**POSTGRESQL – II**

**Limit**

**Offset**

**Ex:**

select \* from employee limit 5

select \* from employee offset 2

select \* from employee offset 2 limit 2

**Like:** Used to match pattern

Wild Chars \_ , %

S% - starting with s

%s - ending with s

%s% - contains s

\_a% -second letter is a

\_ \_s% -third char is s

Ex:

select \* from employee where ename like '\_a%'

select \* from employee where ename like 'R%'

select \* from employee where ename like '%a%'

**In**

Used to check the condition from a given list of values

Ex: select \* from employee where city in ('Hyd','bang','Delhi')

**Serial**

Automatically inserts serial numbers into a column

Ex:

create table products(

pno serial,

pname varchar(30),

price int

)

insert into products (pname, price) values('Chatger', 350)

select \* from products

**Working with Joins**

Joins is the concept of joining one or more table columns into a single result set

Joins used to retrieve the data from more than one table

For joining two tables , they must have atleast one common column.

**Inner Join**

**Outer Join – left outer join , right outer join , full outer join**

**Cross Join**

**Self Join**

**Equi join**

**Inner Join**

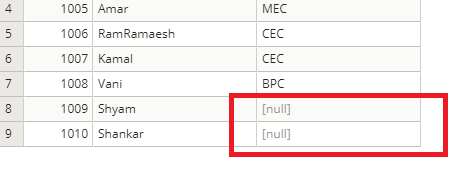
A INNER JOIN creates a new result table by combining column values of two tables (table1 and table2) based upon the join-predicate. Display the records form the two tables which satisfy the join condition

Ex:

select student.sid, student.sname, course.cname from student join course on student.cid = course.cid

**Left Outer Join :**

Combines the two tables columns basing on the join condition and retrieves all rows from first table , if no matching value for the second table column value displays Null



**Right outer join**

Combines the two tables columns basing on the join condition and retrieves all rows from second table , if no matching value for the first table column value displays Null

Ex:

select student.sid, student.sname, course.cname from student right outer join course on student.cid = course.cid

**Full Outer Join**

Combines the two tables columns basing on the join condition and retrieves all rows from two tables , if no matching value displays Null

Ex:

select student.sid, student.sname, course.cname from student Full outer join course on student.cid = course.cid

Examples :

select student.sid, student.sname, course.cname

from student join course

on student.cid = course.cid

select student.sid, student.sname, course.cname

from student left outer join course

on student.cid = course.cid

select student.sid, student.sname, course.cname

from student right outer join course

on student.cid = course.cid

select student.sid, student.sname, course.cname

from student full outer join course

on student.cid = course.cid

select s.sid, s.sname, c.cname

from student as s full outer join course as c

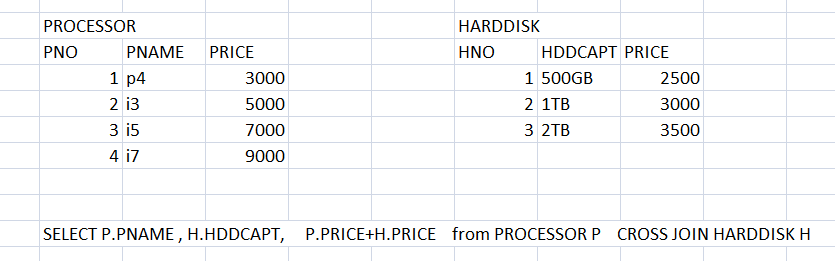
on s.cid = c.cid

**Cross Join**

A CROSS JOIN matches every row of the first table with every row of the second table. If the input tables have x and y columns, respectively, the resulting table will have x+y columns.

Ex:

Select processor.pname, hdd.disk, processor.price+ hdd.price from processor cross join hdd



**Self join**

Joins the same table twice to get précised data

To join the same table twice we requires alias names for the table to avoid ambiguity

Ex:

select e.eno, e.ename, e1.ename from employee as e join employee e1 on e.mngrid = e1.eno

**UNION**

The PostgreSQL **UNION** clause/operator is used to combine the results of two or more SELECT statements without returning any duplicate rows.

To use UNION, each SELECT must have the same number of columns selected, the same number of column expressions, the same data type, and have them in the same order but they do not have to be the same length.

The UNION ALL operator is used to combine the results of two SELECT statements including duplicate rows. The same rules that apply to UNION apply to the UNION ALL operator as well.

Ex:

SELECT \* FROM STUDENT1 UNION SELECT \* FORM STUDENT2

SELECT \* FROM STUDENT1 UNIONALL SELECT \* FORM STUDENT2

**Joining three tables ex:**

select s.sname, c.cname, l.city

from student as s

join course as c on s.cid = c.cid

join clocation as l on c.cid= l.cid

**Working with SubQueries**

A subquery-also referred to as an *inner query* or*inner select*-is a SELECT statement embedded within a data manipulation language (DML) statement or nested within another subquery. You can use subqueries in SELECT, INSERT, UPDATE, and DELETE statements wherever expressions are allowed. For instance, you can use a subquery as one of the column expressions in a SELECT list or as a table expression in the FROM clause.

A DML statement that includes a subquery is referred to as the *outer query*. The following guidelines provide details about how to implement subqueries in your outer queries or in other subqueries:

* You must enclose a subquery in parenthesis.
* A subquery must include a SELECT clause and a FROM clause.
* A subquery can include optional WHERE, GROUP BY, and HAVING clauses.

Ex:

**select \* from employee where mngrid= (select mngrid from employee where eno =1004)**

Will display the employees whose manager id is same as eno 1004

First retrives the mngrid of the employee with eno 1004 and then retrives the employees basing on the mngrid

**select \* from employee where city in (select city from employee where eno = 1004 or eno = 1006)**

**Working with Views**

Views are sub tables . views are not real tables. A view never stores the data in it. A view can represent a subset of a real table, selecting certain columns or certain rows from an ordinary table. A view can even represent joined tables. Because views are assigned separate permissions, you can use them to restrict table access so that the users see only specific rows or columns of a table.

A view can contain all rows of a table or selected rows from one or more tables. A view can be created from one or many tables, which depends on the written PostgreSQL query to create a view.

Views, which are kind of virtual tables, allow users to do the following −

* Structure data in a way that users or classes of users find natural or intuitive.
* Restrict access to the data such that a user can only see limited data instead of complete table.
* Summarize data from various tables, which can be used to generate reports.

Since views are not ordinary tables, you may not be able to execute a DELETE, INSERT, or UPDATE statement on a view. However, you can create a RULE to correct this problem of using DELETE, INSERT or UPDATE on a view.

## Creating Views

The PostgreSQL views are created using the **CREATE VIEW** statement. The PostgreSQL views can be created from a single table, multiple tables, or another view.

The basic CREATE VIEW syntax is as follows −

CREATE VIEW view\_name AS

SELECT column1, column2.....

FROM table\_name

WHERE Condition

You can include multiple tables in your SELECT statement in very similar way as you use them in normal PostgreSQL SELECT query.

Consider, the [COMPANY](https://www.tutorialspoint.com/postgresql/company.sql) table having the columns id, name, age , address, salary

Now, following is an example to create a view from COMPANY table. This view would be used to have only few columns from COMPANY table −

CREATE VIEW COMPANY\_VIEW AS

SELECT ID, NAME, AGE

FROM COMPANY;

Now, you can query COMPANY\_VIEW in a similar way as you query an actual table. Following is the example −

SELECT \* FROM COMPANY\_VIEW;

## Dropping Views

To drop a view, simply use the DROP VIEW statement with the **view\_name**. The basic DROP VIEW syntax is as follows −

DROP VIEW view\_name;

The following command will delete COMPANY\_VIEW view, which we created in the last section −

DROP VIEW COMPANY\_VIEW;