Module 3

STRUCTURED QUERY LANGUAGE (SQL)

DDL Commands(data definition Language Commands)

1. Create table command

This command is used to create a new table in SQL.

Syntax:

CREATE TABLE table_name (Attribute1 datatype,

Attribute2 datatype,

.

.

Attribute n datatype, primary key(key_name),

CONSTRAINT const_name FOREIGN KEY(key_name) REFERENCES table_name(key_name));

Example:1

Create a table employee with **attributes eid,ename, age, salary and dept id(foreign key)**

Also

Create another table department with **attribute deptid,dname and location**

CREATE TABLE department (deptid int, dname varchar(20),location varchar(30),primary key(deptid));

CREATE TABLE employee (eid int primary key,ename varchar(10),age int, salary int, deptid int, CONSTRAINT pk_id FOREIGN KEY(deptid) REFERENCES department(deptid));

Example:2

student

Reg_no	Sname	Address	Age	mark

CREATE TABLE student (reg_no int primary key, sname varchar(10), address varchar(30),age int, mark int);

Specifying constraints in SQL

CREATE TABLE employee (eno int primary key NOT NULL,ename varchar(20), age int DEFAULT '19', dno int,phone int UNIQUE,

CONSTRAINT pk_id FOREIGN KEY (DNO) REFERENCES department (dno) ON DELETE CASCADE ON UPDATE CASCADE);

From the above table definition constraints are

1. **Primary key** – it is used to identify each record of a table uniquely. It is called key constraint
2. **NOT NULL** – this constraint is used to specify NULL values is not permitted for a particular attribute
3. **DEFAULT**- by using this constraint we can specify a default value for an attribute. The default value is included within single quotes.
4. **FOREIGN KEY**-it is called referential integrity constraint. It is specified when the given table has a foreign key
5. **ON DELETE CASCADE**: This constraint is specified within the foreign key.ie. It Specify that ,if we delete the value of the primary table, then that changes will also reflect in the secondary table in which foreign key belongs
6. **ON UPDATE CASCADE**: This constraint is also specified within the foreign key. It Specify that ,if we delete update the value of the primary table, then that changes will also reflect in the secondary table in which foreign key belongs
7. **UNIQUE**: This constraint is used to specify alternate key or secondary key for a table

II. DROP COMMAND

It is used to destroy the created table. After the execution of this command the table along all of its data will be destroyed

Syntax:
DROP TABLE table_name;

Example : drop table student;

III. RENAME COMMAND

It is used to rename the existing table

Syntax:
RENAME oldtable_name TO newtable_name;

Example
If we want to change the table name employee to customer, use the following code
RENAME employee TO customer

IV. ALTER COMMAND

There are 3 uses for the alter command

1. To add new columns
2. To remove any column from the table
3. To modify the data type of existing table

1. Add new columns

Syntax:

ALTER TABLE table_name ADD (attribute1 datatype,
Attribute 2 datatype……);

Example:

From the above table employee add new attribute or column named designation

ALTER TABLE employee ADD(designation varchar(20));

2. **Remove any column**

Syntax:
ALTER TABLE table_name DROP COLUMN column_name;

Example:
From the table employee remove the column age
ALTER TABLE employee DROP COLUMN (age);

3. **Modify the data type existing table**

This command is used to modify the data type of the existing column of a table

Syntax:
ALTER TABLE table_name MODIFY (column_name new_datatype);

Example:
Change the datatype of age to number in the table employee
ALTER TABLE employee MODIFY(age number(3));

Restrictions of ALTER command

By using the ALTER command we cannot perform the following tasks

- Change the name of the table
- Change the name of the column
- Decreases the size of the column if table data exists

DML COMMNADS (DATA MANIPULATION COMMANDS)

1. INSERT COMMAND

This command is used to insert values to the table. The values must be inserted in the order in which the columns are created in the table

Syntax:

INSERT INTO table_name VALUES(value1,value2,…….,value n);

OR

INSERT INTO table_name (column_name1,column_name2…) VALUES (value1,value2….)

Example:

Create the following table and insert the values

STUDENT

SID	SNAME	AGE	MARK
1	Anu	18	200
2	Bindhu	17	289
3	Veena	18	250

CREATE TABLE student(sid int primary key, sname varchar(20), age int, mark int);

INSERT INTO student VALUES (1,'Anu',18,200);

INSERT INTO student VALUES (2,'Bindhu',17,289);

INSERT INTO student VALUES (3,'Veena',18,250);

2. SELECT COMMAND

It is used for viewing or retrieving data from the table. We can use logic operators as well as relational operators with in the select command.

Syntax

SELECT <attribute list>

FROM <table list>

WHERE <condition>

<attribute list > is the list of attribute names to be retrieved from the query.

<table list> is the list table names in which these attributes belongs.

<condition> It is a condition that identifies the tuples to be retrieved by the query

Example:

Create the following table , insert the values and do the quires

Employee

ENO	ENAME	GENDER	PLACE	PHONE	AGE	SALARY
1	Anu	female	pala	22435346	30	30000
2	Meenu	female	kottayam	46474585	40	20000
3	Aravind	male	kottayam	35665768	38	30000
4	Veena	female	ponkunnam	12567869	35	25000
5	Alen	male	pala	12345679	48	45000

CREATE TABLE employee (eno int primary key, ename varchar(20),gender varchar(20),place varchar(20),phone int,age int,salary int)

INSERT INTO employee VALUES(1,'anu','female','pala', 22435346,30,30000);

INSERT INTO employee VALUES(2,'meenu','female','kottayam', 46474585,40,20000);

INSERT INTO employee VALUES(3,'aravind','male','kottayam', 35665768,38,30000);

INSERT INTO employee VALUES(4,'veena','female','ponkunnam', 12567869,35,25000);

INSERT INTO employee VALUES(5,'alan','male','pala', 12345679,48,45000);

Queries

1. Retrieve all employees who have the age below 40
   SELECT ename
   From employee
   WHRE age<40
2. Retrieve all employees
   SELECT *
   From employee
3. Retrieve name and age of all male employees
   SELECT ename,age
   FROM employee
   WHERE gender='male'

ALL and DISTINCT

4. Select all employee salary from the employee table

```
SELECT ALL salary
From employee
```

Output

```
30000
20000
30000
25000
45000
```

5. Select distinct employee salary from the employee table

```
SELECT DISTINCT salary
From employee
```

Output

```
30000
20000
25000
45000
```

6. Retrieve all employee details whose age less than 40and are coming from ‘pala’

```
SELECT * FROM employee
WHERE age<40 and place=’pala’
```

7. Retrieve eno and name of all employees who are either coming from ‘pala’ or salary greater than 25000

```
SELECT eno,ename
FROM employee
WHERE place=’pala’ OR salary>25000
```

8. Retrieve eno and name of all female employees who are either coming from ‘kottayam’ and age greater than 20

```
SELECT eno,ename
FROM employee
WHERE place=’kottayam’ and gender=’female’ and age>20
```

LIKE OPERATOR

The **like** operator is used in where clause to search for a specified pattern. The two wild characters used in conjunction with the **like** operators **are % and –**

% represent 0 , one or multiple characters

_ represent single character

LIKE operator	meaning
LIKE a%	Find any value that starts with letter 'a'
LIKE %a	Find any value that ends with letter 'a'
LIKE % or %	Find all values which include the value 'or' in any position
LIKE _r %	Find any value that have the character 'r' is in the second position
LIKE a_ _	Find any value that starts with character 'a' and have only 3 characters in length
LIKE a % m	Find any value that starts with character 'a' and ends with character 'm'

Employee

ENO	ENAME	GENDER	PLACE	PHONE	AGE	SALARY
1	Anu	female	pala	22435346	30	30000
2	Meenu	female	kottayam	46474585	40	20000
3	Aravind	male	kottayam	35665768	38	30000
4	Veena	female	ponkunnam	12567869	35	25000
5	Alen	male	pala	12345679	48	45000
6	Praveen	male	kottayam	65789891	44	35000

From the above table, answer the following

1. Retrieve all employees names who have the age>25 and whose name ends with the character 'u'

   Ans:
   SELECT ename
   FROM employee
   WHERE age>25 AND ename LIKE '%u'

2. Retrieve names and salary of all employees who are coming from 'kottayam' and their name starts with character 'm' and ends with the character 'u'

   Ans:
   SELECT ename,salary
   FROM employee
   WHERE place='kottayam' AND ename LIKE 'm%u'

3. Select all female employees names those who have the character ‘e’ in the third position of their names

   Ans:
   SELECT ename
   FROM employee
   WHERE gender=’female’ AND ename LIKE ‘_ _e %’

4. Retrieve all male employees names, age and salary in which whose name have the character ‘a’ in the third position and whose name ends with the character ‘n’ and have maximum of 7 characters in length

   Ans:
   SELECT ename, age,salary
   FROM employee
   WHERE gender=’male’ AND ename LIKE ‘_ _ a _ _ _ n’

BETWEEN OPERATOR

This operator is used to select values within a specified range

Example:

1. Select all employee details whose salary between 20000 and 40000

   Answer:
   SELECT * FROM employee
   WHERE salary BETWEEN 20000 AND 40000

2. Select name of all employees who are coming from ‘kottayam’ and whose name start with letter ‘a’ and whose salary lies between 15000 and 30000

Ans:
SELECT ename FROM employee
WHERE place='kottayam' AND ename LIKE 'a%' AND salary BETWEEN 15000 AND 30000

ORDERING OF QUERY RESULT

SQL allow us to order the tuples in the result of a query by using the values of one attribute that appears in that query. **For ordering the result of a query we can use the ORDER BY clause. Query results either can be ordered in ascending form or can be in descending form**. The default ordering is in the ascending form

Q: Arrange all female employee names in the ascending order of their salary

```
SELECT ename, salary
FROM employee
WHERE gender='female'
ORDER BY salary ASC
```

Q: Display all employee names and salary that are coming from 'kottyam' and arrange the result in the descending order of their age

```
SELECT ename,salary,age
FROM employee
WHERE place='kottayam'
ORDER BY age DESC
```

Ambiguous Attributes and Aliasing

In SQL same name can be used for two or more attributes in different tables. In this situation when combing these tables there exists a conflict between these tables. These attributes are called ambiguous attribute. So to avoid such ambiguous attributes, we should prefix the table name before the attribute name and separate them with a period (.)

Consider the following tables

Employee

Eno	Ename	Esal	Dno
1	Anu	10000	1
2	Anju	20000	1
3	Binil	15000	2
4	Biju	25000	2
5	Veena	24000	3
6	varun	18000	2

Department

Dno	Dname	Floor
1	Sales	1
2	Promotion	3
3	Production	2
4	HR	1

When combining these 2 tables, there is conflict between Dno in employee table and Dno in department table. So Dno is an ambiguous attribute. So when specifying this attribute when should use the syntax

Table name. attribute name

Q. Retrieve employee name and salary of all employees who are working in sales department

SELECT ename, salary

FROM employee , department

WHERE employee.Dno = department.Dno and dname='sales'

Q. Retrieve empid and name of all employees those who are working in the first floor

SELECT eno, ename

FROM employee , department

WHERE employee.Dno = department.Dno and floor=1

Using Aliasing

When referencing attributes from more than one tables, we can use aliases instead of table names

For example we can rewrite the above query using aliases as

Q. Retrieve employee name and salary of all employees who are working in sales department

SELECT e.ename, e.salary

FROM employee e, department d

WHERE e.Dno = d.Dno and d.dname='sales'

Q. Retrieve empid and name of all employees those who are working in the first floor

SELECT e.eno, e.ename

FROM employee e, department d

WHERE e.Dno = d.Dno and d.floor=1

Renaming of attributes in SQL

It is possible to rename any attribute that appears in the result of a query. To do that use the qualifier 'as' followed by new attribute name

Example:

Display all employee names working in sales department and display the result with the name 'sales employees'

SELECT e.ename AS sales employees

FROM employee e, department d

WHERE e.Dno = d.Dno and d.dname='sales'

Use of Arithmetic operators in query

We can use basic arithmetic operators (+,-,*, /) on attributes to perform any of the arithmetic operations on attribute values

Example: write a query to increase the salary of all employees working in sales department to 10% and show the results as 'new salary'

SELECT e.ename , e.esal, e.esal+(e.esal*.1) AS new salary

FROM employee e, department d

WHERE e.Dno = d.Dno and d.dname='sales'

2. DELETE COMMAND IN SQL

The delete command is used to remove tuples from the tables based on a condition. We can delete several number of tuples based a condition by using a single delete command.

Syntax:

DELETE FROM table_name WHERE <condition>

Q . delete all employees whose name starts with v.

DELETE FROM employee WHERE ename LIKE 'v%'

Q. delete all employees in sales department

DELETE FROM employee e, department d

WHERE e.dno=d.dno and d.dname='sales'

3. **UPDATE COMMAND IN SQL**

The update command is used to modify the values of one or more selected tuples depending on a condition. We cannot modify the primary key values

Syntax:

UPADTE tablename SET attribute_name=new value

WHERE <condition>

Q. modify the name of the employee as 'bijil' with the employee no 3

UPDATE employee SET ename='bijil' WHERE eno=3

Q. modify the salary of the employee 'veena' to 30000

UPDATE employee SET esal=30000 WHERE ename='veena'

SET OPERATIONS IN SQL

SET operations allow the result of multiple querys to be combined into single result set. set operations' include the following operators

- UNION
- INTERSECT
- EXCEPT

Employee

Eno	Ename	Esal	Pno	Dno
1	Anu	10000	1	1
2	Anju	20000	1	2
3	Vineeth	30000	2	1
4	Arjun	13000	1	3

5	Veena	22000	2	3
6	Meena	35000	1	2
7	Appu	31000	2	2

Department

Dno	Dname
1	Sales
2	Production
3	Promotion

Project

Pno	Pname
1	Product X
2	Product Y

UNION Operator

It combines the result of two SQL queries into a single table. Consider the following examples

Q1: select all employee names those who are either working in sales department or working in promotion department

Ans:

```
SELECT e.ename
FROM employee e, department d
WHERE e.dno=d.dno and d.dname='sales'
UNION
SELECT e.ename
FROM employee e, department d
WHERE e.dno=d.dno and d.dname='promotion'
```

Q2: select all employee names those who are either working in ‘production’ department or involved in a project called ‘product X’

Ans:

SELECT e.ename

FROM employee e, department d

WHERE e.dno=d.dno and d.dname=’production’

UNION

SELECT e.ename

FROM employee e, project p

WHERE e.pno=p.pno and p.pname=’Product X’

INTERSECT Operator

This operator takes the result of two queries and returns only the rows that appears in both result set

Q: Select all employee names those who are working in ‘sales’ department and involving in a project called ‘Product X’

Ans:

SELECT e.ename

FROM employee e, department d

WHERE e.dno=d.dno and d.dname=’sales’

INTERSECT

SELECT e.ename

FROM employee e, project p

WHERE e.pno=p.pno and p.pname=’Product X’

Q: Select all employee names who have the salary greater than 30000 and working in 'promotion' department

Ans:

SELECT e.ename

FROM employee

WHERE esal>30000

INTERSECT

SELECT e.ename

FROM employee e, department d

WHERE e.dno=d.dno and d.dname='promotion'

EXCEPT Operator

This operator is also called minus operator. It is used to combine two selected statements and returns tuples from the first select statements that are not present in the second select statement.ie EXCEPT operator returns only tuples which are not available in the second query

Q: select all employee names in 'sales' department who are not involving in a project called 'product X '

Ans:

SELECT e.ename

FROM employee e, department d

WHERE e.dno=d.dno and d.dname='sales'

EXCEPT

SELECT e.ename

FROM employee e, project p

WHERE e.pno=p.pno and p.pname='Product X'

Aggregate functions in SQL

These functions are used to perform mathematical operations on tuples. Aggregate functions are used to summarize information from multiple tuples. There are 5 type of aggregate functions

- Count
- Sum
- Max
- Min
- Avg

COUNT function returns the no: of tuples specified in the query

Example: count total number of employees working in company

Ans: **SELECT COUNT(*) FROM employee;**

Q:select number of employees whose salary between 20000 and 40000

Ans: **SELECT COUNT(ename) FROM employee WHERE esal BETWEEN 20000 and 40000**

Q:select number employees working in sales department

Ans: **SELECT COUNT(e.ename)**

FROM employee e, department d

WHERE e.dno=d.dno and d.dname='sales'

SUM,MAX,MIN,AVG can be applied to set of values and returns respectively Sum, maximum, minimum and average of those values

Q: display maximum and minimum salary of employees

Ans: **SELECT MAX(esal),MIN(esal)**

FROM employee

Q: display average salary of employees in sales department

Ans: **SELECT AVG(e.esal)**

FROM employee e, department d

WHERE e.dno=d.dno and d.dname='sales'

NESTED QUERIES IN SQL

Nested query is a query which is written inside another query.the result of inner query is used in the execution of outer query.there are mainly 2 types of nested queries

1. Independent nested queries

2. Co-related nested queries

Independent nested query

In independent nested queries, query execution starts from inner most queries to outermost queries. **The execution of inner query is independent of outer query. But the result of inner query is used for the execution of outer queries.**

Various operators like IN, NOT IN, ALL are used for writing nested queries

Example:

Q: select all employee names those who are working in sales department

```
Ans:  SELECT ename
      FROM employee
      WHERE dno IN (   SELECT dno
                       FROM department
                       WHERE dname='sales')
```

Q: select all employee names who are not working in a project called 'product X'

```
Ans:  SELECT ename
      FROM employee
      WHERE pno NOT IN (   SELECT pno
                           FROM project
                           WHERE pname='product X')
```

Q: select all employee names whose salary is greater than the salary of all employees in 'production' department

Ans: SELECT ename

FROM employee

WHERE esal > ALL (SELECT e.esal

FROM employee e ,department d

WHERE e.dno=d.dno and dname='production')

Q: select all employee names who are not working in 'promotion' department

Ans: SELECT ename

FROM employee

WHERE dno NOT IN (SELECT dno

FROM department

WHERE dname='promotion')

Co-related nested queries

Here the execution of inner query depends on the execution of outer query and vice versa. That is, the inner and outer queries are dependent on each other; It is also called synchronized sub queries

EXISTS and UNIQUE functions in SQL

The EXISTS function in SQL is used to check whether the result of nested query is empty or not. EXISTS function returns TRUE, if the nested query result contain atleast one tuple. Otherwise it returns FALSE

That is ,we can write EXISTS function instead of IN operator

Q: select all employee names those who are working in sales department

Ans: SELECT ename

FROM employee

WHERE EXISTS (SELECT dno

FROM department

WHERE dname='sales')

Q: select all employee names who are working in a project called 'product X'

Ans: SELECT ename

FROM employee

WHERE EXISTS (SELECT pno

FROM project

WHERE pname='product X')

UNIQUE(Q)

It returns TRUE if there are no duplicate tuples in the result of a query. Otherwise it returns FALSE. This function is used to test whether the result of a nested query contain duplicate values

GROUP BY and HAVING clause

GROUP BY clause

It is used with aggregate functions to produce summary reports from the database and group the results based on attribute.

The GROUP BY clause specifies the grouping attribute which should also appears in the SELECT clause

Q: Find out the number of employees working in each department

SELECT d.dname, COUNT (e.ename)

FROM employee e, department d

WHERE e.dno=d.dno

GROUP BY d.dname

Q: Find out the maximum salary of employees in each department

SELECT d.dname, MAX (e.esal)

FROM employee e, department d

WHERE e.dno=d.dno

GROUP BY d.dname

HAVING clause

HAVING clause is used along with GROUP BY clause to group tuples based on certain condition. That is HAVING clause specifies certain conditions that is used for the grouping of tuples

Example: if you want to find out the number of employees in each department with the condition that department should contain more than 2 employees

SELECT d.dname, COUNT (e.ename)

FROM employee e, department d

WHERE e.dno=d.dno

GROUP BY d.dname

HAVING COUNT (e.ename) > 2

JOIN OPERATIONS IN SQL

SQL join operation combines rows from one or more tables based on a related column between them. There are different types of join operation in SQL

- Inner join or join
- Left join or left outer join
- Right join or right outer join
- Full outer join

Consider the following tables

country

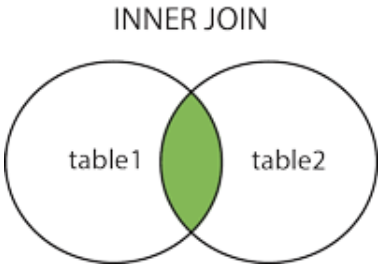
cid	cname
C1	INDIA
C2	NEPAL
C3	SRILANKA

State

sid	cid	Sname
S1	C1	Kerala
S2	C1	Punjab
S3	C2	Kadmandu
S4	null	California

INNER JOIN OR JOIN

It returns the records that have matching values in both tables. That is we can represent inner join as



To apply the inner join write the following statements

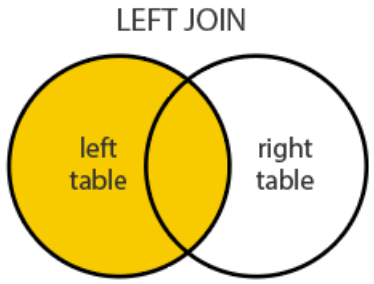
SELECT * FROM country INNER JOIN state ON country.cid = state.cid

Result

cid	cname	sid	cid	sname
1	INDIA	1	1	Kerala
1	INDIA	2	1	Punjab
2	NEPAL	3	2	Kadmandu

LEFT JOIN OR LEFT OUTER JOIN

It returns all records from the left table and the matched records from the right table. That is, all the data from the left table will be displayed irrespective of any matching entry with the right table. We can represent the left join as



To apply the left join write the following statement

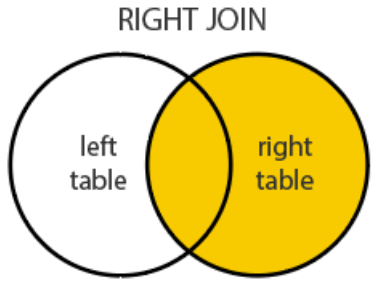
SELECT * FROM country LEFT JOIN state ON country.cid = state.cid

Result

cid	cname	Sid	cid	sname
1	INDIA	1	1	Kerala
1	INDIA	2	1	Punjab
2	NEPAL	3	2	Kadmandu
3	SRILANKA	NULL	NULL	NULL

RIGHT JOIN OR RIGHT OUTER JOIN

It returns all records from the right table and the matched records from the left table. That is, all the data from the right table will be displayed irrespective of any matching entry with the left table. We can represent the right join as



To apply the right join write the following statement

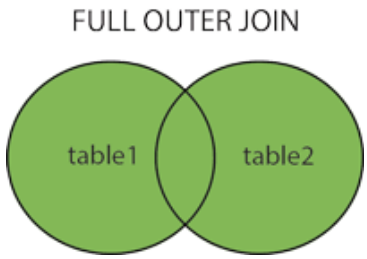
SELECT * FROM country RIGHT JOIN state ON country.cid = state.cid

Result

cid	cname	Sid	cid	sname
1	INDIA	1	1	Kerala
1	INDIA	2	1	Punjab
2	NEPAL	3	2	Kadmandu
NULL	NULL	4	NULL	california

Full outer join

It returns all the records from both table .It show all the matching records as well as non-matching records from both table. It is a combination of inner join, left join and right join. We can represent full join as



To apply the full outer join write the following statement

SELECT * FROM country FULL OUTER JOIN state ON country.cid = state.cid

Result

cid	cname	Sid	cid	sname
1	INDIA	1	1	Kerala
1	INDIA	2	1	Punjab
2	NEPAL	3	2	Kadmandu
3	SRILANKA	NULL	NULL	NULL
NULL	NULL	4	NULL	california

VIEWS IN SQL (VIRTUAL TABLE)

A view is a single table that is derived from other real tables. The other real table can be base tables or previously defined views. A view does not exists in physical form, it is considered to be a virtual table.ewe can create view from two or more tables.

A view contains rows and columns just like a real table. But the fields in a view are the fields from one or more real tables in the database

Syntax:

CREATE VIEW view_name AS

SELECT attribute1,attribute 2……

FROM table 1,table 2……

WHERE conditions

Q: create a view called works-on from the table's employee and project. The view should contain the name and salary of all employees working in a project called 'product X'

Answer:

CREATE VIEW works_on AS

SELECT e.ename,e.esal

FROM employee e, project p

WHERE e.pno=p.pno and p.pname='Product X'

To display the view, use then statement

SELECT * FROM view_name

Example: SELECT * FROM works_on;

Q: create a view called 'empsales' from the tables employee and department. The view should contain the names of all employees working 'sales' department

Answer:

CREATE VIEW empsales AS

SELECT e.ename

FROM employee e, department d

WHERE e.dno=d.dno and d.dname='sales'

To display the view, use then statement

SELECT * FROM empsales

DELETING ROWS

To remove or drop views, we can use the following command

Syntax:

DROP VIEW view_name;

Ex: To remove the already created view 'empsales'

DROP VIEW empsales;

UPDATING VIEWS

We can update the values from the created view in the same manner as we update the values from the real table

Syntax:

UPDATE view_name

SET attribute_name=new value

WHERE condition

Ex:update the salary of the employee anu to 45000 in the view 'works_on'

UPDATE works_on

SET esal=45000

WHERE ename='anu'

RESTRICTIONS ON UPDATE COMMAND

1. A view with single defining table is updatable
2. A view with multiple tables are generally not updatable
3. A view defined using GROUP BY and aggregate functions are not updatable