Introduction to Oracle SQL

By : Ammar Sajdi
Palestine Engineering Co
POBOX 17187
Amman – Jordan
E-mail: : Ammar.Sajdi@realsoft-me.com
Phone :+962-6-5826602/01

Logging on to SQL * Plus .

1. SQLPLUS .
Enter your user name and [Return]
SQL * Plus will display the prompt " Enter Password :".
Enter your password and press [Return] again.

For your protection, your password will not appear on the screen.

SQL* Plus will display its prompt :

SQL >

2. SQLPLUS username

You will be prompted for your password.

3. SQLPLUS username / password.

You will be logged on the SQL* Plus. In this case the password will be displayed.

For Windows Users; Find Oracle group icon and double click the SQL*Plus icon.

Fill in your username and press [TAB]
Fill in your password and press [TAB]
If you are connecting to a local database press [RETURN], If you are connecting to a SERVER supply the Host String name. Ask you database Administrator for you Host String.

Oracle is shipped with a built-in user for the tutorial. The usename for that purpose is SCOTT and the Password is TIGER

Writing SQL Commands

- SQL commands are not case sensitive (unless indicated otherwise).
- An SQL command entered at the SQL prompt, and subsequent lines are numbered. This is called the SQL buffer.
- Only one statement can be current at any time within the buffer, and can be run in a number of ways :

    - Place a semi - colon (;) at the end of last clause.
    - Place a ; or / on last line in the buffer.
    - Place a / at SQL prompt.
    - Issue R [UN] command at SQL prompt.

Any one of the following statements is valid :

SQL>*SELECT * FROM EMP;*

SQL>SELECT
*
FROM
EMP
;

SQL>SELECT *
FROM EMP;

SQL* Plus Commands

SQL commands (like SELECT) are the vehicle to the data via the Oracle kernel. SQL * Plus commands are used fundamentally for controlling the environment, formatting query results and controlling files. The commands identified here are miscellaneous and you may wish to use them in the following exercises.

SQL * Plus commands are entered at the SQL > prompt on one line, they do not open up a buffer.

Command	Description
SAVE filename	allows current contents of SQL buffer to be saved to a file.
GET filename	call up contents of previously saved file into the buffer.
START filename	runs a previously saved command file, Command files are discussed in Section 6, SQL* Plus Reports. “
ED filename	uses default editor in order to edit contents of a saved file.
SPOOL filename	writes all subsequent commands and / or put to the named file. The spool file is assigned the .LST suffix
SPOOL OFF | OUT	OFF closes spool file and OUT turns off spooling and sends spool file to printer.

DESC[RIBE]	displays any database table structure.
HELP	invokes Oracle internal help facility.
HOST command	invokes Operating System shell.
CONN[ECT]userid / password	invokes another Oracle logon from within current logon id.
EXIT	leaves SQL*Plus.
PROMPT text	displays text when running a command file.

Using Files

Spool Files

The screen display produced by SQL*Plus can be captured permanently with the SPOOL command. This will copy everything that appears on the screen to an operating system file.

Spooling will continue until one of the other SPOOL commands is used.

SPOOL OFF will close the spool file.

SPOOL OUT will automatically direct the closed spool file to the system printer (when connected).

When you finish an SQL*Plus session and you have not closed the spool files all open spool files are automatically closed.

Data Definition Language (DDL)

ORACLE Data Structures

- Tables can be created at any time, even with the users still on - line using the database.

- Data length is variable, in that only the characters / numbers specified are stored. Leading spaces and trailing blanks are not stored.

- There is no need to specify the size of any table. This is ultimately defined by how much space has been allocated to the database as a whole.

- Table structures can be modified on - line.

Creating a Table

The name you choose for a table must follow the standard rules for naming an ORACLE database object.

1. The name must begin with a letter, A-Z or a-z

2. It may contain letters, numerals, and the special character - (underscore). The characters $ and # are also legal, but their use is discouraged.

3. The name is the same whether upper or lower case letters are used e,g, EMP, emp, and eMP are all the same table.

4. It may be up to 30 characters in length.

5. The name must not duplicate the name of another table or view in your account.

NAME	VALID
EMP96	YES
96EMP	No; Does not begin with a character
AMMAR_SAJDI	Yes
AMMAR SAJDI	No; Contains a blank space
INSERT	No; Reserved Word

Column Types

When you create a table, you need to specify the column’s data type. The table

below shows the most important data types.
The data type may be followed by one or more numbers in parentheses which give information about the column's width. The column's width determines the maximum width that values in the column may have . CHAR columns must have a width specification; NUMBER columns may have a width specification, but need not.

CHAR(n)	Fixed length character values with max. length equal to 255. All n bytes are used, when necessary filled with blanks
VARCHAR2(n)	Variable length character values with max. length of 2000
VARCHAR(n)	Currently identical to VARCHAR2. Oracle recommends that you use VARCHAR2 as VARCHAR can be used in the future for other data types
NUMBER(v,n)	Numeric data type for integer and real values v= Places before the decimal point with a max of 38 , n = places after the decimal.
DATE	Stores both date and time
LONG	Data type for character data of variable length with max length equal to 2GB
LONG RAW	Data type of binary data of up to 2GB
RAW(n)	Data Type for storing binary data

Creating a Table

To create a table in an ORACLE database use the CREATE TABLE command.

```
CREATE TABLE table_ name
( column name     type   ( size )    [ NULL / NOT NULL],
  column name     type   ( size )    [NULL / NOT NULL],
  .... ) ;
```

```
SQL>CREATE TABLE DEPT
( DEPTNO        NUMBER (2)   NOT NULL,
  DNAME         CHAR  (12),
  LOC           CHAR ( 12 ) ) ;
```

CREATE Table EMP

To create the EMP table, enter :

```
CREATE TABLE EMP
( EMPNO         NUMBER (4)   NOT NULL,
ENAME           CHAR (10) ,
JOB             CHAR (10) ,
```

```
MGR          NUMBER (4) ,
HIREDATE     DATE,
SAL          NUMBER (7 , 2 ) ,
COMM         NUMBER (7 , 2 ) ,
DEPTNO       NUMBER (2)  NOT NULL ) ;
```

The following message is displayed :

Table created

To see a description of the EMP table, enter :

DESCRIBE EMP ;

Name	*Null ?*	*Type*
EMPNO	*NOT NULL*	*NUMBER (4)*
ENAME		*CHAR (10)*
JOB		*CHAR (10)*
MGR		*NUMBER (4)*
HIREDATE		*DATE*
SAL		*NUMBER (7, 2)*
COMM		*NUMBER (7, 2)*
DEPTNO	*NOT NULL*	*NUMBER (2)*

Note the following points :

- EMPNO is defined NOT NULL, this column uniquely identifies the row. -DEPTNO is also NOT NULL, the values in the column are used to JOIN the EMP table to DEPT table.

- SAL and COMM have data type NUMBER (7,2). Total of 7 digits, two of which are after decimal point ie. 12666.54, 37997,99.

- The DESCRIBE command can be used to display the structure of any table in the DATA DICTIONARY (for more on the DATA DICTIONARY refer to latter part of this section). The command can be abbreviated to DESC.

Creating a Table with Rows from Another Table

There is a second form of CREATE TABLE statement in which the table is created with rows in place, derived from another table :

```
CREATE TABLE DEPT
[( column-name    { NULL/NOT NULL } , .... )]
AS SELECT statement
```

- The table will be created with specified columns and the rows retrieved by the SELECT statement inserted into it.

- If all the columns in the SELECT statement have well- defined names, (i.e. no expressions, etc.) , the column specifications may be omitted.

- If column specifications are given, the number of columns must equal the number of items in the SELECT list.

Example using *CREATE ... AS SELECT ...*
To create a table DEPT30 holding the employee

```
SQL> CREATE TABLE DEPT30
 AS
 SELECT   EMPNO, ENAME,SAL
 FROM     EMP
 WHERE    DEPTNO = 30;
```

Table created.

To see the description of DEPT30, enter:

```
SQL> DESC DEPT30
```

Name	Null?	Type
EMPNO	NOT NULL	NUMBER(4)
ENAME		VARCHAR2(10)
SAL		NUMBER(7,2)

Altering a Table

Use the ALTER TABLE command to change the definition of a table.
Use the ADD keyword to add a column to an existing table.

```
ALTER  TABLE name
ADD     ( column type )
```

To add a column to the EMP table that will hold the NATIONALITY of an employee, enter .

```
SQL> ALTER TABLE EMP
  ADD (NATIONALITY  VARCHAR2(20));
```

Table altered.

```
SQL> DESC EMP
Name                          Null?    Type
------------------------------ -------- ----
EMPNO                         NOT NULL NUMBER(4)
ENAME                         VARCHAR2(10)
JOB                           VARCHAR2(9)
MGR                           NUMBER(4)
```

```
HIREDATE                  DATE
SAL                       NUMBER(7,2)
COMM                      NUMBER(7,2)
DEPTNO                    NOT NULL NUMBER(2)
NATIONALITY               VARCHAR2(20)
```

Use MODIFY keyword to modify the definition of an existing column

```
SQL> ALTER TABLE EMP
  MODIFY (ENAME VARCHAR2(25));

Table altered.

SQL> DESC EMP
Name                           Null?          Type
------------------------------ -------- ----
EMPNO                          NOT NULL       NUMBER(4)
ENAME                                         VARCHAR2(25)
JOB                                           VARCHAR2(9)
MGR                                           NUMBER(4)
HIREDATE                                      DATE
SAL                                           NUMBER(7,2)
COMM                                          NUMBER(7,2)
DEPTNO                         NOT NULL       NUMBER(2)
NATIONALITY                                   VARCHAR2(20)
```

There are three changes that you cannot make :

1. You may not change a column containing nulls from NULL to NOT NULL

2. You may not add a new column that is NOT NULL. Make it null, fill it completely and then change it to NOT NULL.

3. You may not decrease the size of a column or change its data type, unless it contains no data.

The DESCRIBE Command

IF the structure of a table or view is unknown to the user, the DESCRIBE command can be used as was demonstrated several times earlier. The Describe command is SQL*PLUS command and is not part of the standard SQL LANGUAGE. It is a command that only exists in SQL*Plus environment

The RENAME Command

The RENAME command is a handy command that can be used by the creator of a TABLE (and other objects to be discussed later like VIEWS and SYNONYMS) to change the name of the database object

To RENAME a database object, the syntax is

RENAME old TO new

to rename the DEPT table to DEPARTMENT, enter

SQL>RENAME DEPT TO DEPARTMENT;

Dropping a Table

To remove the definition of an ORACLE table use the DROP TABLE command

SQL>DROP TABLE EMP;

(Note: Advanced usage of the Drop table is : DROP TABLE XXX CASCADE CONSTRAINTS. More information can be found in Topics concerning Constraints).

-Dropping a table loses all the data in it and all the indexes.

-Any VIEWS and SYNONYMS will remain by they become invalid.

-All pending transactions will be committed.

Note: In a later section we will study the subject of Constraints and Referential Integrity. Constraint are implemented using the CREATE TABLE and ALTER TABLE SQL Commands.

Data Manipulation Language (DML)

INSERT New Rows Into a Table

The INSERT command is used to add rows to a table.
The syntax of the INSERT command is :

```
INSERT  INTO  tablename [( column, column, .... )]
                    VALUES  ( value, value, ..... ) ;
```

It is possible to insert a new row with values in each column, in which case the column list is not required. It is recommended that the COLUMN LIST is ALWAYS specified. If the list is not specified the software will require modification whenever the table definition is altered .

To insert a new department, enter :

```
SQL>INSERT  INTO DEPT   ( DEPTNO, DNAME, LOC )
VALUES   ( 50, ' MARKETING' , ' AMMAN' ) ;
```

Note that this command adds only one row at a time to a table .

To enter a new department omitting department name, the column list must be specified :

```
SQL>INSERT   INTO  DEPT  ( DEPTNO, LOC)
   VALUES  ( 50,  `AMMAN `) ;
```

Alternatively if the department name is not known, a NULL could be specified :

```
INSERT INTO DEPT  ( DEPTNO, DNAME, LOC )
VALUES  ( 50,  NULL, `AMMAN `) ;
```

CHARACTER and DATE values must be enclosed in single quotes .

Using Substitution Variables to INSERT Rows.

As previously mentioned, INSERT is a one-row-at-a time command. Using Substitution Variables it is possible to speed up input :

```
SQL>INSERT  INTO  DEPT  ( DEPTNO, DNAME, LOC )
VALUES  ( &D_NUMBER, `&D_NAME`, `& LOCATION ) ;
```

When the command is run, values are prompted for every time.

INSERTING DATE and TIME Information

When inserting a DATE value, the format DD-MON-YY is usually used. With this format the century defaults to the 20th Century (19nn). The date also contains time information, which if not specified defaults to midnight

The default DD-MON-YY depends on the territory substring of the specified NLS_LANG. One could also change the default date format for a session by using the
ALTER SESSION SET NLS_DATE_FORMAT. i.e

SQL>ALTER SESSION SET NLS_DATE_FORMAT = 'DD/MM/YY';

Coping Rows from Another Table

```
INSERT  INTO table [( column,  column, .... )]
          SELECT  select- list
           FROM     table ;
```

This form of the insert statement allows you to insert several rows into a table where the values are derived from the contents of existing tables in the database .

To copy all information on department 10 into the HISTORY table, enter:

```
SQL>INSERT INTO HISTORY
      ( EMPNO, ENAME, SAL, JOB, HIREDATE)
            SELECT EMPNO,ENAME,SAL,JOB,HIREDATE
            FROM EMP
            WHERE DEPTNO = 10 ;
```

Note the keyword ` VALUES` is not used here .

Updating Rows

The UPDATE statement allows you to change values in rows in a table ;

```
UPDATE  table  [alias]
SET   column  [, column ...] = {expression, subquery }
{ WHERE  condition };
```

For example;

To UPDATE Scott's row enter:

```
SQL> UPDATE EMP
            SET JOB='SALESMAN',
            SAL      = SAL*1.1
            WHERE EMPNO = 4799;
```

If the WHERE clause is omitted, all rows in the table will be updated.

It is possible to use nested subqueries and correlated subqueries in the UPDATE statement .

Suppose you had a table of new commission figures for certain employees.
For example, the COMMISSION table below is to be used to update certain rows of the EMP table.

COMMISSION

EMPNO	COMM
7499	1100
7654	500
7844	3500
7844	2000
7844	1500

EMP

EMPNO	COMM
7499	300
7654	1400
7844	0

The changes listed in the COMMISSION table can be applied to the EMP table using a correlated subquery and a nested subquery as show below

Example 1:

```
UPDATE  EMP
SET COMM= (SELECT COMM FROM COMMISSION C
                    WHERE C.EMPNO = EMP.EMPNO)
WHERE EMPNO IN (SELECT EMPNO FROM COMMISSION);
```

The COMMISSION table might contain more than one entry for a given employee as illustrated below:

EMPNO	COMM
7499	1100
7654	500
7844	3500
7844	2000
7844	1500

You want to REPLACE the commission values on the EMP table with the TOTAL commission for each employee listed in the COMMISSION table.

To accomplish this, use the following SQL:
Example 2:

```
UPDATE EMP
SET COMM = (SELECT SUM(COMM) FROM COMMISSION C
                    WHERE C.EMPNO = EMP.EMPNO)
WHERE EMPNO IN (SELECT EMPNO FROM COMMISSION);
```

The EMP table now reflects the changed commissions:

EMP

EMPNO	COMM
7499	1100
7654	650
7844	7000

If you want to exchange (swap) two fields

SQL> update emp set sal=comm, comm=sal

This is very different from
SQL> update emp set sal=comm;
SQL> update emp set comm=sal;

Deleting Rows from a Table

The DELETE command allows you to remove one or more rows from a table.

```
SQL>DELETE  FROM  EMP
WHERE  DEPTNO  = 10 ;
```

If the WHERE clause is omitted, all rows will be deleted.

Transaction Processing

A transaction is an operation against the database which comprises a series of changes to one or more tables.

There are two classes of transactions. DML transactions which can consist of any number of DML statements and which ORACLE treats as a single entity or logical unit of work, and DDL transactions which can only consist of one DDL statement.

There can be no ` half-way` situation during the execution of the transaction, where some changes specified within the transaction are made to the database and others are not made. For every transaction either all the changes made to the database are made permanent, or none of the changes are carried out (they are all discarded).

A DML transaction begins when the first executable DML command is encountered and ends when one of the following occurs:

- COMMIT / ROLLBACK
- DDL command is issued
- Certain errors (such as deadlock)

- Logoff

- Machine failure

A DDL statement is automatically committed and therefore implicitly ends a transaction.

In order for changes to become permanent, they must be committed to the database. The COMMIT command makes database changes permanent; ROLLBACK allows us to discard or abandon changes. The change, or changes, made to the database between two commit commands therefore make up a transaction.

Until changes are committed, they may be abandoned by typing ROLLBACK. ROLLBACK will return the data to the state it was in immediately after your last COMMIT by discarding all changes made since the last commit.

When a transaction is interrupted by a serious error, for example a system failure, the entire transaction is automatically rolled back. This prevents the error from causing unwanted changes to your data, and returns your tables to their status at the time of the last commit. In this way SQL* Plus protects the integrity of your tables.

An automatic rollback is often caused by a system failure, such as an unintentional system reset or power blackout. Error in entering commands, such as misspelling a column name or trying to perform an unauthorized operation on another user's table, do not interrupt a transaction or cause an automatic rollback. This is because the error are detected at parse time (when the SQL statement is scanned and checked), not during execution time.

A new transaction is started following a commit or rollback - that is when the first executable DML or DDL statement is encountered.

AUTOCOMMIT

COMMIT / ROLLBACK may be issued manually or automatically by using the AUTOCOMMIT option of the SET command . The AUTOCOMMIT option controls when database changes are made permanent .

There are two settings :

Command	Description
SET AUTO[COMMIT]ON	COMMIT issued automatically when each INSERT, UPDATE or DELETE command is issued .
SET AUTO[COMMIT] OFF	COMMIT command can be issued explicitly by the user. Also COMMIT is

issued if <control> Z is pressed, when a DROP, ALTER or CREATE command is issued, or if you exit from SQL*Plus ROLLBACK can be issued explicitly by the user, to reinstate the database.

Read Consistency

Database users make two types of access to the database :

- read operations (select statement)
- write operations (insert, update and delete statements)

The database reader and writer must be guaranteed a consistent view of the data. Readers must not view data which is in the process of being changed. Writers also need to be sure that changes to the database are done in a consistent way : That the changes made by one writer do not disrupt or conflict with changes another writer is making .

The purpose of read consistency is to ensure that each user sees data as it existed at the last commit, before a DML operation has started .

Read consistency is implemented by keeping a partial copy of the database in the Rollback Segment . (Rollback segment is a temporary storage area for pending transactions, this segment is controlled by Oracle and not by the user).

When an insert, update, or delete operation is made to the database, ORACLE will take a copy of the data before it is changed and write it to the Rollback Segment.

All readers, except those who issued the changes, still see the database as it existed before the changes started - they view the Rollback Segment `snapshot` of the data.

However, before changes are committed to the database, only the user who is modifying the data sees the database with the alterations incorporated.
Everyone else sees the rollback segment. This guarantees that readers of the data read consistent data that is not currently undergoing change.

When a DML command is committed, the change made to the database becomes visible to anyone executing a SELECT statement . The modifications are made `universal ` and all users now see the database with the changes incorporated.

The space occupied by the `old` data in the rollback segment is freed for re-use .

If the transaction is rolled back, the changes are `undone`:

- the original data in the Rollback are copied back to the database.
- all users see the database as it existed before the transaction began .

Concurrency and Locking

What is a lock ?

ORACLE allows any number of readers to access the same data at the same time, because reading does not change the data. However, ORACLE permits only one writer to a record of a table at a time.

A lock is the mechanism used to control concurrent access to data on a multi-user system . It prevents simultaneous update by two or more users and ensures that a row in a table or column definition cannot be altered while updates to the data in the progress.

Oracle therefore, has a default ROW LEVEL LOCKING mechanism that is automatically handled by ORACLE RDBMS.

When a user ,for example, updates a record, ORACLE will automatically lock this record and will prevent other users from accessing this for all kind of writing operations (Update, Delete). When the operation is committed or rolled back, the lock is automatically released.

The Basic QUERY Block

The SELECT statement retrieves information from the database, implementing all the operators of Relational Algebra.

In its simplest from, it must include :

1. a SELECT clause, which lists the columns to be displayed (i.e. it is essentially a PROJECTION).

2. a FROM clause, which specifies the table involved.

To list all department numbers, employee names and manager numbers in the EMP table you enter the following :

```
SELECT  DEPTNO, ENAME, MGR
FROM  EMP;
```

Note that column names are separated by a comma

DEPTNO	ENAME	MGR
20	SMITH	7902
30	ALLEN	7698
30	WARD	7698
20	JONE	7839
30	MARTIN	7698
30	BLAKE	7839
10	CLARK	7839
20	SCOTT	7566
10	KING	
30	TURNER	7698
20	ADAM	788
30	JAMES	698
20	FORD	7566
10	MILLER	7782

It is possible to select all columns from the table , by specifying an * after the SELECT command word.

```
SQL> SELECT * FROM EMP;
```

Other Items on the SELECT Line

It is possible to include other items in the SELECT line.

- Arithmetic Expressions

- Column Aliases

- Concatenated Columns
- Literal
- Functions

All of these options allow the user to query data and manipulate it for query purposes; for example, performing calculations, joining columns together, or displaying literal text strings.

Arithmetic Expressions

Arithmetic Expressions may contain column names, constant numeric values and the arithmetic operators :

Operators	Description
+	add
-	subtract
*	multiply
/	divide

The priority is * , / first, then +, - second (left to right if there are several operators with the same priority).

Parentheses may be used to specify the order in which operators are executed, if for example, addition is required before multiplication .

*SQL> SELECT ENAME, SAL*12 FROM EMP;*

ENAME	*SAL*12*
SMITH	*9600*
ALLEN	*19200*
WARD	*15000*
JONES	*35700*
MARTIN	*15000*
BLAKE	*34200*
CLARK	*29400*
SCOTT	*36000*
KING	*60000*
TURNER	*18000*
ADAMS	*13200*
JAMES	*11400*
FORD	*36000*
MILLER	*15600*

Column Aliases

When displaying the result of a query, SQL*Plus normally uses the selected column's name as the heading . In many cases it may be fairly cryptic or meaningless. You can change a column's heading by using an Alias.

To display the column heading ANNSAL for annual salary instead of SAL*12 user the alias:

*SQL> SELECT ENAME, SAL*12 ANNSAL, COMM FROM EMP;*

ENAME	*ANNSAL*	*COMM*
SMITH	*9600*	
ALLEN	*19200*	*300*
WARD	*15000*	*500*
JONES	*35700*	
MARTIN	*15000*	*1400*
BLAKE	*34200*	
CLARK	*29400*	
SCOTT	*36000*	
KING	*60000*	
TURNER	*18000*	*0*
ADAMS	*13200*	
JAMES	*11400*	
FORD	*36000*	
MILLER	*15600*	

14 rows selected.

The Concatenation Operator

The Concatenation Operator (||) allows columns to be linked to other columns, arithmetic expressions or constant values to create a character expression. Columns on either side of the operator are combined to make one single column.

To combine EMPNO and ENAME and give the expression the alias EMPLOYEE, enter:

SQL> SELECT EMPNO||ENAME EMPLOYEE FROM EMP;

EMPLOYEE

7369SMITH
7499ALLEN
7521WARD
7566JONES
7654MARTIN
7698BLAKE
7782CLARK
7788SCOTT
7839KING
7844TURNER
7876ADAMS
7900JAMES
7902FORD

7934MILLER

Literals

A literal is any character, expression, number included on the SELECT list which is not a column name or a column alias.

A literal in the SELECT list is output for each row returned. Literal strings of free format text can be included in the query result, and are treated like a column in the select list.

Date and character literals must be included in single quotes (‘); number literal do not need single quotes. The following statement contains literals selected with concatenation and a column alias:

```
SQL> SELECT EMPNO||'-'||ENAME EMPLOYEE, 'WORKS IN DEPARTMENT', DEPTNO
  2  FROM EMP;

EMPLOYEE       'WORKSINDEPARTMENT'    DEPTNO
---------------  ------------------------------  ---------
7369-SMITH      WORKS IN DEPARTMENT      20
7499-ALLEN      WORKS IN DEPARTMENT      30
7521-WARD       WORKS IN DEPARTMENT      30
7566-JONES      WORKS IN DEPARTMENT      20
7654-MARTIN     WORKS IN DEPARTMENT      30
7698-BLAKE      WORKS IN DEPARTMENT      30
7782-CLARK      WORKS IN DEPARTMENT      10
7788-SCOTT      WORKS IN DEPARTMENT      20
7839-KING       WORKS IN DEPARTMENT      10
7844-TURNER     WORKS IN DEPARTMENT      30
7876-ADAMS      WORKS IN DEPARTMENT      20
7900-JAMES      WORKS IN DEPARTMENT      30
7902-FORD       WORKS IN DEPARTMENT      20
7934-MILLER     WORKS IN DEPARTMENT      10
```

Functions

Functions have a self-descriptive names followed by various parameters (arguments) in parentheses. They transform the column values or literals to which they apply, and are covered in detail later in Section 4, “ Functions “.

Handling Null Values

A null value is a value which is either unavailable, unassigned, unknown or inapplicable. A null value is not the same as zero. Zero is a number . A null value in ORACLE Version 5 is truly non-existent and takes up no disk space at all . This is different in Version 7.

Null values are handled correctly by SQL.

If any column values in an expression are null, the result is null. In the following statement, only Salesmen have a remuneration result:

```
SQL> SELECT ENAME, SAL*12 + COMM ANNUAL_SAL
FROM EMP;

ENAME                ANNUAL_SAL
-------------------- --------------------
SMITH
ALLEN                       19500
WARD                        15500
JONES
MARTIN                      16400
BLAKE
CLARK
SCOTT
KING
TURNER                      18000
ADAMS
JAMES
FORD
MILLER

14 rows selected.
```

In order to achieve a result for all employees, it is necessary to convert the null value to a number. We use the NVL function to convert a null value to a non-null value.

```
SQL> SELECT ENAME, SAL*12 + NVL(COMM,0) ANNUAL_SAL
FROM EMP;

ENAME                ANNUAL_SAL
-------------------- --------------------
SMITH                        9600
ALLEN                       19500
WARD                        15500
JONES                       35700
MARTIN                      16400
BLAKE                       34200
CLARK                       29400
SCOTT                       36000
KING                        60000
TURNER                      18000
ADAMS                       13200
JAMES                       11400
FORD                        36000
MILLER                      15600

14 rows selected.
```

NVL expects two arguments:

1. an expression
2. a non-null value.

Note that you can use the NVL function to convert a null number, date or even character string to another number, date or character string, as long as the data types match.

NVL(DATACOLUMN,'01-JAN-88');
NVL(NUMBERCOLUMN,9)

The DISTINCT Clause

To eliminate duplicate values in the result, include the DISTINCT clause in the SELECT command :

To eliminate the duplicate values displayed in the previous example, enter :

```
SQL> SELECT DISTINCT DEPTNO
FROM EMP;

  DEPTNO
 ---------
    10
    20
    30
```

Multiple columns may be specified in the DISTINCT clause and the DISTINCT works on all selected columns.

To display distinct values of DEPTNO and JOB enter:

```
SQL> SELECT DISTINCT DEPTNO,JOB
FROM   EMP;

DEPTNO     JOB
 ---------     ---------
    10         CLERK
    10         MANAGER
    10         PRESIDENT
    20         ANALYST
    20         CLERK
    20         MANAGER
    30         CLERK
    30         MANAGER
    30         SALESMAN

9 rows selected.
```

Note that the DISTINCT command word can only be referred to once and must immediately follow the SELECT command word.

The ORDER BY Clause

Normally the order of rows returned in a query result is undefined. The ORDER BY

clause may be used to sort the rows. If used, ORDER BY must always be the last clause in the SELECT statement .

```
SQL> SELECT ENAME, SAL
  2  FROM EMP
  3  ORDER BY ENAME
  4  ;

ENAME                SAL
---------------      ---------
ADAMS               1100
ALLEN               1600
BLAKE               2850
CLARK               2450
FORD                3000
JAMES                950
JONES               2975
KING                5000
MARTIN              1250
MILLER              1300
SCOTT               3000
SMITH                800
TURNER              1500
WARD                1250

14 rows selected.
```

Default Ordering of Data

The default sort order is ASCENDING :

- Numeric values lowest first .
- Data values earliest first.
- Character values alphabetically.

Reversing the Default Order

To reverse this order, the command word DESC is specified after the column name in the ORDER BY clause.

To reverse the order of the HIREDATE column, so that the latest dates are displayed first, enter:

```
SQL> SELECT ENAME, HIREDATE
        FROM EMP
        ORDER BY HIREDATE DESC;

ENAME                HIREDATE
-------------------------  --------------
ADAMS                12-JAN-83
SCOTT                09-DEC-82
MILLER               23-JAN-82
```

```
JAMES            03-DEC-81
FORD             03-DEC-81
KING             17-NOV-81
MARTIN           28-SEP-81
TURNER           08-SEP-81
CLARK            09-JUN-81
BLAKE            01-MAY-81
JONES            02-APR-81
WARD             22-FEB-81
ALLEN            20-FEB-81
SMITH            17-DEC-80

14 rows selected.
```

Ordering by Many Columns

It is possible to ORDER BY more than one column. The limit is in fact the number of columns on the table. In the ORDER BY clause specify the columns to order by, separated by commas. If any or all are be reversed specify DESC after any or each column.

```
SQL>    SELECT DEPTNO, JOB, ENAME
          FROM EMP
        ORDER BY DEPTNO, SAL DESC


   DEPTNO JOB              ENAME
---------- ---------- ----------
       10  PRESIDENT  KING
       10  MANAGER    CLARK
       10  CLERK      MILLER
       20  ANALYST    SCOTT
       20  ANALYST    FORD
       20  MANAGER    JONES
       20  CLERK      ADAMS
       20  CLERK      SMITH
       30  MANAGER    BLAKE
       30  SALESMAN   ALLEN
       30  SALESMAN   TURNER
       30  SALESMAN   WARD
       30  SALESMAN   MARTIN
       30  CLERK      JAMES

14 rows selected.
```

Note: To order by a column, it is not necessary for that column to be in the SELECT list as the above example indicates.

Order By and NULL Values
With ORACLE version 6 and 7 (NOT 5), NULL values will always appear first whatever order is specified

The WHERE Clause

The WHERE clause may compare values in columns, literal values, arithmetic expressions or functions. The WHERE clause expects 3 elements :

1. A column name
2. A comparison operator
3. A column name, constant or list of values

Logical Operators

These logical operators will test the following conditions :

Operator	Meaning
=	equal to
>	greater than
>=	greater than or equal to
<	less than
<=	less than or equal to

To list the names, number, jobs and departments of all clerks, enter :

```
SQL> SELECT ENAME, EMPNO, JOB, DEPTNO
   FROM EMP
   WHERE JOB = 'SALESMAN';

ENAME                EMPNO     JOB              DEPTNO
-------------------- ----------  ---------------  --------------
ALLEN                7499      SALESMAN         30
WARD                 7521      SALESMAN         30
MARTIN               7654      SALESMAN         30
TURNER               844       SALESMAN        30
```

To find all employees with salaries greater than 2400, enter;

```
SQL> SELECT ENAME, JOB, HIREDATE, DEPTNO
  FROM EMP
  WHERE SAL > 2400;

ENAME                JOB           HIREDATE    DEPTNO
-------------------- ---------- ---------------  -------------
JONES                MANAGER    02-APR-81        20
BLAKE                MANAGER    01-MAY-81        30
CLARK                MANAGER    09-JUN-81        10
SCOTT                ANALYST    09-DEC-82        20
```

```
KING            PRESIDENT  17-NOV-81       10
FORD            ANALYST    03-DEC-81       20

6 rows selected.
```

You can compare a column with another column in the same row, as well as with a constant value.

For example, suppose you want to find those employees whose commission is greater than their salary, enter:

```
SQL> SELECT ENAME, SAL , COMM
  FROM EMP
  WHERE COMM > SAL;

ENAME                SAL     COMM
-------------------- ----------     ---------
MARTIN               1250     1400
```

SQL Operators

There are four SQL operators which operate with all data types :

SQL Operators

Operator	Meaning
BETWEEN..AND...	between two values (inclusive)
IN(list)	match any of a list of values
LIKE	match a character pattern
IS NULL	is a null value (delete BETWEEN)

The BETWEEN Operator

Tests for values between, and inclusive of, low and high range.

Suppose we want to see those employees whose salary is between 1000 and 2000:

```
SQL> SELECT ENAME , SAL
FROM EMP
WHERE SAL BETWEEN 1000 AND 2000;

ENAME                SAL
-------------------- ---------
ALLEN                1600
WARD                 1250
```

```
MARTIN          1250
TURNER          1500
ADAMS           1100
MILLER          1300

6 rows selected.
```

Note that values specified are inclusive, and the lower limit must be specified first.

The IN Operator

Test for values in a specified list.

To find all employees who have a given MGR number, enter:

```
SQL> SELECT EMPNO, ENAME
FROM EMP
WHERE EMPNO IN (7499, 7902,7788);

  EMPNO  ENAME
---------    -------------------------
   7788     SCOTT
   7902     FORD
   7499     ALLEN
```

The LIKE Operator

Sometimes you may not know the exact value to search for . Using the LIKE operator it is possible to select rows that match a character pattern, The character pattern matching operation may be referred to as a `wild-card` search. Two symbols can be used to construct the search string .

Symbol	Represents
%	any sequence of zero or more characters
-	any single character

To list all employees whose names start with an S, enter:

```
SQL> SELECT ENAME FROM EMP
       WHERE ENAME LIKE 'S%';

ENAME
-------------------------
SMITH
SCOTT
```

To list all employees who have a name exactly 4 characters in length

```
SQL> SELECT ENAME
  2  FROM EMP
```

```
  3  WHERE ENAME LIKE '_ _ _ _';

ENAME
-------------------------
WARD
KING
FORD
```

IS NULL Operator

The IS NULL operator specifically tests for values that are NULL.

So to find all employees who have no manager, you are testing for a NULL value:

```
SQL> SELECT ENAME, MGR
FROM EMP
WHERE MGR IS NULL;

ENAME                MGR
-------------------- ---------
KING
```

Negating SQL Operators

Operator	Description
NOT BETWEEN	not between two given values
NOT IN	not in given list values
NOT LIKE	not like string
IS NOT NULL	is not a null value

To find employees whose salary is not between a range, enter:

```
SQL> SELECT ENAME, SAL
FROM  EMP
WHERE SAL NOT BETWEEN 1000 AND 2000;

ENAME                SAL
-------------------- ---------
SMITH                 800
JONES                2975
BLAKE                2850
CLARK                2450
SCOTT                3000
KING                 5000
JAMES                 950
FORD                3000

8 rows selected.
```

Querying Data with Multiple Conditions

The AND and OR operators may be used to make compound logical expression

To find all employees who are either clerks and/or all employees who earn between 1000 and 2000, enter:

```
SQL> SELECT EMPNO, ENAME, JOB, SAL
  FROM EMP
 WHERE SAL BETWEEN 1000 AND 2000
 OR    JOB = 'CLERK';

    EMPNO ENAME                JOB          SAL
--------- ------------------------- --------------      ---------
     7369 SMITH                CLERK           800
     7499 ALLEN                SALESMAN       1600
     7521 WARD                 SALESMAN       1250
     7654 MARTIN               SALESMAN      1250
     7844 TURNER               SALESMAN      1500
     7876 ADAMS                CLERK          1100
     7900 JAMES                CLERK           950
     7934 MILLER               CLERK          1300

8 rows selected.
```

Operator Precedence

All operators are arranged in a hierarchy that determines their precedence. In an expression, operations are performed in order of their precedence, from highest to lowest. Where operators of equal precedence are used next to each other, they are performed from left to right.

1. All of the comparison and SQL operators, have equal precedence :

   = , !=, <, >, <=, >=, BETWEEN...AND, IN, LIKE, IS NULL.

2. NOT (to reverse a logical expression's result ie. WHERE not (SAL>2000)

3. AND

4. OR

SELECT Summary

So far the following clauses have been covered in the SELECT command:

```
SELECT   [DISTINCT] {*,column [alias],...}
FROM     table
WHERE    condition(s)
ORDER BY  {column,expr} [ASC|DESC};
```

Note that ORDER BY is always specified last.

Running Standard Queries with Substitution Variables.

Single Ampersand(&) Substitution Variables.

You can use substitution variables in a command file or SELECT statement to represent values to be provided at run time. A variable can be thought of as a container in which values are stored temporarily.

A variable is prefixed by a single ampersand (&), and a value is assigned to it.

The following statement prompts the user for a department number at run time.

```
SQL> SELECT  ENAME, SAL
 2  FROM    EMP
 3  WHERE   DEPTNO = &DEPT_NO;
```

Enter value for dept_no: 10

```
old   3: WHERE   DEPTNO = &DEPT_NO
new   3: WHERE   DEPTNO = 10
```

```
ENAME                    SAL
-------------------------   ---------
CLARK                    2450
KING                     5000
MILLER                   1300
```

Note how &dept_no in the old clause was substituted with 10 in the new clause.

With the single ampersand the user is prompted every time the command is executed, because the variable is not defined and consequently the value entered is not saved.

It is possible to prompt for a column name or even a table name at run time

```
SQL> SELECT  ENAME , &COL_NAME
   FROM EMP
  WHERE SAL > 4000;

Enter value for col_name: JOB

old   1: SELECT  ENAME , &COL_NAME
new   1: SELECT  ENAME , JOB

ENAME                JOB
------------------------- -------------
KING                 PRESIDENT
```

Double Ampersand Substitution Variable

If the variable is prefixed with a double ampersand (&&), SQL*Plus will only prompt for the value once. The variable prompted for is defined system level and used again whenever the SQL statement is run.

```
SQL> SELECT ENAME, JOB
FROM EMP
WHERE   DEPTNO = &&DEPT_NO;

Enter value for dept_no: 10
old   3: WHERE   DEPTNO = &&DEPT_NO
new   3: WHERE   DEPTNO = 10

ENAME                JOB
-------------------------- ---------
CLARK                MANAGER
KING                 PRESIDENT
MILLER               CLERK
```

Use the DEFINE command and you will find that DEPT_NO is defined for the session

```
SQL> DEFINE
DEFINE DEPT_NO        = "10" (CHAR)
```

You can use the SQL*Plus DEFINE command to determine if a variable is already defined. If the variable is defined, it displays the assigned value.

The DEFINE command is also used to create a user variable.

The DEFINE command

A value may be assigned to a variable using the SQL*Plus command DEFINE. The value defined may be referenced in a SELECT statement or command file by prefixing the variable name with an ampersand (&). The variables may be emptied by using the UNDEFINE command.

Example

```
SQL> DEFINEMY_DEPT ='30'
SQL> SELECT ENAME,SAL
        FROM EMP
        WHERE DEPTNO = &MY_DEPT;

old   3: WHERE DEPTNO = &MY_DEPT
new   3: WHERE DEPTNO = 30

ENAME                SAL
-------------------------- ---------
ALLEN                1600
WARD                 1250
MARTIN               1250
BLAKE                2850
TURNER               1500
```

JAMES 950

6 rows selected.

The ACCEPT Command

The ACCEPT command allows a variable to be created and a value, which is entered at runtime, stored in it. This variable can then be referenced in a SQL statement. ACCEPT is often used in a command file. There are benefits in using ACCEPT to define substitution variables:

-Data types can be checked.

-Prompts can be made far more explanatory.

-Response value can be hidden

The syntax of the command is:

ACCEPT variable [NUMBER|CHAR] [PROMT 'text'] [HIDE]

Example

```
SQL> ACCEPT  MY_JOB CHAR PROMPT  'ENTER JOB : '

ENTER JOB : 'SALESMAN'

SQL> SELECT ENAME, SAL FROM EMP
        WHERE JOB=&MY_JOB;
old   2: WHERE JOB=&MY_JOB
new   2: WHERE JOB='SALESMAN'

ENAME                    SAL
-------------------------  ---------
ALLEN                     1600
WARD                     1250
MARTIN                   1250
TURNER                   1500
```

FUNCTIONS

Introduction to Functions

Functions are used to manipulate data items. They accept one or more arguments and return one value. An argument is a user-supplied constant, variable or column reference. The format for a function is as follows:

function_name (argument 1, argument2, ...)

Functions can be used to:

- perform calculations on data
- modify individual data items
- manipulate output for groups of rows
- alter date formats for display
- convert column data types

There are different types of functions:

- CHARACTER
- NUMBER
- DATE
- CONVERSION
- FUNCTIONS THAT ACCEPT ANY DATA TYPE AS INPUT
- GROUP

Single Row Functions

- Act on each row returned in the query.
- Return one result per row.
- expect one or more user arguments.
- may be nested
- can be used anywhere that user variables, columns, expressions can be used for example; SELECT, WHERE , ORDER BY clauses

The following table explains the notations that will be used while describing functions

Explanation of Notation

Notation	Meaning
Col	Any named database columns
Value	Any literal value (character/date/number)
n	A number
'string'	character string
chars	a number of specified characters
date	date column, or date value

CHARACTER and NUMBER Functions

Character Functions

Single row character functions accept character data as input and can return both character and number values.

The following functions influence the case of character values

LOWER(Col|value) Force alpha character values into lower case.

Enter:

```
SQL> SELECT LOWER(ENAME), LOWER('aNy VaLuE')
   FROM EMP WHERE DEPTNO = 30;

LOWER(ENAME)   LOWER('AN
-------------------------- ---------
allen                      any value
ward                       any value
martin                     any value
blake                      any value
turner                     any value
james                      any value

6 rows selected.
```

UPPER (col|value) Forces alpha character values into upper case

INITCAP (col|values) Forces the first letter of each word of the string to be capitalized

To display the department names and locations in mixed case, enter

```
SQL> SELECT INITCAP(DNAME), INITCAP(LOC)
       FROM  DEPT;

INITCAP(DNAME) INITCAP(LOC)
-------------- -------------
Accounting     New York
Research       Dallas
Sales          Chicago
Operations      Boston
```

LPAD and RPAD functions pad(يلصق) character values to a specified length

LPAD(col|value,n,'string') The total length of the string should be n
The leading spaces are filled with 'string'

```
SQL> SELECT ENAME, LPAD (SAL,15,'.')
  2  DROM EMP
  3  WHERE SAL < 1500;

ENAME                LPAD(SAL,15,'.'
-------------------------- ---------------
SMITH                .....800
WARD                 .....1250
MARTIN               .....1250
ADAMS                .....1100
JAMES                .....950
MILLER               .....1300
```

```
SQL> COLUMN GRAPH FORMAT A50
SQL> SELECT SAL, LPAD(' ',ROUND(SAL/100,0),'*') AS GRAPH FROM EMP;

    SAL GRAPH
--------- --------------------------------------------------
    800 *******
   1600 ****************
   1250 *************
   2975 ******************************
   1250 *************
   2850 ******************************
   2450 ************************
   3000 ******************************
   5000 **************************************************
   1500 ***************
   1100 **********
    950 *********
   3000 ******************************
   1300 *************

14 rows selected.
```

RPAD (col|value,n,'string") The total length is n and the trailing spaces are filled with 'string'.

SUBSTR(col|value,pos,n) Extracts a character string from the original string. the extraction starts at position POS and affects a string of length n. NOTE, if n is omitted the string is extracted from POS to the end.

The next example displays the following Substrings;

- Two characters from the literal AMMAR starting at the second positions
- Contents of DNAME beginning with the third character till the end.

```
SQL> SELECT SUBSTR('AMMAR',2,2), SUBSTR(DNAME,3)
       FROM DEPT;

SU      SUBSTR(DNAME
------  ------------
MM      COUNTING
MM      SEARCH
MM      LES
MM      ERATIONS
```

6 rows selected.

INSTR(col|value,'string') Finds the character position of first occurrence of the 'string'

INSTR(col|value,'string',pos,n) Find the character position of the nth occurrence of 'string' in column or literal value starting at the position number POS

```
SQL> SELECT DNAME, INSTR(DNAME,'C',1,2)
  FROM DEPT;

DNAME          INSTR(DNAME,'C',1,2)
-------------  ----------------------------
ACCOUNTING                        3
RESEARCH                          0
SALES                             0
OPERATIONS                        0
```

Another Example:

Assume that you have a table containing the following field which contains the year in the last two digits (95)

123/AB/95
87G12/33AC/95

and you want to extract the year from the string. Note that the position of the year is not fixed

.

The following SQL will get the correct answer.

```
SQL> SELECT SUBSTR('&&X',INSTR('&X','/',1,2)+1,2) FROM DUAL
  2  ;
Enter value for x: 123/AB/95
old   1: SELECT SUBSTR('&&X',INSTR('&X','/',1,2)+1,2) FROM DUAL
new   1: SELECT SUBSTR('123/AB/95',INSTR('123/AB/95','/',1,2)+1,2) FROM DUAL

SU
----
95

SQL> UNDEF X

SQL> SELECT SUBSTR('&&X',INSTR('&X','/',1,2)+1,2) FROM DUAL
  2  ;

Enter value for x: 87G12/33AC/95
old   1: SELECT SUBSTR('&&X',INSTR('&X','/',1,2)+1,2) FROM DUAL
new   1: SELECT SUBSTR('87G12/33AC/95',INSTR('87G12/33AC/95','/',1,2)+1,2) FROM

SU
----
95
```

LTRIM(col|value,'char') Removes form the left leading occurrence(s) of char (or combination of leading chars) specified. If char is not specified will trim any blanks from the left.

```
SQL> SELECT ENAME, LTRIM(ENAME,'S') FROM EMP
WHERE SAL < 1500;

ENAME                LTRIM(ENAME,'S')
-------------------- ------------------------
SMITH                MITH                -- NOTE THAT LEADING S WAS REMOVED
WARD                 WARD
MARTIN               MARTIN
ADAMS                ADAMS
JAMES                JAMES
MILLER               MILLER

6 rows selected.
```

RTRIM has the same function of LTRIM but the chars are removed from right instead of left. This function can be particularly helpful in removing extraneous blanks from the end of data in a column.

Example

```
UPDATE EMP
SET ENAME = RTRIM(ENAME)
```

SOUNDEX(col|value) Returns a character string representing the sound of word(s) for each column or literal value.

```
SQL> SELECT ENAME,SAL FROM EMP
WHERE SOUNDEX(ENAME) = SOUNDEX('MELLER');

ENAME                SAL
-------------------- ---------
MILLER               1300
```

LENGTH(col|value) Returns the number of character (or numbers) in the column or literal value

```
SQL> SELECT LENGTH('AMMAR'), LENGTH(DNAME),
FROM DEPT

LENGTH('AMMAR') LENGTH(DNAME)    DNAME
--------------- ---------------- -------------
              5               10 ACCOUNTING
              5                8 RESEARCH
              5                5 SALES
              5               10 OPERATIONS
```

TRANSLATE (col|value,from,to) Translates for output the character *from* to the character *to.* More than one character can be matched. All occurrences of *from* are replaced with corresponding character in *to.* It is one character to one character correspondence.

```
SQL> SELECT TRANSLATE ('AMMAR','M','X'),TRANSLATE('SAJDI','S','M')
       FROM DUAL

TRANS     TRANS
--------- ---------
AXXAR     MAJDI
```

Note that each occurrence of the letter 'M' was translated to letter 'X'. Practical usage of this function can be Code Page translation and Encryption

REPLACE (col|value,from,to) Returns char with every occurrence of *from* replaced with to. If *to* is omitted or null, all occurrences of *from* are removed. If from is null, char is returned. This function provides a superset of the functionality provided by the TRANSLATE function. TRANSLATE provides single character, one to one, substitution. REPLACE allows you to substitute one string for another as well as to remove character strings.

EXAMPLE:

```
SQL>  SELECT REPLACE('AMMAR and AMMAN','AM','O') "Changes"
      FROM DUAL;
```

```
Changes
-------------
OMAR and OMAN
```

ASCII(col|value) Returns the decimal representation in the database character set of the first byte of char.

EXAMPLE:

```
SQL>SELECT ASCII('A')  FROM DUAL

 ASCII('A')
----------
   65
```

CHR(col|value) Returns the character having the binary equivalent to n in the database character set.

EXAMPLE:

```
 SQL>  SELECT CHR(65) "Character"  FROM DUAL

  Character
  ---------
  K
```

Number Functions

Number functions accept numeric input and return numeric values. Most of these functions return values that are accurate to 38 decimal digits.

ROUND(col|value,n) Returns col or value rounded to n places right of the decimal point; if n is omitted, to 0 places. n can be negative to round off digits left of the decimal point. n must be an integer.

```
SQL> SELECT ROUND (65.823,1),
 2       ROUND (65.823),
 3       ROUND (42.999,-1),
 4       ROUND (SAL*17/100,2)
```

```
 5  FROM EMP WHERE DEPTNO =20;

ROUND(65.823,1)  ROUND(65.823)   ROUND(42.999,-1)   ROUND(SAL*17/100,2)
---------------  --------------- ----------------- -------------------
        65.8             66              40                136
        65.8             66              40              505.75
        65.8             66              40                510
        65.8             66              40                187
        65.8             66              40                510
```

TRUNC(col|value,n) truncates the column or value to n decimal places, or if n is omitted no decimal places. if n is negative numbers left of decimal place are truncated to zero.

```
SQL> SELECT TRUNC (99.99), TRUNC (99.99,-1)
FROM DUAL;

TRUNC(99.99)        TRUNC(99.99,-1)
---------------     ----------------------
        99                  90
```

ABS(col|value) Returns the absolute value

```
SQL>  SELECT ABS(-20), COMM-SAL,ABS(COMM-SAL)
      FROM EMP
      WHERE DEPTNO = 30

 ABS(-20)    COMM-SAL          ABS(COMM-SAL)
---------    -----------------  ---------------------
     20      -1300               1300
     20       -750               750
     20      150                 150
     20
     20      -1500               1500
     20

6 rows selected.
```

MOD(value1,value2) find the remainder of value1 divided by value2

```
SQL> SELECT   MOD(3,2) FROM DUAL;

MOD(3,2)
-------------
    1
```

CEIL(col|value) Returns smallest integer greater than or equal to the column, expression of value.

```
SQL> SELECT CEIL(99.1),CEIL(-11.3)
FROM DUAL

CEIL(99.1)    CEIL(-11.3)
----------    -----------
       100            -11
```

FLOOR (col|values) Returns largest inter less than or equal to the column, expression or value.

```
SQL>  SELECT floor(99.1),floor(-11.3)
  FROM DUAL

FLOOR(99.1)  FLOOR(-11.3)
-----------  ------------
         99           -12
```

SIGN(col|value) Return -1 if a column, expression or value is a negative number, returns 0 if zero and +1 if positive .

```
SQL>SELECT SIGN(-15) "Sign"   FROM DUAL

   Sign
   ----
     -1
```

The following mathematical functions are available with ORACLE7 and are accurate to 36 decimal digits.

COS( ), COSH( ),EXP( ), LN( ), LOG( ), SIN( ), SINH( ), TAN( ), TANH( ), SQRT( ), POWER()

EXAMPLE

```
SELECT COS(180 *3.14159265359/180)    "Cosine of 180 degrees"
FROM DUAL

Cosine of 180 degrees
---------------------
                   -1
```

Nested functions

Single row functions can be nested to any depth. If functions are nested, they are evaluated from the inner to the outer functions(s).

Suppose that you want to find out the number of times a specified character occurs in a string. How would you achieve this?

You can nest the LENGTH and TRANSLATE functions to achieve the required result. The following example allows you to count the number of S's in a string

```
SQL> SELECT DNAME, LENGTH(DNAME)-LENGTH(TRANSLATE(DNAME,'AS','A'))
FROM DEPT;

DNAME          LENGTH(DNAME)-LENGTH(TRANSLATE(DNAME,'AS','A'))
-------------- -----------------------------------------------
ACCOUNTING                                                   0
RESEARCH                                                     1
SALES                                                        2
OPERATIONS                                                   1


SQL> SELECT DNAME,TRANSLATE(DNAME,'AS','A'),
   LENGTH(DNAME)-LENGTH(TRANSLATE(DNAME,'AS','A'))
  FROM DEPT;

DNAME          TRANSLATE(DNAM LENGTH(DNAME)-LENGTH(TRANSLATE(DNAME,'AS','A'))
-------------- -------------- -----------------------------------------------
ACCOUNTING     ACCOUNTING                                                   0
RESEARCH       REEARCH                                                      1
SALES          ALE                                                          2
OPERATIONS     OPERATION                                                    1
```

-Notice that A is changed to A, and S has no corresponding character to which it can be changed, S is therefore changed to nothing - removed from the string. The A serves as a placeholder.

-Now, subtract the length of the string - from which we have eliminated all S's, from the length of the original string (string with S's included).

-The result is a value representing a count of the number of occurrences of the character S in the string

Date Functions

ORACLE Date Storage.

Oracle stores dates in an internal numeric format, representing:

- Century
- Year
- Month

- Day
- Hours
- Minutes
- Seconds

The default display / input format for any date is DD- MON- YY. Oracle dates can rang between 1 st Jan 4712 BC and 31 st Dec 4712 AD.

SYSDATE is a pseudo-column that returns the current date and time. You can use SYSDATE just as you would use any other column name. For example, you can display the current date by selecting SYSDATE from a table. It is customary to select SYSDATE from a dummy table called DUAL. The DUAL table is owned by the SYSTEM and may be accessed by all users. It contains one column, and one row with a value `X`. The DUAL table is useful when you want to return a value once only- for instance, the value of a constant, pseudo_column or expression that is not derived from a table with `user` data.

To display the current date

```
SQL>SELECT SYSDATE FROM SYS.DUAL;
```

You could have easily SELECTed SYSDATE FROM EMP, but 14 rows of the same SYSDATE would have been returned, one for every row of the EMP table. DUAL is preferred because it conveniently returns one row only.

Using Arithmetic Operators.

Due to the fact that the date is stored as a number, it is possible to perform calculations with dates using arithmetic operators such as addition and subtraction. You can add and subtract number constants as well as other dates from dates.

The operations you may perform are:

date + number	Adds a number to a date producing a date.
date-number	Subs a number from a date producing a date.
date - date	Subs one date from another, producing a number of days
date+number/24	Adds a number of hours to a date producing a date.

```
SQL>SELECT HIREDATE, HIREDATE+7, SYSDATE-HIREDATE
  FROM EMP
  WHERE HIREDATE LIKE '%MAY%'

HIREDATE     HIREDATE+7       SYSDATE-HIREDATE
---------------  ------------------  ------------------------------
01-MAY-81    08-MAY-81        5473.3738
```

Subtracting SYSDATE from the HIREDATE column of the EMP table returns the number of days since each employees was hired.

MONTHS_BETWEEN Find the number of months between *date1*and *date2*
(date1,date2)

```
SQL> SELECT MONTHS_BETWEEN(SYSDATE,HIREDATE),
     MONTHS_BETWEEN('20-AUG-96','1-JAN-96')
     FROM   EMP
     WHERE  SAL < 1350;

MONTHS_BETWEEN(SYSDATE,HIREDATE)  MONTHS_BETWEEN('20-AUG-96','1-JAN-96')
--------------------------------  ------------------------------------
                       184.27024                            7.6129032
                       182.10895                            7.6129032
                        174.9154                            7.6129032
                       159.43153                            7.6129032
                       172.72186                            7.6129032
                       171.07669                            7.6129032

6 rows selected.
```

ADD_MONTHS(date,n) Add n number of calendar months to date.

```
SQL> SELECT ADD_MONTHS ('20-DEC-95',4) FROM DUAL;

ADD_MONTH
---------
20-APR-96
```

NEXT_DAY(date1,char) Date of the next specified day of the week following date1. Char may be a number representing a day.

```
SQL> SELECT HIREDATE, NEXT_DAY(HIREDATE,'FRIDAY')
FROM EMP
WHERE DEPTNO =20;

HIREDATE      NEXT_DAY(
---------     ---------
17-DEC-80     19-DEC-80
02-APR-81     03-APR-81
09-DEC-82     10-DEC-82
12-JAN-83     14-JAN-83
03-DEC-81     04-DEC-81
```

LAST_DAY(date1) Finds the date of the last day of the month that contains date1

```
SQL>SELECT LAST_DAY(SYSDATE),LAST_DAY('12-FEB-96')
     FROM DUAL

LAST_DAY(SYSDATE)             LAST_DAY
------------------------------ ---------
```

30-APR-96 *29-FEB-96*

Note that LAST_DAY automatically takes care of leap year (السنه الكبيسه)

The ROUND function applies to dates

ROUND(date1)	Returns date1 with the time set to 12:00 AM (midnight). This is useful when comparing dates that may have different times.
ROUND(date1,'MONTH')	Returns the first day of the month containing date1 if date1 is in the first half of the month; otherwise returns the first day of the following month.
ROUND(date1,'YEAR')	Return the first day of the year containing date1 if date1 is in the first half of the month; otherwise returns the first day of the following year.

```
SQL> SELECT SYSDATE, ROUND(SYSDATE,'MONTH')
FROM SYS.DUAL;

SYSDATE     ROUND(SYS
----------- -----------
25-APR-96   01-MAY-96
```

TRUNC(DATE1,'char')	Returns the first day of the month contained in date1 when char ='MONTH'. if char='YEAR', it finds the first day of the year containing date1.

Conversion Functions

SQL Provides a number of functions to control data type conversions. These conversion functions convert a value from one datatype to another. For example, the TO_CHAR(number|date ,'nls_param') converts NUMBER datatype to a value of VARCHAR2 datatype, using the optional number format fmt. If you omit fmt, the number is converted to a VARCHAR2 value exactly long enough to hold its significant digits.

The 'nlsparams' specifies these characters that are returned by number format elements:

* decimal character
* group separator
* local currency symbol
* international currency symbol

This argument can have this form:

'NLS_NUMERIC_CHARACTERS = "dg"
NLS_CURRENCY = "text"

NLS_ISO_CURRENCY = "text" '

The characters d and g represent the decimal character and group separator, respectively. They must be different single-byte characters. Note that within the quoted string, you must use two single-quotes to represent one around the parameter values.

If you omit 'nlsparams' or any one of the parameters, this function uses the default parameter values for your session.

EXAMPLE:

```
SELECT TO_CHAR(17145,'L099G999', 'NLS_NUMERIC_CHARACTERS ='.,''
NLS_CURRENCY = ''JD'' ') "Char"   FROM DUAL
```

```
Char
--------------
JD 017,145
```

The TO_CHAR(date,'date picture','nls param') function is frequently used to change a date format from the default to an alternative display format
if you omit the date picture, the date is converted to a VARCHAR2 value in the default
date format. The 'nlsparams' specifies the language in which month and day names and abbreviations are returned. This argument can have this form:

'NLS_DATE_LANGUAGE = language'

If you omit nlsparams, this function uses the default date language for your session.

EXAMPLES:

```
SQL>SELECT TO_CHAR(HIREDATE,'Month DD, YYYY')   "New date format"
  FROM emp
  WHERE ename = 'SMITH'

New date format
-------------------------------
December 17, 1980
```

```
SQL>  SELECT SYSDATE, TO_CHAR(SYSDATE,'MONTH , MON, DY' ) FROM DUAL;

SYSDATE       TO_CHAR(SYSDATE,'MONTH,MON,DY')
--------------  ------------------------------------------------------------------------
27-APR-96       APRIL     , Apr, Sat
```

Note that the pictures MONTH, DAY in the output are automatically padded with blanks to a length of 9. To remove the blank padding use the FM (Fill Mode) prefix

```
SQL> SELECT SYSDATE , TO_CHAR(SYSDATE,'FMMONTH, YYYY') FROM DUAL;
```

```
SYSDATE      TO_CHAR(SYSDATE,'FMMONTH,YYYY')
-----------  ------------------------------------------------------------
27-APR-96    APRIL, 1996
```

The TO_CHAR can also be used to extract the time of day and display it is specific format:

```
SQL> SELECT TO_CHAR(SYSDATE, 'HH24:MI:SS') FROM DUAL;

TO_CHAR(SYSDATE,'HH24:MI:SS')
---------------------------------------------------------
19:54:06
```

The following is a short list of popular date formats. Refer to ORACLE documentation for a full list

PICTURE	**MEANING**
YYYY	Year
SYYYY	Prefix “BC” date with “-”
YEAR	Year spelled out as in nineteen ninety-six
MONTH	Name of month, padded with blanks to length of 9
MON	Name of month, 3 letter abbreviation as in APR
WW	Week of Year
DDD	Day of year
HH	Hours of day (1-12)
HH24	Hours of day (0-23)
SSSSS	Seconds Past mid night (0-863999).

The following suffixes may be added to the codes above

TH	adding TH to DD for example 4TH
SP	Spelled out numbers (DDSP for FOUR)
SPTH	DDSPTH for FOURTH

Note. The codes are case sensitive add will affect display of date elements

DAY	SUNDAY
Day	Sunday
day	Sunday
MONTH	MAY
MON	MAY
Mon	May
mon	may
etc..	

TO_NUMBER(char|varchar)
Converts char,varchar2, which contains a number to a NUMBER

```
SQL> SELECT EMPNO, ENAME FROM EMP
       WHERE SAL < TO_NUMBER('1500')

   EMPNO        ENAME
---------       -------------------------
    7369         SMITH
    7521         WARD
    7654        MARTIN
    7876        ADAMS
    7900        JAMES
    7934        MILLER

6 rows selected.
```

TO_DATE ('char','fmt')

Converts char of CHAR or VARCHAR2 datatype to a value of DATE datatype. The fmt is a date format specifying the format of char. If you omit fmt, char must be in the default date format. If fmt is 'J', for Julian, then char must be a number.

To show all employees hired on June 4, 1984m we can use the TO_DATE function

```
SQL> SELECT EMPNO , ENAME , HIREDATE
       FROM EMP
       WHERE HIREDATE = TO_DATE('JANUARY 12, 1983','MONTH DD,YYYY');

EMPNO      ENAME                    HIREDATE
-----------  -------------------------  ---------------
    7876     ADAMS                    12-JAN-83
```

Miscellaneous Functions

DECODE

The decode function is best illustrated by example

```
SQL>  SELECT ENAME ,DEPTNO , DECODE (DEPTNO , 10 , 'EXCELLENT'
                    ,20, 'VERY GOOD',   'OK')
                    FROM EMP

ENAME                DEPTNO     DECODE(DE
-------------------------- ---------          ---------
SMITH                 20         VERY GOOD
ALLEN                 30         OK
WARD                  30         OK
JONES                 20         VERY GOOD
MARTIN                30                OK
BLAKE                 30         OK
CLARK                 10         EXCELLENT
SCOTT                 20         VERY GOOD
KING                  10                EXCELLENT
TURNER                30         OK
```

The decode function evaluate the expression (in this case DEPTNO). If it is equal to 10 then it is decoded to EXCELLENT, else if it is equal to 20 then it is decoded to VERY GOOD. For all other values it is decoded to OK.
The decode is a very powerful function and it has many practical uses.

NVL(col|value,val)	If col or value is equal to NULL it is converted to val, otherwise no conversion takes place.

```
SQL>SELECT SAL, COMM, SAL+COMM , SAL+NVL(COMM,0)
  FROM EMP
 WHERE SAL < 1340
```

SAL	COMM	SAL+COMM	SAL+NVL(COMM,0)
800			800
1250	500	1750	1750
1250	1400	2650	2650
1100			1100
950			950
1300			1300

6 rows selected.

Note that SAL+COMM evaluates to NULL if COMM is NULL. This because A+B is equal to NULL if either A or B or Both are NULL. The NVL(COMM,0) converts any COMM whose value is NULL to 0.

GREATEST (col|value,col|value,..)	Returns the greatest of the list of values.

```
SQL> SELECT SAL, NVL(COMM,0), GREATEST(SAL, NVL(COMM,0))
  FROM EMP WHERE SAL<1300;
```

SAL	NVL(COMM,0)	GREATEST(SAL,NVL(COMM,0))
800	0	800
1250	500	1250
1250	1400	1400
1100	0	1100
950	0	950

LEAST (col|value,col|value,..)	Returns the Least of the list of values
VSIZE(col|value)	Returns the number of bytes in ORACLE's Internal representation

```
SQL> SELECT ENAME,VSIZE(ENAME), HIREDATE, VSIZE(HIREDATE)
FROM EMP
WHERE SAL < 1300;
```

ENAME	VSIZE(ENAME)	HIREDATE	VSIZE(HIREDATE)
SMITH	5	17-DEC-80	7

WARD	*4*	*22-FEB-81*	*7*
MARTIN	*6*	*28-SEP-81*	*7*
ADAMS	*5*	*12-JAN-83*	*7*
JAMES	*5*	*03-DEC-81*	*7*

GROUP FUNCTION

Group functions operate on sets of rows. They return results based on groups of rows, rather than one result per row as returned by single row functions.
By default all the rows in a table are treated as one group. The GROUP BY clause of the SELECT statement is used to divide rows into smaller groups.

The group functions are listed below. All group functions except COUNT (*) ignore null values.

All of the above functions operate on a number of rows (for example, a whole table) and are therefore known as GROUP (or AGGREGATE) functions.

DISTINCT makes a group function consider only distinct (non-duplicate) values; ALL makes it consider every value including all duplicates. The default is ALL.

Using Group Functions

To calculate the average salary of all employees, enter:

```
SQL> SELECT AVG(SAL) FROM EMP;

 AVG(SAL)
---------
2073.2143
```

Note that the rows in the EMP table are treated as one group.

To find the mimimum salary earned by a manager, enter:

```
SQL> SELECT MIN(SAL) FROM EMP
WHERE JOB='MANAGER';

 MIN(SAL)
---------
    2450
```

TO count the number of employees in department 10, enter:

```
SQL> SELECT COUNT(*) FROM EMP WHERE DEPTNO=10;

 COUNT(*)
---------------
      3
```

The GROUP BY Clause

The Group By clause, can be used to divide the rows in a table into smaller groups, Group functions may be used to return summary information for each group.

To calculate the average salary for each different job, enter:

```
SQL> SELECT JOB, AVG(SAL) FROM EMP
GROUP BY JOB;

JOB             AVG(SAL)
---------       ---------
ANALYST         3000
CLERK           1037.5
MANAGER         2758.3333
PRESIDENT        5000
SALESMAN         1400
```

WHERE clause may be used with GROUP BY

```
SQL> SELECT JOB, AVG(SAL) FROM EMP
WHERE JOB IN ('SALESMAN','MANAGER')
GROUP BY JOB;

JOB              AVG(SAL)
---------        ---------
MANAGER          2758.3333
SALESMAN         1400
```

Groups Within Groups

We can also use the GROUP BY clause to provide result for groups within groups. To display the average monthly salary bill for each . job type within a department enter.

```
SQL> SELECT DEPTNO, JOB,AVG(SAL) FROM EMP
 2  GROUP BY DEPTNO, JOB;

  DEPTNO    JOB            AVG(SAL)
---------- ---------       ---------
      10    CLERK           1300
      10     MANAGER        2450
      10    PRESIDENT       5000
      20    ANALYST         3000
      20    CLERK           950
      20    MANAGER         2975
      30     CLERK          950
      30    MANAGER         2850
      30    SALESMAN        1400

9 rows selected.
```

Group Functions and Individual Results

The following SQL Statement returns the maximum salary for each job group. However, the result is not very meaningful because JOB is not displayed in the result.

```
SQL> SELECT MAX(SAL) FROM EMP
```

```
   GROUP BY JOB;

 MAX(SAL)
---------------
    3000
    1300
    2975
    5000
    1600
```

It is optional to display JOB, However, the values displayed will have more meaning if they are related to a job group

```
SQL> SELECT MAX(SAL), JOB FROM EMP
   GROUP BY JOB;

 MAX(SAL)   JOB
--------------   ---------
    3000       ANALYST
    1300       CLERK
    2975       MANAGER
    5000       PRESIDENT
    1600       SALESMAN
```

NOTE: Group functions cannot be retrieved with individual items that are not included in the GROUP BY clause.

For example:

```
SQL> select deptno , min(sal) from emp;
select deptno , min(sal) from emp
       *
ERROR at line 1:
ORA-00937: not a single-group group function
```

The command in invalid because DEPTNO has a value for each row in the table, while MIN(SAL) has a single value for the whole table.

To correct the error, we must group the individual items

```
SQL> SELECT DEPTNO, MIN(SAL) FROM EMP
GROUP BY DEPTNO;

   DEPTNO  MIN(SAL)
---------------  ---------
      10         1300
      20         800
      30         950
```

The HAVING Clause

Use the HAVING clause if you wish to specify which groups are to be displayed,

i.e. restrict the groups that you return.

To show the average salary for all departments employing more than three people, enter.

```
SQL> SELECT DEPTNO, AVG(SAL) FROM EMP
  GROUP BY DEPTNO
  HAVING COUNT(*) >3

   DEPTNO   AVG(SAL)
  ---------- ----------
     20         2175
     30       1566.6667
```

Note that the HAVING clause may precede the GROUP BY clause, but it is recommended that GROUP BY is listed first because it is more logical to do so.
Groups are formed and group functions are calculated before the HAVING clause is
applied to select groups for output.

The Order of Clause in the SELECT Statement

The WHERE clause may still be used to exclude rows form consideration. The order of the clauses is then:

```
SELECT column(s)
FROM   table(s)
WHERE row condition(s)
GROUP BY column(s)
HAVING  group of rows condition(s)
ORDER BY column(s)
```

Formatting The Output of SQL

SQL vs. SQL*Plus Commands

SQL*Plus is an environment within which two kinds of commands may be issued:
1. SQL commands (Such as SELECT).
2. SQL*Plus command (Such as SAVE).

SQL*Plus commands differ from SQL commands:

1. They are not related to any particular SELECT statement
2. They are entered on a single line. However, a continuation character, the hyphen, may be used if the command is too long for entry on a single line.

SQL*Plus commands may be used to affect the presentation of data retrieved by SELECT statements, and are therefore useful for producing reports, as well as controlling the environment and file handling.

Some of the general features possible in a SQL*Plus report are illustrated next:

SET Commands

The SET commands control the environment in which SQL*Plus is currently operating. To find the value of a SET variable use the SHOW command.

EXAMPLES

SET FEEDBACK OFF|ON Disables, Or enables the display of the number of records displayed by a query

SQL> SELECT ENAME, SAL FROM EMP WHERE SAL < 2000;

ENAME	*SAL*
SMITH	*800*
ALLEN	*1600*
WARD	*1250*
MARTIN	*1250*
TURNER	*1500*
ADAMS	*1100*
JAMES	*950*
MILLER	*1300*

8 rows selected. *-- This line is the feedback line, when feedback is set to off, it will no longer appear.*

SET HEADING OFF|ON

S QL> SET HEADING OFF

```
SQL> SELECT ENAME, SAL FROM EMP WHERE DEPTNO  = 20;


SMITH                800
JONES                2975
SCOTT                3000
ADAMS                1100
FORD                 3000
```

Note that column heading was suppressed as an outcome of HEADING OFF

SET PAUSE ON

This is a useful command which makes SQL*Plus waits for you to press {Return] before displaying a new page of output. One could also specify Text which SQL*Plus should display before waiting

```
SQL>SET PAUSE ON
SQL>  SET PAUSE 'PLEASE PRESS RETURN TO CONT'
SQL>  SET HEADING ON
SQL>  SELECT ENAME, SAL FROM EMP WHERE DEPTNO = 20;

PLEASE PRESS RETURN TO CONT       -- SQL*Plus waits for you to press Enter

ENAME                     SAL
------------------------- ---------
SMITH                      800
JONES                     2975
SCOTT                     3000
ADAMS                      1100
FORD                      3000
```

SET NUMWIDHT n

When SQL*Plus displays numeric data items, it uses a default width of 9 digits. Some times this is too long for your data a consumes a lot of space. You can modify this width using SET NUMWIDTH

Before using NUMWIDTH

```
SQL> SHOW NUMWIDTH
numwidth 9
SQL> SELECT * FROM EMP WHERE DEPTNO = 20 ;

PLEASE PRESS RETURN TO CONT

   EMPNO    ENAME                JOB          MGR HIREDATE       SAL     COMM    DEPTNO
--------- ------------------------- --------- --------- --------- -------       ----      --------     -----------
NATIONALITY
--------------------
    7369       SMITH              CLERK        7902 17-DEC-80          800                  20


    7566       JONES              MANAGER      7839 02-APR-81                               20
```

7788	*SCOTT*	*ANALYST*	*7566 09-DEC-82*	*3000*	*20*
7876	*ADAMS*	*CLERK*	*7788 12-JAN-83*	*1100*	*20*
7902	*FORD*	*ANALYST*	*7566 03-DEC-81*	*3000*	*20*

```
SQL> SET NUMWIDTH 4
SQL> SELECT * FROM EMP WHERE DEPTNO = 20;
PLEASE PRESS RETURN TO CONT

EMPNO  ENAME       JOB         MGR  HIREDATE  SAL COMM DEPTNO  NATIONALITY
-----  ----------  ----------  ---- --------- ----- ---------- ----------
 7369  SMITH       CLERK       7902 17-DEC-80  800                    20
 7566  JONES       MANAGER     7839 02-APR-81 2975                    20
 7788  SCOTT       ANALYST     7566 09-DEC-82 3000                    20
 7876  ADAMS       CLERK       7788 12-JAN-83 1100                    20
 7902  FORD        ANALYST     7566 03-DEC-81 3000                    20
```

SET LINESIZE n

Sets the number of characters that SQL*Plus will display on a line before the beginning a new line. The default value is 80 and the Max value is 500.

SET NEWPAGE n

Sets the number of blank lines to be printed between the bottom title of each page and the top title of the next page. The default is 1. A value of 0 makes SQL*Plus send a form feed between pages. This clears the screen on most terminals.

SET PAGESIZE n

Sets the number of lines per page. The default is 24; properly fit for a screen page and not A4 printer page. For reports printed on A4 paper, a value of 54 (Plus a NEWPAGE value of 6) is ideal

SET TIMING ON

Makes SQL*Plus give timing statistics on each SQL statement run.

```
SQL> SET TIMING ON
SQL> SELECT ENAME , SAL FROM EMP
   WHERE DEPTNO = 10;
PLEASE PRESS RETURN TO CONT

ENAME                SAL
-------------------- ----
CLARK                2450
KING                 5000
MILLER               1300

 real: 1600        -- THIS IS IN MILLISECONDS. THEREFORE THE TIMING HERE IS 1.6s
```

SET TIME ON

Not to be confused with SET TIMING. This command will give you the current time with the SQL prompt

SQL> SET TIME ON
08:33:32 SQL> SQL> SET TIME ON
08:33:32 SQL> SELECT DNAME FROM DEPT
08:34:06 2 ;
PLEASE PRESS RETURN TO CONT

DNAME

ACCOUNTING
RESEARCH
SALES
OPERATIONS

real: 1540
08:34:09 SQL>

Note that the effects of previous SET Commands are retained

08:34:09 SQL> SHOW TIMING
timing ON
08:35:37 SQL> SHOW NUMWIDTH
numwidth 4
08:35:45 SQL> SHOW PAUSE
pause is ON and set to "PLEASE PRESS RETURN TO CONT"
08:35:52 SQL>

To Disable:

SQL> SET TIMING OFF
SQL> SET TIME OFF
SQL>SET PAUSE OFF

TO see system wide setting you can use the command

SQL> SHOW ALL

If you are using SQL*Plus under windows you can click on the **O**ptions Menu and invoke the Set **O**ptions submenu. You can then set and reset all SET commands graphically using the mouse.

SET AUTOCOMMIT ON|OFF
This SET command will make SQL*Plus issue an automatic COMMIT after each SQL command.

SET SPACE n
This command sets the spaces between column headings when displayed on the screen. The default is 1.

SET UNDERLINE 'character'

This set command allows you to change the underline character of the column header. The default is underscore '-'

```
SQL> SET UNDERLINE '='
SQL> SET SPACE 3
SQL> SELECT * FROM DEPT;

  DEPTNO          DNAME                    LOC
=========    ==============    =============
     10          ACCOUNTING          NEW YORK
     20          RESEARCH            DALLAS
     30          SALES               CHICAGO
     40          OPERATIONS          BOSTON

SQL> SET SPACE 0
SQL> SELECT * FROM DEPT;

  DEPTNODNAME       LOC
====================================
     10ACCOUNTING      NEW YORK
     20RESEARCH        DALLAS
     30SALES           CHICAGO
     40OPERATIONS      BOSTON
     12هندسه           سودان
```

There are more SET commands than the ones described in this training course literature. However, the one mentioned here are the most important.

When you exit the SQL*Plus session all SETTING that you made in this session are erased. When you log on again the default settings will prevail.

Customizing default setting for you SQL*Plus sessions

If you want to create environment of your own setting, you must create a file called LOGIN.SQL in your current directory. Include the SET COMMAND you want in this file. You would write the SET command is exactly the same way you would if you were entering them interactively in an SQL*Plus session. SQL*Plus will read this file every time a new SQL*Plus is session is invokes.

Example of such file

LOGIN.SQL

```
SET TIME ON
SET PAUSE 'RETURN'
SET PAUSE ON
```

COLUMN Options

The COLUMN command let you control who a particular column will appear when displayed on the SQL*Plus screen. It affects its picture, width and other options. It does not affect how the column is stored in the database.

Notes on the COLUMN Options

1. The column name must refer to a column alias if a column alias is used on the SELECT clause.

2. Once a column statement is issued, it remains active throughout the remainder of an SQL*Plus session. It continues to be set while other tables (Without this column) are displayed.

3. Column options can be cleared during the session.

4. Column option can be included in the LOGIN.SQL file mentioned earlier.

5. To find out the current options set for a column, enter:

SQL> COL column_name / alias

Examine the following Command

SQL>COL DNAME FORMAT A50 HEADING 'Department Name'

*SQL> SELECT * FROM DEPT ;*

DEPTNO	Department Name	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON
12	سودان	هندسه

The FORMAT option changed the picture of the DNAME column to A50 meaning Alphanumeric of with 50 characters and
the HEADING option changed the heading of the column.

SQL> COL SAL FORMAT $99999.99
SQL> SELECT EMPNO, SAL FROM EMP 2 ;

EMPNO	SAL
7369	$800.00
7499	$1600.00
7521	$1250.00
7566	$2975.00
7654	$1250.00
7698	$2850.00
7782	$2450.00
7788	$3000.00

The Picture of SAL is changed to 4 numeric digits and 2 decimals preceded with $

A NUMBER column's width defaults to the value of SET NUMWIDTH. To change the width, use FORMAT followed by an element as specified in the following Table.

Element	Example(s)	Description
9	9999	Determines the display width by the number of digits entered. Does not display leading zeroes.
0	0999	Displays leading zeroes.
	9990	Displays zero instead of a blank when a value is zero.
$	$9999	Prefixes a dollar sign to a value.
B	B9999	Displays a zero value as blank.
MI	9999MI	Displays "-" after a negative value.
PR	9999PR	Displays a negative value in angle brackets.
comma	9,999	Displays a comma in the position indicated.
period	99.99	Aligns the decimal point in the position indicated.
V	999V99	Multiplies value by 10n, where n is the number of "9's" after the "V."
EEEE	9.999EEEE	Displays in scientific notation (format must contain exactly four"E's").
DATE	DATE	Displays value as a date in MM/DD/YY format; used to format NUMBER columns that represent Julian dates.

Note: SQL*Plus default number justification is Right justified.

ERROR MESSAGES Due to Formatting

.. Value too big for format
% .. Value wrong type for format

Other COLUMN Options

There are many COLUMN options; only the important ones are going to be explained here.

WRAP
TRUNC
Allows you to inform SQL*Plus that whenever the actual contents of a column exceeds the width specified, that the additional information is WRAPped round onto the next line or TRUNCated. WRAP is the default.

WORD_WRAPPED
Moves an entire word to the next line rather than split it between two lines

CLEAR
Removes the previous column formatting commands for that column

```
SQL> COLUMN  SAL   -- Will display the current format of SAL
             column   sal ON
             format   $99999.99
             justify left
SQL> COLUMN SAL CLEAR
```

JUSTIFY
Your options are LEFT, CENTER and RIGHT. This options allows to specify justification of the column heading. By default CHAR/DATE are displayed JUSTIFY LEFT and NUMBER headings are JUSTIFY RIGHT. It is important to stress the this options affects the HEADING rather than the actual column values alignment

NULL string
Sets any NULL in the column to the specified string.

PRINT, NOPRINT
Causes the column to be displayed/not displayed on the output list

TEMP
specifies that the format for this column is applied for one SQL query only.

```
SQL> COLUMN DEPTNO NOPRINT TEMP
SQL> SELECT * FROM DEPT;

DNAME          LOC
==========   === =============
ACCOUNTING    NEW YORK
RESEARCH       DALLAS
SALES           CHICAGO
OPERATIONS    BOSTON


SQL> SELECT * FROM DEPT ;

  DEPTNO     DNAME              LOC
========= ========== =============
    10          ACCOUNTING    NEW YORK
    20          RESEARCH       DALLAS
    30          SALES            CHICAGO
    40          OPERATIONS    BOSTON
```

Examples of column formatting

```
SQL> COLUMN DEPTNO FORMAT 099 HEADING 'DEPT.'
SQL> COLUMN JOB FORMAT A9 HEADING 'JOB' JUSTIFY RIGHT
SQL> COLUMN EMPNO FORMAT 9999 HEADING 'EMP|NO'
```

```
SQL> COLUMN REM FORMAT 999,999.99 HEADING TOTAL
SQL> SELECT DEPTNO,JOB,EMPNO,SAL*12+NVL(COMM,0) REM
  2  FROM EMP WHERE DEPTNO=30;

                              EMP
DEPT.           JOB           NO       TOTAL
===== ========= ===== ===========
 030  SALESMAN  7499      19,500.00
 030  SALESMAN  7521      15,500.00
 030  SALESMAN  7654      16,400.00
 030  MANAGER   7698      34,200.00
 030  SALESMAN  7844      18,000.00
 030  CLERK     7900       11,400.00

6 rows selected.
```

This query result displays characteristics of the COLUMN Formatting options. One of the headings was split over two lines using one vertical bar (|). Notice that the JOB header is right aligned while the JOB values are left aligned

```
SQL> COLUMN COMM NULL 'NO COMM'
SQL> SELECT DEPTNO,JOB,EMPNO,SAL*12+NVL(COMM,0) REM, COMM
       FROM EMP WHERE DEPTNO=30;

                              EMP
DEPT.           JOB           NO       TOTAL      COMM
===== ========= ===== ======= =====
 030  SALESMAN  7499      19,500.00        300
 030  SALESMAN  7521      15,500.00        500
 030  SALESMAN  7654      16,400.00       1400
 030  MANAGER   7698      34,200.00 NO COMM
 030  SALESMAN  7844      18,000.00          0
 030  CLERK     7900       11,400.00 NO COMM
```

Notice that the NULL option has been used to force a character string to be displayed where a null would normally appear.

The TITTLE and BITTLE Commands

The TTITLE and BTITLE commands are used to produce titles on a page

TTITLE 'character string'
Prints the current date in the top left hand corner of each page. The page number in the top right hand corner and centers the title on the line below

BTITLE 'character string'
Prints the text centered at the bottom of each page. In both cases a '|' character will cause the following text to be centered on the next line.

```
SQL>TTITLE
SQL> BTITLE          --displays the current TTITLE or BTITILE

SQL> TTITLE OFF              --cancels the display of a previously defined title
SQL> BTITLE OFF
```

EXAMPLE

```
SQL> TTITLE 'COMPANY REPORT|PRODUCED BY PERSONNEL DEPT'
SQL> BTITLE 'COMPANY CONFIDENTIAL'
SQL> /

Fri May 03                                                    page
                              COMPANY REPORT
                        PRODUCED BY PERSONNEL DEPT

                                  EMP
DEPT.          JOB                NO      TOTAL      COMM
===== ========== ===== ======== =====
  030  SALESMAN                  7499     19,500.00       300
  030  SALESMAN                  7521     15,500.00       500
  030  SALESMAN                  7654     16,400.00      1400
  030  MANAGER                   7698     34,200.00 NO COMM
  030  SALESMAN                  7844     18,000.00         0
  030  CLERK                     7900      11,400.00 NO COMM

                           COMPANY CONFIDENTIAL

6 rows selected.
```

The TTITLE and BTITLE commands can include a number of clauses enabling the appearance of the title to be specified in more detail.

TTITLE *clause*

Clause can be
SKIP n
Skips to the start of a new line n times. If n is omitted, skips one line; if n is 0, backward to the start of the current line.

TAB n
Skips forward n print positions (back if n is negative)

LEFT, CENTER, RIGHT
left-align, center or right-align data on the current line. The data items which follow this clause are aligned as a group, up to the end of the TTITLE command or the next LEFT, CENTER, RIGHT or COLUMN. (Center and RIGHT use the LINESIZE value to calculate the positions of the data items.

SQL.PNO
The system variable for the current page number.

SQL.LNO

The system variable for the current Line n umber

SQL.USER
The system variable for the username

EXAMPLE

```
SQL> TTITLE LEFT 'PAGE : 'SQL.PNO  RIGHT 'PRODUCED BY ACCOUNTING DEPT' SKIP 2 -
> CENTER 'CONFIDENTIAL SALES REPORT' SKIP 2 CENTER '---------------------------------'
SQL> BTITLE CENTER  'END OF REPORT' SKIP  CENTER '--------------------------------'
SQL> SELECT ENAME, SAL FROM EMP
        WHERE COMM IS  NULL;

PAGE :       1                                          PRODUCED BY ACCOUNTING

                        CONFIDENTIAL SALES REPORT
                          -------------------------------------
ENAME                   SAL
==========          ============== =========
SMITH                    800
JONES                   2975
BLAKE                   2850
CLARK                   2450
SCOTT                   3000
KING                    5000
ADAMS                   1100
JAMES                   950
FORD                    3000
MILLER                  1300



                              END OF REPORT
                             --------------------------

10 rows selected.
```

SQL*PLUS COMMAND FILE

This file is like a batch file. It can contain SET commands, COLUMN, TTITLE, BTITLE and STANDARD SELECT statement. It is a file that usually has SQL extension (x.sql). It can be run from within SQL*Plus by using the START command

EXAMPLE

Create the following file using your editor and call test.sql:-

```
COLUMN SAL FORMAT 9999.99
TTITLE PALESTINE ENGINEERING CO.
SET HEADING ON
SET FEEDBACK OFF

SELECT ENAME, SAL FROM EMP WHERE SAL < 2000
/
SET HEADING OFF
SET FEEDBACK ON
COLUMN SAL CLEAR
TTITLE OFF
```

SQL>START test.sql

Suppressing Duplicate Values and Breaking a Report into Secitons

The rows of a report may be broken up into sections by using the BREAK command, By BREAKing on a column, the display of duplicate values in the column is suppressed. You may also leave blank lines or start a new page between section. Since a break will occur each time the column value changes, you should remember to ORDER BY the column in the SELECT statement or your report will be split into meaningless sections.

There can only be one BREAK command active at any one time; therefore, if you require multiple breaks they should all be specified as one BREAK command. You should list the break columns in order of importance, i.e. major breaks first

Break Options
Breaks can be active on:

- Column
- Row
- Page
- Report

A BREAK on REPORT will allow final summary calculations. At any break, the following options may be specified

Options	Description
PAGE	Throws a page when value in column changes.
SKIP n	Skips *n* number of lines when values changes.
DUP[LICATE]	Duplicates values. The default in NODUP[LICATE]

Examples:

BREAK ON REPORT ON DEPTNO PAGE ON JOB SKIP 2
BREAK ON REPORT ON DEPTNO PAGE ON JOB DUP

To clear the breaks issue the command:

CLEAR BREAKS

To display current breaks issue the command:

BREAK

```
SQL> BREAK ON DEPTNO SKIP 1
SQL> select deptno , job ,ename from emp
  order by deptno;
```

```
  DEPTNO  JOB         ENAME
--------- ---------   -------------------------
    10    MANAGER     CLARK
          PRESIDENT   KING
          CLERK       MILLER

    20    CLERK        SMITH
          CLERK       ADAMS
          ANALYST     FORD
          ANALYST     SCOTT
          MANAGER     JONES

    30    SALESMAN    ALLEN
          MANAGER     BLAKE
          SALESMAN    MARTIN
          CLERK       JAMES
          SALESMAN    TURNER
          SALESMAN    WARD

14 rows selected.
```

Summary Calculation

The COMPUTE command performs calculations on breaks established by the BREAK command.

SYNTAX : COMPUTE clause(s) OF column(s) ON break(s)

where clause is
AVG, COUNT, MAX,MIN,NUMBER,STD,SUM,VAR

There may be many COMPUTE commands, although it is often easier to specify all computes required in one command.

For example:

```
SQL>COMPUTE SUM AVG OF SAL COMM ON DEPTNO

SQL>  select deptno , job ,ename, sal from emp
 order by deptno

  DEPTNO  JOB         ENAME                    SAL
--------- ---------   ------------------------ ---------
    10    MANAGER     CLARK                     2450
          PRESIDENT   KING                      5000
          CLERK       MILLER                    1300
*********                                      ---------
avg                                             2916.6667
sum                                             8750

    20    CLERK       SMITH                     800
          CLERK       ADAMS                     1100
          ANALYST     FORD                      3000
          ANALYST     SCOTT                     3000
```

```
            MANAGER   JONES              2975
*********                                ---------
avg                                      2175
sum                                      10875

     30     SALESMAN  ALLEN              1600
            MANAGER   BLAKE              2850
            SALESMAN  MARTIN             1250
            CLERK     JAMES              950
            SALESMAN  TURNER             1500
             SALESMAN  WARD              1250
*********                                ---------
avg                                      1566.6667
sum                                      9400

14 rows selected.
```

ADVANCED EXAMPLE

It is required that we produce the following output (Report)

```
Sat May 11                                              page1
                          COMPANY REPORT

JOB          Department 10   Department 20   Department 30   Total by Job
---------    -------------   -------------    -------------  ------------
ANALYST      .00             6,000.00        .00             6,000.00
CLERK        1,300.00        1,900.00        950.00          4,150.00
MANAGER      2,450.00        2,975.00        2,850.00        8,275.00
PRESIDENT    5,000.00         .00            .00             5,000.00
SALESMAN     .00             .00            5,600.00        5,600.00
             -------------   -------------   -------------   ------------
             8,750.00        10,875.00       9,400.00        29,025.00

                          CONFIDENTIAL
```

Here are the steps to follow :-

1) Divide the SAL column into 3 separate departments using DECODE function

```
SQL> SELECT JOB,
  2   DECODE (DEPTNO,10,SAL,0) D10,
  3   DECODE (DEPTNO,20,SAL,0) D20,
  4   DECODE (DEPTNO,30,SAL,0) D30
  5  FROM EMP;

JOB           D10       D20       D30
---------     ---------  --------- ---------
CLERK         0         800        0
SALESMAN      0         0          1600
SALESMAN      0         0          1250
MANAGER       0         2975       0
SALESMAN      0         0          1250
```

```
MANAGER      0       0       2850
MANAGER      2450    0       0
ANALYST      0       3000    0
PRESIDENT    5000    0       0
SALESMAN     0       0       1500
CLERK        0       1100    0
CLERK        0       0       950
ANALYST      0       3000    0
CLERK        1300    0       0

14 rows selected.
```

2- Summarise the rows into groups

```
SQL> SELECT JOB,
    SUM(DECODE (DEPTNO,10,SAL,0)) D10,
    SUM(DECODE (DEPTNO,20,SAL,0)) D20,
    SUM(DECODE (DEPTNO,30,SAL,0)) D30
    FROM EMP
    GROUP BY JOB;

JOB          D10          D20       D30
---------    ---------    ---------          ---------
ANALYST      0            6000               0
CLERK        1300         1900               950
MANAGER      2450         2975      2850
PRESIDEN     5000         0                  0
SALESMAN     0            0                  5600
```

3- Add a dummy column in order to compute the total for each department,or alternatively BREAK ON REPORT and compute at the break level

```
SQL> SELECT JOB,
    SUM(DECODE (DEPTNO,10,SAL,0)) D10,
    SUM(DECODE (DEPTNO,20,SAL,0)) D20,
    SUM(DECODE (DEPTNO,30,SAL,0)) D30,
    SUM(SAL)  TOTAL ,
    SUM(0) DUMMY
    FROM EMP
    GROUP BY JOB;

JOB          D10          D20       D30              TOTAL            DUMMY
---------    ---------    ---------          ---------          ----------          ------------
ANALYST      0            6000               0                  6000                0
CLERK        1300         1900               950                4150                0
MANAGER      2450         2975      2850               8275               0
PRESIDEN     5000         0                  0                  5000                0
SALESMAN     0            0                  5600               5600                0
```

4- Finally format the report with COLUMN command and any other SQL*Plus commands required

```
SQL> SET PAGESIZE 16
SQL> COLUMN D10 HEADING 'Department 10' FORMAT 99,999.99
SQL> COLUMN D20 HEADING 'Department 20' FORMAT 99,999.99
SQL> COLUMN D30 HEADING 'Department 30' FORMAT 99,999.99
SQL> COLUMN TOTAL HEADING 'Total by Job' FORMAT 999,999.99
SQL> COLUMN DUMMY NOPRINT       NOTE: THIS MEANS DO NOT PRINT DUMMY COLUMN
SQL> BREAK ON DUMMY
```

```
SQL> COMPUTE SUM OF D10 D20 D30 TOTAL ON DUMMY
SQL> TTITLE 'COMPANY REPORT'
SQL> BTITLE 'CONFIDENTIAL'

SQL>SELECT JOB,
    SUM(DECODE (DEPTNO,10,SAL,0)) D10,
    SUM(DECODE (DEPTNO,20,SAL,0)) D20,
    SUM(DECODE (DEPTNO,30,SAL,0)) D30,
    SUM(SAL)  TOTAL ,
    SUM(0) DUMMY
    FROM EMP
    GROUP BY JOB;
```

Sat May 11 *page1*

COMPANY REPORT

JOB	*Department 10*	*Department 20*	*Department 30*	*Total by Job*
ANALYST	*.00*	*6,000.00*	*.00*	*6,000.00*
CLERK	*1,300.00*	*1,900.00*	*950.00*	*4,150.00*
MANAGER	*2,450.00*	*2,975.00*	*2,850.00*	*8,275.00*
PRESIDENT	*5,000.00*	*.00*	*.00*	*5,000.00*
SALESMAN	*.00*	*.00*	*5,600.00*	*5,600.00*
	8,750.00	*10,875.00*	*9,400.00*	*29,025.00*

CONFIDENTIAL

This Kind Of Report Is Called A Matrix Report

Extracting Data From More Than One Table

To distinguish between the DEPTNO column in EMP and the one in DEPT, enter :

```
SQL> SELECT ENAME, DEPT.DEPTNO, DNAME
   FROM EMP, DEPT
  WHERE EMP.DEPTNO = DEPT.DEPTNO
  ORDER BY DEPT.DEPTNO
```

Using Table Aliases

Table names can also be very tedious when typing in repeatedly. Temporary labels (or aliases) can be used in the FROM clause. These temporary names are valid only for the current select statement. Table aliases should also be specified in the SELECT clause. This effectively `speeds` up the query, in that the statement contains very specific information.

```
SQL> SELECT E.ENAME, D.DEPTNO, D.DNAME
       FROM EMP E, DEPT D
       WHERE E.DEPTNO = D.DEPTNO
       ORDER BY D.DEPTNO;
```

Equi -Join

In order to ascertain, manually, which department any employee is in, we would compare the value in the employee's DEPTNO column with the same DEPTNO values in the DEPT table . The relationship between the EMP and DEPT table is an equi-join, in that values in the DEPTNO column on both tables are equal .

A join condition is specified in the WHERE clause:

SELECT	column (s)
FROM	tables
WHERE	join condition is ..

To join two tables EMP and DEPT, enter :

```
SQL>  SELECT ENAME, JOB, DNAME, DEPT.DEPTNO
 FROM EMP,DEPT
WHERE EMP.DEPTNO = DEPT.DEPTNO
```

ENAME	JOB	DNAME		DEPTNO
SMITH	CLERK	RESEARCH		20
ALLEN	SALESMAN	SALES	30	
WARD	SALESMAN	SALES		30
JONES	MANAGER	RESEARCH		20
MARTIN	SALESMAN	SALES	30	
BLAKE	MANAGER	SALES	30	
CLARK	MANAGER	ACCOUNTING		10

```
SCOTT           ANALYST     RESEARCH            20
KING            PRESIDENT   ACCOUNTING          10
TURNER          SALESMAN    SALES       30
ADAMS           CLERK       RESEARCH            20
JAMES           CLERK       SALES       30
FORD            ANALYST     RESEARCH            20
MILLER          CLERK       ACCOUNTING          10

14 rows selected.
```

You will see that every employee now has his respective department name displayed. Notice that join condition specifies table names prior to the column name. This is a requirement when column names are the same in both tables. It is necessary to `tell` ORACLE exactly which column you are referring to.

This requirement is also essential when referring to those columns which may be ambiguous in a SELECT or ORDER BY clause. Anyhow it is a good practice to to always specify table names prior to the column name.

Products

When a join condition is invalid or omitted completely the result is a PRODUCT, and all combinations of rows will be displayed.

```
SQL>SELECT A.ENAME, A.SAL,A.DEPTNO,B.DNAME
FROM DEPT B , EMP A
WHERE SAL < 1300

ENAME                SAL    DEPTNO DNAME
-------------------- ---------  --------- --------------
SMITH                800      20 ACCOUNTING
SMITH                800      20 RESEARCH
SMITH                800      20 SALES
SMITH                800      20 OPERATIONS
SMITH                800      20 هندسه
WARD                1250       30 ACCOUNTING
WARD                1250       30 RESEARCH
WARD                1250       30 SALES
WARD                1250       30 OPERATIONS
WARD                1250       30 هندسه
MARTIN               1250       30 ACCOUNTING
MARTIN               1250       30 RESEARCH
MARTIN               1250       30 SALES
MARTIN               1250       30 OPERATIONS
MARTIN               1250       30 هندسه
ADAMS                1100       20 ACCOUNTING
ADAMS                1100       20 RESEARCH
ADAMS                1100       20 SALES
ADAMS                1100       20 OPERATIONS
ADAMS                1100       20 هندسه
JAMES                950       30 ACCOUNTING

ENAME                SAL    DEPTNO DNAME
-------------------- ---------  --------- --------------
JAMES                950       30 RESEARCH
```

```
JAMES           950     30 SALES
JAMES           950     30 OPERATIONS
JAMES           950     30 هندسه

25 rows selected.
```

Therefore in the absence of a WHERE condition, each row in EMP is linked in turn to each row in DEPT.

Non-Equi-Join.

The relationship between the EMP and SALGRADE table is a non-equi-join, in that no column in EMP corresponds directly to a column in SALGRADE. The relationship is obtained using an operator other than equal (=). To evaluate an employee's grade, his/her salary must be between any one of the low and high salary ranges.

The BETWEEN operator is used to construct the condition, enter:

```
SQL> SELECT E.ENAME, E.SAL, S.GRADE
  FROM EMP E, SALGRADE S
 WHERE E.SAL BETWEEN S.LOSAL AND S.HISAL

ENAME                SAL      GRADE
-------------------- -------- --------
SMITH                 800     1
ADAMS                1100     1
JAMES                 950     1
WARD                 1250     2
MARTIN               1250     2
MILLER               1300     2
ALLEN                1600     3
TURNER               1500     3
JONES                2975     4
BLAKE                2850     4
```

Other operators such as <= and >= could be used, however between is the simplest. Remember the low value first, high value last when using BETWEEN . Again table aliases have been specified, not because of possible ambiguity, but for performance reasons. In order to join together all three tables, it is necessary to construct two join conditions. To join four tables a minimum of three joins would be required.

A simple rule is :

> The number of tables minus one = minimum number of join conditions.

This rule may not apply if your table has a concatenated primary key that uniquely identifies each row (primary keys are covered later in the manual).

Outer Joins

If a row does not satisfy a join condition, then the row will not appear in the query result. In fact, in the equi-join condition of EMP and DEPT, department 40 does not appear. This is because three are no employees in department 40.

The row(s) can be returned if an outer join operator is used in the join condition . The operator is a plus sign enclosed in parenthesis (+), and is placed in the `side` of the join (table) which is deficient is information . The operator has the effect one or more NULL rows, to which one or more rows from the non-deficient table can be joined. One NULL row is created for every additional row in the non-deficient table.

```
SQL> SELECT E.ENAME, D.DEPTNO, D.DNAME
  2  FROM EMP E, DEPT D
  3  WHERE E.DEPTNO (+) = D.DEPTNO;
```

ENAME	DEPTNO	DNAME
CLARK	10	ACCOUNTING
KING	10	ACCOUNTING
MILLER	10	ACCOUNTING
	12	هندسه
SMITH	20	RESEARCH
ADAMS	20	RESEARCH
FORD	20	RESEARCH
SCOTT	20	RESEARCH
JONES	20	RESEARCH
ALLEN	30	SALES
BLAKE	30	SALES
MARTIN	30	SALES
JAMES	30	SALES
TURNER	30	SALES
WARD	30	SALES
	40	OPERATIONS

16 rows selected.

Joining a Table to Itself

It is possible using table labels (aliases) to join a table to itself, as if it were two separate tables. This allows rows in a table to be joined to rows in the same table. The following query displays all employees who earn less than their managers.

```
SQL> SELECT E.ENAME , E.SAL, M.ENAME, M.SAL
 FROM EMP E , EMP M
 WHERE E.MGR = M.EMPNO
AND   E.SAL < M.SAL;
```

ENAME	SAL	ENAME	SAL
SMITH	800	FORD	3000
ALLEN	1600	BLAKE	2850
WARD	1250	BLAKE	2850
JONES	2975	KING	5000

MARTIN	*1250*	*BLAKE*	*2850*
BLAKE	*2850*	*KING*	*5000*
CLARK	*2450*	*KING*	*5000*
TURNER	*1500*	*BLAKE*	*2850*
ADAMS	*1100*	*SCOTT*	*3000*
JAMES	*950*	*BLAKE*	*2850*
MILLER	*1300*	*CLARK*	*2450*

11 rows selected.

Please note that the FORM clause refers to EMP twice, and therefore EMP has been given an alias in both cases - E and M. Assign meaning full table aliases, for example
E means employees and M means managers (it does help!).

The join clause in English reads: " where an employee's manager number is the same
as his manager's employee number".

Set Operators

Intersect, Union and Minus are covered in this section.

UNION, INTERSECT and MINUS are useful in constructing queries that refer to different tables. They combine the results of two or more select statements into one result. A query may therefore consist of two or more SQL statements linked by one
or more set operators. Set operators are often called Vertical Joins, because the join is not according to rows between two tables, but columns.

In the following three examples, the queries are the same, but the operator is different in each case yielding different query result.

UNION

To return all distinct rows retrieved by either of the queries, enter:

```
SQL> SELECT JOB FROM EMP
   WHERE DEPTNO = 10
   UNION
   SELECT JOB FROM EMP
   WHERE DEPTNO = 30;

JOB
---------
CLERK
MANAGER
PRESIDENT
SALESMAN
INTERSECT
```

INTERSECT

To return only rows retrieved by both of queries, enter :

```
SQL> SELECT JOB FROM EMP
  WHERE DEPTNO = 10
 INTERSECT
 SELECT JOB FROM EMP
 WHERE DEPTNO = 30;

JOB
---------
CLERK
MANAGER
```

MINUS

To return all rows retrieved by first query that are not in the second, enter:

```
SQL> SELECT JOB FROM EMP WHERE DEPTNO = 10
   MINUS
   SELECT JOB FROM EMP WHERE DEPTNO = 30;

JOB
---------
PRESIDENT
```

It is possible to construct queries with many set operators. If multiple set operators are used, the execution order for the SQL statements is from top to bottom. Brackets may be used to produce an alternative execution order.

Rules When Using Set Operators.

1. SELECT statements must select the same number of columns.
2. Corresponding Columns must be of the same datatype.
3. Duplicate rows are automatically eliminated, (DISTINCT cannot be used.)
4. Column names from the first query appear in the result.
5. ORDER BY clause appears at the end of statement.
6. ORDER BY column position only.
7. Set Operators can be used in Sub-queries, see Section 11.
8. Select statements are executed from top to bottom.
9. Multiple set operators are possible with parentheses if necessary to alter the sequence of execution.

Subqueries

Single Row Subqueries

To find the employee who earns the minimum salary in the company (minimum salary is an unknown quantity), two steps must be taken:

1. Find the minimum salary .

```
SQL> SELECT MIN(SAL) FROM EMP;

 MIN(SAL)
---------
     800
```

2. Find the employee who earns the minimum salary

```
SQL> SELECT ENAME,JOB,SAL FROM EMP
   WHERE SAL = (SELECT MIN(SAL) FROM EMP);

ENAME                JOB             SAL
-------------------------- ---------       ---------
SMITH                CLERK         800
```

Nested Subqueries

A subquery is a SELECT statement that is nested within another SELECT statement.

For example:

```
SELECT column 1, column 2, .....
FROM table
WHERE column =
                ( SELECT column
                FROM table
                WHERE condition )
```

The subquery is often referred to as a sub-select or inner select or inner select, it generally executes first and its output is used to complete the query condition for the main or outer query. Using subqueries allows a developer to build powerful commands out of simple ones. The nested subquery can be very useful when you need to select rows from a table with a condition that depends on the data in the itself.

How are nested subqueries processed ?

A select statement can be considered as a query block. The example above consists of two query blocks - the main query and the inner query .

The inner statement or query block is executed first, producing a query result: 800 The main query block is then processed and uses the value returned by the inner query to complete its search condition. In essence . the main query would kindly look like this:

```
SELECT ENAME, SAL, DEPTNO
FROM EMP
WHERE SAL = 800;
```

In the above example, the 800 is a single value. The subquery that returned the value of 800 is called a single row subquery. When a subquery. When a subquery returns only one row, a single row comparison or logical operator should be used. For example: = ,<,>,<=, etc.

To find all employees who have the same job as BLAKE we would enter:

```
SQL> SELECT ENAME, JOB FROM EMP
WHERE JOB = (SELECT JOB FROM EMP
WHERE ENAME='BLAKE');

ENAME                JOB
-------------------- ---------
JONES                MANAGER
BLAKE                MANAGER
CLARK                MANAGER
```

Subqueries That Return More Than One Row

The following query attempts to find the employees who earn who lowest salary in each department .

```
SQL>SELECT ENAME, SAL, DEPTNO FROM    EMP
WHERE SAL IN ( SELECT MIN ( SAL)
FROM EMP
GROUP BY DEPTNO);

ENAME                SAL                  DEPTNO
-------------------- ---------            ---------
SMITH                800                  20
JAMES                 950       30
MILLER               1300                 10
```

Notice that the inner query has a GROUP BY clause. This means that it may return more than one value. We therefore need to use a mufti-row comparison operator. In this case, the IN operator MUST be used because it expects a list of values.

The result obtained does not show the department in which the qualifying employees work. In addition, because we are only comparing salary values, the inner query could return a value simply because it matches the lowest salary for one of the

departments, not employee's own department. Therefore, the query should be rewritten in order to match the combination of employee's salary and department number with the minimum salary and department number.

```
SQL>SELECT ENAME, SAL, DEPTNO
   FROM EMP
   WHERE (SAL, DEPTNO) IN (SELECT MIN(SAL), DEPTNO FROM EMP
  GROUP BY DEPTNO)

ENAME                  SAL      DEPTNO
---------------------- --------- ---------
SMITH                  800      20
JAMES                  950      30
MILLER                 1300     10
```

The query above compares a pair of columns.

Notice that the columns on the LEFT of the search condition are in parenthesis and that each column is separated with a comma.

Columns listed in the SELECT clause of the subquery MUST be IN THE SAME ORDER as the bracketed column list on the WHERE clause of the outer query.

Columns returned by the inner query must also match in number and datatype the columns to which they are compared in the outer query .

When a subquery returns more than one row and a single row comparison operator is used, SQL*PLUS issues the following error message:

```
SQL> SELECT ENAME,SAL,DEPTNO FROM EMP
  WHERE SAL = (SELECT MIN(SAL) FROM EMP
 GROUP BY DEPTNO);

ERROR:
ORA-01427: single-row subquery returns more than one row


no rows selected

SQL> SELECT ENAME,JOB FROM EMP
   WHERE JOB = (SELECT JOB FROM EMP
   WHERE ENAME = 'SSSS');

no rows selected
```

ANY or ALL Operators

The ANY or ALL operators may be used for subqueries that return more than one row. They are used on the WHERE or HAVING clause in conjunction with the logical operators (=.!=, <,>,>=,<=).

ANY compares a value to EACH value retained by a subquery.

To display employees who earn more than the lowest salary in Department 30, enter:

```
SQL> SELECT ENAME, SAL, JOB, DEPTNO
  FROM EMP
  WHERE SAL> ANY (SELECT DISTINCT SAL FROM EMP
          WHERE DEPTNO = 30)
  ORDER BY SAL DESC

ENAME                SAL      JOB              DEPTNO
-------------------- ---------  ---------------  ---------
KING                 5000     PRESIDENT        10
SCOTT                3000     ANALYST          20
FORD                 3000     ANALYST          20
JONES                2975     MANAGER          20
BLAKE                2850     MANAGER          30
CLARK                2450     MANAGER          10
ALLEN                1600     SALESMAN         30
TURNER               1500     SALESMAN         30
MILLER               1300     CLERK            10
WARD                 1250     SALESMAN         30
MARTIN               1250     SALESMAN         30
ADAMS                1100     CLERK            20

12 rows selected.
```

The lowest salary in Department 30 is $950 (James`). The main query has returned employees who earn a salary that is greater than the lowest (minimum) salary in department 30. So '>ANY' means more than the minimum. `= ANY` is equivalent to IN.

When using ANY, the DISTINCT keyword is frequently used to prevent rows being selected several times.

ALL compares a value to EVERY value returned by a subquery.

The following query finds employees who earn more than every employee in Dept. 30

```
SQL> SELECT ENAME, SAL, JOB, DEPTNO FROM EMP
  WHERE SAL> ALL (SELECT DISTINCT SAL
  FROM EMP WHERE DEPTNO = 30)

ENAME                SAL      JOB              DEPTNO
-------------------- ---------  -------------    ---------
JONES                2975     MANAGER          20
SCOTT                3000     ANALYST          20
KING                 5000     PRESIDENT        10
FORD                 3000     ANALYST          20
```

The highest salary in Dept 30 is $2850(Blake's), so the query has returned those employees whose salary is higher than $2850. That is, greater than the highest(max.) salary for Dept 30, and consequently more than every salary in that department.

HAVING Clause with Nested Subqueries

Nested subqueries can also be used in the HAVING clause.

(Remember that WHERE refers to SINGLE rows and HAVING to GROUPS of rows specified in the GROUP BY clause).
For example, to display the Department (s) which have an average salary bill greater than Dept. 30, enter:

```
SQL> SELECT (SAL) FROM EMP
  HAVING AVG(SAL) > (SELECT AVG(SAL) FROM EMP
              WHERE DEPTNO = 30)
  GROUP BY DEPTNO;

 DEPTNO  AVG(SAL)
 ---------  ---------
    10        2916.6667
    20        2175
```

To construct a query which finds the job with the highest average salary, enter :

```
SQL> SELECT JOB, AVG(SAL) FROM EMP
GROUP BY JOB
 HAVING AVG(SAL) = (SELECT MAX(AVG(SAL)) FROM EMP
            GROUP BY JOB);

JOB                AVG(SAL)
----------------   ---------
PRESIDENT       5000
```

The inner query first finds the average salary for each different job group, and the MAX function picks the highest average salary. That value (500) is used in the HAVING clause. The GROUP BY clause in the main query is needed because the main query's SELECT list contains both an aggregate and non-aggregate column.

Ordering Data with Subqueries

You may NOT have an ORDER BY clause in a subquery.

The RULE remains that you can have ONLY ONE ORDER BY clause for a select statement and, if specified, it must be the last clause in the select command.

Nesting Subqueries

Subqueries may be nested (used within another subquery):

Display the name, sal for employees whose salary is greater than the highest salary in any SALES department .

```
SQL> SELECT ENAME, SAL FROM EMP
   WHERE SAL> (SELECT MAX(SAL) FROM EMP
                WHERE DEPTNO = (SELECT DEPTNO FROM DEPT
                                WHERE DNAME='SALES'));
```

ENAME	*SAL*
JONES	*2975*
SCOTT	*3000*
KING	*5000*
FORD	*3000*

Guidelines

- The inner query must be enclosed in parenthesis, and must be on the right hand side of the condition.

- The subquery may not have an ORDER BY clause.

- The ORDER BY clause appears at the end of the main select statement.

- Multiple columns on the select list of the inner query must be in the same order as the columns appearing on the condition clause of the main query. The dataType and number of columns listed must also correspond.

- Set operators may be used in a subquery.

- Subqueries are always executed from the most deeply nested to the least deeply nested, unless they are correlated subqueries, (discussed later).

- Logical and SQL operators may be used as well as ANY and ALL.

- Subqueries can :

  - Return one or more rows.

  - Return one or more columns

  - Use group by or group functions

  - Be used in multiple AND or OR predicates of the same outer query

  - Join Tables

  - Retrieve from a different table than the outer query

  - Appear in SELECT, UPDATE, DELETE, INSERT, CREATE TABLE statements

  - Correlate with an outer query

Correlated Subqueries

A Correlate subquery is a nested subquery which is executed once for each

` candidate row` considered by the main query and which on execution uses a value from a column in the outer query . This causes the correlated subquery to be processed in a different way to the ordinary Nested Subquery.

A Correlated subquery is identified by the use of an outer query's column in the inner query's predicate clause.

With a normal nested subquery, the inner select runs first and it executes once, returning values to be used by the main query. A Correlated Subquery, on the other hand, executes once for each row (candidate row) considered by the outer query . The inner query is driven by the outer query.

Steps to Execute a Correlated Subquery

1. Get candidate row (fetched by outer query).

2. Execute inner query using candidate row's value

3. Use value (s) resulting from inner query to qualify or disqualify candidate

4. Repeat until no candidate row remains

To find employees who earn a salary greater than the average salary for their department, enter :

```
SQL> SELECT ENAME, JOB, SAL ,DEPTNO FROM EMP E
    WHERE SAL > (SELECT AVG(SAL) FROM EMP WHERE DEPTNO = E.DEPTNO);

ENAME                JOB            SAL      DEPTNO
-------------------------- ---------         ---------  ---------
ALLEN                SALESMAN    1600       30
JONES                MANAGER     2975       20
BLAKE                 MANAGER    2850       30
SCOTT                ANALYST      3000       20
KING                 PRESIDENT   5000       10
FORD                 ANALYST      3000       20

6 rows selected.
```

Note that the alias is necessary only to avoid ambiguity in column names.

Let us analyze the above example using the EMP table:

The Main Query

1. Select first candidate row - Smith in department 20 earning 20 earning $ 800.

2. EMP in FROM clause has alias E which qualifies depth column referenced in inner query's WHERE clause.

3. WHERE clause compares 800 against value returned by inner query.

The Inner Query

4. Computes AVG(SAL) for employee's department.

5. WHERE department value is candidate's department (E.DEPTNO) - value passed into inner query from outer query's DEPTNO column.

6. AVG(SAL) for Smith's department - 20 - is $ 2175.

7. Candidate row does not meet condition, so discard.

8. Repeat from step 1 for next candidate row, ALLEN in department 30 earning $ 1600.

The selection of candidate rows continues with those meeting the condition appearing in the query result.

Remember, a Correlated subquery is signaled by a column name, a table name to table alias that refers to the value of a column in each candidate row of the outer select. Also the Correlated subquery executes repeatedly for each candidate row in the main query.

Update commands can contain correlated subqueries:

```
UPDATE EMP E
SET (SAL,COMM) = (SELECT AVG(SAL)*1.1 , AVG(COMM) FROM EMP
                  WHERE DEPTNO = E.DEPTNO),
HIREDATE = '11-JUN-81'
```

Operators

When you are nesting select statements the logical operators are all valid as well as ANY and ALL. In addition the EXISTS operator may be used.

EXISTS Operator

The EXISTS operator is frequently used with Correlated Subqueries. It tests whether a value is there (NOT EXISTS ensures that something is not there). If the value exists it returns TRUE, if does not exist FALSE is flagged.

To find employees who have at least one person reporting to them, enter:

```
SQL> SELECT EMPNO,ENAME,JOB FROM EMP E
  WHERE EXISTS (SELECT EMPNO FROM EMP
  WHERE EMP.MGR =E.EMPNO)
 ORDER BY EMPNO

  EMPNO ENAME                JOB
--------- -------------------------  ---------
   7566 JONES                MANAGER
   7698 BLAKE                MANAGER
   7782 CLARK                MANAGER
```

7788 SCOTT	*ANALYST*
7839 KING	*PRESIDENT*
7902 FORD	*ANALYST*

6 rows selected.

Why Use a Correlated Subquery ?

The Correlated subquery is one way of ` reading` every row in the table, and comparing values in each row against related data,. It is used whenever a subquery must return a different result or set of result for each candidate row considered by the
main query.

The inner select is normally executed once for each candidate row.