ORACLE(SQL)

INTRODUCTION TO RDBMS
DATABASE CONCEPTS

What is a Database ?

A Database Management System (DBMS) consists of a collection of interrelated data and a set of programmes to access that data. The collection of data, usually reffered to as the "DB" , contain information about one particular enterprise. The primary goal of a DBMS is to provide an environment, i.e., both convinient & efficient to use, in retrieving and storing database information.

Database systems are designed to manage large bodies of information. The management of data involves both the definition of structures for the storage information and the provision of mechanisms for the manipulation of Info. In addition, the database system, should provide for the safety of information stored, despite system crashes or attempts at unauthorised access. If the datat is to be stored among several users, the system must avoid possible anomalous results.

The importance of the Info in most organisations, and hence the value of the database has lead to the development of a large body of concepts and techniques for the efficient management of data.

DBMS offers the following servies:

1.DataDefinition: This is the method of defining and storing a set of data.
2.Data Maintenance: Takes care to see that each record is containing information about one particular item.
3. Data Manipulation: Allows user to Insert, Updata, Delete and Sort data in the database.
4. Data Display : This helps in viewing the data by the user.
5. Data Integrity : This ensures the accuracy of the data.

Data Abstraction

A database management system is a collection of interrelated files and a set of programs that allow users to access and modify these files. A major pupose of a database system is to provide users with an abstract view of the data. Tht is, the system hides certain details of how the data is stored and maintained. However in order for the system to be usable, data must be retrived efficiently. This concern has lead to the design of complex data structures for the reepresentation of data in the database. Since many database system users are not computer-trained. The complexity is hidden fromthem through several layers of abstraction in order to simplify their interaction with the system.

Evolution of RDBMS concept

A mathematician called E.F.T. CODD who was working with IBM doing research in statistics, applied the principles of relationships in statistics to data management and came up with 12 rules. Using mathematics he proved that if all these 12 rules are incorporated into database core technology, there would be a revolution in the speed of any DBM Systems while managing data; even when used on networkoperating systems.

A software companies took up the challenge and all products like Oracle, Ingres, Sybase, Unify, Inform have been trying to successfully implement all of CODD'S 12 laws into their DBMS core thechnology. Of all the mentioned above products ORACLE has implemented the maximum number of rules, fully of partially.

Codd's Rules

A relational database system uses only its relational capabilities to manage the information stored in its database.

1.**Information Representation**:

A information stored in a relational database is represented only by database item values, m which are stored in the tables that make up the database. Associations between data items are not logically represented in any other way, such as, by the use of pointers from one table to the other.

2.Logical Accessibility:

Every data item value stored in a relational database is accessible by stating the name of the table it is stored in, the name of colume under which it is stored and the value of the primary key that defines then row in which it is stored.

3.Representation of Null Values:

The database management system has a consistenet method for representing null values. For example, null values for numeric data must be distinct from zero or any other nuemeric value and for character data it must be different from a string of blanks or any other character value.

4.Catalog Facilities:

The logical description of a relational datanase os represented in the same manner as ordinary data. This is done so that the facilities of the relational database management system itself can be used to maintain as database description.

5.Data Language:

A relational database management system may support many types of languages for describing data and accessing the database. However there must be atleast One language that uses ordinary character strings to support the definition of data, the definition of views, the manipulation of data, constraints on data integrity, information concerning authorization and the boundaries for recovery of units.

6.View Updatability:

Any view that can be defined using combination of base table, that are theoritically updatable, is capable of being updated by the relational database management system.

7.Insert, Update and Delete:

Any operand that describe the results of single retrieval operatin is capable of being applied of an insert, update or delete opertion as well.

8. Physical data independence:

Charges made to physical storage representation or access methods do not require changes to be made to application program.

9.Logical data independence;

Changes made to tables, that do not modify any data stored in that table, do not require changes to be application program.

10.Integrity Constraint:

Constraints that apply to entity integrity and referential integrity are specificable by the data language implemented by the database management system and not by the statements code into the applications program.

11.Database Distribution:

The data language implemented by the relational database management system supports the ability to distribute the data base without requiring changes to be made to application program. This facility must be provided in the data language, whether or not the database management system itself supports distributed databases.

12.Non Subversion:

If the relational database management system supports facilities that allow application programs to operate on the table a row at a time, an application program using this type of database access is prevented from by passing entity integrity or referential integrity constrainsts that are defined for the database.

13.Characterristics of a relational datamodel:

The relational database model eliminated all parent child relationship and instead represented all data in database as simple row/column tables of data value.

- A relation is similar a table with rows/column of data values. The rows of table are referred to a 'Attributes' Several tables of equal length placed one below the other create a table.
- Each table is an independet entity and there is no physical relationship between tables.
- Most data management systems based on the relational model have a built-in support for query language like ANSI SQL or QBE (Query By Example). These queries are simple English Constructs that allow adhoc data manipulation from table.
- Relational model of data management is based on set theory Built-in query language is designed in the RDBMS, so that it can manipulate sets of data ccone or tuples.
- The user interface used with relational models is non-porcedural because only what needs to be done is specified and not how it has to be done Using any of the other methods you have not only to specify what needs to be done but how it has to be done as well.

Objects in RDBMS :

There are 3 types of objects in any RDBMS.

1.Structural Objects : We have to use only SQL statements to create these objects. The following objects come under structural objects.

1. Tables
2. Views
3. Synonyms
4. Indexes
5. Clusters
6. Database links
7. Swquences
8. Snapshots

2. Procedural Objects : These objects are created using PL/AQL and Embedded SQL.

1. Stored Procedures
2. Stored functions
3. Packages
4. Database triggers
5. Snapshot logs

3. Non-visual Objects: These are created using SQL statements only.

1. Table space

STRUCTURED QUERY LANGAUAGE

Introduction

SQL :

To interact with a relational database, you must use an access language that embodies the principles of relational algebra SQL (Structure Query Language) is a set of commands used to access data from the database. The American national Standards Institute (ANSI) has adopted SQL as standard language for RDBMS.

SQL *Plus :

SQL *PLUS is a program, that provides

- As environment for SQL commands to by keyed directly or indirectly from a command file.
- Formatting commands which are used for customized output.
- Set options used to change SQL *PLUS environment.

SQL Commands:

The SQL commands can be broadly divided into the following categories.

- Data Definition Language Commands (DDL)
- Data Manipulation Language commands (DML)
- Data Control Language commands (DCL)

Data Definition Language :
The DDL commands are used for structuring the dtabase. The commands CREATE, ALTER AND DROP are used to create new objects, alter the structure of existing objects or completlely remove objects from the system.

Data Manipulation Language:
DML commands are used to extract records from a database. SELECT statements are used to query the database. The UPDATE, INSERT and DELETE statements alter emitting database rows, place new records into a database or remove recordsfrom the database respectively.

DataControl Language:
DCL statements such as GRANT, REVOKE, COMMIT and ROLLBACK control access to databases and affirm or revoke database transactions. DCL statements also control who has access to what tables, who can log on to the oracle system, and what privileges each user has for various database tables. The COMMIT and ROLLBACK commands permit groups of database transactions to be made permanent or to be nullified.

SQL *PLUS DATATYPES :

DATATYPE	DESCRIPTION
CHAR (SIZE)	Values of this data type are fixed length character strings of maximum length 255 characters. The default size is 1.
VARCHAR/ VARCHAR2(SIZE)	Values of this data types are variable length character strings of maximum length 2000.
DATE	Valid dtes range from january 1 4712BC to December 31, 4712 AD. Default format is DD-MM-YY as in 01-Feb-87.
LONG	Character data of variable length up to 65, 55 characters. You Are LONG column per table.
NUMBER	This data type is used to store numbers (fixed or floating point) of Virtually any magnitde may be stored upt to 38 digits of precision.
RAW(SIZE)	Raw binary data, size bytes long, Maximum size is 255 bytes. Values must be inserted as character stings in hexadecimal Notation.

CREATING A TABLE
The create table command is used to define a new table syntax : CREATE TABLE tablename

Column 1 datatype (size) [NOT NULL],
Column 2 datatype (size) [NOT NULL];

Example :

CREATE TABLE EMP (EMPNO NUMBER (5) NOT NULL,
ENAME CHAR (10), JOB CHAR (9), DOJ DATE,
SAL NUMBER (7,2), COMM NUMBER (7,2));

NOT NULL option can be specified to avoid the user from inserting a record into the table with that particular field as empty, i.e the record is not inserted until & unless a value has been entered into that field.

The table name should be unique.

ALTERING A TABLE: ALTER TABLE command is used to change the definition of a table.

SYNTAX:

ALTER TABLE table name

ADD	(column_space[column_constrant])
	ENABLE clause
MODIFY	DISABLE clause
DROP	options

ADD clause:

ADD keyword is used to add a column and/or constraint to an omitting table.

To add a column COMM to the EMP table;

ALTER TABLE EMP ADD (COMM NUMBER (7.,2));

To add a table contraint to an existing table, which specifies that the monthly salary must not exceed $ 5000.

MODIFY clause:

MODIFY keyword is used to modify the definition of an existing column.

ALTER TABLE name MODIFY (column type [NULL]);

To change the width of JOB to 15 chanracters;

ALTER TABLE EMP MODIFY (ENAME CHAR (25));

DROP clause:

DROP clause to used to remove a constraint from a table.

Syntax:

ALTER TABLE table name

DROP CONSTRAINT constraint name [CASCADE]
DROP PRIMARY KEY
UNIQUE (column1, column2, …….)

To drop primary key constraint

ALTER TABLE EMP DROP PRIMARY KEY.

The cascade option of the DROP clause causes any dependent constraint also to be dropped. For example.

ALTER TABLE DEPT DROP PRIMARY KEY CASCADE.

Would also cause the foreign key constraint on EMP.DEPTNO jto be dropped.

DROPPING TABLE:

To remove the definition of an ORACLE table. Drop TABLE command is used DROP TABLE table_name [CASCADE CONSTRAINTS]
Dropping a table losers all the data init and all the indexes associated with it. The CASCADE CONSTRAINTS option will also remove dependent refrential integrity constraints.

Example: -

DROP TABLE EMP;

Note:

- Any VIEWS or SYNONYMS that were bared on the deleted table will but they become invalid.
- Any pending transactions are committed.

DATA MAINPULATION LANGUAGE

INSERT NEW ROWS INTO A TABLE:

The INSERT command is used to add rows to a table.

SYNTAX:

INSERT.INTO table name [(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.

Insert a new Department:

```
INSERT INTO DEPT (DEPTNO, DNAME, LOC)
VALUES (50, 'SALES', 'MADRAS');
```

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

```
INSERT INTO DEPT (DEPTNO.Loc) values (50,'MADRAS');
```

If the department name is not known, a NULL could be specified.

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

NOTE: CHARACTER and DATE values must be enclosed in single quotes. The INSERT command adds only one row at a time the table.

USING SUBSTITUION VARIABLES TO INSERT ROWS:

It is possible to speed up input using substitution variables. When the command in run, values are prompted for every time.

```
INSERT INTO DEPT (DEPTNO,DNAME,LOC)
VALUES (&D-NUMBER, '&D-NAME', & 'LOCATION')
```

INSERTING DATE AND TIME;

The general format for date is DD-MON-YY. With this format the century defaults to the 220th century (19nn). The date also contains time information, which if not specified defaults to midnight (00:00:00). If a DATE needs to be entered in another century and a specified-time is also required, TO_DATEfunction is used.

```
INSERT INTO EMP
(EMPNO, ENAME, JOB, HIREDATE, DEPTNO)
VALUES (328,'JAMES', 'MANAGER', TO-DATE ('21/0/2001 9:30', 'DD/MM/YYYY HH:MI'), 30)';
```

COPYING ROWS FROM ANOTHER TABLE:

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

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 D10 table:

```
INSERT    INTO D10 (EMPNO, ENAME, JOB, HIREDATE)
          SELECT EMNO, ENAME, JOB, HIREDATE
          From EMP WHERE DEPTNO=10;
```

Note: The keyword 'VALUES' is not used here

UPDATING ROWS:

The UPDATE statement allows to chage values in rows in table.

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

To UPDATE jones row,

```
UPDATE EMP SET JOB = 'MANAGER', SAL = SAL *1.2
WHERE ENAME = 'JONES';
```

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

It ispossible to use nested sub queries and correlated sub queries in the UPDATE statement.

The changes in the COMMISSION table can be applied to the EMP table using a corelated sub query and a nested sub query as shown below;

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

DELETING ROWS:

THE DELETE command in used to delete rows from a table.

To delete all the rows from EMP table.

```
DELETE * FROM EMP;
```

To delete all information about department 10 from the EMP table

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

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 transaction 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.

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

- COMMIT/ROLLBACK
- DDL command is used
- Certain errors (such as dead lock)
- Log off (EXIT from SQL * Plus)
- Machine failure

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

After one transaction ends, the next executable SQL statement will automatically start the next transaction.

COMMIT:

Syntax : COMMIT [WORK]

- Markes changes in the current transation permanent.
- Erases all savepoints in the transaction.
- Ends the transaction
- Releases the transaction's lock.
- The keyword work is optional.
- You should explicitly and transaction is application progress using the COMMIT (or ROLLBACK) statement. If you do not explicitly commit the transaction and the program terminates abnormally, the last uncommitted transaction will be rolle back.
- Implicit (automatic)commits occur in the following situations.

befor a DDL command
after a DDL command
at normal disconnect from the data base.

DDL statements always cause commits at the time they are executed. If you enter a DDL statement causes a commit to occur prior to its own execution, ending the current transaction. Then, if the DDL statement executes successfully, it too is committed.

SAVEPOINT:

Synatax : SAVEPOINT savepoint_name;

- Can be used to divide a transaction into smaller portions.
- Savepoint allows you to arbitrarily "hold" your work at any point of time, with the option later committing that work or undoing all or a portion of it. Thus, for a long transaction, you can save parts of it as you proceed, finally committing or rolling back. If you make an error, you need not resubmit every statement.
- If you create a second savepoint with the same name as an earlier savepoint, the earlier savepoint is deleted.
- The maximum number of savepoint per user process default to This limit can be changed.

ROLLBACK:

SYNTAX: ROLLBACK [WORK] TO [SAVEPOINT] savepoint-name.

- The ROLLBACK statement is used to undo work.
- The keyword is optional. Rolling back to a SAVEPOINT is also optional.
- If you use ROLLBACK without a SAVEPOINT clause, it:

ends the transaction
undoes all changes in current transaction
releases the transaction's lock.

AUTO COMMIT:

COMMIT/ROLLBACK may be issued manually or automatically by using the AUTOCOMMIT option of the SET command. The AUTOCOMMIT option contracts 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 when a DROP,
ALTER or CREATE command is SQL *	issued, or if you exit from Plus.

INTEGRITY CONSTRAINTS:

Integrity constraints allow relational database systems to easily expose the standard data integrity rules that are part of the relational model. The integrity rules are declared as part of the table's definition.

DOMAIN INTEGRITY:

Domain integrity makes sure that a database doesn't contains values which are not legal. A row cannot be in a table unless each of the row's column values is a member in the domain of the corresponding column, for example there can not be a row in the CUSTOMER table with the character 'A' in 'CUST_ID' because the 'CUST_ID' column holds numbers and 'A ' is not a member of the domain of numbers. This integrity in take care by declaring the datatype while creating the table.

For example:

CREATE TABLE CUSTOMER (CUST_ID NUMBER(2), CUST_NAME CHAR(20), DOJ DATE);

The table created will allow only numbers for CUST_ID, any character for the field value of CUST_ID , any valid dates for the field values of DOJ.

ENTITY INTEGRITY:

Entity integrity means that every row in a table must be unique. If a table has entity integrity, you can uniquely identify every row in jthe table because there are no duplicate rows. To ensure entity integrity, a developer designates a column or a set of columns in a table as its primary key, the primary key columns should not contain NULL values. A table can have only one primary key.

For example:

1. CREATE TABLE customer (cust-id NUMBER(2) PRIMARY KEY, cust_name CHAR (20));
2. CREATE TABLE ATTE (MONTH NUMBER(2), PRESENT NUMBER(2), CL NUMBER(2), ADVANCE NUMBER(3), PRIMARY KEY (MONTH, EMP_ID));

REFERENTIAL INTEGRITY: -

Referential integrity, also known as relation integrity defines the relationships among different columns and tables in a relational database. It is called referential integriy because the values in one column or a set or columns refer to or more match the values in a related column or set of columns.

For example:

When entering the PURCHASE ORDER, the ITEM_ID entered should be existing in the ITEM MASTER.

CREATE TABLE PURC_ORDER (ITEM_ID NUMBER(2), QTY NUMBER(3), PRICE NUMBER(5,2, FOREIGN KEY (ITEM_ID) REFERENCES ITEM_MASTER);

The above table will not accept the item code which is not existing in the ITEM-MASTER. The field which is referred as Foreign key in jthe PURC_ORDER, whould be declared as PRIMARY KEY or UNIQUE KEY in the ITEM_MASTER. The dependent column or set of columns in a foreign key. The referenced column or set columns in a parent key which must be a primary key or unique key. I.e., any field which is FOREIGN KEY should be declared as PRIMARY KEY in the table in which it is referred from.

UNIQUE AND CHECK INTEGRITY:

A table can have only one primary key. In many cases developers need to eliminate duplicate values from other columns as well. For this a developer can designate a non-primary key column or set of columns as an alternate key or a unique key. A table cannot have duplicate values in a unique key, just as it can't in a primary key.

A developer might find it necessary to narrow a column's domain even more. For example, onlyu valid state code abbreviations 'AP','KAR','TN','MP', should be in the STATE column of the CUSTOMER table. This integrity can be taken care by using CHECK option.

CREATE TABLE CUSTOMER (CUST_ID NUMBER(2), PRIMARY KEY, CUST_NAME CHAR(25), STATE CHARE(S), PHONE CHAR(6), UNIQUE(PHONE), CHECK(STATE IN ('AP','KAR','TN','MP')));

ON DELETE CASCADE OPTION:

The delete cascade referential action of the constraint is to cascade and delete a parent row to all dependent child rows.

For example, if we have 2 tables ORDERS, ITEMS Where ORDERS table contains data about the orders placed i.e., the order number, order date, supplier code and ITEMS table containing the information about the different items that have been placed in the order i.e., the order number, item code, item qty and item price, then while creating the items table the command will be given as follows.

CREATE TABLE ITEMS (ORDER_ID NUMBER(3), ITEM_CODE NUMBER(2), QTY NUMBER(3), PRICE NUMBER(5,2) NPRIMARY KEY (ORDER_ID, ITEM_CODE), FOREIGN KEY(ORDER_ID) REFERENCES ORDERS ON DELETE CASCADE, FOREIGN KEY (ITEM_CODE) REFERENCES ITEM_MASTER).

If the user deletes records from parent row i.e., from ORDERS table then the corresponding records from ITEMS table will be deleted automatically. This action is called restrict referential action.

THE BASIC QUERY STATEMENT:

THE SELECT statement is used to retrive information from the database. 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 employee members, employee names and job in the EMP table we enter:

SELECT DEPTNO, ENAME, MGR FROM EMP;

EMPNO	ENAME	JOB
7902	FORD	ANALYST
7934	MILLER	CLERK
7839	KING	PRESDENT
7844	TURNER	SALES MAN
7900	JAMES	CLERK

To list all the columns;

SELECT * FROM EMP;

OTHER ITEMS IN THE SELECT CLAUSE

- Attribute expressions
- Column aliases
- Concatenated columns
- Concatenated columns
- Literals

All these options allow user to query data and manipulate it to get the required output, 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

If the arithmetic expression contains more than one operator, the priority is for *,/first and then +,-. The operation is from left to right if there are several operators with the same priority.

In the following example, the multiplication (250*12) is evaluated first, then the salary value is added to the result.

```
SELECT ENAME, SAL + 250 * 12
FROM EMP;
```

Parenthesis has to be used to specify the orders in which the operators have to be executed. If for example, addition is required before multiplication, then the query will be;

```
SELECT ENAME, (SAL +250)*12
FROM EMP;
```

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 cryptic or meaningless. The columns heading can be changed by using an alias. A column alias gives a column an alternative heading on output. By default, alias headings will be forced to uppercase and cannot contain blank spaces, unless the alias is enclosed in double quotes.

To display the column heading ANNSAL for annual salary instead of SAL * 12;

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

THE CONCATENATION OPERATOR: The concatenation operator(II) allows columns to be linked to other columns, arethmetic expressions or constant values to create a character expression columns on eithere side of the operator are combined to make one single column.

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

SELECT EMPNO II ENAME EMPLOYEE

FROM EMP;

EMPLOYEE

7902 FORD
7934 MILLER
7839 KING
7844 TURNER
7900 JAMES

LITERALS: A literal is any character expression, number included 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. Date & character literals must be enclosed within single quotes(‘); Numeric literals do not need single quotes.

The following statement contains literals selected with concatenation and a column alias:

```
SELECT      EMPNO II '__' II ENAME EMPLOYEE.
            'WORKS IN DEPARTMENT' DEPTNO
```

```
FROM    EMP;
```

HANDLING NULL VALUES: If a row lacks a data value for a partcular item, that value is said to be null. A null value is a value wich is either unavoidable, unassigned, unknown or inapplicable. A null value is not the same as zero, because zero is a number. Null values take up one byte of internal 'storage' space.

If any column values in an expression are null, the rsult is null. In order to achieve a result for all employees, in the given example, 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.

```
SELECT      ENAME, SAL * 12+NVL(COMM,O) ANNUAL_SAL

FROM        EMP;
```

ENAME	ANNUAL SAL
ADAMS	13200
JAMES	11400
FORD	36000
SCOTT	36000
KING	60000
TURNER	18000

NVL expects two arguments: 1). an expression
2). a non-null value

The NVL function can be used to convert a null number, date or even character string to another number, data or character string, as long as the data types match.

```
NVL (DATECOLUMN, '08-JUL-86')
NVL (NUMBERCOLUMN, 9)
NVL (CHARCOLUMN, 'STRING')
```

DISTINCT CLAUSE: Unless indicated otherwise. SQL * PLUS displays the results of query without eliminating duplicate entries.

To eliminate duplicate values in the result. DISTINCT qualifier is included in the SELECT statement. DISTINCT qualifier can only be referred once and must immediately follow the SELECT command word.

Example:

```
SELECT      DISTINCT    DEPTNO
FROM            EMP;
```

DEPTNO
10
20
30

Multiple columns may be specified after the DISTINCT qualifier and the DISTINCT affects all selected columns.

To display a list of all the different cdombination of job and department numbers:

```
SELECT      DISTINCT    DEPTNO.     JOB
FROM            EMP;
```

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

ORDER BY CLAUSE

The order of rows returned in a query result is normally undefined The IRDER BY clause may be used to sort rows. If used, ORDER BY must be the last clause in the SELECT statement To order by a column. It is not necessary to have selete it.

To sort by ENAME, enter.

```
SELECT      ENAME, JOB, SAL * 72
FROM        EMP
ORDER BY    ENAME;
```

ENAME	JOB	SAL * 12
ADAMS	CLERK	13200
ALLEN	SALESMAN	19200
BLAME	MANAGER	34200
FORD	ANALYST	36000
JAMES	CLERK	11400

The default sort order is ASCENDING

- Numeric values lowest first
- Date values earlist first
- Character values alphabetically

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 ENAME column, enter:

```
SELECT      ENAME, JOB, SAL * 12
FROM        EMP
ORDER BY    ENAME DESC;
```

It is possible to sort using more than one column.In the ORDER BY clause, specify the columns, sperated by commas. If any or all are to be reversed, specify DESC after any or each column. To order by two column and display is reverse order of salary, enter:

```
SELECT      DEPTNO, JOB, ENAME
FROM        EMP
```

```
ORDER BY    DEPTNO, SAL DESC;
```

WHERE CLAUSE: The WHERE clause corresponds to the Restriction operator of Relational algebre. It contains a condition that rows must meet in order to be displayed.

Syntax:

```
SELECT      columns
FROM        table
WHERE       certain conditions are met
```

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

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

Comparision operators used in the WHERE clause can be divided into two categories, Logical &SQL.

LOGICAL OPERATORS:

These oerators will test the following conditions.

OPERATOR	MEANINGS
=	equal to
>	greater than
>=	greater than or equal to
<	less than
<=	less than or equal to

CHARACTER STRINGS & DATES IN THE WHERE CLAUSE.

ORACLE columns may be Character, Number or Date character strings and dates in the WHERE clause must be enclosed in single quotation marks and character strings must match case with the column value.

To list the names, job and departments of all clerks;

```
SELECT      ENAME, JOB, DEPTNO

FROM        EMP

WHERE       JOB='CLERK'
```

ENAME	JOB	DEPTNO
ADAMS	CLERK	20
SMITH	CLERK	20
JAMES	CLERK	30
MILLER	CLERK	10

To find all department names with department numbers greater than 20;

```
SELECT    DNAME, DEPTNO
FROM      DEPT
WHERE     DEPTNO>20
```

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

For example, to find those employees whose commission is greater than their salary.

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

ENAME	SAL	COMM
ALLEN	1600.00	1800.00

SQL OPERATORS:

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

OPERATOR	DESCRIPTION
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

BETWEEN OPERATOR: Tests for values between, and inclusive of, low and high range.
To see thaose employees whose salary is between 1000 and 2000

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

Note: The lower limit must be specified first and the values specified are inclusive.

IN OPERATOR: Tests for values in a specified list

To find all employees who have one of the three specified salaries.

```
SELECT    EMPNO, ENAME, SAL
FROM      EMP
WHERE     SAL IN (800, 1000, 3000)
```

EMPNO	ENAME	SAL
7876	ADAMS	1000
7788	SCOTT	3000
7376	SMITH	800
7902	FORD	3000

LIKE OPERATOR: sometimes we may not know the exact value to search for Using the LIKE operator, it is prossible to select rows that match a character pattern. The charactr pattern matching operator may be refered to as a 'Wild-card' search. Two symbols can be used to constant the search string.

SYMBOL	REPRESENTS
%	any sequence of zero or more characters
__	any single character

To list all employees whose name starts with an s, enter.

```
SELECT    ENAME
FROM      EMP
WHERE     ENAME LIKE 'S%'
```

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

```
SELECT    ENAME
FROM      EMP
WHERE     ENAME LIKE '- - - - - - - '
```

IS NULL OPERATOR: The IS NULL operator specifically tests for values that are NULL.
To find all employees who have no commission:

```
SELECT    ENAME, COMM
FROM      EMP
WHERE     COMM IS NULL;
```

NEGATING EXPRESSIONS:

OPERATOR	DESCRIPTION
NOT BETWEEN	not between two given values
NOT IN	not in the given list of values
NOT LIKE	not like string
IS NOT NULL	is not a null value

To find employees whose salary is not between a range:

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

To find those employees whose name does not atart with S:

```
SELECT    ENAME, EMPNO
FROM      EMP
WHERE     JOB NOT LIKE 'S%'
```

To find all employees who have a commission;

```
SELECT      ENAME, COMM
FROM        EMP
WHERE       COMM IS NOT NULL;
```

Note: - If a NULL value is used in a comparision, then the comparison operator should be either IS or IS NOT NULL. If these operators are not used and NULL values are compared, the result is always FALSE.

For example, COMM!=NULL is always FALSE. The result is false because a NULL clause may not be either equal or unequal to any other value.

QUERYING DATA WITH MULTIPLE CONDITIONS: The AND and OR operators may be used to make compound logical expressions. The AND ope4ator expects both the conditions to be 'true'; where as OR operator expects any one of the conditions to be 'true'.

To find all clerks who earn between 1000 and 2000.

```
SELECT      ENAME, JOB, SAL
FROM        EMP
WHERE       SALARY BETWEEN 1000 AND 2000
AND         JOB='CLERK';
```

ENAME	JOB	SAL
ADAMS	CLERK	1,800.00
MILLER	CLERK	1,300.00

To find all employees who are either clerks or all employees who earn between 1000 and 2000;

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

ENAME	JOB	SAL
SMITH	CLERK	800.00
ALLEN	SALESMAN	1,600.00
WARD	SALESMAN	1,250.00
MARTIN	SALESMAN	1,500.00
ADAMS	CLERK	1,100.00

You may combine AND and OR in the same logical expression AND has a higher precedence than OR.

Since AND has a higher precedence then OR, the following SQL statement returns all managers with salaries ove r$1500, and all salesmen.

```
SELECT      EMPNO, ENAME, JOB, SAL, DEPTNO
FROM        EMP
WHERE       SAL > 1500
```

```
AND     JOB='MANAGER'
OR      JOB='SALESMAN';
```

To select all managers and salesmen with salaries over $1500.

```
SELECT  EMPNO, ENAME, JOB, SAL, DEPTNO
FROM    EMP
WHERE   SAL > 1500
AND     (JOB='NANAGER'
OR      JOB=SALESMAN');
```

The parenthesis specify the order in which the operators should be evaluated. In the above SQL statement the OR operator is evaluated before the AND operator.
SELECT SUMMARY:

The following cluases have been covered in the SELECT command.

```
SELECT      [ DISTINCT ] {*, column [ alias ],……}
FROM        TABLE
WHERE       conditions
ORDER BY    { column, expr} [ ASC I DESC ];
```

DESCRIPTION:

SELECT	Selects at least one column
Alias	may be used for columns on select list only
*	denotes all columns
DISTINCT	can be used to eliminate duplicates
FROM Table	denotes the table where the columns Originate
WHERE	restricts query to rows that meet a condition It may contain column value expressions and literals.
AND I OR	may be used in a WHERE clause to construct more complex conditions AND takes priority over OR.
( )	can be used to force priority
ORDER BY	always appears last. Sort order may be specified on one or more columns.
ASC I DESC	ascending order is the default

SQL * PLUS EDITING COMMANDS:

Because SQL *PLUS does not store SQ: *Plus commands in the buffer, you edit a SQL *Plus command entered directly to the command prompt by using [Backspace] or by re-entering the command.

You can use a number of SQL *Plus commands to edit the SQL command or PL/SQL block currently stored in the buffer. Or, you can use a host operating system editor to edit the buffer contents.

Table shows several SQL * Plus commands that allow you to examine or change the command in the buffer without re-entering the command.

COMMAND	ABBREVIATION	PURPOSE
APPEND text	A text	add text at the end of a line
CHANGE/text	C/old/new	change old to new in a line
CHANGE/text	C/text	delete text from a line
CLEAR BUFFER	CL BUFF	delete all lines
DEL	(none)	delete a line
INPUT	I	add one or more lines
INPUT text	I text	add a line consisting of text
LIST	L	lists all lines in the SQL buffer
LIST n	Ln or n	list one line
LIST *	L*	list the current line
LIST LAST	L LAST	list the last line
LIST m n	L m n	list a range of lines (m to n)

TABLE: SQL *Plus EDITING COMMANDS:

You will find these commands useful if you must type a command or wish to modify a command you have entered.

Listing the Buffer contents

Any editing command other than LIST affects only a single line in the buffer, this line is called the current line. It is marked with an asterisk when you list the current command or block.

Suppose you want to list the current command. Use the LIST command as shown below.

SQL >LIST

1. SELECT EMPNO, ENAME, HOB, SAL
2. *FROM EMP WHERE SAL < 2500.

Notice that the semicolon you entered at the end of the SELECT command is not listed. This semicolon is necessary to mark the end of the command when you enter it, but SQL * Plus does not store it in the SQL buffer. This makes editing more convenient, since it means you can add a new line to the end of the buffer without removing a semicolon from the line that was previously the last.

Editing the Current Line;

The SQL *Plus CHANGE command allows you to edit the current line. Various actions detemine which line is the current line:

List a given line to make it the current line.

If you get an error message, the line containing the error automatically becomes the current line.

Example; Making an Error in command Entry

Suppose you try to select the DEPTNO column but mistakenly enter it as DEPTNO: Enter the following command, purposely misspelling DEPTNO in the first line:

SQL > SELECT DEPTNO, ENAME, SAL

2.FROM EMP
3.WHERE DEPTNO=10;

You see this message on your screen:

SELECT DPTNO, ENAME, SAL

ERROR at line 1:
ORA-0904: invalid column name

Examine the error message; it indicates an invalid column name in line 1 of the query, the asterisk shows the point of error-the mistyped column DPTNO>

Instead of re-entering the entire command, you can correct the mistake by editing the command in the buffer, the line containing the error is now the current line, use the CHANGE command to correct the mistake.

To change DPTNO to DEPTNO, change the line with the CHANGE command:
SQL > CHANGE/DPTNO/DEPTNO

The corrected line appears on your screen:

I * SELECT DEPTNO, ENAME, SAL

Now that you have corrected the error, you can use the RUN command to run the command again:

SQL > RUN

Lists the command, and then runs it:

1. SELECT DEPTNO, ENAME, SAL.
2. FROM EMP
3. WHERE DEPTNO = 10

DEPTNO	ENAME	SALARY
10	CLARK	$ 2,450
10	KING	$ 5,000
10	MILLER	$ 1,300

Ading a New Line

To insert a new line after the current line, use the INPUT command.

Example: Adding a Line

Suppose you want to add a fourth line to the SQL command you modified in Example. Since line 3 is already the current line, enter INPUT (which may be abbreviated to I) and press [Return] SQL * Plus prompts you for the new line;

SQL > INPUT
4

Enter the new line. Then press [Return] SQL *Plus prompts you again for a new line:

4 ORDER BY SAL

Press [Return] again to indicate that you, will not enter any more lines, and then use RUN to verify and rerun the query.

Appending Text to a Line

To add text to the end of a line in the buffer, use the APPEND command.

1. Use the LIST command (or just the line number) to list the line you want to change.
2. Enter APPEND followed by the text you want to add. If the text you want to add begins with a blank, separtate the word APPEND from the first character of the text by two blanks: one to separate APPEND from the text, and one to go into the buffer with the text.

Example: Appending Text to a Line

To append a space and the clause DESC to line 4 of the current query, first list line 4:

```
SQL > LIST 4
4 * ORDER BY SAL
```

Next, enter the following command (be sure to type 2 spaces bet's APPEND and DESC);

```
SQL > APPEND DESC
4 * ORDER BY SAL DESC
```

Use RUN to verify and rerun the query.

Deleting a Line

To delete aline in the buffer, use the DEL command]

1. Use the LIST command (or just the line number) to list the line you want to delete.
2. Enter DEL

DEL makes the following line of the buffer (if ny the current line. Thus, you can deleter several consecutive lines by making the first of them the current line, then entering DEL several times.

Editing Commands with a system Editor

You can run your first operating system's default text editor without leaving SQL * Plus by entering the EDIT command.

SQL > EDIT

CHANGING COLUMN HEADINGS:

When displaying column headings, you can either use the default heading or you can change it using the COLUMN command. The sections below describe how the default headings are derived and how you can alter them with the COLUMN command

COLUMN column_name HEADING column_heading

Example: Changing a Column heading

To produce a report from EMP with new headings specified for DEPTNO. ENAME, and SQL, enter the following commands.

```
SQL > COLUMN DEPTNO HEADING Department
SQL > COLUMN SAL HEADING Salary
SQL > COLUMN COMM HEADING Commission
SQL > SELECT DEPTNO, ENAME, SAL, COMM
2 FROM EMP
3 WHERE JOB='SALESMAN';
```

SQL *Plus displays the following output

Department	Employees	Salary	Commission
30	ALLEN	1600	300
30	WARD	1250	500
30	MARTIN	1250	1400
30	TURNER	1500	0

FORMATTING COMMANDS:

FOR[MAT] format

Specifies the display format of the, column: the format specification must be a text constant such as A 10 or $9,999 – not a variable.

NUMBER FORMATS:

The elements of a number format model are:

Element	Picture	Description	Example(s)
9	999999	Determines the display width by the number of digits entered. Does not display leading zeroes.	1234
0	099999	Displays leading zeroes	001234
$	$999999	Prefixes a dollar sign to	$1234
B	B9999.99	Displays a zero value as blank	1234.00

MI	999999MI	Displays "-" after a negative Brackets.	1234-
PR	999999PR	displays a negative value in Angle brackets	<1234>
Comma	999,999	Displays a comma in the Position indicated	1,234
Period	999999,99	Aligns the decimal point In the position indicated.	1234.00
V	999V99	Multiplies value by 10n Where n is the number of "9'S" after the "V".	1234.00
EEEE	99.999EEEE	Displays in scientific Notation(format must contain Exactly four "E's").	1,234E+03

TABLE: NUMBER FORMATS

Examples:

To make the ENAME column 20 characters wide and display EMPLOYEE NAME on two lines at the top

SQL > COLUMN ENAME FORMAT A20 HEADING 'EMPLOYEE NAME'

To format the SAL column so that it shows millions of dollars, rounds to cents, uses commas to separate thousands, and displays $o.oo when a value is zero.

SQL > COLUMN SAL FORMAT $9,999,990.99

CHANGING THE SQL *PLUS ENVIRONMENT:

Establishes an aspect of the SQL *Plus environment for your current session.

SET OPTION:

Syntax: SET system_variable value

Where system_variable value represents a system variable followed by a value as shown below:

ARRAY(SIZE){20iN}	Set the number of rows that may be fetched from the database At one time. Valid values are 1 to 5000
ECHO{OFF / ON}	Causes commands in a command file being executed to be Displayed

LINE[ESIZE]{80/N}	Specifies the number of characters per line.
PAGES[IZE]{14/N}	Waits for carriage return befor displaying in the page
SERVEROUT[PUT] {OFF/ON}[SIZE n]	Sets the server output ON or OFF
VER[IFY] {OFF/ON}	Display query after substitution of variables, if used

TTITLE & BTTITLE Commands:

Places and formats a specified title at the top of each report page, or lists the current TTITLE definition.

Syntax:

TTI[TLE][printspec[text/veriable] …] [OFF/ON]

Where printspec represents one or more of the following clauses used to place and format the text.

Syntax:

BTI[TLE] [printspec[text/variable]…] [off/on]

Example:

SQL > TTITLE 'ABC COMPANY LIMITED HYDERABAD'

Similarly BTITLE can be used for specifying the bottom title.

SQL > BTITLE CENTER 'COMPANY CONFIDENTIAL'

Options

COL n
S[KIP][n]
TABn
LE[FT]
CE[NTER]
R[IGHT]
BOLD

Note: If you do not enter a printspec clause before the first occurrence of text, TTITLE left justifies the text, Enter TITLE with co clauses to list th3 current TITILE definition.

While using TITLE the system date and the page number will print. When using the options of TITLE command, the system date and the page number will not be printed.

SHOW Command

SHO[W] option

Where option represents one of the following terms or clauses:

BTI[TLE]	the BTITLE definition
REL[EASE]	the release number of this version of oracle
SPOO[L]	show whether the output is currently being spooled
USER	Your user ID
WRAP	the current truncate setting

BREAK command

The BREAK command suppresses duplicate values by default in the column or expression you name, thus, to suppress the duplicate values in a column specified in and ORDER NBY clause, use the BREAK command in its simplest form:

BREAK ON {column_name/ROW/PAGE/REPORT}[options]

Example: Suppressing duplicate Values in a Break Column

To suppress the display of duplicate department numbers in the query results shown above, enter the following commands:

Options

SKIP n	skip n number of lines at break
PAGE	skip to next page at break

```
SQL > BREAK ON DEPTNO

SQL > SELECT DEPTNO, ENAME, SAL

2 FROM EMP
3 WHERE SAL < 2500
4 ORDER BY DEPTNO;
```

SQL *Plus displays the following output:

DEPTNO	ENAME	SAL
10	CLARK	2450
	MILLER	1300
20	SMITH	800
	ADAMS	1100
30	ALLEN	1600
	JAMES	950

You can insert blank lines or begin a new page each time the value changes in the break column, to inser n blank lines, use the BREAK command in the following form:

BREAK ON break_column SKIP N

To skip the number of lines defined to be a page, use the command in this form:

BREAK ON break_column SKIP PAGE

Example: Inserting Space when a Break column's Value Changes

To place one blank line between departments

SQL > BREAK ON DEPTNO SKIP 1

Inserting Space after Every Row

You may wish to insert blank lines or a blank page after every row, to skip n lines after every row, use BREAK in the following form:

You can remove the current break definition by entering the CLEAR command with the BREAKS clause:

CLEAR BREAKS

COMPUTE command:

Calculate and prints summary lines, using various standard computations, on subsets of selected rows, Or, lists all COMPUTE definitions.

Syntax:

COMP[UTE] [function….. OF {expr/column/alias}….. ON {expr/column/alias/REPORT/ROW}}

Function	Computers	Applies to Datatypes
AVG	Average of non-null values	NUMBER
COU[NT]	Count of non-null values.	All types MAXI[IMUM]
	Maximum value	NUMBER, CHAR, VARCHAR2 (VARCHAR)
NUM(BER)	count of row	all types
STD	Standard deviation of non-null values	NUMBER
SUM	Sum of non-null values	NUMBER
VAR(IANCE)	Variable of non-null	NUMBER

OF{expr/column/alias}…..

Specifies the columns) or expression9s) you wish to use in the computation. The COMPUTER command will be ignored if the columns used in the OF clause are not specified in the SELECT command.

ON{expr/column/alias/FEPORT/ROW}

Specifies the column of the table to be used as break COMPUTE prints the computed value and restarts the computation when the ON expression's value changes.

Example:

To compute the average and miximum salary for the accounting and sales departments:

```
SQL >  BREAK ON DNAME SKIP 1

SQL > COMPUTE AVG MAX OF SAL ON DNAME
SQL > SELECT DNAME, ENAME, SAL
2 FROM DEPT, EMP
3 WHERE DEPT. DEPTNO=EMP.DEPTNO
4 AND DNAME IN ('ACCOUNTING','SALES')
5 ORDER BY DNAME;
```

The following output results:

```
DNAME          ENAME      SAL

ACCOUNTING     CLARK      2450
               KIND       5000
               MILLER     1300

**********                ———
avg                       2916.66667

maximum                   5000
SALES          ALLEN      1600
          WARD            1250
          MARTIN          1250
          TURNER          1500
          JAMES            950
          BLAKE           2850
**********                ———

avg                       1566.66667
maximum                   2850
```

SPOOL:

Stores query results in an operating system file and, opetionally, sends the file to a default printer Also lists the current spooling status.

Syntax:

SQL[OL][file name[.ext]/OFF/OUT]

File_name [.ext] Represents the name of the file to which you wish to spool. SPOL followed by file_name begins spooling displayed output to the named file. If you do not specify an extension, SPOOL uses a default extension(LST or LIS on most systems).

OFF	Stops spooling
OUT	Stops spooling and sends the file to your host computer's standard (default) printer.

Enter SPOOL with no clauses to list the current spooling status.

Example:

To record your displayed output in a file named emplist using the default file extension, enter.

SQL > SPOOL emplist

Show the name of the current spool

SQL > SPOOL
Currently spooling to emplist list

HELP

Syntax: HELP [topic]

Enter HELP without topic to get help on the help system

SQL > HELP COMP

Displays help on COMPUTE followed by help on comparison operators.

GROUP FUNCTIONS: Functions which apply to a group of records & returns a single vaue for a group of records.

They may be referred to by various names: summarizing functions, or aggregate functions, or set functions, or group by functions.

The addition of DISTINCT within the group function makes the group function consider only distinct values of the expression; the addition of ALL makes it consider all values including duplicates. If neither is specified, ALL is the default.

Functions	Description
AVG[DISTINCT/ALL]N]	Average value of n, ignoring null values
COUNT([DISTINCT/ALL]expr)	Number of rows where expr evaluates to something other than NULL
COUNT(*)	Counts the number of rows including duplicates and those with Nulls.
MAX([DISTINCT/ALL]expr)	Maximum value of expr
MIN([DISTINCT/ALL]expr)	Minimum value of expr
STDDEV([DISTINCT/ALL]n)	Standard deviation of n, ignoring null values
SUM([DISTINCT/ALL]n)	Sum values of n, ignoring null values
VARIANCE([DISTINCT/ALL]n)	Variance of n, ignoring null values

Note: Any of these functions may be applied to numeric values character and data values can only be used with MIN, MAX and COUNT. Null values are ignored in computing SUM,AVG,STDDEV and VARIANCE.

COUNT Function: The COUNT function is used to count the number of entities returned by a query. NULL entities are ignored.

Syntax:

```
SELECT      COUNT (entity)
FROM              table
WHERE       conditions are true
```

Example: Count the number of employees who have job title (that is, the number of rows where JOB is not NULL)

```
SQL > SELECT      COUNT(JOB)  FROM EMP;

COUNT(J)B)

   12
```

count the number of different job title in the emp table.

```
SQL > SELECT COUNT(DISTINCT JOB) FROM EMP;

COUNT(DISTINCT JOB)

        5
```

Count the number of people who work in department 10.

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

COUNT(*)

  12
```

MIN/MAX Functions:

Compute the maximum and maximum salary in the company.

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

MIN(SAL)          MAX(SAL)

  800               9000
```

To find earliest date on which someone was hired, and the latest date on which someone was hired.

```
SQL >   SELECT MIN(HIREDATE), MAX(HIREDATE) FROM EMP;

MIN(HIREDATE)      MAX(HIREDATE)

  15-FEB-78          18-JUL-85
```

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.

Syntax:

```
SELECT      grouped-by-columns or functions
FROM        table, table,…….
[WHERE      conditions are true]
GROUP BY    column, column……
```

To calculate the average salary for each different job:

SELECT JOB, AVG(SAL) FROM EMP GROUP BY JOB:

JOB	AVG(SAL)
ANALYST	3000
CLERK	1025.5
MANAGER	2702.33
PRESIDENT	5000
SALESMAN	1300

To show the average salary for each JOB including managers:

SELECT JOB, AVG(SAL) FROM EMP WHERE JOB!=MANAGER GROUP BY JOB,

Nested Group Functions:

One level of grouping

SELECT DEPTNO, SUM(SAL), AVG(SAL) FROM EMP GROUP BY DEPTNO;

DEPTNO	SUM(SAL)	AVG(SAL)
10	8750	2916.67
20	10575	2.175
30	9400	1566.67

Two levels of grouping

SELECT AVG(SUM(SAL)), SUM(AVG(SAL)) FROM EMP GROUP BY DEPTNO;

AVG(SUM(SAL))	SUM(AVG(SAL))
9675	6658.33
8564	6152.00

GROUPING ON More than one column:

Count the number of people who hold each type of job in each department,

SELECT DEPTNO, JOB, COUNT(*) FROM EMP GROUP BY DEPTNO, JOB;

WHERE and GROUP BY clauses Together:

The WHERE clause can be used before grouping, to select the rows upon while grouping will be performed.

To determine the average salary for analysts and salesmen and determine the number of analysts or salesmen in each department.

SELECT DEPTNO, AVG(SAL), COUNT(*) FROM EMP WHERE JOB IN

('ANALYST','SALESMAN') GROUP BY DEPTNO;

Note: If you include functions in a SELECT command, you may not select individual results as well UNLESS the individual column appears in the GROUP BY clause.

In other words, group functions cannot the retrieved with individual items that are not included in the GROUP BY clause.

Example:

SELECT DEPTNO, MIN(SAL) FROM EMP;

The above statement is illegal and produces an error mentioning that DEPTNO is not a single value for the whole items.

The correct statement is

SELECT DEPTNO, MIN9(SAL) FROM EMP GROUP BY DEPTNO;

DEPTNO	MIN(SAL)
10	1300
20	800
30	950

DEPTNO, in the above SELECT statement, is no longer an individual item: it is the name of a group.

If more than one column with individual values is entered in the SELECT statement, you must group all the indivudual columns.

HAVING Clause:

The HAVING clause specifies which groups should be returned. It acts on the GROUP BY clause.

The conditions in the HAVING clause must involve only GROUP BY items or aggregate functions. Functions which are not selected may be referenced in the HAVING clause.

Syntax:

SELECT	group-by-columns or functions
FROM	table

```
[WHERE      conditions are true]
GROUP BY    column……..
HAVING      desired group characteristics
```

To determine which departments have more than two people holding a particular job.

```
SELECT      DEPTNO, JOB, COUNT(*)
FROM        EMP
GROUP BY    DEPTNO, JOB HAVING COUNT(*) > 3
```

DEPTNO	JOB	COUNT(*)
30	SALESMAN	4

WHERE and HAVING together: Better performance is activated by using the WHERE clause to eliminate rows before they are grouped.

To find all departments that have atleast two salesmen.

```
SELECT      DEPTNO
FROM        EMP
WHERE       JOB='SALESMAN'
            GROUPBYDEPTNO
HAVING      COUNT(*)>=2;
```

The order of clauses in the SELECT statement:

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

EXTRACTING DATA FROM MORE THAN ONE TABLE

JOINS: A join is used when a SQL query requires data from more than one table onj the database.

Rows in one table may be joined to rows in another according to common values existing in correspondence columns. There are two main types of join condition:

1. Equi-join
2. Non-equi-join

Equi-join: An equi-join links rows from tables based on the equality of a common attribute, such as a department number which occurs as DEPTNO in both the EMP and DEPT tables.

Syntax:

```
SELECT Columns from tables named in jthe from clause
FROM   table1, table2,……
WHERE TABLE1.Column=table2.column
```

Example: For each employee, show the name of the department and its loction.

SQL > SELECT ENAME, JOB, EMP, DEPTNO, DNAME, LOC FROM EMP, DEPT
WHERE EMP.DEPTNO=DEPT.DEP.TNO'

Or

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

Equi-joins with Row Selection conditions

The WHERE Clause may be used to specify row selection conditions are well as join columns.

Example: List the name, job, department, number, department name, and location all clerks.

SQL > SELECT ENAME, JOB, DEPT.*FROM EMP, DEPT WHERE EMP.DEPTNO=DEPT.DEPTNO AND JOB='CLERK';

Join Guidelines:

- The tables to be joined are specified in the FROM clause
- The WHERE clause specifies how to join or merge the tables together, as well as "search criteria" as seen with single table queries.
- Columns from either table may be names in the SELECT clause
- Columns which have the same name in multiple tables naed in the FROM clause must be uniquely identified by specifying TABLE NAME.COLUMNAME table. * is a shorthand notation for all columns of a joined table.
- If the columns are uniquely named across the tables, it is not necessary to specify table name column name in the WHERE or SELECT clause.
- As may tables as desired may be joined together.
- The "maching criteria" for the tables is called the join predicate or join criteria.
- Columns specified in join conditions should be indexed
- More than one pair of columns may be used to specify the join condition between any two tables.

When n tables are joined, it is necessary to have of least n-1 two-table join conditions to avoid the Cartesian product (four table join requires specification of join criteria for three pairs. Of tables)

Cartesian Joins: If you omit the join clause, a Cartesian join is performed. A Cartesian product matches every row of one table to every row of the other table.

Example:

SQL > SELECT ENAME, LOC FROM DEPT, EMP
WHERE JOB='CLERK';

Outer Joins: The outer join operator is a plus sign enclosed in parentheses(+). It forces a row containing null values to be generated to match every value of the second table for which there would normally be no match.

Syntax:

SELECT Columns
FROM table1, table2
WHERE table1.column(+)=table2.column

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

The missing rows 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 on the side of the join(table) Which is deficient in information. The operator has the effect of creating 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.

```
SELECT    E.NAME,D.DEPTNO,D.DNAME
FROM      EMP E, DEPT D
WHERE     E.DEPTNO(+)=D.DEPTNO
AND       D.DEPTNO IN (30,40);
```

ENAME	DEPTNO	DNAME
AMIT	30	SALES
BANU	30	SALES
RAVI	30	SALES
GOPI	30	SALES
VAMSHI	30	SAKES
KIRAN	40	OPERATIONS

Outer join Guidelines:

- To include rows in a join result even when they have no match in a joined table, use an outer join.
- An outer join causes SQL to supply to “dummy” row for rows not meeting the join condition.
- No data is actually added or altered in the table; the dummy rows exist only for the purposes of the outer join.
- Use the notation(+) after the table/column combination in the WHERE clause that needs the dummy rows (for example, WHERE EMP>DEPTNO(+)=DEPT>DEPTNO).
- If multiple columns are required to match for the join, all or none of the columns in a table’s join predicates should have the (+).
- Only one of the tables in a join relation can be outer joined (hat is, you cannot put the (+) on both the tables).
- A table can only be outer joined to one another table.
- Rows that do not have a match on the join condition but which are returned becaues of the outer join may be located by searching for rows with the NULL condition.

Self Joins:

A self join is used to match and retrieve rows that have a matching value in different columns of the same table.

- A table can be joined with itself as though it were two separate tables.
- Self-join is useful to join one row in a table to another in the same table.
- As with any other join, the join is on columns that contains the same type of information.
- The table must be giben an alias to synchronize which columns are coming from which table.

The following query displays all employees who earn less than their managers.

```
SELECT      E.NAME,EMP_NAME,E.SAL,EMP_SAL,
            M.ENAME,MGR_NAME,M.SAL,MGR_SAL
            EMP E, EMP M
FROM        EMP E, EMP M
WHERE       E.MGR=M.EMPNO
AND         E.SAL<M.SAL;
```

NON-EQUI_JOINS:

Equi-joins are based on the equality of values in the joined tables. Non equi-joins can be performed using the operators.

!= < <= >= BETWEEN

Example: The relationship between the EMP and SALGRADE tables 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(=).

```
SELECT      E.NAME,E.SAL,S.GRADE
FROM        EMP E, SALGRADE S
WHERE       E.SAL BETWEENS.LOSAL AND S.HISAL;
```

RULES FOR JOINING TABLES:

In order to join together three tables, it is necessary to construct two join conditions, to join four tables, a minumum of three joins would be required.

The 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.

Note: You can specify join conditions and other (non-join) conditions together.

SUBQUERIES

A subquery is a SELECT statement that is rested within another SELECT statement and which returns immediate results.The results of one query can be dynamically substituted into the WHERE clause of another query.

```
SELECT      Columns, column2
FROM        table
WHERE       column=(SELECT      column
            FROM                table
            WHERE               condition)
```

Single row subqueries: To find the employee who earns the maximum salary in the company (maximum salary is an unknown quantity), two steps are involved:

1. Find the maximum salary

2. Find the employee who earns the maximum salary

We can combine the two queries as follows:

```
SELECT    ENAME,JOB,SAL
FROM      EMP
WHERE     SAL=(SELECT MAX(SAL) FROM EMP);
```

The above example consists of two query blocks is executed first', producing a query result. The main query is processed and uses the value returned by the inner query to complete its search condition. When a subquery returns only one row, a single row comparision or logical operator should be used. For example: +,<,>,=etc.,

Subqueries that return more than one row: The following query attempts to find the employees who earn the lowest salary in each department.

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

ENAME	SAL	DEPTNO
SMITH	950	20
JAMES	1050	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 multirow comparison operator IN 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 department, not necessity the employees own department. Therefore, the query should be rewrites in order to match the combination of employees salary and department number with the minimum salary and department number;

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

Note: 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 that columns to which they; are compared in the outer query.

Example:

```
WHERE (numcolumn, charcolumn)=
          (SELECT datacolumn, numcoluumn, charcolumn…….) is illegal.
```

SOME/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 is conjunction with the logical operator (=,!=, <,>, <=)

Any (or its synonym SOME) compares a value to each value returned by a subquery. The IN operator is equivalent to "=ANY"

To display employees who earn more than the lowest salary in department 30.

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

Here ">ANY" means more than the minumum.
When using SOME/any, the DISTINCT keyword is frequently used to prevent rows from being selected several times.

ALL Operator: The ALL operator forces the specified relation for the column of a WHERE clause to be true for all values returned by the subquery. NOT IN is equivalent to "I=ALL"

Find employees who earn more than every employee in department 30

```
SELECT      ENAME, SAL, JOB, DEPTNO
FROM        EMP
WHERE       SAL>ALL(SELECT DISTINCT SAL
            FROM    EMP WHERE DEPTNO=20)
ORDER BY    SAL DESC;
```

HAVING CLAUSE WITH NESTED SUBQUERIES:- Noted subqueries can also be used in the HAVING clause. We have to remember that WHERE refers to single row and HAVING to groups of rows specified in the GROUP BY clause.

To find the job with the highest average salary:

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

ORDERING DATA WITH SUBQUERIES: You may NOT have an ORDER BY clause in a subquery. You can have only one ORDER BY clause for a SELECT STATEMENT andm if specified, it must be the last clause in the SELECT command.

NESTING SUBQUERIES: Sub queries may be used within another subquery. The limit on the levels of nesting for a subquery is 255.

To display the name, job and hire date for eomployees whose salary is greater than the highest salary in any SALES department.

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

The subquery may not have 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.

CORRELATED SUBQUERIES: A correlated 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 columnn in the outer query.

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 considered 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. the values)resulting from inner query to qualify or disqualify candidate.
4. Repeat until no candidate row, remains.

```
SELECT      EMPNO,ENAME,SAL,DEPTNO FROM EMP E
WHERE       SAL>(SELECT AVG9SAL) FROM EMP
                WHERE DEPTNO=E.DEPTNO)
```

Update commands can contain correclated subqueries:

```
UPDATE      EMP E
SET (SAL,COM)       =(SELECT AVG(SAL)*1.1,AVG(COMM)
                    FROM EMP WHERE DEPTNO =
                    E.DEPTNO
                    AND HIREDATE='11-JUN-86');
```

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.

EXIST OPERATOR: The EXISTS operator is frequently used with correlated subqueries to check the existence of rows in the inner query. To find employees who have atleast one person reporting to them.

To find employees who have atleast one person reporting to them

```
SELECT      EMPNO,EMPNAME,JOB,DEPTNO FROM EMP E
WHERE       EXISTS(SELECT EMPNO FROM EMP
                WHERE EMP,MGR=E.EMPNO)
```

NOT EXISTS is more reliable if the subquery returns any NULL values since a NOT-IN condition evaluates to false when NULLS are included in the compared list.

```
SELECT      ENAME,JOB FROM EMP
WHERE       EMPNO NOT IN (SELECT MGR FROM EMP).
```

No rows are returned by the above query, since the MGR column contains a NULL value.

The correct query is.

```
SELECT      ENAME, JOB FROM EMP
WHERE       NOT EXISTS(SELECT MGR FROM EMP WHERE MGR=E,EMPNO);
```

OPERATORS: The set operators 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 coulmns. It is possible to construct queries with many set operations. If multiple set operators are used, the execution order for the SQL statements is from top to bottom, Parenthesis may be used to change the execution order.

UNION and UNION ALL:

To teturn All distinct rows retrieved by either of the queries:

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

To return all rows (including duplicates) retrieved by either of the queries

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

INTERESECT:-

To return only rows retrieved by both of the queries:

```
SELECT      JOB  FROM EMP WHERE DEPTNO=10
INTERSECT
SELECT      JOB  FROM EMP WHERE DEPTNO =30
```

MINUS:-

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

```
SELECT      JOB  FROM EMP WHERE DEPTNO=10
MINUS
SELECT      JOB  FROM EMP WHERE DEPTNO=30
```

ORDER BY:- ORDER BY can be used only once in a query. It used, the ORDER BY clause must be placed at the end of the query. Also, because you may select different columns in each SELECT you cannot name the columns in the ORDER BY clause. Instead, columns in the ORDER BY must be referred to by their relative positions in the SELECT list.

```
SELECT      EMPNO,ENAME,SAL FROM EMP
UNION
SELECT      ID, NAME,SALARY FROM EMP_HISTORY
ORDER BY 2;
```

Notice that on the ORDER BY clause a numeral (2) is used to represent the position of the ENAME column on the SELECT list.

VIEWS

- A view is like a 'window' through which data on tables can be viewed or changed.
- A view is stored as a SELECT statement only. It is virtual table that is a table that does not physically exist in its own right, but appears to the user as if it exists.
- A view has no data of its own. It manipulates data in the underlying base table.

Views are useful for the following reasons:

- resulting access to the database
- allowing uses to make simple queries to retrieve the results from co,plicated queries.
- One view can be used to transparently retrieve data from several tables. Also views allow the same data to be seen by different user in different ways.

Simple views derive data from only one table and contains no functions or GROUP of data where as complex views derive data from many tables and contains functions and GROUPS of data.

Create Views Syntax:

```
CREATE [OR REPLACE] [FORCE] VIEW view-name
        [ (column1, column2…….)]

AS SELECT statement
        [ WITH CHECK OPTION CONSTRAINT constraint name]
```

column1, column2, etc. are the names to be given to columns in the view and must correspond to the items in the select list.

To create a simple view called DEPT10, containing employee details of department 10;

```
CREATE VIEW DEPT10 AS

SELECT EMPNO, ENAME, SAL FROM EMP WHERE DEPT=10;
```

After the view is created, it can then be used like any table.

To create a complex view called DEPT_SUMMARY, containing group functions and data from two tables.

```
CREATE VIEW
DEPT_SUMMARY(NAME,MINSAL,MAXSAL,AVGSAL)
AS SELECT DNAME,MIN(SAL),MAX(SAL),AVG(SAL) FROM EMP, DEPT
WHERE EMP.DECTNO=DEPT.DEPTNO
GROUP BY DNAME;
```

Note that alternative column names have been specified for the view. A view cannot contain an ORDER BY clause.

If a column alias is used in the query, a view column alias is not required for example:

```
CREATE    VIEW   DEPT20
AS SELECT     ENAME,SAL*12 ANNSAL FROM EMP
WHERE    DEPTNO=20;
```

OR REPLACE Option: This option allows a view to be created even if one exists with this name already, thus replacing the old version of the view for its owner.

FORCE Option: This option creates the view even if the base table doesn't exist, or there are insufficient table privileges. However, the table must exist before the view may be used.

Drop View Command: The command removes the view definition from the database. Rows and columns are not affected since they are stored on the tables from which the view was derived. Views or other application based on a dropped view become invalid.

Syntax:

```
DROP   VIEW viewname;
```

TABLE PRIVILEGES: You own each table, view, sequence and synonym that you create. Unless you want to share an object with other ORACLE users, only you or any DBA can access it.

To give another user access to one of the database objects, GRANT command is used.

```
GRANT          privileges ON object TO user;
```

The table below shows the privileges that can be granted on tables or views.

PRIVILEGE	OBJECT
SELECT	data in a table or view
INSERT	rows in a table or view
UPDATE	rows or specific columns in table or view
DELETE	rows from a table or view
ALTER	column definitions in a table
INDEX	index to a table
REFERENCES	refer to a table named within a table or column constraint

ALL

Note that ALTER,INDEX and SELECT privileges apply to SWQUENCES.

To grant SMITH the privilege to SELECT from your table DEPT:

```
GRANT          SELECT ON DEPT TO SMITH;
```

To grant update privilege on specific columns to SMITH;

```
GRANT UPDATE (DNAME,LOC)
ON DEPT TO SIMTH;
```

To grant several privileges at once, enter all the privileges separated by commas. Similarly, to grant privileges to more than one user, enter the user names separated by commas.

To grant INSERT and UPDATE privileges on DEPT to both SMITH and MILLER;

```
GRANT INSERT,UPDATE ON DEPT
TO SMITH, MILLER;
```

To grant all privileges on DEPT TO ADAMS,

```
GRABT ALL ON DEPT TO ADAMS;
```

Passing on privileges that you have been granted: when you grant an access privilege, the user who receives the grant normally does not receive authority to pass the privilege on to others. To give a user authority to pass privileges on, use the clause WITH GRANT OPTION.

To grant the SELECT privilege on EMP to JAMES, with authority to grant that privilege to others;

```
GRANT SELECT ON EMP TO JAMES WITH GRANT OPTION;
```

Public privilege: Allows the owner of a table to grant access to all users with a single command.

```
GRANT SELECT ON EMP TO PUBLIC;
```

Sequence: To grant access on your sequence M.SEQ to ADAMS.

```
GRANT SELECT ON M_SEQ TO ADAMS;
```

The Revoke Command: To withdraw a privilege you have granted, use the REVOKE command. When you use REVOKE, the privileges you specify are revoked from the users you name, and from any other users the whom they have granted those privileges.

To revoke all privileges on DEPT from ADAMS;

```
REVOKE ALL ON DEPT FROM ADAMS;
```

To find out which users have privileges on your tables, views or sequences, query the data dictionary views USERS_TAB_GRANTS, or USERS_COL_GRANTS.

Creating a synonym for a Table, View or Sequences:

Synonyms are used for reasons of security and covenience, including:

- to reference a table, sequence or view without specifying the owner of the object.
- to provide another name for the table

For performance reasons, it is not advisable to use synonyms when referring tables in applications.

To refer to a table owned by another user, you need to prefix the table name with the name of the user who created it followed by a period(.).

To refer to a table owned by another user, you need to prefix the table name with the name of the user who created it followed by a period(.).

To refer to the EMP table owned by SCOTT;

```
SELECT * FROM SCOTT. EMP;
```

Alternatively, you may create a synonym (another name) for the table or view.

To refer to SCOTT table EMP as just 'EMP'.

```
CREATE SYNONYM EMP FROM SCOTT. EMP;
```

Now when you query Scott's EMP table, you simply enter;

```
SELECT * FROM EMP;
```

Only DBA cam create PUBLIC synonyms that all users can access.

```
CREATE PUBLIC SYNONYAM synonym-name
FOR [OWNER] object-name;
```

A synonyam can be removed by entering;

DROP [PUBLIC] SYNONYM synonym-name;

THE SEQUENCE GENERATOR: In order to generate sequence numbers automatically, you must define a SEQUENCE using the CREATE SEQUENCE statement.

```
CREATE SEQUENCE [user] sequence-name
    [increment by n]
    [state with n]
    [maxvalues n/nomaxvalue]
    [minvalue n/nominvalues]
```

All clauses are optional and are explained below

User	owner of sequence, Defaults to the user submitting CREATE SEQUENCE command
Sequence-name	name of sequence and should follow SQL object naming conventions.
INCREMENT BY	determines the interval between sequence numbers you can enter any Non-zero integer 9either positive or negative value). The default value is 1.
START WITH	specifies the first sequence number to be created. Default is 1 for Assending sequences and MAXVALUE for descending sequence.
MIN VALUE	minimum value sequence will generate (lower NOMINVALUE bound).
MAXVALUE	highest value sequence will produce. NOMAXVALUE provides upper Bound for sequences. Default is 1 for descending sequences. Default Is 1 for descending sequences and 10e27-1 for ascending sequences.

To produce a sequence you must have resource privilege, the following command creates a sequence for a DEPTNO coulmn of the DEPT tabls

CREATE SEQUENCE dept-seq INCREMENT BY 10 START WITH 10 MAXVALUE 1000;

After a sequence has been created it can be used to generate unique sequence numbers;

The pseudo-columb NEXTVAL is used to extract successive sequence numbers from a specified sequence. When you reference NEXTVAL a new sequence number is generated.

SELECT dep-seq, NEXTVAL FROM SYS<DUAL:

NEXTVAL:
10

If you reexecute the above SQL statement the value increases by 10. If you reference NEXTVAL multiple times for a single SQL statement each reference will return the same value.

NEXTVAL is most useful in DML commands, for example, when you are populating table you can use a specified sequence to produce unique values for the primary key column of a table, the following example uses the dept-seq sequence to add the first unique primary key value to the DEPT table.

INSERT INTO DEPT VALUES (dept-seq. NEXT VAL, 'MKTG','HYDERABAD');

Using Sequence Number with CURVEL: To refer to a sequence number that has just been generated (the current sequence number) use the pseudeo-column CURRVAL. CURRVAL tells you the last value returned number in the current user session before you can reference CURRVAL> sequences are treated in a similar way to tables and can be altered or dropped, the owner of a sequence can grant other users either ALTER or SELECT privilege on the sequence, and the WITH GRANT OPTION IS also valid.

Altering a Sequence:

The ALTER SEQUENCE command is used to modify an existing sequence.

ALTER SEQUENCE [user,] sequence-name
[INCREMENT BY n]
MAXVALUE n/NOMAXVALUE]
[MINVALUE n/NOMINVALUE]

For example to set a new MAXVALUE for the dept-seq sequence;
ALTER SEQUENCE dept-seq maxvalue 20000;

Removing a sequence: use the DROP SEQUENCE command to remove a sequence definition from the data dictinary.The syntax is

DROP SEQUENCE dept-seq:

Listing sequences: All sequence definitions are sotred in a sequence table in the data dictinary, to see the sequences to which you have access, query the data dictinary views USER,SEQUENCES or ALL.SEQUENCES.

INDEXES: Indexes can be created to speed up the retrieved of rows from table and enforce uniquencess on values in a column of a table. Once created, indexes are used automatically by ORACLE for accessing data. However indexes have to be created judiciously, because at time index affects the performance by increasing the DML overhead.

Types of Index:-

Type	Description
UNIQUE	ensures that values in specified columns are unique.
NON INIQUE	ensures fastest possible query results (default)
SINGLECOLUMN	Only one column exists in the index
CONCATENATED	Upto 16 columns can be specified in the index, for either Performance or uniqueness check purposes.

Creating an Index: Oracle indexes can be created online by means of the CREATE INDEX command.

CREATE [UNIQUE] INDEX index-name
ON table (column1(,<column2>]….)

To create an index called-ENAME which can be used to speed up queries in ENAME:

CREATE INDEX-ENAME ON EMP(ENAME);

Unique indexes are created automatically as a result of a PRIMARY KEY or UNIQUE constraint on a table, they may, however, also be created via the CREATE UNIQUE INDEX command.,

To prevent the entry of duplicate values in the EMDNO column.

CREATE UNIQUE INDEX I-EMPNO ON EMP(EMPNO);

DROP Index Command:

To remove an index definition from the data dictionary.

DROP INDEX indexname;