception needs to declare an exception parameter. A specific Oracle error code is also included in the declare section to associate the given exception name by using the PRAGMA EXCEPTION_INIT command.

```
SQL> DECLARE
  2    v_itemid binary_integer;
  3    v_cid  binary_integer;
  4    v_ord_date date;
  5    v_ord_qty binary_integer;
  6    v_delver_date date;
  7    v_eid         binary_integer;
  8    e_null_id EXCEPTION;
  9    PRAGMA EXCEPTION_INIT (e_null_id, -1400);
 10
 11  BEGIN
 12    v_itemid := 90003;
 13    v_cid := 50005;
 14    v_ord_date := to_date('23-May-2002', 'dd-mm-yyyy');
 15    v_ord_qty := 13;
 16    v_delver_date := to_date('30-May-2002', 'dd-mm-yyyy');
 17    v_eid := 521;
 18    INSERT INTO orders VALUES(null, v_ord_date, v_cid, v_itemid,
 19    v_ord_qty, v_delver_date, v_eid);
 20
 21  EXCEPTION
 22     WHEN e_null_id THEN
 23     DBMS_OUTPUT.PUT_LINE('Please make sure no NULL value is entered in a
 24     NOT NULL field.');
 25
 26  END;
 27  /
Please make sure no NULL value is entered in a NOT NULL field.

PL/SQL procedure successfully completed.
```

- **User-defined Exception**

Example: use a user-defined exception to advice the user the current state of the stock on hand for the specified item.

In the program, two exceptions are defined and declared. ‘e_enoughstock’ exception is raised when the quantity on hand for the specified item is more than 20. It informs the user that there is sufficient stock on hand. ‘e_reorder’ exception is raised when the quantity on hand for the item is less than 20. It informs the user that there is insufficient stock in the inventory.

```
SQL> DECLARE
  2     v_itemid       BINARY_INTEGER;
  3     v_qoh          BINARY_INTEGER;
  4     e_enoughstock  EXCEPTION;
  5     e_reorder      EXCEPTION;
  6
  7  BEGIN
  8     v_itemid := 90002;
  9
 10     SELECT itemid, qoh
 11     INTO v_itemid, v_qoh
 12     FROM items
 13     WHERE itemid = v_itemid;
 14
 15     IF v_qoh > 20 THEN
 16       RAISE e_enoughstock;
 17     ELSE
 18       RAISE e_reorder;
 19     END IF;
 20
 21  EXCEPTION
 22     WHEN e_enoughstock THEN
 23    DBMS_OUTPUT.PUT_LINE('There is sufficient stock in the inventory.');
 24     WHEN e_reorder THEN
 25     DBMS_OUTPUT.PUT_LINE('The inventory does not have sufficient stock
 26     for item ID '||v_itemid||'.');
 27
 28  END;
 29  /
The inventory does not have sufficient stock for
item ID 90002.

PL/SQL procedure successfully completed.
```

12. PL/SQL PROGRAM UNITS

Program units are named PL/SQL blocks. They fall into three main categories: (Ault, 2001)

- *Procedures* to perform actions
- *Functions* to compute a value
- *Packages* to bundle logically related procedures and functions

They are created based on sophisticated business rules and application logic, and are stored within the database. The user may also move code between the Oracle Server and an application. By consolidating business rules with the database, they no longer need to be written into each application, hence, saving time during application creation and simplifying the maintenance process. (Ault, 2001) This improves the performance dramatically.

12.1 Procedures

- **Creating a Procedure**

Example: create a procedure 'change_itemprice' to update item price for the specified item to the specified amount.

```
SQL> CREATE OR REPLACE PROCEDURE change_itemprice
  2  (v_item_id IN NUMBER,
  3  v_price IN NUMBER) IS
  4  BEGIN
  5  UPDATE items
  6  SET price = v_price
  7  WHERE itemid = v_item_id;
  8  COMMIT;
  9  END change_itemprice;
 10  /

Procedure created.
```

When this procedure is invoked, Oracle will take the parameters for the item ID and the new item price, and update the price for that specified item.

- **Invoking Procedures from SQL*Plus**

Procedures are called as stand-alone executable statements. They are invoked using EXECUTE command in the SQL* Plus environment.

Example: Update item price for item '90002'

Before the price is changed.

```
SQL> select price from items
  2  where itemid = 90002;

     PRICE
----------
      1500
```

Invoke procedure 'change_itemprice'.

```
SQL> EXECUTE change_itemprice (90002, 1600);

PL/SQL procedure successfully completed.
```

Verify the change in price for item '90002'.

```
SQL> select price from items
  2  where itemid = 90002;

     PRICE
----------
      1600
```

12.2 Functions

- **Create a Function**

Example: create a function 'Order_total' to return the total price for the specified order.

```
SQL> CREATE OR REPLACE FUNCTION Order_total
  2  (v_orderid IN NUMBER) RETURN NUMBER
  3  IS
  4    ordprice number;
  5    ordquantity number;
  6    ordertotal number;
  7
  8  BEGIN
  9    select ord_qty, price
 10    Into ordquantity, ordprice
 11    from orders o, items i
 12    where o.itemid = i.itemid and orderid = v_orderid;
 13    ordertotal := ordquantity*ordprice;
 14
 15    RETURN (ordertotal);
 16  END;
 17  /

Function created.
```

- **Invoking functions from SQL*Plus**

Unlike procedures, functions are invoked as part of an expression. The user-defined functions can be called just like the built-in function. They can be called from various SQL clauses.

Example 1: use 'order_total' function to calculate the total price for order number 3333.

SOLUTION1:

```
SQL> select order_total(3333)
  2  from dual;

ORDER_TOTAL(3333)
-----------------
             4500
```

SOLUTION2

```
SQL> select orderid, order_total(orderid)
  2  from orders
  3  where orderid = 3333;

   ORDERID ORDER_TOTAL(ORDERID)
---------- --------------------
      3333                 4500
```

Example 2: Return the totals for all the orders made by customer number 50003.

```
SQL> select sum(order_total(orderid))
  2  from orders
  3  where cid = 50003;

SUM(ORDER_TOTAL(ORDERID))
-------------------------
                     4804
```

Example 3: Return the order ID that has a total more than $3000.

```
SQL> select orderid
  2  from orders
  3  where order_total(orderid) > 3000;

   ORDERID
----------
      3333
      3335
```

12.3 Packages

Packages are containers that bundle together procedures, functions, and data structures. They consist of an externally visible package specification and a package body.

- **Creating Packages Specification**

Package specification contains the function headers, procedure headers, and externally visible data structures. It is created with CREATE PACKAGE command.

Example: create a package specification named ‘orderspackage’ that will contain the aforementioned function and procedure.

```
 SQL> CREATE OR REPLACE PACKAGE orderspackage AS
  2   PROCEDURE change_itemprice (v_item_id IN NUMBER, v_price IN NUMBER);
  3   FUNCTION Order_total (v_orderid IN NUMBER) RETURN NUMBER;
  4  END orderspackage;
  5  /

Package created.
```

- **Creating Packages Body**

Package body contains the declaration, executable, and exception sections of all the bundled procedures and functions. It is created with CREATE PACKAGE BODY command.

Example: create the package body for 'orderspackage'.

All the codes for the body of the 'change_itemprice' procedure and 'order_total' function are included within the 'orderspackage' packages body.

```
SQL> CREATE OR REPLACE PACKAGE BODY orderspackage AS
  2  PROCEDURE change_itemprice (v_item_id IN NUMBER, v_price IN NUMBER)
  3   IS
  4   BEGIN
  5     UPDATE items
  6     SET price = v_price
  7     WHERE itemid = v_item_id;
  8   END change_itemprice;
  9  FUNCTION Order_total (v_orderid IN NUMBER) RETURN NUMBER
 10   IS
 11     ordprice number;
 12     ordquantity number;
 13     ordertotal number;
 14   BEGIN
 15     select ord_qty, price
 16     Into ordquantity, ordprice
 17     from orders o, items i
 18     where o.itemid = i.itemid and orderid = v_orderid;
 19     ordertotal := ordquantity*ordprice;
 20     RETURN (ordertotal);
 21   END ORDER_TOTAL;
 22  END ORDERSPACKAGE;
 23  /

Package body created.
```

- **Referencing Package Contents**

Example: execute the procedure 'change_itemprice' within the 'orderspackage'.

```
SQL> execute ORDERSPACKAGE.CHANGE_ITEMPRICE(90006, 9);

PL/SQL procedure successfully completed.
```

Verify the change on item 90006. The price has been modified from $8 to $9.

```
SQL> select price from items
  2  where itemid = 90006;

     PRICE
----------
         9
```

12.4 Triggers

Triggers are programs that are executed automatically in response to a change in the database.

- **Creating a Trigger**

Example: create a trigger to update the item's QOH field in the ITEMS table in the situation when a new order is created, or the existing order quantity (ord_qty) is being updated or the existing order is being deleted.

```
SQL> CREATE OR REPLACE TRIGGER ITEMQOHTRIGGER
  2  BEFORE insert or update or delete of ord_qty on ORDERS
  3  For each row
  4  BEGIN
  5    IF INSERTING THEN
  6      UPDATE ITEMS SET qoh = qoh - :NEW.ord_qty;
  7    ELSIF UPDATING THEN
  8      UPDATE ITEMS SET qoh = qoh + :OLD.ord_qty - :NEW.ord_qty;
  9    ELSIF DELETING THEN
 10      UPDATE ITEMS SET qoh = qoh + :OLD.ord_qty;
 11    END IF;
 12  END;
 13  /

Trigger created.
```

Verify the effect of 'itemqohtrigger' on QOH filed in the ITEMS table.

The current QOH state for item 90004.

```
SQL> SELECT * FROM ITEMS;

    ITEMID ITEMDESC                          PRICE        QOH
---------- ------------------------------ ---------- ----------
     90001 HITACHI monitor 17inch              900         50
     90002 HITACHI monitor 19inch             1600         12
     90003 sony 56k modem                       99         68
     90004 Microsoft keyboard                   40         87
     90005 sony 52x CDROM drive                120         94
     90006 TDK flopy disc x 12                   9        112
```

Action1: insert a new order for item '50004' with order quantity of 15 in the ORDERS table.

```
SQL> insert into orders values
  2  (S_order_id.NEXTVAL, to_date('25/05/2002', 'dd/mm/yyyy'), '50004',
'90004', '15',to_date('03/06/2002', 'dd/mm/yyyy'), '524');

1 row created.
```

When querying item '90004' from ITEMS table, the QOH has decreased from 87 to 72 due to the new order.

```
SQL>  SELECT * FROM ITEMS
  2   WHERE ITEMID = 90004;

    ITEMID ITEMDESC                          PRICE        QOH
---------- ------------------------------ ---------- ----------
     90004 Microsoft keyboard                   40         72
```

Action 2: Update the new order by increasing the order quantity for item '90004' to 20.

```
SQL> UPDATE orders
  2  SET ORD_QTY = 20
  3  WHERE orderid = 3340;

1 row updated.
```

The updating on the order quantity has also taken effect on the QOH of item '90004'.

```
SQL> SELECT * FROM ITEMS
  2  WHERE ITEMID = 90004;

    ITEMID ITEMDESC                        PRICE        QOH
---------- ------------------------------ ---------- ----------
     90004 Microsoft keyboard                  40         67
```

Action 3: delete the new order.

```
SQL> DELETE FROM ORDERS
  2  WHERE ORDERID = 3340;

1 row deleted.
```

The deletion of the new order has brought the QOH of item '90004' back to its original state.

```
SQL> SELECT * FROM ITEMS
  2  WHERE ITEMID = 90004;

    ITEMID ITEMDESC                        PRICE        QOH
---------- ------------------------------ ---------- ----------
     90004 Microsoft keyboard                  40         87
```

- **Enabling and disabling Triggers**

Triggers can be enabled or disabled by using ALTER TRIGGER command or ALTER TABLE command.

ALTER TRIGGER command

The user may enable / disable specific triggers with this command. In order to use this command, the user must have ALTER ANY TRIGGER system privilege.

Example 1: enable the 'itemqohtrigger'

```
SQL> ALTER TRIGGER itemqohtrigger ENABLE;

Trigger altered.
```

Example 2: disable the 'itemqohtrigger'

```
SQL> ALTER TRIGGER itemqohtrigger DISABLE;

Trigger altered.
```

ALTER TABLE command

The user may enable / disable all triggers associated with the specified table with this command. In order to use this command, the user must have ALTER ANY TABLE system privilege.

Example 1: enable all triggers associated with table ORDERS.

```
SQL> ALTER TABLE orders enable all triggers;

Table altered.
```

Example 2: disable all triggers associated with table ORDERS.

```
SQL> ALTER TABLE orders disable all triggers;

Table altered.
```

CHAPTER 3 Database Architecture and Administration

13. SOFTWARE INSTALLATION

This section is the documentation of installing Oracle 8i Enterprise Edition and Oracle 9i Application Server on the windows platform.

13.1 Oracle 8i release 8.1.7 Enterprise Edition

Installation Failure

Problem description:

After selecting install from the CD, the hourglass appears then disappears and the Universal Installer never runs. If you looked in the Task Manager, you would see a process called JREW.EXE for a brief moment then it would go away (as if terminated) immediately after executing the Oracle Client Setup.EXE.

Possible causes for the problem:

Setup.exe does nothing on the machine maybe because of a problem with the Java Runtime Environment (JRE) and Pentium 4 processor. The version of JRE using in the Oracle 8i release 8.1.7 Enterprise Edition is not compatible with the one used in the P4 processor.

Resolution/Workaround

1. Copy the entire CD down to the hard drive in a temporary folder.

2. Download JRE release 1.1.8-009 from java.sun.com and install it. The Java code URL is: http://java.sun.com/products/jdk/1.1/jre/download-jre-windows.html .Then go to C:\Program Files\JavaSoft\JRE\1.1\bin and copy the file 'symcjit.dll' over the one in the Oracle 8.1.7 client installation area here:
   stage\Components\oracle.swd.jre\1.1.7.30\1\DataFiles\Expanded\jre\win32\bin

3. Once the step 2 is done, the setup.exe will install the oracle 8.1.7 enterprise edition.

Installation Summary

```
Global database name: project.world
Database system identifier (sid): project
SYS account password: change_on_install
SYSTEM account password: manager
```

Error message

An error message has occurred at the end of the installation.

```
Java.exe has generated errors and will be closed by windows.
```

Apart from this, the installation was successful.

13.2 Oracle 9i Application Server release 1 (version 1.0.2.2) installation

Pre-installation steps:
The following steps need to be completed before installing Oracle 9i Application server according to the Oracle9i AS installation guide by Oracle cooperation.

1. Create the password file required by Oracle 9i AS Database Cache. Enter the following in DOS prompt.

   c:\oracle\ora81\bin\orapwd file=pwdproject.ora password=change_on_install entries=10

2. Increase the size of the users tablespace. Enter the following SQL command in the SQL/plus.

   alter database datafile 'c:\oracle\oradata\project\users01.dbf' resize 250M;

3. Modify TNS files located at c:\oracle\ora81\network\admin

   listener.ora => change port 1521 to 1526, port 2481 to 2486
   sqlnet.ora => no change
   tnsnames.ora => change port 1521 to 1526

4. Service startup

   Go to start => settings => control panel => Administrative tools => services Change startup type of *oracleorahome81 http server* to manual.

Installation Failure
Problem description:
The same installation problem occurred. After selecting install from the CD, the hourglass appears then disappears and the Universal Installer never runs.

Possible causes for the problem:
Setup.exe does nothing on the machine. This maybe because of a problem with the Java Runtime Environment (JRE) and Pentium 4 processor. The version of JRE using in the Oracle 9i As release1 v(1.0.2.2) is not compatible with the one used in the P4 processor.

Resolution/Workaround

1. Copy the disc 1 down to the hard drive in a temporary folder.

2. Go to C:\Program Files\JavaSoft\JRE\1.1\bin and copy the file 'symcjit.dll' over the one in the Oracle 9iAS installation area here:

   install/Stage/Components\oracle.swd.jre\1.1.7.30\1\DataFiles\Expanded\jre\win32 \bin
   and
   stage\Components\oracle.swd.jre\1.1.7.30\1\DataFiles\Expanded\jre\win32 \bin

3. Once the step 2 is done, the setup.exe will install the oracle 9i As disc 1. The remaining follow the instruction through.

An error message has occurred during the installation:

```
vbj.exe – entry point Not Found.
The procedure entry point VBJRuntime could not be located in the dynamic
link                        library                        vbjruntim.dll.
```

Apart from this, the installation was successful

14. ADMINISTRATOR AUTHENTICATION METHOD

14.1 Using Password File Authentication

1. Create the password file using the password utility ORAPWD. Enter the following command in DOS prompt.

   c:\>orapwd file=ora81/database/pwdproject password=admin entries=10

2. Set the REMOTE_LOGIN_PASSWORDFILE parameter to EXCLUSIVE. The REMOTE_LOGIN_PASSWORDFILE is located in the file **init.ora** which is located at oracle/admin/project/pfile

3. Connect to the database as follows:

   SQL> connect /as sysdba
   or
   SQL>connect internal/oracle

15. MANAGING AN ORACLE INSTANCE

The instance in Oracle 8i release 3 (v 8.1.7) can be managed from the Oracle Enterprise Manager Console and DBA studio

Starting and shutdown the Instance using the following command in SQL/Plus

15.1 Start Up a Database

In order to start or stop an Oracle instance, the user must have the SYSDBA or SYSOPER privilege. Otherwise, the following error message will be generated.

```
SQL> startup pfile=project.world.ora
ORA-01031: insufficient privileges
```

The STARTUP command is used to start the instance, mount a database, and open the database for normal operation.

```
Sql> connect /as sysdba
Sql> startup pfile=project.world.ora


Total System Global Area  142039068 bytes
Fixed Size                    75804 bytes
Variable Size              58081280 bytes
Database Buffers           83804160 bytes
Redo Buffers                  77824 bytes
Database mounted.
Database opened.
```

Where ‘project.world.ora’ is the parameter file for database ‘ Project’.

15.2 Manage Sessions

Once when a database connection is made, Oracle starts a session. Information in relation to a session is contained in the V$SESSION view.

Example: query the V$SESSION view to find out who is connected to the database and what program they are running now.

```
SQL> select username, program
  2  from V$SESSION;
```

```
USERNAME                       PROGRAM
-----------------------------  ------------------------------------
                              ORACLE.EXE

SYS                            SQLPLUSW.EXE

DBSNMP                          dbsnmp.exe

IVY                             jre.exe

IVY                             jre.exe

IVY                             jre.exe
```

```
IVY                            jre.exe

IVY                            jre.exe

IVY                            jre.exe

IVY                            jre.exe

IVY                            jre.exe

IVY                            jre.exe

MARY                           SQLPLUSW.EXE


13 rows selected.
```

A user session can be terminated by using the ALTER SYSTEM command. In order to do this, the SID and SERIAL# are required from the V$SESSION view.

Example: terminate user MARY's session.

Firstly, query the SID and SERIAL# of user MARY's session from V$SESSION view.

```
SQL> select username, sid, serial#, status
  2  from V$SESSION
  3  WHERE username = 'MARY';

USERNAME                              SID    SERIAL# STATUS
------------------------------ ---------- ---------- --------
MARY                                   22        855 INACTIVE
```

Then issue the ALTER SYSTEM command with the SID and SERIAL# to kill that particular session.

```
SQL> ALTER SYSTEM KILL SESSION '22, 855';

System altered.
```

Verify that the session has been terminated.

```
SQL> select username, sid, serial#, status
  2  from V$SESSION
  3  WHERE username = 'MARY';

USERNAME                              SID    SERIAL# STATUS
------------------------------ ---------- ---------- --------
MARY                                   22        855 KILLED
```

15.3 Shutdown a Database

The SHUTDOWN command is used to shut down the current instance.

```
SQL> SHUTDOWN immediate
Database closed.
Database dismounted.
ORACLE instance shut down.
```

16. CREATE A DATABASE

16.1 Creating a Database

The database is a collection of physical files that work together with an area of allocated memory and background processes. It requires careful planning and is done in multiple steps.

Example: Create Database db01

1. Declare the variable ORACLE_SID in operating system prompt with the following command:

```
c:\>set Oracle_SID=db01
```

2. Create the database service and the password file with ORADIM

```
c:\>ORADIM –NEW –SID db01 –INTPWD 000 –STARTMODE auto
 -PFILE ora81\database\initdb01.ora
```

3. Set up the path for the new database.

- oracle\oradata\db01
- oracle\admin\db01

4. Modify an existing init.ora file to reflect the parameters for the new database. Save the modified parameter file as ‘initdb01.ora’.

```
|db_name                              =                              "DB01"
db_domain                             =                               world
instance_name                            =                            DB01
service_names                         =                          DB01.world
control_files = ("C:\oracle\oradata\DB01\control01.ctl",
"C:\oracle\oradata\DB01\control02.ctl",
"C:\oracle\oradata\DB01\control03.ctl")
background_dump_dest            =            C:\oracle\admin\DB01\bdump
user_dump_dest = C:\oracle\admin\DB01\udump
```

5. Connect to the database via SQL*PLUS as sysdba

```
SQL> connect /as sysdba
```

6. Issue the following command to start up the database in NOMOUNT mode

```
SQL>startup nomount pfile=”c:\oracle\ora81\database\initdb01.ora”
ORACLE instance started.
Total System Global Area  142039068 bytes
Fixed Size                    75804 bytes
Variable Size              58081280 bytes
Database Buffers           83804160 bytes
Redo Buffers                  77824 bytes
```

7. Once the database has been started, use the following database creation script to create the database ‘DB01’.

```
create database db01
  Maxinstances 1
  maxloghistory 5
  maxlogfiles 5
  maxlogmembers 5
  maxdatafiles 100
datafile 'c:\oracle\oradata\db01\system01.dbf'
   size 325m reuse autoextend on next 10240k maxsize unlimited
   character set we8iso8859p1
   national character set we8iso8859p1
logfile
   group 1 ('c:\oracle\oradata\db01\redo01.log') size 100M,
   group 2 ('c:\oracle\oradata\db01\redo02.log') size 100M,
   group 3 ('c:\oracle\oradata\db01\redo03.log') size 100M;
```

However, the following error message s displayed and the database creation is failed.

```
ERROR at line 1:
ORA-01501: CREATE DATABASE failed
ORA-01991: invalid password file 'C:\oracle\ora81\DATABASE\PWDproject.ORA'
```

After setting the Remote_login_passwordfile parameter in the ‘initdb01.ora’ file to **shared**, the database is created successfully.

16.2 Creating the Data Dictionary

After a database is created, it can be functional only when the data dictionary is created. (Sarin, 2000)
In order to generate the data dictionary, run the scripts, catalog.sql and catproc.sql in SQL/PLUS after connecting to the instance and open the database db01.

```
SQL:>Start ‘c:\oracle\ora81\rdbms\admin\catalog.sql’;
SQL:>Start ‘c:\oracle\ora81\rdbms\admin\catproc.sql’;
```

17. MAINTAINING CONTROL FILES

17.1 Create New Control Files

The control file names and locations are listed in the CONTROL_FILES parameter in the initial parameter file. When creating a database, Oracle creates as many control files as are specified in the initial parameter file.

If more control files are required in the database, the subsequent steps need to be followed. The creation of this additional control file is based on the existing one.

Example: create an additional control file for database 'DB01'.

1. Shut down the database.

```
SQL> shutdown immediate;
Database closed.
Database dismounted.
ORACLE instance shut down.
```

2. Make a copy of the existing control file to a different device using operating system commands.

   The new control file 'control04.ctl' is created based on 'control01.ctl'.

```
C:\oracle\oradata\db01\copy control01.ctl control04.ctl
```

3. Edit or add the CONTROL_FILES parameter and specify names for all the control files.

   Adding the additional file 'control04.ctl' in the initial parameter file of database 'DB01'

```
control_files =
("C:\oracle\oradata\DB01\control01.ctl",
"C:\oracle\oradata\DB01\control02.ctl",
"C:\oracle\oradata\DB01\control03.ctl",
"C:\oracle\oradata\DB01\control04.ctl")
```

4. Start up the database.

```
SQL> startup pfile='c:\oracle\ora81\database\initdb01.ora'
ORACLE instance started.

Total System Global Area  142039068 bytes
Fixed Size                    75804 bytes
Variable Size              58081280 bytes
Database Buffers           83804160 bytes
Redo Buffers                  77824 bytes
Database mounted.
Database opened.
```

Verify the additional control file by querying from the V$CONTROLFILE view.

```
SQL>  SELECT * FROM V$CONTROLFILE;

STATUS
-------
NAME
--------------------------------------------

C:\ORACLE\ORADATA\DB01\CONTROL01.CTL


C:\ORACLE\ORADATA\DB01\CONTROL02.CTL


C:\ORACLE\ORADATA\DB01\CONTROL03.CTL


C:\ORACLE\ORADATA\DB01\CONTROL04.CTL
```

18. MAINTAINING REDO LOG FILES

18.1 Obtaining Log and Archive Information

To find out if the database is configured to run in ARCHIVELOG mode, the following SQL commands can be used to obtain archive information.

1. Enter the following command in SQL prompt.

```
SQL> archive log list
```

This command derives the following information, which specify the current setting of:

- Database log mode
- Automatic archival
- Archive destination
- Oldest online log sequence
- Current log sequence

```
Database log mode              No Archive Mode
Automatic archival             Disabled
Archive destination            C:\oracle\ora81\RDBMS
Oldest online log sequence     1476
Current log sequence           1478
```

2. The other way to obtain archive information is to query the dynamic performance views V$DATABASE.

```
SQL> select name, log_mode
  2  from v$database;

NAME      LOG_MODE
--------- ------------
PROJECT   NOARCHIVELOG
```

This command retrieves the database log mode of current database from the dynamic performance views V$DATABASE.

3. Another way is to query the dynamic performance views V$INSTANCE and look at its archiving mode.

```
SQL> select archiver
  2  from v$instance;

ARCHIVE
-------
STOPPED
```

4. Use Oracle Enterprise Manager Console or DBA studio to obtain Archive information.

- Launch Console and enter login information `Start → Programs → Oracle – Orahome81 → Enterprise Manager → Console`
- Expand the working database, **Project**, and then expand **Instance**
- Right click on **database**, and select **Edit**
- Click on **Archive** tab to obtain archive information

18.2 Obtaining Log Group Information

To obtain information regarding log group, it can be accessed through query from the dynamic performance view V$THREAD, V$LOG, V$LOGFILE.

1. By querying from the dynamic performance view V$THREAD, information about the number of online redo log groups, the current log group, and the sequence number can be returned.

```
SQL> select groups, current_group#, sequence#
  2  from v$thread;

    GROUPS CURRENT_GROUP#  SEQUENCE#
---------- -------------- ----------
         3              3       1479
```

2. By querying from the dynamic performance view V$LOG, information about the online redo log files from the control file can be returned.

```
SQL> select group#, sequence#, bytes, members, status
  2  from v$log;

    GROUP#  SEQUENCE#      BYTES    MEMBERS STATUS
---------- ---------- ---------- ---------- ----------------
         1       1477    1048576          1 INACTIVE
         2       1478    1048576          1 CURRENT
         3       1476    1048576          1 INACTIVE
```

The returned information indicates that group# 2 is the current online redo log group and is active. Whereas the group# 1 and 3 are no longer needed for instance recovery. It may or may not be archived.

3. The following query returns the names of all the members of a group by querying the dynamic performance view V$LOGFILE.

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

    GROUP# STATUS
---------- -------
MEMBER
------------------------------------------------------------
         1 STALE
C:\ORACLE\ORADATA\PROJECT\REDO03.LOG

         2
C:\ORACLE\ORADATA\PROJECT\REDO02.LOG

         3
C:\ORACLE\ORADATA\PROJECT\REDO01.LOG
```

The returned information indicates that contents of group#1 are incomplete whereas group#2 and #3 are in use.

18.3 Controlling Log Switches

- **Forcing Log Switches**

The following SQL command can force a log switch.

```
SQL> alter system switch logfile;

System altered.
```

If querying again from the previously mentioned dynamic performance view, we can see the log has switched now.

```
SQL>  select groups, current_group#, sequence#
  2   from v$thread;

    GROUPS CURRENT_GROUP#  SEQUENCE#
---------- -------------- ----------
         3              1       1480
```

```
SQL>  select group#, sequence#, bytes, members, status
  2   from v$log;

    GROUP#  SEQUENCE#      BYTES    MEMBERS STATUS
---------- ---------- ---------- ---------- ----------------
         1       1480    1048576          1 CURRENT
         2       1478    1048576          1 INACTIVE
         3       1479    1048576          1 ACTIVE
```

```
SQL> select * from V$logfile;

    GROUP# STATUS
---------- -------
MEMBER
------------------------------------------
         1
C:\ORACLE\ORADATA\PROJECT\REDO03.LOG

         2
C:\ORACLE\ORADATA\PROJECT\REDO02.LOG

         3
C:\ORACLE\ORADATA\PROJECT\REDO01.LOG
```

18.4 Adding Online Redo Log Groups

Example: create additional redo log group# 4 for database Project.

```
SQL> Alter Database project
  2    add logfile ('/oracle/oradata/project/REDO04.LOG') size 1M;

Database altered.
```

18.5 Adding Redo Log Members

Example: create new members to existing redo log file group#1 and group#2.

```
SQL> Alter database project
  2    add logfile member
  3   '/oracle/oradata/project/REDO01b.LOG' to group 1,
'/oracle/oradata/project/REDO02b.LOG' to group 2;

Database altered.
```

18.6 Dropping a Redo Log Group

Example: drop redo log group# 4.

```
SQL> Alter database project
  2  drop logfile group 4;

Database altered.
```

18.7 Dropping a Redo log Member

Example: drops the redo log member ‘redo01b.log’ of group# 1.

```
Alter database project
drop logfile member '/oracle/oradata/project/REDO01B.log';
```

19. MANAGING TABLESPACES AND DATA FILES

19.1 Creating Tablespace

- **Dictionary_Managed Tablespace**

In dictionary_managed tablespaces, all extent information is stored in the data dictionary. (Enterprise DBA Part 1A Volume 1)

Example:

```
SQL> Create tablespace pro_data
  2  datafile '/oracle/oradata/project/pro_data_01.dbf' size 100M,
  3  '/oracle/oradata/project/pro_data_02.dbf' size 100M
  4  Minimum extent 500k
  5  default storage (initial 500k
  6     Next 500k
  7    Maxextents 500
  8    Pctincrease 0 );

Tablespace created.
```

The SQL statement creates a tablespace named ‘pro_data’. The datafile specified are created with a size of 100MB each.

- **Locally Managed Tablespace**

Compare to Dictionary_Managed tablespace, Locally managed tablespaces manage space more efficiently, provide better methods to reduce fragmentation, and increase reliability. The table space is created by include ‘EXTENT MANAGEMENT LOCAL’ clause. (Enterprise DBA Part 1A Volume 1)

Following is an example of creating a locally managed tablespace with Oracle managing the extent allocation.

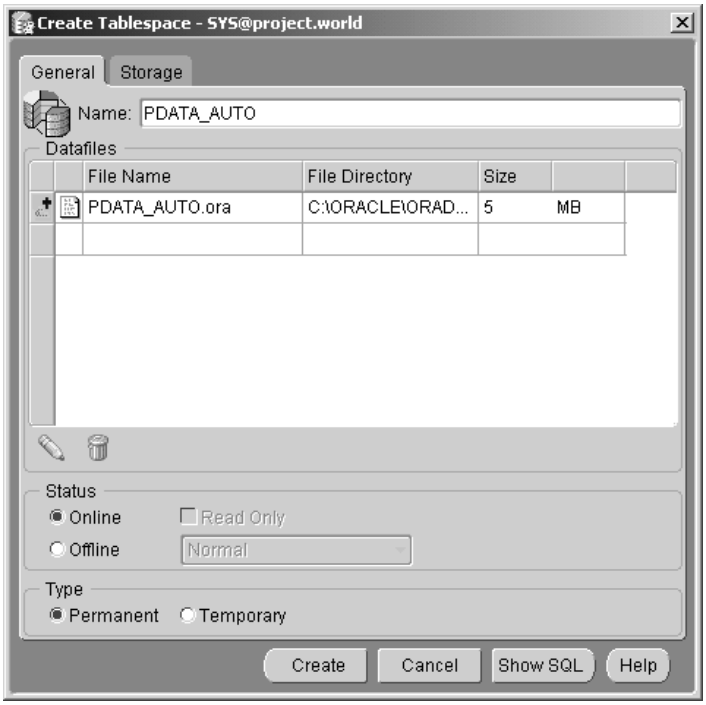
```
SQL> CREATE TABLESPACE my_datafile
  2  DATAFILE '/oracle/oradata/project/my_datafile_01.dbf' SIZE 100M
  3  EXTENT MANAGEMENT LOCAL AUTOALLOCATE;

Tablespace created.
```

19.2 Creating Tablesapce with DBA studio

1. Launch DBA studio and connect to the database ‘project’ directly

```
Start -> Programs -> Oracle_orahome81 -> Database Administration ->
DBA studio
```

2. Expand database ‘project’ and then expand storage.
3. Right click on tablespace and select create.
4. Enter the name for the tablespace, ‘pdata_auto’ and set all other settings as default.



5. Click the Create bottom and a message box will display to inform you the tablespace has been created successfully.

19.3 Changing the Storage Settings

The settings for tablespaces can be changed by using the ALTER TABLESPACE command.

Following is an example of changing the settings for tablespace 'pro_data'.

```
SQL> Alter Tablespace pro_data
  2  minimum extent 1M;

Tablespace altered.

SQL> Alter Tablespace pro_data
  2  Default storage (
  3   initial 1m
  4   next 1m
  5   maxextents 700);

Tablespace altered.
```

The storage settings can also be changed through DBA studio or Enterprise Management Console.

- **Read Only Tablespace**

When no changes are allowed to make on any data, the tablespace can be made read only.

```
SQL> ALTER TABLESPACE my_datafile READ ONLY;
```

When the command is issued, the tablespace goes into a transitional read-only mode in which no further DML statements are allowed. (Sarin, 2000)

19.4 Dropping Tablespaces

A tablespace from the database can be removed using the following command. In the following example, the tablespace 'pro_data' is removed from database 'project'.

```
SQL> Drop tablespace pro_data including contents;

Tablespace dropped.
```

The tablespace can also be dropped using Oracle DBA studio.

1. Launch DBA studio and connect to the database 'project' directly

Start → Programs → Oracle_orahome81 → Database Administration → DBA studio

2. Expand database 'project' and then expand storage.
3. Right click on tablespace and select remove.
4. Click yes to confirm.

19.5 Resizing of Data Files Automatically

The size of tablespaces can be increased automatically by enabling the AUTOEXTEND function.

- **Enabling AUTOEXTEND for a New Data File**

Example: create a new data file 'pdata_auto_01.dbf' with automatic extension enabled.

```
SQL> Alter Tablespace pdata_auto
  2  add datafile '/oracle/oradata/project/pdata_auto_01.dbf'
  3  size 100M
  4  autoextend on
  5  next 10m
  6  Maxsize 500m;

Tablespace altered.
```

- **Enabling AUTOEXTEND for an Existing Data File**

Example: enable the AUTOEXTEND for the previously created data file 'my_datafile_01.dbf'.

```
SQL> Alter database
  2  datafile '/oracle/oradata/project/my_datafile_01.dbf'
  3  AUTOEXTEND ON;

Database altered.
```

- **Use Oracle Enterprise Manager to Enable Automatic Resizing**

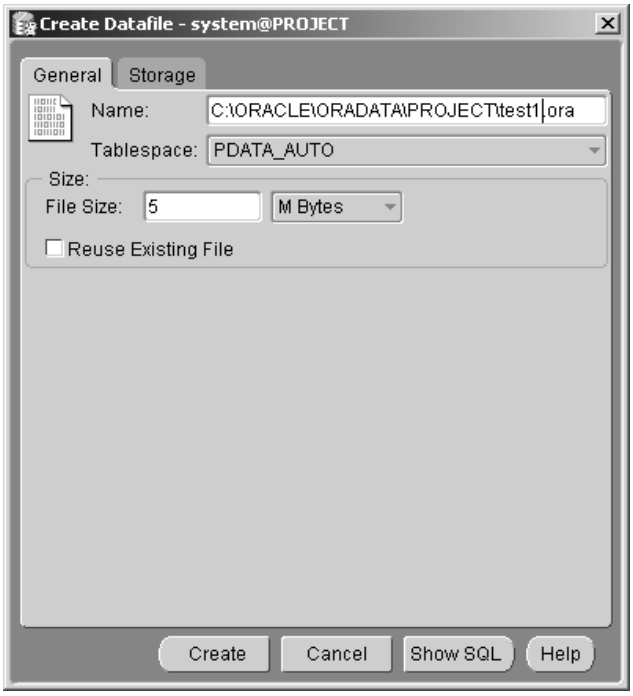
  1. Launch **DBA studio** and connect to the database 'project' directly as SYSDBA

     `Start → Programs → Oracle_orahome81 → Database Administration → DBA studio`
  2. Expand database 'project' and then expand storage.
  3. Expand datafiles and right click on datafile 'pdata_auto_01.dbf' and select edit then the following window appears.

4. In the Auto Extend tab, select the Enable Auto Extend check box.
5. Click ok to confirm.

The DBA studio can be also used to add a data file in the existing tablespaces.

1. Launch DBA studio follow the steps as described earlier.
2. Expand the Tablespaces folder.
3. Select the Tablespace that you wish to add a new datafile. Right click on the selected tablespace (ie. Pdata_auto) and select **add Datafile…**. The following window will appear.



4. Specify the datafile name (ie. Test1.ora) and file size in general tab and AUTOEXTEND in storage tab.

5. Click on create bottom.

19.6 Resizing of Data Files Manually

The alternative way to increase the tablespace size is using RESIZE command.

Example: resize datafile 'pdata_auto_01.dbf' to 200 M.

```
SQL> Alter Database project
  2  datafile '/oracle/oradata/project/pdata_auto_01.dbf'
  3  RESIZE 200M;

Database altered.
```

20. MANAGING ROLLBACK SEGMENTS

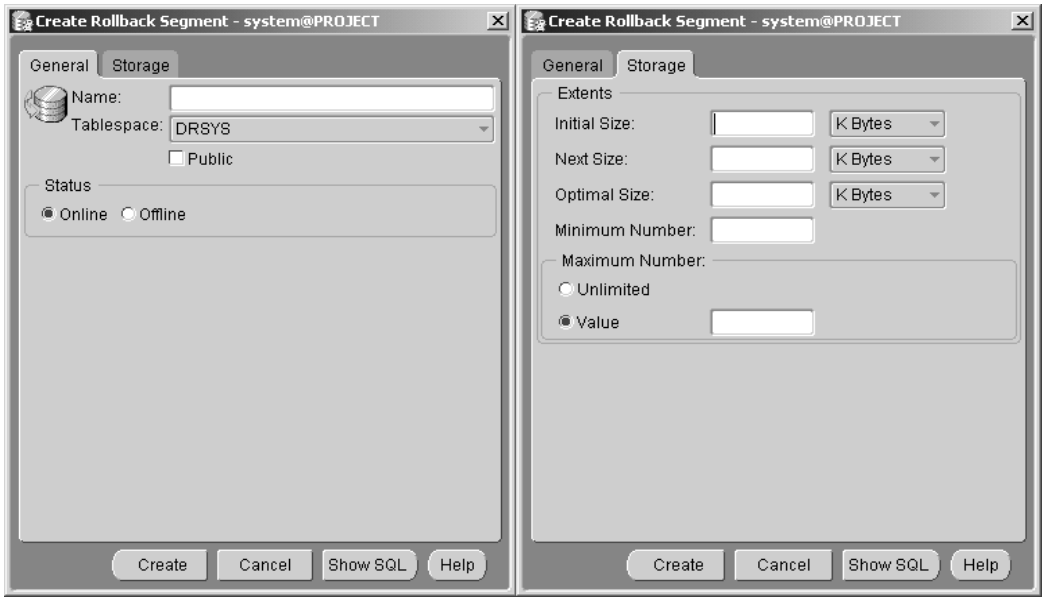
20.1 Create Rollback Segments

Example: create a rollback segment 'rbstest01' in the tablespace 'RBS'.

```
SQL> create rollback segment rbstest01
  2  tablespace RBS
  3  storage (
  4   initial 120k
  5   next 120k
  6   minextents 20
  7   maxextents 100
  8   optimal 2400k);

Rollback segment created.
```

The DBA studio can be also used to create rollback segment.

1. Launch DBA studio follow the steps as described earlier.
2. Expand the storage.
3. Right click on the Rollback segment and select create. The following window will appear.



4. Enter the information required in both general tab and storage tab.
5. Click on Create.

After the Rollback segment is created, DBA studio can also be used to change the settings, shrink a rollback segment, taking a rollback segment online or offline or drop a rollback segment.

20.2 Dictionary Views about Rollback Segments

Verify which rollback segments in the system are available for use by transactions.

```
SQL> select segment_name, status
  2  from dba_rollback_segs;

SEGMENT_NAME                   STATUS
------------------------------ ---------------
SYSTEM                         ONLINE
RBS0                           ONLINE
RBS1                           ONLINE
RBS2                           ONLINE
RBS3                           ONLINE
RBS4                           ONLINE
RBS5                           ONLINE
RBS6                           ONLINE
RBSTEST01                      OFFLINE

9 rows selected.
```

21. MANAGING TABLES

21.1 Creating a Table

- **Creating a table using create table clause**

Tables are created using CREATE TABLE clause.

Example: create EMPLOYEES table

```
SQL> Create table employees
  2  (eid number(3) constraint employees_eid_pk primary key,
  3  efname varchar2(10),
  4  elname varchar2(15),
  5  edob date,
  6  eadd varchar2(25),
  7  ephone number(8),
  8  epcode number(4),
  9  Did number(3))
 10  tablespace new_tablespace;

Table created.
```

- **Creating a table from a query**

Example: create a new table from the EMPLOYEES table for employees belong to sales department '101'.

```
SQL> CREATE TABLE sales_department
  2  (eid, efname, elname, eadd, ephone)
  3  TABLESPACE new_tablespace
  4  PCTFREE 0
  5  STORAGE (INITIAL 128K NEXT 128K PCTINCREASE 0)
  6  AS
  7  SELECT eid, efname, elname, eadd, ephone
  8  FROM employees
  9  WHERE did = '101';

Table created.
```

- **Partitioning**

Range-Partitioned Table

Example: create a range-partitioned table named ORDERS.

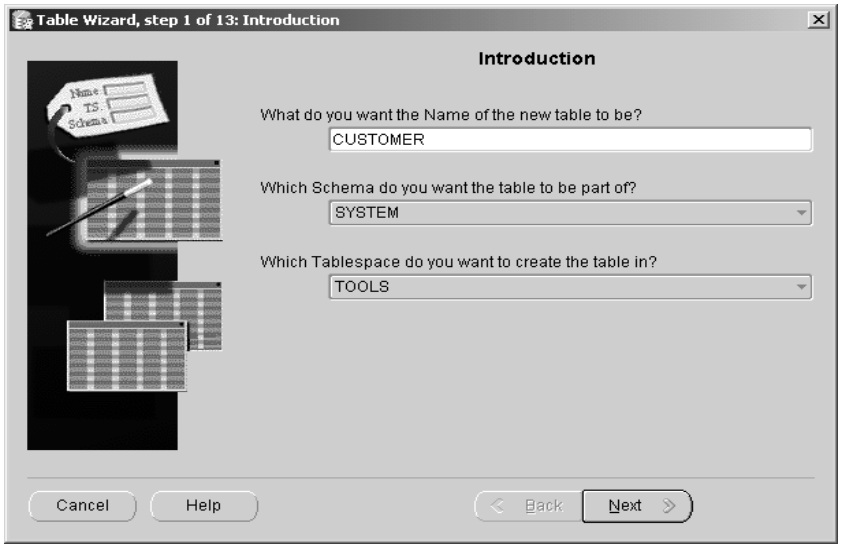
```
SQL> Create table orders
  2  (orderid number(4) constraint orders_orderid_pk primary key,
  3  ord_date date,
  4  cid number(5) constraint orders_cid_fk references customers(cid),
  5  itemid number(5) constraint orders_itemid_fk references items(itemid),
  6  ord_qty number(5),
  7  delver_date date,
  8  eid number(3) constraint orders_eid_fk references employees(eid))
  9  PARTITION BY RANGE (ORD_DATE)
 10  (PARTITION ORD1 VALUES LESS THAN
 11         (TO_DATE('31-03-2002', 'DD-MM-YYYY'))
 12         TABLESPACE ORD_DATA,
 13  PARTITION ORD2 VALUES LESS THAN (MAXVALUE)
 14         TABLESPACE ORD_DATA2)
 15  STORAGE (INITIAL 128K NEXT 128K PCTINCREASE 0)
 16  NOLOGGING;

Table created.
```

PARTITION BY RANGE specifies that the table should be range partitioned according to the 'ORD_DATE'. Records with 'ORD_DATE' prior to 31-03-2002 will be stored in partition ORD1. The MAXVALUE parameter specifies that the partition bound is infinite.

The main reason for partitioning is to better manage a table. If a table is created spread across many data files, and one disk fails, the entire table will need to be recovered. However, if the table is partitioned, only that partition needs to be recovered. In addition, SQL statements can access the required partition(s) rather than reading the entire table. This improves the performance of speed.

- **Creating a Table using DBA studio ~ Schema Manager**

1. Launch DBA studio and connect to the database 'project' directly as sysdba.
2. Select **object -> create** from the menu bar.
3. Choose **Table** from the list of objects, check the **use Wizard** option, and click Create.
4. A table wizard will appear on the screen and require the user to enter information needed for the new table throughout the steps. The information required are the schema, tablespace, columns information, define the primary key for the table if uses, define other constraints if applicable, and storage information.



5. At the last step of the Table Wizard, a SQL statement is generated in relation to the new table.

```
CREATE TABLE "SYSTEM"."CUSTOMER"
   ("CID" NUMBER(10, 3) NOT NULL,
    "CNAME" VARCHAR2(15) NOT NULL,
    "CADD" VARCHAR2(50) NOT NULL,
    "CPHONE" NUMBER(15, 8) NOT NULL,
    CONSTRAINT "CUSTOMER_CID_PK" PRIMARY KEY("CID"),
    UNIQUE("CID"))
    TABLESPACE "TOOLS"
```

In the above SQL statement, a table CUSTOMER has been created. It contains columns of CID, CFNAME, CLNAME, CADD and CPHONE in which the CID is defined as the primary key. This CUSTOMER table is part of the '**system**' schema and is created in the '**tools'** tablespace.

6. After the table is created, any change that needs to be made can be done in the schema management or using ALTER TABLE clause.

21.2 Analysing Tables

- **Validating Structure**

Action: Create a table named INVALID_ROWS based on the script utlvalid.sql supplied from Oracle, located in the rdbms/admin directory of the software installation.

```
SQL> @c:\oracle\ora81\rdbms\admin\utlvalid.sql

Table created.
```

This table holds information of corrupted blocks

Example: To verify the integrity of each data block of the ORDERS table.

```
SQL> ANALYZE TABLE ORDERS VALIDATE STRUCTURE;

Table analyzed.
```

If Oracle encounters bad rows, they are inserted into the `INVALID_ROWS` table.

No bad rows are generated from ORDERS table.

```
SQL> select * from invalid_rows;

no rows selected
```

- **Finding Migrated Rows**

The LIST CHAINED ROWS clause of the ANALYZE command can be used to find the chained and migrated rows of a table. When chained rows are encountered, ORACLE writes the ROWID of such rows to the CHAINED_ROWS table.

Create a table named CHAINED_ROWS based on the script utlchain.sql supplied from Oracle, located in the rdbms/admin directory of the software installation.

```
SQL> @c:\oracle\ora81\rdbms\admin\utlchain.sql

Table created.
```

Example: analyse EMPLOYEES table to find migrated rows.

```
SQL> ANALYZE TABLE employees LIST CHAINED ROWS;

Table analyzed.
```

Find the number of migrated rows of EMPLOYEES table from the CHAINED_ROWS table. As the result indicates, there is no chain rows occurred in the EMPLOYEES table.

```
SQL> SELECT COUNT(*)
  2  FROM CHAINED_ROWS
  3  WHERE OWNER_NAME = 'system'
  4  AND TABLE_NAME = 'EMPLOYEES';

  COUNT(*)
----------
         0
```

- **Collecting Statistics**

Example: gather statistics on system's EMPLOYEES table.

```
SQL> ANALYZE TABLE EMPLOYEES COMPUTE STATISTICS;

Table analyzed.
```

The statistics generated is stored in the data dictionary, ie. DBA_TABLES. The following query retrieves statistics of average row length, the number of empty blocks, number of rows and number of chained rows in EMPLOYEES table.

```
SQL> select AVG_ROW_LEN, EMPTY_BLOCKS, num_rows, chain_cnt
  2  from dba_tables
  3  where table_name = 'EMPLOYEES';

AVG_ROW_LEN EMPTY_BLOCKS   NUM_ROWS  CHAIN_CNT
----------- ------------ ---------- ----------
         59           61          8          0
```

21.3 Obtain ROWID information

The DBMS_ROWID package provides several functions that can be used to convert between ROWID formats and to translate between ROWID and its individual components.

Example: find out which files and blocks contain the orders from customer cid '50001'.

```
SQL> select distinct dbms_rowid.rowid_relative_fno(rowid) as "file",
  2  dbms_rowid.rowid_block_number(rowid) as "Block"
  3  from customers
  4  where cid='50001';

      file      Block
---------- ----------
         5         19
```

21.4 Retrieve Extent Information

Using the following clause enables us to verify the current extent information of a particular table.

```
SQL> select count(*)
  2  from dba_extents
  3  where segment_name= 'ORDERS'
  4  and owner = 'SYSTEM';

  COUNT(*)
----------
         7
```

21.5 Allocate extents manually

Allocating extents manually might be required in order to control the distribution of extents of a table across files. In addition, the dynamic extension of tables can be prevented if we allocate extents manually before loading data in bulk.

The extents can be allocated manually using ALTER TABLE……ALLOCATE EXTENT… command. For example:

```
SQL> alter table system.orders
  2  allocate extent(size 200k);

Table altered.

SQL> select count(*)
  2  from dba_extents
  3  where segment_name= 'ORDERS'
  4  and owner = 'SYSTEM';

  COUNT(*)
----------
        10
```

In the above example, the table **syste.orders** has been assigned additional 200k sizes of extents. When retrieving the extent information again using the clause mentioned earlier, we can find that the number of extents has increased from 7 to 10.

21.6 Dropping a Table

When a table is no longer needed, it can be dropped using DROP TABLE clause.

However, when trying to drop table ORDERS, the following error message is generated, and the drop table failed.

```
ERROR at line 1:
ORA-04098: trigger 'SYS.JIS$ROLE_TRIGGER$' is invalid and failed re-
validation
```

The cause of the error message is that a trigger was attempted to be retrieved for execution and was found to be invalid. This also means that compilation/authorization failed for the trigger. The action I took was to resolve the compilation/authorization errors.

1. Log in SQL*PLUS as SYS
2. Issue the following statement

```
SQL> alter trigger SYS.JIS$ROLE_TRIGGER$ compile;

Trigger altered.
```

3. Drop the table again using the DROP TABLE statement. The table then is dropped successfully.

```
SQL> drop table orders;

Table dropped.
```

22. MANAGING INDEXES

22.1 Creating Indexes

- **B-Tree indexes**

B-Tree index is organized in a structure like a tree, with nodes and leaves, and it restructures itself automatically whenever a new row is inserted in the table. (Sarin, 2000) It is most useful on columns with a significant amount of variety in their data. For example, a customer name column, which contains many distinct values, would be a good candidate.

```
SQL> Create index customername_idx
  2  on customers (cname)
  3  tablespace index_tbs;

Index created.
```

- **Reverse-Key Indexes**

Reverse-key indexes can be used to avoid unbalanced indexes in certain situations, such as when ascending values are being inserted into a table, while lower values are deleted. (Sarin, 2000)

```
Create UNIQUE index employee_eid_idx
on employees(eid)
tablespace index_tbs
REVERSE;
```

- **Bitmap indexes**

Unlike B-Tree indexes, Bitmap indexes are appropriate for columns with only a few distinct values. In addition, they should only be used if the data is infrequently updated, as they add to the cost of all data manipulation transactions against the tables they index. (Sarin, 2000) For example, employees' postcode column would be appropriate for Bitmap index.

```
SQL> Create BITMAP index epcode_idx
  2  on employees(epcode)
  3  tablespace index_tbs;

Index created.
```

- **Function-Based Indexes**

Function-based index is one of the new types of indexes introduced in Oracle 8i. It is index that stores pre-computed values of functions or expressions. (Oracle8i Administrator's guide, 1999) A function-based index can be created with either a B-tree or bitmap index structure:

Example: create a function-based index base on the delivery date (DELVER_DATE) in the ORDERS table.

```
SQL> CREATE INDEX deliverdate_idx
  2  ON orders(delver_date);

Index created.
```

When DELVER_DATE is referenced in the SELECT or DELETE statement, the function-based index 'deliverdate_idx' will be used by the optimiser to retrieve the data. For example, when querying the following select statement, the 'deliverdate_idx' index will be used. Without the function-based index, the ORDERS table would need to perform a full table scanned.

```
SELECT orderid, itemid
FROM orders
WHERE delver_date = to_date ('22-APR-2002', 'DD-MM-YYYY');
```

22.2 Reorganising Indexes

- **Changing Storage Parameter for Indexes**

The storage parameters and block utilisation parameters can be modified by using the ALTER INDEX command. In the following example, the storage parameter of CUSTOMERNAME_IDX index has been changed to use a next extent size of 400k and a maximum extent size of 100.

```
SQL> ALTER INDEX customername_idx
  2  storage(next 400k
  3  maxextents 100);

Index altered.
```

- **Rebuilding Indexes**

The rebuild capability allows users to recreate an index without having to drop the existing index. The currently available index is used as the data source for the index instead of using the table as the data source.
In the following example, the customername_idx index is rebuilt in the index_tbs02 tablespace.

```
SQL> ALTER INDEX customername_idx Rebuild
  2  tablespace index_tbs02;

Index altered.
```

22.3 Dropping indexes

When an index is corrupt, invalid or no longer needed, it may need to be dropped. DROP INDEX is the command used to drop an index. In the following example, the index 'customername_idx' is dropped.

```
SQL> DROP INDEX CUSTOMERNAME_IDX;

Index dropped.
```

23. MANAGING USERS

23.1 Creating a new user

In order to create a new user, the CREATE USER system privilege needs to be granted. Since all DBA users accounts have this privilege, firstly log in SQL*PLUS as sysdba. Then enter the CREATE USER command.

The following shows a sample of CREATE USER statement.

```
SQL> connect/ as sysdba
Connected.
SQL> CREATE USER Mary
  2  IDENTIFIED BY usermary
  3  DEFAULT TABLESPACE users
  4  TEMPORARY TABLESPACE temp;

User created.
```

In the above example, a new user MARY has been created with password *usermary*. Objects created by this user will be stored by *user* tablespace. The temporary tablespace is *temp* in which the user to sort operations and also to store temporary LOBS and temporary tables.

At this point, MARY is not yet allowed to perform any tasks in the database. As shown in the following example, the attempt to connect MARY to the database was failed. The error message indicates that Mary lacks privileges to log in to the database.

```
SQL> connect mary/usermary;
ERROR:
ORA-01045: user MARY lacks CREATE SESSION privilege; logon denied
```

Therefore, after creating a new account, we need to grant system privilege to the new user, which gives the right to perform certain type of action in the database.

23.2 Granting System Privileges / Roles

Oracle provides three standard roles granted to new users. They are: (Oracle8i Administrator's Guide,1999)

The Connect Role

This role is usually given to occasional users. It gives the user the CREATE SESSION and ALTER SESSION privileges, and the ability to create tables, views, sequences, clusters, synonyms, and links to other databases.

The Resource Role

This role is usually granted to more sophisticated and regular users of the database and permits them to perform many of the same actions allowed with the CONNECT role. In addition, the user also has the privilege to create their own tables, sequences, procedures, triggers, indexes, and clusters.

The DBA Role

The DBA role includes all system privileges, as well as the ability to grant all privileges to other users.

Privilege is given to the user by using GRANT command. In the following example, the new user is granted with connect and resource privilege / role.

```
SQL> Grant create session to Mary;

Grant succeeded.

SQL> connect mary/usermary
Connected.
```

After being granted the CREATE SESSION privilege, user MARY is now able to log on to the database.

Enter the following CREATE TABLE command to create a table in Mary's account.

```
SQL> CREATE TABLE mtable
  2  (id INTEGER constraint mtable_pk primary key,
  3  name varchar2(10));
CREATE TABLE mtable
*
ERROR at line 1:
ORA-01031: insufficient privileges
```

However, the table creation failed and the error message indicates that Mary does not have sufficient privileges to create table in the database. This is because CREATE SESSION privilege merely allow users to log on to Oracle.

Connect to the database as SYSTEM, and grant CONNECT role to Mary.

```
SQL> connect system/manager
Connected.

SQL> grant connect to mary;

Grant succeeded.
```

Enter the CREATE TABLE statement again after log on to MARY's account.

```
SQL> connect mary/usermary
Connected.
SQL> CREATE TABLE mtable
  2  (id INTEGER constraint mtable_pk primary key,
  3  name varchar2(10));
CREATE TABLE mtable
*
ERROR at line 1:
ORA-01950: no privileges on tablespace 'USERS'
```

However, this time appears another error message, which indicates that User Mary must be given a resource quota on the USERS tablespace to be able to create and store objects within it. In the following command, using ALTER USER…QUOTA command to give the user 15M of space to store objects owned by user Mary in the tablespace USERS.

```
SQL> CONNECT SYSTEM/MANAGER;
Connected.
SQL> ALTER USER Mary QUOTA 15M ON USERS;

User altered.
```

After the quota space has been specified, user MARY is able to create objects using CREATE TABLE command.

```
SQL> CONNECT MARY/USERMARY
Connected.
SQL> CREATE TABLE mtable
  2  (id INTEGER constraint mtable_pk primary key,
  3  name varchar2(10));

Table created.
```

23.3 Dropping a User

Users can be dropped using DROP USER command.

```
SQL> DROP USER mary;
DROP USER mary
*
ERROR at line 1:
ORA-01922: CASCADE must be specified to drop 'MARY'
```

This response means that the MARY account cannot be dropped with the `DROP USER mary;` statement because there are objects exist within the user MARY's schema. As the error message indicats, CASCADE needs to be included in the statement in order to drop the user and also the objects owned by the user.

```
SQL> DROP USER mary CASCADE;

User dropped.
```

24. MANAGING PRIVILEGES

There are two types of Oracle privileges: system privileges and object privileges. System privileges include rights to perform system-wide database actions; the following are examples of system privileges: (Oracle8i Administrator's guide, 1999)

- GRANT ANY PRIVILEGE
- CREATE ROLE
- CREATE USER
- ALTER ANY INDEX
- DROP ANY TRIGGER
- SELECT ANY TABLE

Object privileges consist of rights to perform specific actions on specific schema objects. Examples of object privileges include:

- SELECT
- UPDATE
- ALTER
- DELETE
- EXECUTE
- INSERT
- INDEX
- REFERENCES

24.1 Granting Privileges

A user can grant privileges on any objects he or she owns whilst the DBA can grant any system privileges.

- **Granting System Privileges**

To grant a system privilege, the user must have either the GRANT ANY PRIVILEGE system privilege, or the system privilege that you are attempting to grant must have been granted to the user with the ADMIN OPTION.

For example, connect as SYSTEM, create two users, ROBERT and MARY with the following privileges.

```
SQL> grant create session, create table to mary;

Grant succeeded.

SQL> Grant create session to Robert;

Grant succeeded.
```

The above commands gives both MARY and ROBERT the ability to connect to ORACLE, and gives MARY extra capability to create tables. The privileges are granted by SYSTEM since SYSTEM has GRANT ANY PRIVILEGE and GRANT ANY ROLE privileges.

Now, grant CREATE USER system privilege to user Mary with admin option.

```
SQL> GRANT CREATE USER to Mary with ADMIN OPTION;

Grant succeeded.
```

```
SQL> connect mary/usermary
Connected.

SQL> CREATE USER Kate identified by cat;

User created.

SQL> connect system/manager
Connected.
SQL> grant create session to kate;

Grant succeeded.
```

```
SQL> connect mary/usermary
Connected.
SQL> grant create user to Kate;

Grant succeeded.

SQL> connect kate/cat;
Connected.
SQL> CREATE USER Vincent identified by sky;

User created.
```

The with admin option clause permits the grantee to give the SYSTEM privilege on other users. In this case, user MARY is able to grant other user (KATE) with CREATE USER system privilege. Therefore, Kate is able to create user VINCENT successfully.

- **Granting Object Privileges**

The object privilege can usually be granted by the owner of the object or by other user who has been given the object privilege with the WITH GRANT OPTION clause.

Grant Object Privileges by the Object Owner

For example, user MARY owns the CUSTOMERS table. In order for Robert to access the CUSTOMER table, the SELECT object privilege needs to be granted.

In the following, connect as SYSTEM, and grant ROBERT the SELECT privilege using GRANT command.

```
SQL> connect system/manager
Connected.
SQL> GRANT SELECT ON Mary.customers to Robert;
GRANT SELECT ON Mary.customers to Robert
                    *
ERROR at line 1:
ORA-01031: insufficient privileges
```

However, the above error message is generated. This is because that the object privilege can only be granted by the owner of the object or any other user who received the privilege from the owner with GRANT OPTION.

When reconnecting to the system as MARY and give ROBERT the SELECT privilege, the GRANT was succeed.

```
SQL> connect mary/usermary
Connected.
SQL>
SQL> GRANT SELECT ON Mary.customers to Robert;

Grant succeeded.
```

When connect as ROBERT and try to query the CUSTOMERS table, user ROBERT is able to select data from that table.

```
SQL> connect robert/userrob
Connected.
SQL> select * from Mary.customers;

       CID CFNAME          CLNAME
---------- --------------- ---------------
CADD                                                CPHONE
-------------------------------------------------- ----------
       101 Paul            Wilson
123 Clayton road                                    96782415

       102 Lucas           Roy
41 Glenhuntly road                                  95712800

       103 John            Smith
38 Mcmaster court                                   97580322


       CID CFNAME          CLNAME
---------- --------------- ---------------
CADD                                                CPHONE
-------------------------------------------------- ----------
       104 Lisa            Morrison
1 Flinder street                                    96508016

       105 Michael         Chen
3500 Lancell road                                   98133049
```

Grant Object Privileges with Grant Option

In the following example, user MARY, the CUSTOMERS table owner, grants SELECT privilege on her table to the user KATE with grant option.

```
SQL> CONNECT MARY/USERMARY
Connected.

SQL> GRANT SELECT ON customers to Kate WITH GRANT OPTION;

Grant succeeded.

SQL> CONNECT KATE/CAT
Connected.

SQL> GRANT SELECT ON MARY.CUSTOMERS TO VINCENT;

Grant succeeded.
```

The WITH GRANT OPTION clause allows the recipient of that grant to pass along the privileges he or she has received to other users. In this case, KATE is able to grant the SELECT privilege on MARY's CUSTOMERS table to user VINCENT even

though she is not the owner of the object. Therefore, VINCENT is allowed to query MARY'S CUSTOMERS table from his account.

```
SQL> CONNECT VINCENT/SKY
Connected.
SQL>  SELECT * FROM MARY.CUSTOMERS
  2   WHERE CID = 103;

       CID CFNAME          CLNAME
---------- --------------- ---------------
CADD                                                CPHONE
-------------------------------------------------- ----------
       103 John            Smith
38 Mcmaster court                                   97580322
```

24.2 Revoking Privileges

Privileges granted can be taken away by using REVOKE command.

- **Revoke System Privileges**

In order to revoke a system privilege from a user, It needs to be performed by users have that privilege with THE ADMIN OPTION. However, it is not necessary to be the user whom you granted the privilege from.

```
SQL> CONNECT MARY/USERMARY
Connected.

SQL> REVOKE CREATE USER FROM KATE;

Revoke succeeded.
```

```
SQL> CONNECT KATE/CAT
Connected.
SQL> CREATE USER IAN IDENTIFIED BY SWIM;
CREATE USER IAN IDENTIFIED BY SWIM
                              *
ERROR at line 1:
ORA-01031: insufficient privileges
```

Since user MARY has revoked the CREATE USER system privilege from user KATE, any attempt by user KATE to create a new user will result in the error message above.

- **Revoke Object Privileges**

Unlike revoking system privileges, revoking object privileges can only be done by the user who granted you with that object privilege at the first place.

```
SQL> connect system/manager;
Connected.
SQL> revoke select on mary.customers from robert;
revoke select on mary.customers from robert
*
ERROR at line 1:
ORA-01927: cannot REVOKE privileges you did not grant
```

The above error message indicates that SYSTEM does not have the right to revoke ROBERT's SELECT privilege since SYSTEM was not the one who granted ROBERT that privilege.

```
SQL> connect mary/usermary
Connected.
SQL> REVOKE SELECT ON customers from robert;

Revoke succeeded.
```

Once reconnecting as user MARY, the SELECT privilege is taken away successfully from ROBERT. This can be proved by connecting as ROBERT and querying the CUSTOMERS table from MARY's schema. An error message indicates that the table is no longer available to ROBERT.

```
SQL> connect robert/userrob;
Connected.
SQL> select * from mary.customers;
select * from mary.customers
                  *
ERROR at line 1:
ORA-00942: table or view does not exist
```

However, revoking object privileges has a cascading effect. The example further demonstrates that the user VINCENT's SELECT privilege on MARY's CUSTOMERS table is taken away when revoking the privilege from user KATE. This is because VINCENT's SELECT privilege is granted from KATE.

```
SQL> connect mary/usermary
Connected.
SQL>
SQL> REVOKE select ON mary.customers from Kate;

Revoke succeeded.
```

```
SQL> connect vincent/sky
Connected.
SQL> select
  2
SQL> SELECT * FROM MARY.CUSTOMERS;
SELECT * FROM MARY.CUSTOMERS
                  *
ERROR at line 1:
ORA-00942: table or view does not exist
```

25. MANAGING ROLES

A role is a named set of privileges that can be given to users and other roles. (Sarin, 2000) Using roles makes the task of managing security easier than granting privileges to individual users.

25.1 Creating a Role

A role is created with CREATE ROLE statement. In the following example, a role 'staff' is created.

```
SQL> CREATE ROLE staff;

Role created.
```

The DBA can also create a role with a password to prevent unauthorized use of the privileges granted to the role.

```
SQL> CREATE ROLE Supervisor
  2  identified by busy;

Role created.
```

25.2 Granting Privileges to a Role

After the role is created, we need to assign privileges to it. The privileges are assigned using GRANT statement.
The following example shows 'STAFF' has been assigned CREATE TABLE and CREATE SESSION system privileges.

```
SQL> GRANT create table, create session to staff;

Grant succeeded.
```

In the next example, the 'STAFF' role has also been granted the SELECT, INSERT, UPDATE and DELETE object privileges on user MARY's CUSTOMERS table.

```
SQL> connect mary/usermary
Connected.
SQL> GRANT SELECT, INSERT, UPDATE, DELETE
  2  ON mary.customers TO staff;

Grant succeeded.
```

25.3 Assign Roles

Roles can be granted to other roles or to users.

- **Granting a role to another role**

```
SQL> GRANT STAFF to SUPERVISOR;

Grant succeeded.
```

In this example, the STAFF role is granted to the SUPERVISOR role. Even though no privileges are granted directly to the SUPERVISOR role, it will now inherit any privileges that have been granted to the STAFF role.

- **Granting a role to users**

Instead of granting each privilege to each user, we can grant privileges to the role and then grant the role to each user. This greatly simplifies the administrative tasks involved in managing privileges.

```
SQL> CREATE USER ZOY IDENTIFIED BY FLY;

User created.

SQL> GRANT STAFF to zoy;

Grant succeeded.
```

In this example, STAFF role is granted to user ZOY. Therefore, ZOY is now inheriting all the privileges that have been granted to the STAFF role. As shown following, ZOY is now able to query from Mary's CUSTOMERS table.

```
SQL> connect zoy/fly
Connected.
SQL> select * from mary.customers
  2  where cid = 101;

       CID CFNAME          CLNAME
---------- --------------- ---------------
CADD                                                CPHONE
-------------------------------------------------- ----------
       101 Paul            Wilson
123 Clayton road                                    96782415
```

25.4 Revoking Privileges from a role

ROVOKE command is used to revoke a privilege from a role.

```
SQL> REVOKE delete ON employees from staff;

Revoke succeeded.
```

In the example, users of STAFF role will be unable to perform delete command on the EMPLOYEES table.

25.5 Dropping a Role

DROP ROLE command is used to drop a role. As shown in the following example:

```
SQL> connect system/manager;
Connected.
SQL> DROP ROLE supervisor;

Role dropped.
```

The SUPERVISOR role and its associated privileges are removed from the database entirely.

26. MANAGING PROFILES

A profile is a set of limits on the use of database resources. Profiles are used to limit users' system and database resources and to maintain password restrictions. (Oracle8i Administrator's Guide, 1999)

The following are the steps to enforce resource limit using profiles

26.1 STEP 1: Create profiles

A profile is created using CREATE PROFILE command. In the example, profile 'staff_profile' is created to limit the number of failed login attempts, the number of session allow for each user, etc.

```
SQL> CREATE PROFILE staff_profile LIMIT
  2  FAILED_LOGIN_ATTEMPTS 2
  3  SESSIONS_PER_USER 1
  4  CPU_PER_SESSION 10000
  5  IDLE_TIME 30
  6  CONNECT_TIME 360;

Profile created.
```

26.2 STEP 2: Assign profiles to the user.

- **For new user:**

Example: create new user JOHN and assign the 'staff_profile' profile to the user.

```
SQL> CREATE USER John IDENTIFIED BY userjohn
  2  DEFAULT TABLESPACE new_tablespace
  3  TEMPORARY TABLESPACE temp
  4  PROFILE staff_profile;

User created.
```

When the profile is not specified during the user creation, the Oracle Server will assign its default profile, called DEFAULT,to users. This profile specifies unlimited resources for each of its parameters.

- **For existing user:**

ALTER USER command is used to change existing user's profile setting.

```
SQL> ALTER USER Mary
  2  PROFILE staff_profile;

User altered.
```

26.3 STEP 3 : Enabling Resource Limits

In order to make the Oracle Server enforce resource limits, the value of the initialization parameter RESOURCE_LIMIT in the init.ora file must be set to TRUE. This can be done through:

Setting the initialization parameter RESOURCE_LIMIT to TRUE in the parameter file 'init.ora'. Then restart the databse.

or

Enforce the resource limits by enabling the parameter with the ALTER SYSTEM command so that there is no need to restart the database.

```
SQL> ALTER SYSTEM SET RESOURCE_LIMIT=TRUE;

System altered.
```

CHAPTER 4 Performance Tuning

27. SQL TUNING

MEASURE THE PERFORMANCE OF SQL QUERIES

Problems with poorly performing applications can frequently be traced to poorly written SQL statements. The following are commonly used tools for measuring the performance of SQL statements, SQL TRACE and TKPROF, Explain Plans and AUTOTRACE. Gathering and analysing application information in the form of trace files and execution plans is the first step to understanding how the application is performing.

27.1 SQL TRACE and TKPROF

Oracle trace files contain session information for the process that created them. The information is useful for performance tuning and system troubleshooting.

In order to examine a SQL statement, the DBA should obtain the trace output for the statement and use the TKPROF utility to convert the trace file into a form that can be more easily understood. The steps for setting up and running TRACE utility and TKPROF utility are as follows:

EXAMPLE: USER-SELF TRACE for user IVY

- **STEP 1: Set Initialization Parameters**

When tracing application code to gather performance statistics, the initialisation parameter timed_statistics should be set to true. The initialisation parameters max_dump_file_size and user_dump_dest should also be specified.

Action: setting the timed_statistics parameter via an ALTER SESSION command.

```
SQL> ALTER SESSION set TIMED_STATISTICS= TRUE;

Session altered.
```

Using the default value for the following parameters.

```
max_dump_file_size = 10240
user_dump_dest = C:\oracle\admin\project\udump
```

- **STEP 2: Activate the Tracing Process**

Action: enable tracing process within the user's session by using ALTER SESSION command.

```
SQL> ALTER SESSION SET sql_trace= TRUE;

Session altered.
```

- **STEP 3: Run the SQL statement**

Action: execute the following SQL statement

```
select eid, efname, elname, eadd ephone, did
from employees
where did = 101
order by eid;


select eid, efname, elname,eadd, epcode, did
from employees
where did in (101);


select orderid, ord_date, cid
from orders
where eid = 520;
```

The output trace file is created in the directory specified for the user_dump_dest parameter.

- **STEP 4: Disable the TRACE utility**

Action: disable tracing process with the following command.

```
SQL> ALTER SESSION SET SQL_TRACE = FALSE;

Session altered.
```

- **STEP 5: Format the TRACE file with TKPROF**

Action 1: Identify the generated trace file for user IVY.

```
SQL> SELECT s.username, p.spid
  2  FROM v$session s, v$process p
  3  WHERE s.paddr = p.addr
  4  AND p.background is null;

USERNAME                       SPID
------------------------------ ---------
                               1376
                               1324
                               1380
                               1384
SYS                            1200
DBSNMP                         1312
IVY                            1056
```

The result indicates that a trace file 'ora01056.trc' has been generated for user IVY's session.

Action 2: run TKPROF to convert the 'ora01056.trc' into a readable format. Execute the following command in command prompt.

```
C:>TKPROF c:\oracle\admin\project\udump\ora01056.trc
 d:\project\oracle\trace.txt explain=ivy/000 sys=no
```

The TKPROF translates the TRACE file to a readable format 'trace.txt'.

- **STEP 6: Interpret the Output**

An example of the output is shown below. For the complete list of the TKPROF output (trace.txt), please refer to the Appendix D.

```
select orderid, ord_date, cid
from orders
where eid = 520

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.30       0.49          1          0          1           0
Execute      1      0.00       0.00          0          0          0           0
Fetch        2      0.00       0.00          1          2          4           3
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        4      0.30       0.49          2          2          5           3

Misses in library cache during parse: 1
Optimizer goal: CHOOSE
Parsing user id: 48  (IVY)

Rows     Row Source Operation
-------  ---------------------------------------------------
      3  TABLE ACCESS FULL ORDERS


Rows     Execution Plan
-------  ---------------------------------------------------
      0  SELECT STATEMENT   GOAL: CHOOSE
      3   TABLE ACCESS (FULL) OF 'ORDERS'
```

The example output shows 2 disk reads and 7 memory reads (query plus current). The disk column indicates the physical reads usually when no index is used. The execution path indicates a full table scan confirms that there might be a potential problem.

27.2 EXPLAIN PLAN

Oracle's Explain Plan facility is used to determine how a particular SQL statement is processing. It does not actually execute the SQL statement but only shows what will happen if the statement is executed. (Niemiec, 1999) With this information, it is possible to improve the statement's performance by rewriting the SQL code to eliminate unwanted behaviour.

The steps to evaluate the performance of SQL statement using EXPLAIN PLAN tool are as following:

- **STEP 1: Create the PLAN_TABLE**

The plan table is used to insert the query execution plan in the form of records. This step only needs to be performed once.

Action: create the plan table with the ORACLE 'utlxplan.sql' script.

```
SQL> @c:\oracle\ora81\rdbms\admin\utlxplan.sql

Table created.
```

- **STEP 2: Run EXPLAIN PLAN for the Query to be Optimized**

Action: Populate the PLAN_TABLE with the SQL statement's execution plan using the EXPLAIN PLAN FOR command:

```
SQL> EXPLAIN PLAN FOR
  2  select orderid, eid
  3  from orders o, items i
  4  where o.itemid = i.itemid and
  5  (ord_qty*price) = (select MAX(ord_qty*price) from
  6  orders o, items i where o.itemid = i.itemid);

Explained.
```

- **STEP 3: Query the Plan Table**

Action 1: query the PLAN_TABLE in order to see how the explained statement would be executed:

```
SQL> select id, operation, options, object_name
  2  from plan_table;
```

```
        ID OPERATION                      OPTIONS       OBJECT_NAME
---------- ------------------------------ ---------------  -------------
         0 SELECT STATEMENT


         1 FILTER


         2 NESTED LOOPS


         3 TABLE ACCESS                   FULL             ORDERS

         4 TABLE ACCESS                     BY INDEX ROWID   ITEMS

         5 INDEX                          UNIQUE SCAN      ITEMS_ITEMID_PK

         6 SORT                             AGGREGATE


         7 NESTED LOOPS


         8 TABLE ACCESS                   FULL             ORDERS

         9 TABLE ACCESS                     BY INDEX ROWID ITEMS

        10 INDEX                          UNIQUE SCAN      ITEMS_ITEMID_PK


11 rows selected.
```

Action 2: Retrieving a more readable output by using the following command.

```
SQL> select lpad(' ', 4*(level-2)) ||LEVEL||'.'||nvl(position,0)||' '
  2  ||operation||' '||options||' '||object_name
  3  "Execution Plan"
  4  from plan_table
  5  start with id = 0
  6  connect by prior id = parent_id;
```

```
Execution Plan
----------------------------------------------------------------------------
1.0 SELECT STATEMENT
2.1 FILTER
    3.1 NESTED LOOPS
        4.1 TABLE ACCESS FULL ORDERS
        4.2 TABLE ACCESS BY INDEX ROWID ITEMS
            5.1 INDEX UNIQUE SCAN ITEMS_ITEMID_PK
    3.2 SORT AGGREGATE
        4.1 NESTED LOOPS
            5.1 TABLE ACCESS FULL ORDERS
            5.2 TABLE ACCESS BY INDEX ROWID ITEMS
                6.1 INDEX UNIQUE SCAN ITEMS_ITEMID_PK

11 rows selected.
```

- **STEP 4: Interpret the EXPLAIN PLAN Output**

Action: interpret the EXPLAIN PLAN output derived from last step.

The Explain Plan output is interpreted by starting at the innermost operation in the Explain Plan.

6.1 This step is executed first. The ITEMS_ITEMID_PK index is scanned to produce the list of ROWIDs to the following step.

5.2 For each ROWID returned, this operation will access the ITEMS table by ROWID, get the requested data, and return the data to the following step.

5.1 Oracle takes the resulting rows from the FTS of ORDERS and then compares the results of that operation to each row in the previous operation (5.2) via the ITEMS_ITEMID_PK index.

The TABLE ACCESS FULL operation in the Explain Plan indicates that a Full Table Scan (FTS) occurred for table ORDERS. These operations cause every row in a table to be accessed. This can be a costly operation. The appropriate use of indexing may help to minimize performance problems associated with FTS.

4.1 The NESTED LOOPS operation indicates that the ORDERS table is jointed to the ITEMS table via the ITEMS_ITEMID_PK index. This NESTED LOOPS produces the result of the subquery.

3.2 The rows returned from the last operation will be analysed via the SORT AGGREGATE operation, which will return the maximum (ord_qty*price) value to the user.

5.1 For each row of data received from 3.2, this operation will use the item ID to perform a unique scan to get the ROWID.

4.2 Access the ITEMS table by ROWID and retrieve the data.

4.1 The Oracle optimiser takes the resulting rows from the FTS of ORDERS and then compares the results of that operation to each row in the previous operation (4.2) via the ITEMS_ITEMID_PK index.

3.1 The NESTED LOOPS operation indicates that the ORDERS table is jointed to the ITEMS table via the ITEMS_ITEMID_PK index.

2.1 The FILTER operation indicates that the rest of the conditions of the WHERE clause are applied.

1.0 This indicates it is a SELECT statement.

27.3 AUTOTRACE Utility

EXPLAIN PLAN and statistics about the performance of a query can also be generated via AUTOTRACE.

The steps to evaluate the performance of SQL statement using EXPLAIN PLAN tool are as following:

- **STEP 1: Create the PLAN_TABLE**

The PLAN_TABLE need to exist for the user to insert the EXPLAIN PLAN output record. As it has already been created earlier, this step does not need to be performed again for the example.

- **STEP 2: Grant the Trace Role to The User**

The PLUSTRACE role needs to be granted to the users in order for them to perform the AUTOTRACE utility.

Action 1: create a database role, 'PLUSTRACE' role, by running the ORACLE script 'plustrce.sql'.

```
SQL> @c:\oracle\ora81\sqlplus\admin\plustrce.sql
```

Action 2: grant the 'PLUSTRACE' role to user IVY.

```
SQL> GRANT plustrace TO Ivy;

Grant succeeded.
```

The following error message will be generated if the PLUSTRACE role is not granted to the user to perform the AUTOTRACE.

```
SQL> SET AUTOTRACE ON
SP2-0613: Unable to verify PLAN_TABLE format or existence
SP2-0611: Error enabling EXPLAIN report
SP2-0618: Cannot find the Session Identifier.  Check PLUSTRACE role is
enabled
SP2-0611: Error enabling STATISTICS report
```

- **STEP 3: Enable AUTOTRACE Utility**

Action: issue the following command to activate AUTOTRACE

```
SET AUTOTRACE ON
```

- **STEP 4: Run the Query to be Optimized**

Action: issue the following SELECT statement

```
SQL> SELECT orderid, delver_date
  2  FROM orders
  3  WHERE delver_date =
  4  TO_DATE('April 22, 2002','Month dd, YYYY');
```

Output:

```
   ORDERID DELVER_DA
---------- ---------
      3333 22-APR-02
      3334 22-APR-02


Execution Plan
----------------------------------------------------------
   0      SELECT STATEMENT Optimizer=CHOOSE
   1    0   TABLE ACCESS (FULL) OF 'ORDERS'



Statistics
----------------------------------------------------------
        260  recursive calls
          5  db block gets
         54  consistent gets
          2  physical reads
          0  redo size
        474  bytes sent via SQL*Net to client
        425  bytes received via SQL*Net from client
          2  SQL*Net roundtrips to/from client
          4  sorts (memory)
          0  sorts (disk)
          2  rows processed
```

OPTIMISE SQL STATEMENTS

There are varies methods available to improve SQL statements efficiency. Methods, include hints, indexes and parallel operations, are most commonly used to tune the problem queries. The approach is to experiment with optimiser hints and try alternate forms of the statement.

27.4 Hints

The use of hints in a given query may achieve better performance. Hints provide a mechanism to direct the optimiser to choose a certain query execution plan based on the type of operation required. (Oracle8i Designing and Tuning for Performance, 1999)

- **Hint for Execution Path**

These hints modify the execution path and use the specified optimisation approach, either cost-based or rule-based or both. They will override what is specified in the initialisation parameter. This type of hints includes:

- ALL_ROWS
- CHOOSE
- FIRST_ROWS
- RULE

Example 1: optimise the following query with a hint that can give the best response time.

The elapsed time is derived from the TKPROF output. For the complete list of the result for each experiment, pleas refer to Appendix E.

```
Select * from orders;
```

Elapsed time: 0.42

➢ **TUNING ACTION1: SPECIFY A FULL HINT**

```
SELECT /*+ FULL(ORDERS)*/*
FROM ORDERS;
```

Elapsed time: 0.20

➢ **TUNING ACTION 2: SPECIFY AN INDEX HINT**

```
SELECT /*+ INDEX (ORDERS ORDERS_ORDERID_PK)*/*
FROM ORDERS;
```

Elapsed time: 0.11

➢ **TUNING ACTION 3: SPECIFY AN RULE HINT**

```
SELECT /*+ RULL */*
FROM ORDERS;
```

Elapsed time: 0.02

The FULL hint and INDEX hint belong to access methods hint category. Those hints allow the user to vary the way the actual query is accessed. However, they use higher memory and CPU to perform the query. As a result, the response time is longer. In the example, out of the three hints, RULE hint achieves the best response time.

- **Hints for Join Orders**

Example: Optimise the performance of the following join table query, which returns all the orderid, customer ID, customer name and total amount for each order. The goal of this optimisation is to find the best response time. The AUTOTRACE and TKPROF utilities are activated to generate the EXPLAIN PLAN outputs.

```
SQL> Select orderid, orders.cid, cname AS "Customers", (ord_qty*price) as
"Total"
  2  from orders, customers, items
  3  where orders.itemid = items.itemid and
  4  orders.cid = customers.cid;
```

The AUTOTRACE output

```
Execution Plan
----------------------------------------------------------
   0      SELECT STATEMENT Optimizer=CHOOSE
   1    0   NESTED LOOPS
   2    1     NESTED LOOPS
   3    2       TABLE ACCESS (FULL) OF 'ORDERS'
   4    2       TABLE ACCESS (BY INDEX ROWID) OF 'ITEMS'
   5    4         INDEX (UNIQUE SCAN) OF 'ITEMS_ITEMID_PK' (UNIQUE)
   6    1     TABLE ACCESS (BY INDEX ROWID) OF 'CUSTOMERS'
   7    6       INDEX (UNIQUE SCAN) OF 'CUSTOMERS_CID_PK' (UNIQUE)


Statistics
----------------------------------------------------------
          0  recursive calls
          4  db block gets
         34  consistent gets
          3  physical reads
          0  redo size
       1025  bytes sent via SQL*Net to client
        425  bytes received via SQL*Net from client
          2  SQL*Net roundtrips to/from client
          0  sorts (memory)
          0  sorts (disk)
          8  rows processed
```

TKPROF output

```
call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.04       0.09          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch        2      0.00       0.02          3         34          4           8
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        4      0.04       0.11          3         34          4           8
```

- **TUNING ACTION 1: SPECIFY ORDERED HINT IN THE QUERY**

Action: include an ORDERED hint in the query.

```
SQL> Select /*+ ORDERED */ orderid, orders.cid, cname AS "Customers",
(ord_qty*price) as "Total"
  2  from orders, customers, items
  3  where orders.itemid = items.itemid and
  4  orders.cid = customers.cid;
```

```
Execution Plan
----------------------------------------------------------
   0      SELECT STATEMENT Optimizer=CHOOSE (Cost=5 Card=55 Bytes=5390)

   1    0   HASH JOIN (Cost=5 Card=55 Bytes=5390)
   2    1     HASH JOIN (Cost=3 Card=67 Bytes=4824)
   3    2       TABLE ACCESS (FULL) OF 'ORDERS' (Cost=1 Card=82 Bytes=4264)

   4    2       TABLE ACCESS (FULL) OF 'CUSTOMERS' (Cost=1 Card=82 Bytes=1640)

   5    1     TABLE ACCESS (FULL) OF 'ITEMS' (Cost=1 Card=82 Bytes=2132)


Statistics
----------------------------------------------------------
         82  recursive calls
         12  db block gets
          8  consistent gets
          0  physical reads
          0  redo size
       1001  bytes sent via SQL*Net to client
        425  bytes received via SQL*Net from client
          2  SQL*Net roundtrips to/from client
          3  sorts (memory)
          0  sorts (disk)
          8  rows processed
```

TKPROF output

```
call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.09       0.14          0          0          0           0
Execute      1      0.00       0.02          0          0          0           0
Fetch        2      0.00       0.03          0          4         12           8
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        4      0.09       0.19          0          4         12           8
```

The ORDERED hint forces the ORDERS, CUSTOMERS and ITEMS tables to be accessed in a particular order, based on their orders in the FROM clause of the query. The ORDERS table is the driving table and is accessed first. As the execution plan shows, a HASH join is performed. It indicates that a hash table is created by ORACLE to store the result sets derived from scanning the ORDERS table. Then it scans the second table 'CUSTOMERS' to retrieve corresponding records and following the result would be joined with the ITEMS table, which is accessed last.

As the EXECUTION PLAN indicates, the *consistent gets* and the *physical read* are improved. However, the use of HASH joins increases the use of memory resource. The elapsed time is end up increase to 0.19 second.

- **TUNING ACTION 2: USE THE ORDERED HINT AND VARY THE ORDER OF THE TABLES**

Action: change the order of ITEMS table and CUSTOMERS table in the clause.

```
SQL> Select /*+ ORDERED */ orderid, orders.cid, cname AS "Customers",
(ord_qty*price) as "Total"
  2  from orders, items, customers
  3  where orders.itemid = items.itemid and
  4  orders.cid = customers.cid;
```

```
Execution Plan
----------------------------------------------------------
   0      SELECT STATEMENT Optimizer=CHOOSE (Cost=5 Card=55 Bytes=5390)

   1    0   HASH JOIN (Cost=5 Card=55 Bytes=5390)
   2    1     HASH JOIN (Cost=3 Card=67 Bytes=5226)
   3    2       TABLE ACCESS (FULL) OF 'ORDERS' (Cost=1 Card=82 Bytes=4264)

   4    2       TABLE ACCESS (FULL) OF 'ITEMS' (Cost=1 Card=82 Bytes=2132)

   5    1     TABLE ACCESS (FULL) OF 'CUSTOMERS' (Cost=1 Card=82 Bytes=1640)



Statistics
----------------------------------------------------------
          0  recursive calls
         12  db block gets
          4  consistent gets
          0  physical reads
          0  redo size
        990  bytes sent via SQL*Net to client
        425  bytes received via SQL*Net from client
          2  SQL*Net roundtrips to/from client
          3  sorts (memory)
          0  sorts (disk)
          8  rows processed
```

TKPROF output

```
call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.05       0.05          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch        2      0.00       0.00          0          4         12           8
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        4      0.05       0.05          0          4         12           8
```

The order of the tables in the FROM clause has alter the order in which Oracle perform the HASH join. The elapsed time of this query is down to only 0.05 second. The query performance is much improved.

27.5 Indexes

It is possible to improve SQL queries by forcing the optimiser to scan all the entries from the index instead of the table.

If the index is physically smaller than the table, it will take less time to scan the entire index than to scan the entire table.

Example: use a function index to optimise the following query.

```
SQL> SELECT orderid, itemid
  2  FROM orders
  3  WHERE delver_date <= to_date ('22-APR-2002', 'DD-MM-YYYY');

   ORDERID     ITEMID
---------- ----------
      3331      90003
      3332      90006
      3333      90001
      3334      90003
```

```
Execution Plan
----------------------------------------------------------
   0      SELECT STATEMENT Optimizer=CHOOSE
   1    0   TABLE ACCESS (FULL) OF 'ORDERS'


Statistics
----------------------------------------------------------
        181  recursive calls
          5  db block gets
         42  consistent gets
          0  physical reads
          0  redo size
        573  bytes sent via SQL*Net to client
        425  bytes received via SQL*Net from client
          2  SQL*Net roundtrips to/from client
          4  sorts (memory)
          0  sorts (disk)
          4  rows processed
```

TKPROF output

```
call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        3      0.38       0.78          0          0          2           0
Execute      3      0.00       0.00          0          0          0           0
Fetch        6      0.00       0.02          0          8          8          12
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total       12      0.38       0.80          0          8         10          12

Misses in library cache during parse: 3
Optimizer goal: CHOOSE
Parsing user id: 48  (IVY)
```

The execution plan indicates that Oracle performed a full table scan on the ORDERS table. Not only does the query run slowly, it also exploits higher amount of memory and CPU to perform the query. .

- **ACTION: CREATE FUNCTION INDEX FOR 'DELVER_DATE'**

```
SQL> CREATE INDEX delvery_date_idx
  2  ON IVY.orders(delver_date);

Index created.
```

- **ACTION: EXECUTE THE QUERY AGAIN.**

```
SQL> SELECT orderid, itemid
  2  FROM orders
  3  WHERE delver_date <= to_date ('22-APR-2002', 'DD-MM-YYYY');

   ORDERID     ITEMID
---------- ----------
      3331      90003
      3332      90006
      3333      90001
      3334      90003
```

```
Execution Plan
----------------------------------------------------------
   0      SELECT STATEMENT Optimizer=CHOOSE
   1    0   TABLE ACCESS (BY INDEX ROWID) OF 'ORDERS'
   2    1     INDEX (RANGE SCAN) OF 'DELVERY_DATE_IDX' (NON-UNIQUE)



Statistics
----------------------------------------------------------
          0  recursive calls
          0  db block gets
          4  consistent gets
          0  physical reads
          0  redo size
        573  bytes sent via SQL*Net to client
        425  bytes received via SQL*Net from client
          2  SQL*Net roundtrips to/from client
          0  sorts (memory)
          0  sorts (disk)
          4  rows processed
```

TKPROF output

```
call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        2      0.15       0.43          0          0          1           0
Execute      2      0.00       0.00          0          0          0           0
Fetch        4      0.00       0.02          0          6          4           8
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        8      0.15       0.45          0          6          5           8

Misses in library cache during parse: 2
Optimizer goal: CHOOSE
Parsing user id: 48  (IVY)
```

The query is now faster with the added index. Since the ‘delver_date’ column has been properly indexed, Oracle does not need to perform a full table scan. The number of memory reads is lower and subsequently reduce the physical reads.

28. TUNING THE SHARE POOL

The Shared Pool is the portion of the System Global Area (SGA) that caches the SQL and PL/SQL statements that have been recently issued by application users. The Shared Pool is made up of three components: the Library Cache, the Data Dictionary Cache, and the User Global Area.

MEASURE THE PERFORMANCE OF THE SHARE POOL

The performance of the Shared Pool is measured by calculating the hit ratios for both library cache and data dictionary cache.

28.1 Library Cache

The tuning goals for tuning library cache include: (Oracle8i Designing and Tuning for Performance Release 2 (8.1.6), 1999)

- Reduce misses
- Avoid fragmentation

The performance of the Library Cache is measured by calculating the reload ratio, hit ratio, pinhit ratio, etc. The hit ratio information can be located in the V$LIBRARYCACHE dynamic performance view.

- **Monitor the library cache RELOAD ratio**

```
SQL> select sum(pins) Executions, sum(reloads) "CACHE MISSES WHILE
     EXECUTING", sum(reloads)/sum(pins) "RELOAD RATIO"
  2  from v$librarycache;


EXECUTIONS CACHE MISSES WHILE EXECUTING RELOAD RATIO
---------- --------------------------- ----------------------
    289546                          56              .000193406
```

The sum of the `EXECUTIONS` column indicates that SQL statements, PL/SQL blocks, and object definitions were accessed for execution a total of 289,546 times.

The sum of the `CACHE MISSES WHILE EXECUTING` column indicates that 56 of those executions resulted in library cache misses causing Oracle to implicitly re-parse a statement or block or reload an object definition because it aged out of the library cache.

The ratio of the total misses to total executions is about 0.19%. This value means that only 0.19% of executions resulted in re-parsing. The state of current system is considered well tuned since the reload ratio is below 1 percent.

- **Monitor the library cache HIT ratio**

```
SQL> select SUM(PINS) / (SUM(PINS) + SUM(RELOADS)) "HIT RATIO"
  2  FROM V$LIBRARYCACHE;

HIT RATIO
----------
.999824564
```

A library cache hit ratio of over 99.98 percent is achieved. There is no need to increase the SHARE_POOL_SIZE parameter. Consideration should be given to tune the SHARE_POOL_SIZE parameter if the ratio is below 95 percent.

- **Monitor the PIN HIT ratio**

```
SQL> select (sum(pinhits) / sum(pins)) "PIN HIT RATIO"
  2  FROM V$LIBRARYCACHE;

PIN HIT RATIO
-------------
   .993081998
```

The ratio indicates that every time an item is executed in the system, 99.3 percent of the time that it is already allocated and valid in cache. This ratio should always be maintained to close to 1.

28.2 Data Dictionary Cache

The main tuning goal for tuning data dictionary cache is to avoid dictionary cache misses. The hit ratio information is located in the V$ROWCACHE dynamic performance view.

- **Monitoring the Dictionary Cache HIT ratio**

```
SQL> select  sum(gets) "Gets", sum(getmisses) "Misses",
  2          (1 - (sum(getmisses) / (sum(gets) +
  3   sum(getmisses))))*100 "HitRate"
  4  from   v$rowcache;

      Gets     Misses    HitRate
---------- ---------- ----------
     12557        820 93.8700755
```

The result of this query shows that the application is finding the data dictionary information it needs, already in memory, 93.87 percent of the time. Consideration should be given to tuning the Shared Pool if the Data Dictionary hit ratio is less than 85 percent.

28.3 Monitor SHARE_POOL_SIZE memory

Monitoring the size of free memory available in SHARE_POOL_SIZE also helps to determine if the SHARE_POOL_SIZE parameter needs to be increased.

Action: run the following query to find out how fast memory in the Shared Pool is being depleted and also what percentage of memory is unused.

```
SQL> col value for 999,999,999,999 heading 'Shared Pool Size'
SQL> col bytes for 999,999,999,999 heading 'Free Bytes'
SQL> select    to_number(v$parameter.value) value, v$sgastat.bytes,
  2  (v$sgastat.bytes/v$parameter.value)*100 "Percent_Free"
  3  from  v$sgastat, v$parameter
  4  where v$sgastat.name = 'free memory'
  5  and v$parameter.name = 'shared_pool_size';

Shared Pool Size       Free Bytes Percent_Free
---------------- ---------------- ------------
      31,457,280       25,613,512   81.4231618
      31,457,280          614,400     1.953125
      31,457,280       16,105,472   51.1979167
```

This query was run after starting the database and running other queries for about 2 hours. The result indicates that there is plenty of free memory after the system is run for some time. Therefore, there is no need to increase the SHARE_POOL_SIZE parameter.

OPTIMISE THE SHARE POOL

If the share pool hit ratio is low, the DBA might need to consider to improve the performance of Shared Pool. The technique for optimising the Shared Pool performance is as following:

28.4 Increase Share Pool Size

This is achieved by increasing the value of initialisation parameter, SHARED_POOL_SIZE.

To estimate the adequate size for the Share Pool, run the following query after starting up the database, the instance and running for a period of time.

```
SQL> select sum(a.spspv) "Packages/Views", sum(a.spssql) "SQL Statements",
  2  sum(a.spsusr) "SQL Users", round((sum(a.spspv) + sum(a.spssql) +
  3  sum(a.spsusr)) * 2.5,-6) "Estimated shared_pool_size"
  4  from (select sum(sharable_mem) spspv, 0 spssql, 0 spsusr
  5  from v$db_object_cache
  6  union all
  7  select 0, sum(sharable_mem), 0 from v$sqlarea
  8  where executions > 5
  9  union all
 10  select 0, 0, sum(250 * users_opening) from v$sqlarea) a;

Packages/Views SQL Statements  SQL Users Estimated shared_pool_size
-------------- -------------- ---------- --------------------------
      12954529        1236777       4250                   35000000
```

(Query source: http://www.uaex.edu/srea/shared_pool_size.sql)

28.5 Pin (Cache) PL/SQL Object Statements Into Memory

- **Step 1: Build DBMS_SHARED_POOL**

Action: create the package with the ORACLE 'dbmspool.sql' script.

```
SQL> @c:\oracle\ora81\rdbms\admin\dbmspool.sql

Package created.


Grant succeeded.


View created.


Package body created.
```

- **Step 2: Pin the selected PL/SQL object statements**

Action: pin 'orderspackage' package, created in the SQL and PL/SQL module, in memory using the DBMS_SHARED_POOL.KEEP procedure.

```
SQL> EXECUTE DBMS_SHARED_POOL.KEEP ('orderspackage');
```

29. TUNING THE DATABASE BUFFER CACHE

MEASURE THE PERFORMANCE OF THE DATABASE BUFFER CACHE

29.1 Monitor the Data Buffer Hit Ratio For the Entire Instance

Action: run the following query to determine if the data block buffer is set high enough

```
SQL> SELECT 1 - (physical.value / (blockgets.value+consistent.value))
  2  "Buffer Cache Hit Ratio"
  3  FROM v$sysstat physical,
  4  v$sysstat blockgets,
  5  v$sysstat consistent
  6  WHERE physical.name = 'physical reads'
  7  AND blockgets.name = 'db block gets'
  8  AND consistent.name = 'consistent gets';


Buffer Cache Hit Ratio
----------------------
            .981596242
```

According to Sarin (2000), a data base buffer hit ratio of 1 is ideal. The closer the data buffer hit ration is to 1, the better the performance. The result of this query indicates that around 98 percent of time, Oracle found the data blocks it needed in memory, instead of having to read them from disk. No modification is needed for the buffer cache.

29.2 Monitor the Data Buffer Hit Ratio For an Individual Session

Action: run the following query to calculate a per-session Database Buffer Cache hit ratio. In this example, the `V$SESS_IO` and `V$SESSION` views are used.

```
SQL> SELECT username, osuser,
  2  1 - (io.physical_reads/(io.block_gets+io.consistent_gets))
  3  "Hit Ratio"
  4  FROM v$sess_io io, v$session sess
  5  WHERE io.sid = sess.sid
  6  AND (io.block_gets + io.consistent_gets) ! = 0
  7  AND username IS NOT NULL;

USERNAME                       OSUSER                          Hit Ratio
------------------------------ ------------------------------ ----------
John                           IVY-B2HSLL1HHG7\Ivy            .991962175
IVY                            SYSTEM                         .697812892
MARY                           IVY-B2HSLL1HHG7\Ivy                     1
```

The result of this query shows the database buffer cache hit ratio for user IVY, JOHN and MARY. While the overall database performance is acceptable, user IVY might be experiencing performance issues.

OPTIMISE THE DATABASE BUFFER CACHE

If the database buffer hit ratio is low, the DBA might need to consider improving the performance of the data buffer. The techniques for increasing the database buffer hit ratio are as following:

29.3 Increase Buffer Cache Size

The easiest way to improve the performance of the Database Buffer Cache is to increase its size. The size of the Database Buffer Cache is determined by the DB_BLOCK_SIZE and DB_BLOCK_BUFFERS in the 'init.ora' parameters.

29.4 Cache Tables in Memory

Cache tables can be implemented in three ways:

- **EXAMPLE 1: MODIFY THE EXISTING TABLE EMPLOYEES INTO A CACHE TABLE**

```
SQL> ALTER TABLE employees CACHE;

Table altered.
```

The EMPLOYEES table is cached in memory. This prevents its data from being aged out of the data buffers.

- **EXAMPLE 2: CREATE A NEW CACHE TABLE SALARY.**

```
SQL> CREATE TABLE salary
  2  ( employee_id number,
  3  workmode varchar2(11),
  4  pay_per_hr number,
  5  total_hr number,
  6  bankdetail varchar2(30))
  7  TABLESPACE new_tablespace
  8  STORAGE (INITIAL 50K NEXT 50K PCTINCREASE 0)
  9  CACHE;

Table created.
```

- **EXAMPLE 3: USE HINTS TO CACHE**

```
SQL> select /*+CACHE*/ EID, ELNAME, EFNAME
  2  FROM EMPLOYEES;

       EID ELNAME          EFNAME
---------- --------------- ----------
       520 Peterson        Jim
       521 Manson          Clair
       522 Smith           Todd
       523 Gwen            Rebecca
       524 Joans           Mareen
       525 Chang           Miller
       526 Taylor          Jessica
       528 Spenser         Nancy

8 rows selected.
```

- **EXAMPLE 4: DISPLAY CACHE TABLE INFORMATION**

```
SQL> SELECT owner, table_name
  2  FROM dba_tables
  3  WHERE LTRIM(cache) = 'Y';

OWNER                          TABLE_NAME
------------------------------ ----------------
IVY                            EMPLOYEES
SYSTEM                         SALARY
```

30. TUNING THE REDO LOG BUFFER

The main purpose to tune the Redo Log Buffer is to ensure there is adequate space so that log space requests from server processes and transactions can be satisfied.

MEASURE THE PERFORMANCE OF THE REDO LOG BUFFER

30.1 Using V$SYSSTAT Performance View

```
SQL> SELECT retries.value/entries.value "Redo Log Buffer
  2  Retry Ratio"
  3  FROM v$sysstat retries, v$sysstat entries
  4  WHERE retries.name = 'redo buffer allocation retries'
  5  AND entries.name = 'redo entries';

Redo Log Buffer
Retry Ratio
---------------------------
                 .001230976
```

The result shows, for every entry that is placed in to the Redo Log Buffer by the user server processes since the instance start up, 0.12 % of the time user server processes has to wait and then retry placing the entry in the Redo Log Buffer because LGWR had not yet written the current entries to the online redo log.

Oracle recommends that this Redo Log Buffer Retry Ratio should be less than 1 percent. (Oracle8i Designing and Tuning for Performance Release 2 (8.1.6) 1999) The result returned by the query indicates that the redo log buffer is in the good condition.

30.2 Using V$SESSION_WAIT Performance View

```
SQL> SELECT username, wait_time, seconds_in_wait
  2  FROM v$session_wait, v$session
  3  WHERE v$session_wait.sid = v$session.sid
  4  AND event LIKE 'log buffer space';

no rows selected
```

The SECONDS_IN_WAIT value of the “log buffer space” event indicates the time spent waiting for space in the redo log buffer because the log switch does not occur. This is an indication that the buffers are being filled up faster than LGWR is writing. This may also indicate disk I/O contention on the redo log files.

As no rows are returned form the above query, no tuning action is needed for the redo log buffer.

30.3 Using V$SYSTEM_EVENT Performance View

```
SQL> select event, total_waits, time_waited, average_wait
  2  from v$system_event
  3  where event like 'log file switch completion%';

no rows selected
```

This dynamic performance view reports the number of waits that have occurred since instance startup for a variety of events.

As no rows are returned, it indicates that no waits have occurred.

OPTIMISE THE REDO LOG BUFFER

30.4 Increase Redo Log Buffer Size

The size of the Redo Log Buffer is determined by the LOG_BUFFER in the 'init.ora' parameters. The way to improve the performance of the Redo Log Buffer is to increase the value of this initialisation parameter.

30.5 Reduce Redo Generation

The alternative way to improve the performance of the Redo Log Buffer is to reduce the amount of redo information generated by certain DML statement. This is achieved by using UNRECOVERABLE or NOLOGGING keyword.

- **UNRECOVERABLE keyword**

It is used when creating a table using the CREATE TABLE AS SELECT… SQL command:

- **EXAMPLE : CREATE A TABLE ORDER_HISTORY WITH THE UNRECOVERABLE KEYWORD**

```
SQL> CREATE TABLE order_history
  2  AS
  3  SELECT *
  4  FROM orders
  5  UNRECOVERABLE;

Table created.
```

Tables created in this manner do not generate any redo information for the inserts generated by the CREATE statement's sub-query.

- **NOLOGGING keyword**

  - **EXAMPLE 1: MODIFY THE EXISTING TABLE ORDER_HISTORY INTO A NOLOGGING MODE**

```
SQL> ALTER TABLE order_history NOLOGGING;

Table altered.
```

  - **EXAMPLE 2: CREATE A NEW NOLOGGING TABLE**

```
SQL> CREATE TABLE BACKORDER
  2  (backorderid number,
  3  date_expected date,
  4  date_received date,
  5  qty_received number,
  6  orderid number(5))
  7  TABLESPACE new_tablespace
  8  STORAGE (INITIAL 500k NEXT 500k PCTINCREASE 0)
  9  NOLOGGING;

Table created.
```

In both examples, redo entry generation will be suppressed for all subsequent DML on the ORDER_HISTORY and BACKORDER tables if that DML is of the following types: (Sarin, 2000)

- Direct Path loads using SQL*Loader
- Direct load inserts using the /*+ DIRECT */ hint

  - **EXAMPLE 3: DISPLAY TABLES WITH NOLOGGING ATTRIBUTE**

```
SQL> SELECT owner, table_name
  2  FROM dba_tables
  3  WHERE logging = 'NO';

OWNER                          TABLE_NAME
------------------------------ ------------------
SYS                            ATEMPTAB$
SYSTEM                         DEF$_TEMP$LOB
PORTAL30_DEMO                  RUPD$_EMP
SCOTT                          RUPD$_EMP
SYSTEM                         BACKORDER
IVY                            ORDER_HISTORY

6 rows selected.
```

31. TUNING SORT OPERATION

MEASURE THE PERFORMANCE OF THE SORT OPERATION

```
SQL> select disk.value "Disk", mem.value "Memory",
  2  (disk.value/mem.value) "Ratio"
  3  from v$sysstat mem, v$sysstat disk
  4  where mem.name = 'sorts (memory)'
  5  and disk.name = 'sorts (disk)';

      Disk     Memory      Ratio
---------- ---------- ----------
         3       3788 .000791975
```

There are currently 3788 sorts occurring in memory and 3 sorts occurring on disk. The ratio of disk sorts to memory sorts is 0.07%. In general, the disk sort Ratio should not be more than 5 percent. (Sarin, 2000) It should be kept as low as possible. Therefore, the result indicates that there is sufficient memory for users to perform the sort operations.

OPTIMISE PERFORMANCE OF THE SORT OPERATION

31.1 Increase SORT_AREA_SIZE

One way to improve the performance of sort operation is to increase the value of this initialisation parameter SORT_AREA_SIZE.

Example: change the SORT_AREA_SIZE value using ALTER SESSION command.

```
Alter session set sort_area_size=100000000;
```

31.2 Optimising Sort Performance with Temporary Tablespaces

The overhead of any disk sorts can be minimised by performing all disk sorts in the TEMPORARY tablespace.

Action : issue the following query to check the type of tablespace specified for each user in the database.

```
SQL> select username, temporary_tablespace, b.contents
  2  from dba_users a, dba_tablespaces b
  3  where a.temporary_tablespace = b.tablespace_name;

USERNAME                       TEMPORARY_TABLESPACE           CONTENTS
------------------------------ ------------------------------ ---------
SYS                            TEMP                           TEMPORARY
SYSTEM                         TEMP                           TEMPORARY
ZOY                            SYSTEM                         PERMANENT
VINCENT                        MY_DATAFILE                    PERMANENT
KATE                           SYSTEM                         PERMANENT
ROBERT                         SYSTEM                         PERMANENT
IVY                            TEMP                           TEMPORARY
MARY                           TEMP                           TEMPORARY
JOHN                           TEMP                           TEMPORARY
-------
```

Example 1: create a new TEMPORARY tablespace TEMPA

```
SQL> Create tablespace TempA
  2  datafile '/oracle/oradata/project/TempA_01.dbf' size 100M,
  3  '/oracle/oradata/project/TempA_02.dbf' size 100M
  4  Minimum extent 500k
  5  default storage (initial 500k
  6     Next 500k
  7     Maxextents 500
  8     Pctincrease 0 )
  9  TEMPORARY;

Tablespace created.
```

Example 2: change the contents of a tablespace to temporary for user KATE.

```
SQL> ALTER USER KATE
  2  TEMPORARY TABLESPACE TEMPA;

User altered.
```

Verify the change on the tablespace.

```
SQL> select username, temporary_tablespace, b.contents
  2  from dba_users a, dba_tablespaces b
  3  where a.temporary_tablespace = b.tablespace_name
  4  and USERNAME = 'KATE';

USERNAME                       TEMPORARY_TABLESPACE           CONTENTS
------------------------------ ------------------------------ ---------
KATE                           TEMPA                          TEMPORARY
```

Example 3: change the contents of tablespace ‘my_datafile’ to temporary.

This can only be done when there are no permanent objects such as tables or indexes contained in the tablespace.

```
SQL> ALTER TABLESPACE MY_DATAFILE TEMPORARY;

Tablespace altered.
```

Verify the change on the tablespace.

```
SQL> select username, temporary_tablespace, b.contents
  2  from dba_users a, dba_tablespaces b
  3  where a.temporary_tablespace = b.tablespace_name
  4  and USERNAME = 'VINCENT';

USERNAME                       TEMPORARY_TABLESPACE           CONTENTS
------------------------------ ------------------------------ ---------
VINCENT                        MY_DATAFILE                    TEMPORARY
```

32. TUNING ROLLBACK SEGMENTS

MEASURE THE PERFORMANCE OF ROLLBACK SEGMENTS

32.1 Using V$ROLLSTAT Performance View

```
SQL> select sum(waits)* 100 /sum(gets) "Ratio",
  2  sum(waits) "Waits", sum(gets) "Gets"
  3  from v$rollstat;

     Ratio      Waits       Gets
---------- ---------- ----------
         0          0        769
```

This query is used to find out the chances of wait for a user's Server Process to gain a successful access to a rollback segment. The ratio of the sum of WAITS to the sum of GETS should be less than 5%. (Sarin, 2000) The ratio result is 0%, which indicates no wait has occurred for access to rollback segments. No tuning is needed.

32.2 Using V$WAITSTAT Performance View

```
SQL> SELECT a.class "Class", a.count "Count",
  2  SUM(b.value) "Total Requests",
  3  ROUND(((a.count / SUM(b.value)) * 100), 3)
  4  "Percent Waits"
  5  FROM v$waitstat a, v$sysstat b
  6  WHERE a.class IN ('system undo header',
  7  'system undo block', 'undo header',
  8  'undo block'
  9  )
 10  AND b.name IN ('db block gets', 'consistent gets')
 11  GROUP BY a.class, a.count;

Class                   Count Total Requests Percent Waits
------------------ ---------- -------------- -------------
system undo block           0         170729             0
system undo header          0         170729             0
undo block                  0         170729             0
undo header                 0         170729             0
```

This query is used to find out the number of requests from the Database Buffer Cache and the Rollback Buffer Cache that result in any amount of time spent waiting to gain access to a rollback segment. The result for "Percent Waits" is 0%. If it had been greater than 1%, the DBA would have had to create additional rollback segments to reduce the level of contention.

32.3 Monitor the Status of the Rollback Segments

```
SQL> SELECT
  2  SUBSTR(DS.SEGMENT_NAME,1,22) R_SEGMENT,
  3  SUBSTR(DS.TABLESPACE_NAME,1,20) TABLESPACE,
  4  DS.BLOCKS,
  5  DS.EXTENTS,
  6  DRS.STATUS
  7  FROM DBA_SEGMENTS DS,DBA_ROLLBACK_SEGS DRS
  8  WHERE DS.SEGMENT_NAME = DRS.SEGMENT_NAME
  9  ORDER BY 1;

R_SEGMENT              TABLESPACE               BLOCKS    EXTENTS STATUS
---------------------- -------------------- ---------- ---------- ----------
RBS0                   RBS                         512          8 ONLINE
RBS1                   RBS                         512          8 ONLINE
RBS2                   RBS                         512          8 ONLINE
RBS3                   RBS                         512          8 ONLINE
RBS4                   RBS                         512          8 ONLINE
RBS5                   RBS                         512          8 ONLINE
RBS6                   RBS                         512          8 ONLINE
RBSTEST01              RBS                        1280         20 OFFLINE
SYSTEM                 SYSTEM                       50          5 ONLINE

9 rows selected.
```

OPTIMISE PERFORMANCE OF THE ROLLBACK SEGMENTS

32.4 Increase Rollback Segments

If the demand for rollback segments is high, additional rollback segments can be created by using CREATE ROLLBACK SEGMENT command as was mentioned in the previous module.

32.5 Monitor Rollback Area used by Transaction

Any transaction's rollback areas can be measured by monitoring the changes in the system statistics tables during its run.

Example: monitor the rollback area used by the following transaction.

- **STEP 1: Determine current Rollback Area values**

Action: issues the following query.

```
SQL> SELECT SUM(WRITES) FROM V$ROLLSTAT;

SUM(WRITES)
-----------
      23934
```

- **STEP 2: Run the transaction**

Action: issue the following query to insert a new order in the ORDERS table.

```
SQL> insert into ivy.orders values
  2   ('3339',  to_date('30/05/2002',  'dd/mm/yyyy'),  '50003',  '90004',
'13',to_date('22/06/2002', 'dd/
mm/yyyy'), '520');

1 row created.
```

- **STEP 3: Determine the Rollback area values after the transaction is completed.**

```
SQL> SELECT SUM(WRITES) FROM V$ROLLSTAT;

SUM(WRITES)
-----------
      24628
```

- **STEP 4: Calculate the size of the rollback information**

When the transaction has completed, the size of the rollback information generated can be calculated by using

ENDING_WRITES - BEGINNING_WRITES - 54 = ROLLBACK INFO GENERATED.

(source: www.oracle.com)

24628 – 23934 – 54 = 640

This tells how much rollback space is needed to handle this transaction. By knowing how many transactions will be running at once will give an idea of how much space will be needed in the ROLLBACKS tablespace.

32.6 Isolating Large Transactions

When dealing with large transactions, it is better to dedicate them to larger rollback segments. This is achieved through the following steps:

- **Put the rollback segment on line**

```
SQL> connect /as sysdba
Connected.
SQL> ALTER ROLLBACK SEGMENT rbstest01 ONLINE;

Rollback segment altered.
```

The rollback segment ‘rbstest01’ was created in the previous module.

- **Assign the transaction to the rollback segment**

This is done by using SET TRANSACTION command.

```
SQL> SET TRANSACTION USE ROLLBACK SEGMENT rbstest01;

Transaction set.
```

33. CONCLUSION

Throughout the project, SQL and PL/SQL knowledge was the fundamental element required to perform most major DBA tasks. Various SQL statements and PL/SQL statements have been put into practice to manage and optimise the performance of the database. In the Architecture and Administration module, various simulations were conducted to manage the physical database structures and memory structures. Some of which included database creation, data security and managing parameter files. In the Performance Tuning module, various simulations were also conducted to monitor and measure the performance of the database structures and memory structures. Different methods were explored to overcome the different tuning problems that occurred.

However, the amount of data stored in the database in practice was insufficient. The database in practice was also very new for any potential problems to arise. Thus, a few of the tuning measurements and turning method experiments were unable to show the effect of what it is capable of doing.

In conclusion, the tasks performed by a DBA can be very complex and difficult to manage. Care should be practiced at all times. Combined with experience and the right tools ensure optimal performance of the database.

34. REFERENCES

Ault, M. R., 'Oracle 8i DBA: SQL and PL/SQL', 2001, Coriolis, Arizona.

Morrison, J., Morrison, M. 'A Guide to Oracle8', 2000, Course Technology, Canada

Niemiec, R. J., 'ORACLE Performance Tuning Tips & Techniques', 1999, McGraw-Hill, USA

Oracle8i Administrator's Guide Release 2 (8.1.6) December 1999 Part No. A76956-01
http://otn.oracle.com/docs/products/oracle8i/doc_library/817_doc

Oracle8*i* Documentation Addendum Release 3 (8.1.7) September 2000 Part No. A85455-01
http://otn.oracle.com/docs/products/oracle8i/doc_library/817_doc/addendum.817/index.htm

Oracle8i Designing and Tunning for Performance Release 2 (8.1.6), December 1999
http://otn.oracle.com/docs/products/oracle8i/doc_library/817_doc/server.817/a76992.pdf

Sarin, S. 'ORACLE DBA Tips & Techniques', 2000, McGraw-Hill, USA

Oracle 8i (8.1.7) Enterprise Edition installation guide
Oracle 9i Application Server release 1 (version 1.0.2.2)

Shared_Pool Script:
http://www.uaex.edu/srea/shared_pool_size.sql

Appendix

A. Table Creation
B. Data Insertion
C. Additional Tablespace
D. The trace output file ‘trace.txt’
E. TKPROF output for using different hints

A. TABLE CREATION

- **CUSTOMERS TABLE**

```
CREATE TABLE customers
(cid NUMBER(5) constraint customers_cid_pk primary key,
cname VARCHAR2(10),
cadd VARCHAR2(20),
cpcode number(4),
cphone NUMBER(8));
```

- **ORDERS TABLE**

```
Create table orders
(orderid number(4) constraint orders_orderid_pk primary key,
ord_date date,
cid number(5) constraint orders_cid_fk references customers(cid),
itemid number(5) constraint orders_itemid_fk references items(itemid),
ord_qty number(5),
delver_date date,
eid number(3) constraint orders_eid_fk references employees(eid));
```

- **EMPLOYEES TABLE**

```
Create table employees
(eid number(3) constraint employees_eid_pk primary key,
efname varchar2(10),
elname varchar2(15),
edob date,
eadd varchar2(25),
ephone number(8),
epcode number(4),
Did number(3));
```

- **ITEMS TABLE**

```
Create table items
(itemid number(5) constraint items_itemid_pk primary key,
itemdesc varchar(30),
price number(5),
qoh number(4));
```

- **DEPARTMENTS TABLE**

```
Create table departments
(did number(3) constraint departments_did_pk primary key,
depart_name varchar2(10));
```

- **PERFORMANCE TABLE**

```
Create table Performance
(Rating char(1) constraint Performance_rating_cc
check ((Rating = 'A') or (Rating = 'B') or (Rating = 'C') or
(Rating = 'D') or (Rating = 'E') or (Rating = 'F')),
lowsale number,
highsale number);
```

- **ORDERPRICE TABLE**

```
Create table orderprice
(orderid number(4),
itemid number(5) constraint price_itemid_fk references items(itemid),
ord_qty number(5),
Price number);
```

B. DATA INSERTION

- **CUSTOMERS TABLE**

```
insert into customers values
(50001, 'ColesMyer ', '123 Clayton road', '3600', '96782415');

insert into customers values
(50002, 'Samsung', '41 Glenhuntly road', '3112', '95712800');

insert into customers values
(50003, 'Hilton', '38 Mcmaster court', '3897', '97580322');

insert into customers values
(50004, 'Evian', '1 Flinder street', '3112', '96508016');

insert into customers values
(50005, 'Monash', '3500 Lancell road', '5038', '98133049');
```

- **ORDERS TABLE**

```
insert into orders values
('3331',   to_date('13/04/2002',   'dd/mm/yyyy'),   '50005',   '90003',
'10',to_date('20/04/2002','dd/mm/yyyy'), '521');

insert into orders values
('3332', to_date('14/04/2002', 'dd/mm/yyyy'), '50001', '90006', '47',
to_date('20/04/2002','dd/mm/yyyy'), '520');

insert into orders values
('3333',   to_date('15/04/2002',   'dd/mm/yyyy'),   '50003',   '90001',
'5',to_date('22/04/2002', 'dd/mm/yyyy'), '520');

insert into orders values
('3334',   to_date('16/04/2002',   'dd/mm/yyyy'),   '50002',   '90003',
'15',to_date('22/04/2002', 'dd/mm/yyyy'), '524');

insert into orders values
('3335',   to_date('16/04/2002',   'dd/mm/yyyy'),   '50004',   '90002',
'10',to_date('22/06/2002', 'dd/mm/yyyy'), '524');
```

- **EMPLOYEES TABLE**

```
insert into employees values
('520', 'Jim', 'Peterson', to_date('02/11/1978', 'dd/mm/yyyy'), '13/78 King
Street', '94037878', '1352', '101');

insert into employees values
('521', 'Clair', 'Manson' , to_date('30/03/1974', 'dd/mm/yyyy'), '2/8 Park
Street', '97750322', '2012', '101');

insert into employees values
('522', 'Todd', 'Smith', to_date('17/04/1976', 'dd/mm/yyyy'), '11 Kew
Street', '94219660', '1352', '103');

insert into employees values
('523', 'Rebecca', 'Gwen', to_date('20/01/1975', 'dd/mm/yyyy'), '16/2
Melrose place', '93315716', '1032', '102');

insert into employees values
('524', 'Mareen', 'Joans', to_date('13/04/1979', 'dd/mm/yyyy'), '133/6
Gleniris road', '97305643', '1033', '101');
```

```
insert into employees values
('525', 'Miller', 'Chang', to_date('03/07/1971', 'dd/mm/yyyy'), '23/40 Water
street', '97784106', '2012', '');
```

- **ITEMS TABLE**

```
insert into items values
('90001', 'HITACHI monitor 17inch', '900', '50');

insert into items values
('90002', 'HITACHI monitor 19inch', '1500', '12');

insert into items values
('90003', 'sony 56k modem', '90', '68');

insert into items values
('90004', 'Microsoft keyboard', '40', '87');

insert into items values
('90005', 'sony 52x CDROM drive', '120', '94');

insert into items values
('90006', 'TDK flopy disc x 12', '8', '112');
```

- **DEPARTMENTS TABLE**

```
insert into departments values
(101, 'Sales');

insert into departments values
(102, 'Accounting');

insert into departments values
(103, 'Marketing');
```

- **PERFORMANCE TABLE**

```
insert into performance values
('A', '12001', '20000');

insert into performance values
('B', '9001', '12000');

insert into performance values
('C', '6001', '9000');

insert into performance values
('D', '3001', '6000');

insert into performance values
('E', '1001', '3000');

insert into performance values
('F', '0', '1000');
```

C. ADDITIONAL TABLESPACE

```
CREATE TABLESPACE ord_data
DATAFILE '/oracle/oradata/project/ord_data_01.dbf' SIZE 100M
EXTENT MANAGEMENT LOCAL AUTOALLOCATE;
```

```
CREATE TABLESPACE ord_data2
DATAFILE '/oracle/oradata/project/ord_data_02.dbf' SIZE 100M
EXTENT MANAGEMENT LOCAL AUTOALLOCATE;
```

D. THE TRACE OUTPUT FILE 'TRACE.TXT'

```
TKPROF: Release 8.1.7.0.0 - Production on Fri Jun 1 02:12:41 2002

(c) Copyright 2000 Oracle Corporation.  All rights reserved.

Trace file: c:\oracle\admin\project\udump\ora01056.trc
Sort options: default

********************************************************************************
count    = number of times OCI procedure was executed
cpu      = cpu time in seconds executing
elapsed  = elapsed time in seconds executing
disk     = number of physical reads of buffers from disk
query    = number of buffers gotten for consistent read
current  = number of buffers gotten in current mode (usually for update)
rows     = number of rows processed by the fetch or execute call
********************************************************************************

ALTER SESSION SET sql_trace= TRUE


call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        0      0.00       0.00          0          0          0           0
Execute      1      0.05       0.07          0          0          0           0
Fetch        0      0.00       0.00          0          0          0           0
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        1      0.05       0.07          0          0          0           0

Misses in library cache during parse: 0
Misses in library cache during execute: 1
Optimizer goal: CHOOSE
Parsing user id: 48  (IVY)
********************************************************************************

select eid, efname, elname, eadd ephone, did
from employees
where did = 101
order by eid

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.34       0.49          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch        2      0.00       0.06          2          4          0           3
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        4      0.34       0.55          2          4          0           3

Misses in library cache during parse: 1
Optimizer goal: CHOOSE
Parsing user id: 48  (IVY)

Rows     Row Source Operation
-------  ---------------------------------------------------
      3  TABLE ACCESS BY INDEX ROWID EMPLOYEES
      9   INDEX FULL SCAN (object id 29359)


Rows     Execution Plan
-------  ---------------------------------------------------
      0  SELECT STATEMENT   GOAL: CHOOSE
      3   TABLE ACCESS   GOAL: ANALYZED (BY INDEX ROWID) OF 'EMPLOYEES'
      9    INDEX   GOAL: ANALYZED (FULL SCAN) OF 'EMPLOYEES_EID_PK'
               (UNIQUE)

********************************************************************************

select eid, efname, elname,eadd, epcode, did
from employees
where did in (101)

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.06       0.08          0          0          0           0
```

```
Execute      1      0.00       0.00          0          0          0           0
Fetch        2      0.00       0.00          1          2          4           3
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        4      0.06       0.08          1          2          4           3

Misses in library cache during parse: 1
Optimizer goal: CHOOSE
Parsing user id: 48  (IVY)

Rows     Row Source Operation
-------  ---------------------------------------------------
      3  TABLE ACCESS FULL EMPLOYEES


Rows     Execution Plan
-------  ---------------------------------------------------
      0  SELECT STATEMENT   GOAL: CHOOSE
      3   TABLE ACCESS   GOAL: ANALYZED (FULL) OF 'EMPLOYEES'

********************************************************************************

select orderid, ord_date, cid
from orders
where eid = 520

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.30       0.49          1          0          1           0
Execute      1      0.00       0.00          0          0          0           0
Fetch        2      0.00       0.00          1          2          4           3
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        4      0.30       0.49          2          2          5           3

Misses in library cache during parse: 1
Optimizer goal: CHOOSE
Parsing user id: 48  (IVY)

Rows     Row Source Operation
-------  ---------------------------------------------------
      3  TABLE ACCESS FULL ORDERS


Rows     Execution Plan
-------  ---------------------------------------------------
      0  SELECT STATEMENT   GOAL: CHOOSE
      3   TABLE ACCESS (FULL) OF 'ORDERS'

********************************************************************************

ALTER SESSION SET SQL_TRACE = FALSE


call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.01       0.02          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch        0      0.00       0.00          0          0          0           0
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        2      0.01       0.02          0          0          0           0

Misses in library cache during parse: 1
Optimizer goal: CHOOSE
Parsing user id: 48  (IVY)



********************************************************************************

OVERALL TOTALS FOR ALL NON-RECURSIVE STATEMENTS

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        4      0.71       1.08          1          0          1           0
Execute      5      0.05       0.07          0          0          0           0
Fetch        6      0.00       0.06          4          8          8           9
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total       15      0.76       1.21          5          8          9           9
```

```
Misses in library cache during parse: 4
Misses in library cache during execute: 1


OVERALL TOTALS FOR ALL RECURSIVE STATEMENTS

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse       24      0.40       0.60          0          0          0           0
Execute     31      0.01       0.04          0          0          0           0
Fetch       61      0.00       0.01          1        103          0          46
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total      116      0.41       0.65          1        103          0          46

Misses in library cache during parse: 10

    5  user  SQL statements in session.
   24  internal SQL statements in session.
   29  SQL statements in session.
    3  statements EXPLAINed in this session.
********************************************************************************
Trace file: c:\oracle\admin\project\udump\ora01056.trc
Trace file compatibility: 8.00.04
Sort options: default

       2  sessions in tracefile.
       5  user  SQL statements in trace file.
      24  internal SQL statements in trace file.
      29  SQL statements in trace file.
      16  unique SQL statements in trace file.
       3  SQL statements EXPLAINed using schema:
            IVY.prof$plan_table
              Default table was used.
              Table was created.
              Table was dropped.
     310  lines in trace file.
```

E. TKPROF OUTPUT FOR USING DIFFERENT HINTS

```
select *
from
 orders


call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.25       0.40          1          0          1           0
Execute      1      0.00       0.00          0          0          0           0
Fetch        2      0.00       0.02          1          2          4           8
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        4      0.25       0.42          2          2          5           8

Misses in library cache during parse: 1
Optimizer goal: CHOOSE
Parsing user id: 48  (IVY)

Rows     Row Source Operation
-------  ---------------------------------------------------
      8  TABLE ACCESS FULL ORDERS


Rows     Execution Plan
-------  ---------------------------------------------------
      0  SELECT STATEMENT   GOAL: CHOOSE
      8   TABLE ACCESS (FULL) OF 'ORDERS'
```

- **THE FULL HINT**

```
SELECT /*+ FULL(ORDERS) */*
from orders

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.11       0.20          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch        2      0.00       0.00          0          2          4           8
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        4      0.11       0.20          0          2          4           8

Misses in library cache during parse: 1
Optimizer goal: CHOOSE
Parsing user id: 48  (IVY)

Rows     Row Source Operation
-------  ---------------------------------------------------
      8  TABLE ACCESS FULL ORDERS


Rows     Execution Plan
-------  ---------------------------------------------------
      0  SELECT STATEMENT   GOAL: CHOOSE
      8   TABLE ACCESS (FULL) OF 'ORDERS'
```

- **THE INDEX HINT**

```
SELECT /*+ INDEX (ORDERS ORDERS_ORDERID_PK) */*
from orders

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.07       0.09          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch        2      0.00       0.02          1          4          0           8
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        4      0.07       0.11          1          4          0           8

Misses in library cache during parse: 1
Optimizer goal: CHOOSE
Parsing user id: 48  (IVY)

Rows     Row Source Operation
-------  ---------------------------------------------------
      8  TABLE ACCESS BY INDEX ROWID ORDERS
      9   INDEX FULL SCAN (object id 29398)


Rows     Execution Plan
-------  ---------------------------------------------------
      0  SELECT STATEMENT   GOAL: CHOOSE
      8   TABLE ACCESS (BY INDEX ROWID) OF 'ORDERS'
      9    INDEX (FULL SCAN) OF 'ORDERS_ORDERID_PK' (UNIQUE)
```

- **THE RULE HINT**

```
SELECT /*+ RULE */*
FROM ORDERS

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.02       0.02          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch        2      0.00       0.00          0          2          4           8
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        4      0.02       0.02          0          2          4           8

Misses in library cache during parse: 1
Optimizer goal: RULE
Parsing user id: 48  (IVY)

Rows     Row Source Operation
-------  ---------------------------------------------------
      8  TABLE ACCESS FULL ORDERS


Rows     Execution Plan
-------  ---------------------------------------------------
      0  SELECT STATEMENT   GOAL: HINT: RULE
      8   TABLE ACCESS (FULL) OF 'ORDERS'
```