**DATA DEFINITION LANGUAGE (DDL):** The Data Definition Language (DDL) is used to create and destroy databases and database objects. These commands will primarily be used by database administrators during the setup and removal phases of a database project.

Let's take a look at the structure and usage of four basic DDL commands:

1. CREATE 2. ALTER 3. DROP 4. RENAME

**1. CREATE:**

**(a)CREATE TABLE:** This is used to create a new relation (table)

***Syntax:*** CREATE TABLE <relation_name/table_name >

(field_1 data_type(size),field_2 data_type(size), .. . );

***Example:***

SQL> CREATE TABLE Student (sno NUMBER (3), sname CHAR (10), class CHAR (5));

**2. ALTER:**

**(a)ALTER TABLE ...ADD...:** This is used to add some extra fields into existing relation.

***Syntax:*** ALTER TABLE relation_name ADD (new field_1 data_type(size), new field_2

data_type(size),..);

***Example:*** SQL>ALTER TABLE std ADD (Address CHAR(10));

**(b)ALTER TABLE...MODIFY...:** This is used to change the width as well as data type of fields of existing relations.

***Syntax:*** ALTER TABLE relation_name MODIFY (field_1 newdata_type(Size), field_2

newdata_type(Size),....field_newdata_type(Size));

***Example:***SQL>ALTER TABLE student MODIFY(sname VARCHAR(10),class VARCHAR(5));

**c) ALTER TABLE..DROP...:** This is used to remove any field of existing relations.

***Syntax:*** ALTER TABLE relation_name DROP COLUMN (field_name);

***Example:***SQL>ALTER TABLE student DROP column (sname);

**d)ALTER TABLE..RENAME...:** This is used to change the name of fields in existing relations.

***Syntax:*** ALTER TABLE relation_name RENAME COLUMN (OLD field_name) to

(NEW field_name);

***Example:*** SQL>ALTER TABLE student RENAME COLUMN sname to stu_name;

3. **DROP TABLE:** This is used to delete the structure of a relation. It permanently deletes the records in the table.

***Syntax:*** DROP TABLE relation_name;

***Example:*** SQL>DROP TABLE std;

4. **RENAME:** It is used to modify the name of the existing database object.

***Syntax:*** RENAME TABLE old_relation_name TO new_relation_name;

***Example:*** SQL>RENAME TABLE std TO std1;

**DATA MANIPULATION LANGUAGE (DML):** The Data Manipulation Language

(DML) is used to retrieve, insert and modify database information. These commands will be used by all database users during the routine operation of the database. Let's take a brief look at the basic DML commands:

**1. INSERT 2. UPDATE 3. DELETE**

**1. INSERT INTO:** This is used to add records into a relation. These are three type of INSERT INTO queries which are as

**a) Inserting a single record**

***Syntax:*** INSERT INTO < relation/table name> (field_1,field_2……field_n)VALUES

(data_1,data_2,.....data_n);

***Example:*** SQL>INSERT INTO student(sno,sname,class,address)VALUES

(1,'Ravi','M.Tech','Palakol');

**b) Inserting a single record**

***Syntax:*** INSERT INTO < relation/table name>VALUES (data_1,data_2,.....data_n);

***Example:*** SQL>INSERT INTO student VALUES (1,'Ravi','M.Tech','Palakol');

**c) Inserting all records from another relation**

***Syntax:*** INSERT INTO relation_name_1 SELECT Field_1,field_2,field_n

FROM relation_name_2 WHERE field_x=data;

***Example:*** SQL>INSERT INTO std SELECT sno,sname FROM student

WHERE name = 'Ramu';

**d) Inserting multiple records**

***Syntax:*** INSERT INTO relation_name field_1,field_2,.....field_n) VALUES

(&data_1,&data_2,.....&data_n);

***Example:*** SQL>INSERT INTO student (sno, sname, class,address)

VALUES (&sno,'&sname','&class','&address');

Enter value for sno: 101
Enter value for name: Ravi
Enter value for class: M.Tech
Enter value for name: Palakol

2. **UPDATE-SET-WHERE:** This is used to update the content of a record in a relation.

***Syntax:*** SQL>UPDATE relation name SET Field_name1=data,field_name2=data,

WHERE field_name=data;

***Example:*** SQL>UPDATE student SET sname = 'kumar' WHERE sno=1;

3. **DELETE-FROM**: This is used to delete all the records of a relation but it will retain the structure of that relation.

**a) DELETE-FROM**: This is used to delete all the records of relation.

***Syntax:*** SQL>DELETE FROM relation_name;

***Example:*** SQL>DELETE FROM std;

b) **DELETE -FROM-WHERE:** This is used to delete a selected record from a relation.

***Syntax:*** SQL>DELETE FROM relation_name WHERE condition;

***Example:*** SQL>DELETE FROM student WHERE sno = 2;

**5. TRUNCATE:** This command will remove the data permanently. But structure will not be removed.

## DATA RETRIEVAL LANGUAGE (DRL)

1. **SELECT FROM:** To display all fields for all records.

***Syntax:*** SELECT * FROM relation_name;

***Example :*** SQL> select * from dept;

2. **SELECT FROM:** To display a set of fields for all records of relation.

***Syntax:*** SELECT a set of fields FROM relation_name;

***Example:*** SQL> select deptno, dname from dept;

3. **SELECT - FROM -WHERE:** This query is used to display a selected set of fields for a selected set of records of a relation.

***Syntax:*** SELECT a set of fields FROM relation_name WHERE condition;

***Example:*** SQL> select * FROM dept WHERE deptno<=20;

## TRANSATIONAL CONTROL LANGUAGE (TCL)

TCL commands can only use with DML commands like INSERT, DELETE and UPDATE only.

1. **COMMIT:** This command is used to end a transaction only with the help of the commit command transaction changes can be made permanent to the database.

***Syntax:*** SQL> COMMIT;

***Example:*** SQL> COMMIT;

2. **SAVE POINT**: Save points are like marks to divide a very lengthy transaction to smaller once. They are used to identify a point in a transaction to which we can latter role back. Thus, save point is used in conjunction with role back.

***Syntax:*** SQL> SAVE POINT ID;

***Example:*** SQL> SAVE POINT xyz;

3. **ROLLBACK:** A role back command is used to undo the current transactions. We can role back the entire transaction so that all changes made by SQL statements are undo (or) role back a transaction to a save point so that the SQL statements after the save point are role back.

***Syntax:*** ROLLBACK (current transaction can be role back)

ROLLBACK to save point ID;

***Example:*** SQL> ROLLBACK;

SQL> ROLLBACK TO SAVE POINT xyz;

## *CLAUSES:*

***Aggregative operators:*** In addition to simply retrieving data, we often want to perform some computation or summarization. SQL allows the use of arithmetic expressions. We now consider a powerful class of constructs for computing aggregate values such as MIN and SUM.

1. **Count:** COUNT following by a column name returns the count of tuple in that column. If DISTINCT keyword is used then it will return only the count of unique tuple in the column. Otherwise, it will return count of all the tuples (including duplicates) count (*) indicates all the tuples of the column.

   ***Syntax:*** COUNT (Column name)
   ***Example:*** SELECT COUNT (Sal) FROM emp;

2. **SUM:** SUM followed by a column name returns the sum of all the values in that column.

   ***Syntax:*** SUM (Column name)

   ***Example:*** SELECT SUM (Sal) From emp;

3. **AVG:** AVG followed by a column name returns the average value of that column values.

   ***Syntax:*** AVG (n1, n2...)

   ***Example:*** Select AVG (10, 15, 30) FROM DUAL;

4. **MAX:** MAX followed by a column name returns the maximum value of that column.

   ***Syntax:*** MAX (Column name)

   ***Example:*** SELECT MAX (Sal) FROM emp;

5. **MIN:** MIN followed by column name returns the minimum value of that column.

   ***Syntax:*** MIN (Column name)

   ***Example:*** SELECT MIN (Sal) FROM emp;

**STRING FUNCTIONS:**

**Concat:** CONCAT returns char1 concatenated with char2. Both char1 and char2 can be any

of the datatypes

SQL>SELECT CONCAT('ORACLE','CORPORATION')FROM DUAL;

ORACLECORPORATION

**Ltrim:** Returns a character expression after removing leading blanks.

SQL>SELECT LTRIM('SSMITHSS','S')FROM DUAL;

MITHSS

**Rtrim:** Returns a character string after truncating all trailing blanks

SQL>SELECT RTRIM('SSMITHSS','S')FROM DUAL;

SSMITH

**Equi-join :**

A join, which is based on equalities, is called equi-join.

Example:

Select * from item, cust where item.id=cust.id;

In the above statement, item-id = cust-id performs the join statement. It retrieves rows from both the tables provided they both have the same id as specified by the where clause. Since the where clause uses the comparison operator (=) to perform a join, it is said to be equijoin. It combines the matched rows of tables. It can be used as follows:

- ✓ To insert records in the target table.
- ✓ To create tables and insert records in this table.
- ✓ To update records in the target table.
- ✓ To create views.

**Non Equi-join:**

It specifies the relationship between columns belonging to different tables by making use of relational operators other than'='.

***Example:***

Select * from item, cust where item.id<cust.id;

Table Aliases

Table aliases are used to make multiple table queries shorted and more readable. We give an alias name to the table in the 'from' clause and use it instead of the name throughout the query.

**Self join:**

Joining of a table to itself is known as self-join. It joins one row in a table to another. It can compare each row of the table to itself and also with other rows of the same table.

Example:

select * from emp x ,emp y where x.salary >= (select avg(salary) from x.emp where x. deptno =y.deptno);

**Outer Join:**

It extends the result of a simple join. An outer join returns all the rows returned by simple join as well as those rows from one table that do not match any row from the table. The symbol(+) represents outer join.

- Left outer join
- Right outer join
- Full outer join

**ORDER BY:** This query is used to display a selected set of fields from a relation in an ordered manner base on some field.

***Syntax:*** SELECT <set of fields> FROM <relation_name>

ORDER BY <field_name>;

***Example:*** SQL> SELECT empno, ename, job FROM emp ORDER BY job;

**SUBQUERIES:** The query within another is known as a sub query. A statement containing sub query is called parent statement. The rows returned by sub query are used by the parent statement or in other words A subquery is a SELECT statement that is embedded in a clause of another SELECT statement

You can place the subquery in a number of SQL clauses:

- WHERE clause
- HAVING clause
- FROM clause
- OPERATORS( IN.ANY,ALL,<,>,>=,<= etc..)

## Types

### 1. Sub queries that return several values

Sub queries can also return more than one value. Such results should be made use along with the operators in and any.

### 2. Multiple queries

Here more than one sub query is used. These multiple sub queries are combined by means of ‘and’ & ‘or’ keywords.

### 3. Correlated sub query

A sub query is evaluated once for the entire parent statement whereas a correlated Sub query is evaluated once per row processed by the parent statement.

**VIEW:** In SQL, a view is a virtual table based on the result-set of an SQL statement. A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database.

You can add SQL functions, WHERE, and JOIN statements to a view and present the data as if the data were coming from one single table.

A view is a virtual table, which consists of a set of columns from one or more tables. It is similar to a table but it does not store in the database. View is a query stored as an object.

***Syntax:*** CREATE VIEW <view_name> AS SELECT <set of fields>
FROM relation_name WHERE (Condition)

***Example:***

SQL> CREATE VIEW employee AS SELECT empno,ename,job FROM EMP
WHERE job = ‘clerk’;

SQL> View created.

***Example:***

CREATE VIEW [Current Product List] AS

SELECT ProductID, ProductName

FROM Products

WHERE Discontinued=No;

**UPDATING A VIEW :** A view can updated by using the following Syntax :

**Syntax** : CREATE OR REPLACE VIEW view_name AS

SELECT column_name(s)

FROM table_name

WHERE condition

**DROPPING A VIEW:** A view can deleted with the DROP VIEW command.

**Syntax**: DROP VIEW <view_name> ;