COURSE NAME: DBMS
COURSE CODE:23AD2102A

Topic: **JOINING RELATIONS(INNER, OUTER)**

Session - 10

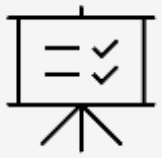
AIM OF THE SESSION

To familiarize students how to join the relations in PostgresQL

To aware the students to define views and compound statements

To grasp the students the concept of user defined statements in PostgresQL

INSTRUCTIONAL OBJECTIVES



This Session is designed to get the enough knowledge to join the relations and user defined functions

LEARNING OUTCOMES

At the end of this session, you should be able to write the compound statements

JOINING RELATIONS

Why PostgresQL JOIN:

- As the name shows, JOIN means to combine something. In case of SQL, JOIN means "to combine two or more tables".
- The PostgresQL JOIN clause takes records from two or more tables in a database and combines it together.
- In the process of joining, rows of both tables are combined in a single table.
- To access more than one table through a select statement.

ANSI standard PostgresQL defines five types of JOIN :

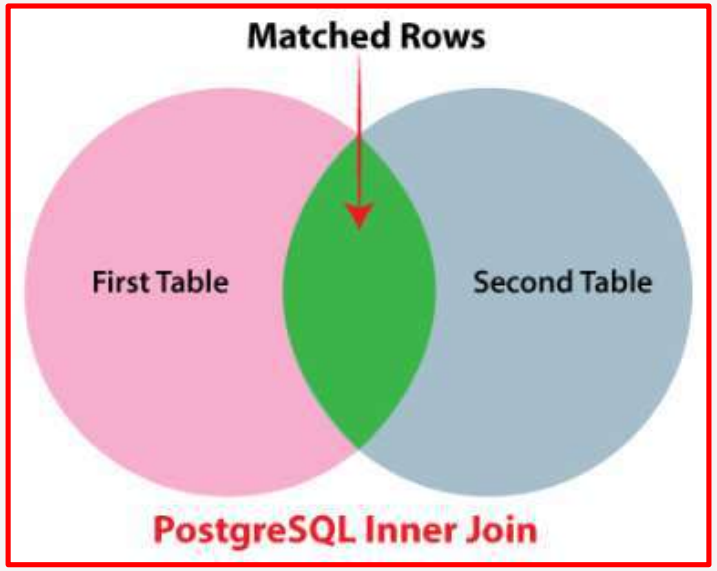
1) Inner Join
2) Left Outer Join
3) Right Outer Join
4) Full Outer Join
5) Cross Join

JOINING RELATIONS

1) Inner Join: The INNER JOIN keyword selects records that have matching values in both tables.

Syntax:

```
SELECT column_name(s)
FROM table1
INNER JOIN table2
ON table1.column_name = table2.column_name;
```



Example:

1. Orders table

OrderID	CustomerID	EmployeeID	OrderDate	ShipperID
10308	2	7	1996-09-18	3
10309	37	3	1996-09-19	1
10310	77	8	1996-09-20	2

JOINING RELATIONS

2. Customers table

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico

Query:

```
SELECT Orders.OrderID, Customers.CustomerName
FROM Orders
INNER JOIN Customers ON Orders.CustomerID = Customers.CustomerID;
```

Note: Before running the query, insert the records into the tables.

Output:

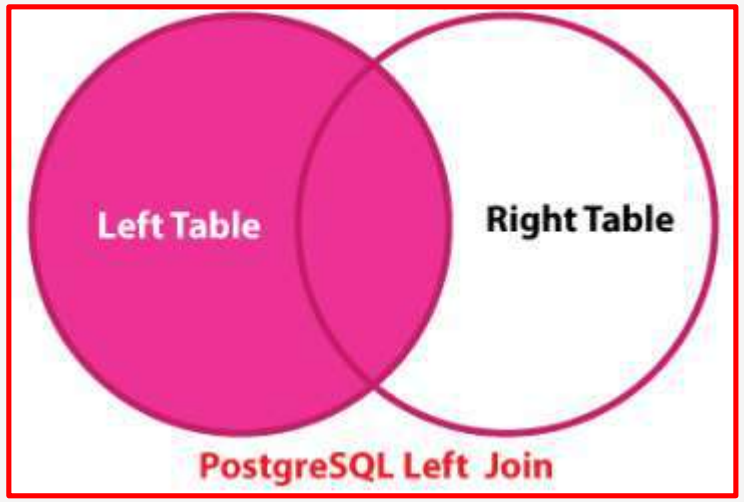
OrderID	CustomerName
10248	Wilman Kala
10249	Tradição Hipermercados
10250	Hanari Carnes
10251	Victuailles en stock
10252	Suprêmes délices

JOINING RELATIONS

2) Left Outer Join: In some databases LEFT OUTER JOIN is called LEFT JOIN. The LEFT JOIN keyword returns all records from the left table (table1), and the matching records from the right table (table2). The result is 0 records from the right side, if there is no match.

Syntax:

```
SELECT column_name(s)
FROM table1
LEFT JOIN table2
ON table1.column_name = table2.column_name;
```



JOINING RELATIONS

Query:

```
SELECT Customers.CustomerName, Orders.OrderID
FROM Customers
LEFT JOIN Orders ON Customers.CustomerID = Orders.CustomerID
ORDER BY Customers.CustomerName;
```

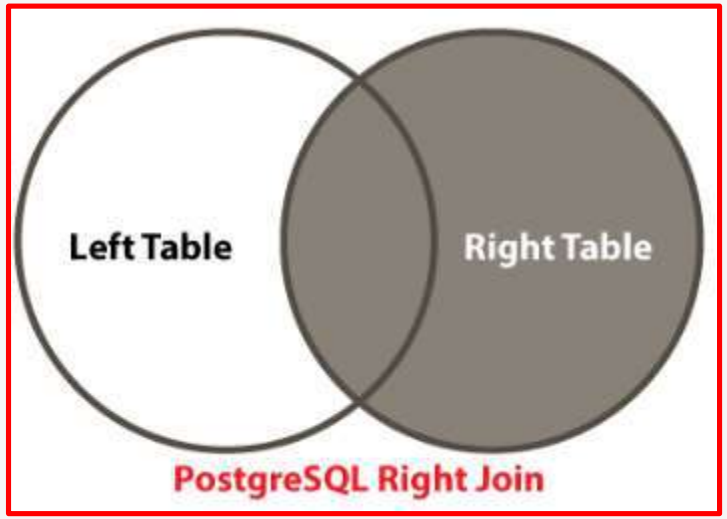
The LEFT JOIN keyword returns all records from the left table (Customers), even if there are no matches in the right table (Orders).

Output:

CustomerName	OrderID
Alfreds Futterkiste	*null*
Ana Trujillo Emparedados y helados	10308
Antonio Moreno Taquería	10365
Around the Horn	10355
Around the Horn	10383

JOINING RELATIONS

3) Right Outer Join: In some databases RIGHT OUTER JOIN is called RIGHT JOIN. The RIGHT JOIN keyword returns all records from the right table (table2), and the matching records from the left table (table1). The result is 0 records from the left side, if there is no match.



Syntax:

```
SELECT column_name(s)
FROM table1
RIGHT JOIN table2
ON table1.column_name = table2.column_name;
```

JOINING RELATIONS

Query:

```
SELECT Orders.OrderID, Employees.LastName, Employees.FirstName
FROM Orders
RIGHT JOIN Employees ON Orders.EmployeeID = Employees.EmployeeID
ORDER BY Orders.OrderID;
```

The RIGHT JOIN keyword returns all records from the right table (Employees), even if there are no matches in the left table (Orders).

Output:

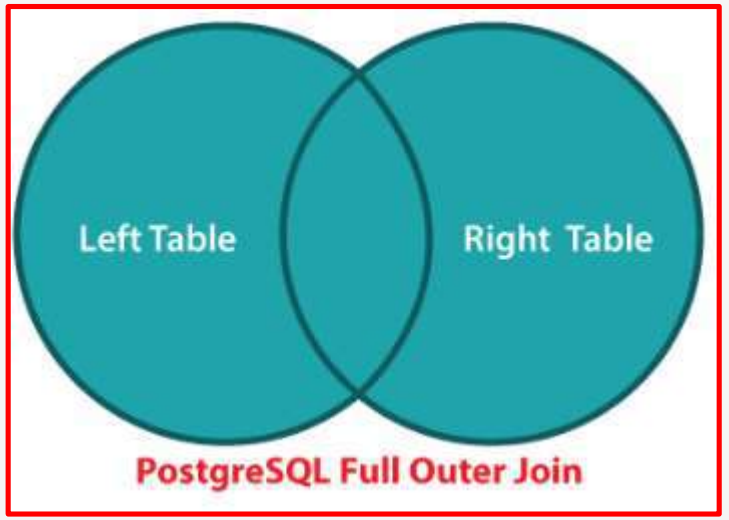
OrderID	LastName	FirstName
	West	Adam
10248	Buchanan	Steven
10249	Suyama	Michael
10250	Peacock	Margaret
10251	Leverling	Janet

JOINING RELATIONS

4) Full Outer Join: In some databases FULL OUTER JOIN is called FULL JOIN. The FULL OUTER JOIN keyword returns all records when there is a match in left (table1) or right (table2) table records. The FULL OUTER JOIN can potentially return very large result-sets.

Syntax:

```
SELECT column_name(s)
FROM table1
FULL OUTER JOIN table2
ON table1.column_name = table2.column_name
WHERE condition;
```



JOINING RELATIONS

Query:

```
SELECT Customers.CustomerName, Orders.OrderID
FROM Customers
FULL OUTER JOIN Orders ON Customers.CustomerID=Orders.CustomerID
ORDER BY Customers.CustomerName;
```

The FULL OUTER JOIN keyword returns all matching records from both tables whether the other table matches or not. So, if there are rows in "Customers" that do not have matches in "Orders", or if there are rows in "Orders" that do not have matches in "Customers", those rows will be listed as well.

Output:

CustomerName	OrderID
Null	10309
Null	10310
Alfreds Futterkiste	*Null*
Ana Trujillo Emparedados y helados	10308
Antonio Moreno Taquería	*Null*

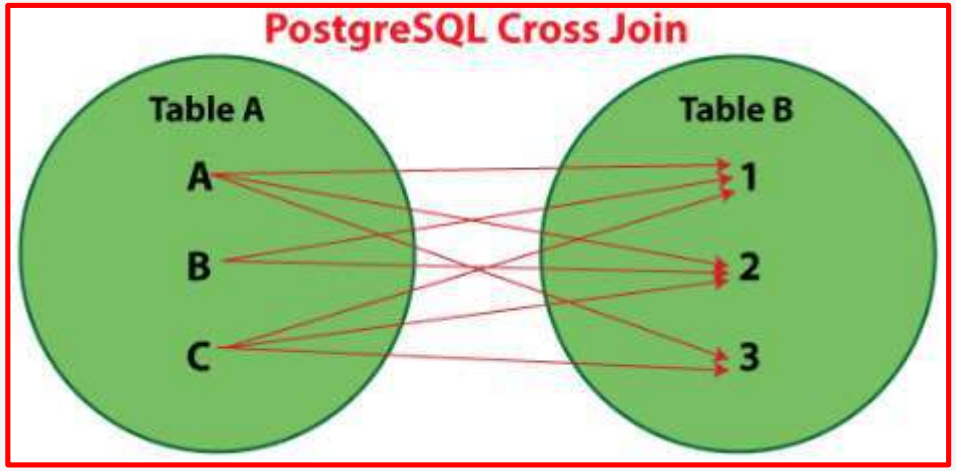
JOINING RELATIONS

5) Cross Join: CROSS JOIN is also known as the Cartesian product / Cartesian join. Cross join defines where the number of rows in the first table multiplied by a number of rows in the second table. Cross Join applies to all columns.

If we add a WHERE clause (if table1 and table2 has a relationship), the CROSS JOIN will produce the same result as the INNER JOIN clause:

Syntax:

```
SELECT column_name(s)
FROM table1
CROSS JOIN table2;
```



JOINING RELATIONS

Query:

```
SELECT Customers.CustomerName, Orders.OrderID
FROM Customers
CROSS JOIN Orders;
```

The CROSS JOIN keyword returns all matching records from both tables whether the other table matches or not. So, if there are rows in "Customers" that do not have matches in "Orders", or if there are rows in "Orders" that do not have matches in "Customers", those rows will be listed as well.

Output:

CustomerName	OrderID
Alfreds Futterkiste	10248
Ana Trujillo Emparedados y helados	10248
Antonio Moreno Taquería	10248
Around the Horn	10248
Berglunds snabbköp	10248

CREATING AND DROPPING VIEWS

- 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:

  a. Structure data in a way that users or classes of users find natural or intuitive.

  b. Restrict access to the data such that a user can only see limited data instead of complete table.

  c. Summarize data from various tables, which can be used to generate reports.

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.

Syntax:

```
CREATE [TEMP | TEMPORARY] VIEW view_name AS
SELECT column1, column2.....
FROM table_name
WHERE [condition];
```

CREATING AND DROPPING VIEWS

Example: Consider, the following COMPANY table is having the following records:

id	name	age	address	salary
1	Paul	32	California	20000
2	Allen	25	Texas	15000
3	Teddy	23	Norway	20000
4	Mark	25	Rich-Mond	65000
5	David	27	Texas	85000
6	Kim	22	South-Hall	45000
7	James	24	Houston	10000

To create a view from COMPANY table. This view would be used to have only few columns from COMPANY table:

```
testdb=# CREATE VIEW COMPANY_VIEW AS
SELECT ID, NAME, AGE
FROM  COMPANY;
```

We can query COMPANY_VIEW in a similar way as we query an actual table as the following:

```
testdb=# SELECT * FROM COMPANY_VIEW;
```

Output:

id	name	age
1	Paul	32
2	Allen	25
3	Teddy	23
4	Mark	25
5	David	27
6	Kim	22
7	James	24

(7 rows)

Dropping Views: To drop a view, simply use the DROP VIEW statement with the view_name. The basic DROP VIEW syntax is as follows:

```
testdb=# DROP VIEW COMPANY_VIEW;
```

COMPOUND STATEMENTS

A Compound Statement is in principal the essential block of the SQL/PSM language (PSM --> "Persistent, Stored Modules" allows us to store procedures as database schema elements. PSM = a mixture of conventional statements (if, while, etc.) and SQL). It enables us to enter the sequence of statements, declare local variables, conditions and subroutines, errors (exceptions) and warnings handling. The declaration of variables can not be intersected by the declaration of cursors, errors handling, etc. The variables can be set to the default value. The range of a compound statement is determined by the pair BEGIN END.

Example:

```
CREATE OR REPLACE FUNCTION report()
RETURNS void AS
$$
  bl:BEGIN
      DECLARE done boolean DEFAULT false;
      DECLARE a, b integer;
      DECLARE cx CURSOR FOR SELECT f.a, f.b FROM Foo f;
      DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = true;

      OPEN cx;
      FETCH cx INTO bl.a, bl.b;

      WHILE NOT done
      DO
        PRINT bl.a, bl.b;
        FETCH cx INTO bl.a, bl.b;
      END WHILE;

      CLOSE cx;
    END bl;
$$ LANGUAGE plpgpsm;
```

USER DEFINED FUNCTIONS

PostgreSQL uses the CREATE FUNCTION statement to develop user-defined functions.

Syntax:

```
CREATE FUNCTION function_name(p1 type, p2 type)
 RETURNS type AS
BEGIN
 -- logic
END;
LANGUAGE language_name;
```

Details of syntax:

a. First, specify the name of the function after the CREATE FUNCTION keywords.

b. Then, put a comma-separated list of parameters inside the parentheses following the function name.

c. Next, specify the return type of the function after the RETURNS keyword.

d. After that, place the code inside the BEGIN and END block. The function always ends with a semicolon (;) followed by the END keyword.

e. Finally, indicate the procedural language of the function e.g., plpgsql in case PL/pgSQL is used.

USER DEFINED FUNCTIONS

Example: We will develop a very simple function named **inc** that increases an integer by 1 and returns the result.

```
CREATE FUNCTION inc(val integer) RETURNS integer AS $$
BEGIN
RETURN val + 1;
END; $$
LANGUAGE PLPGSQL;
```

If the function is valid, PostgreSQL will create the function and return the CREATE FUNCTION statement as the following.

We can call the **inc** function like any built-in functions as follows:

Output

```
SELECT inc(20);
```

	inc integer
1	21

If we call the **inc** function 2 times (nested), the result is as the following:

```
SELECT inc(inc(20));
```

	inc integer
1	22

ACTIVITIES/ CASE STUDIES/ IMPORTANT FACTS RELATED TO THE SESSION

1) Create two tables (employee personal and payroll) and execute different types joins.

2) Create a view for a table student and execute it.

3) Create a user defined function and execute it.

EXAMPLES

1. INNER JOIN: The following statement joins the first table (basket_a) with the second table (basket_b) by matching the values in the fruit_a and fruit_b columns:

```
SELECT
    a,
    fruit_a,
    b,
    fruit_b
FROM
    basket_a
INNER JOIN basket_b
    ON fruit_a = fruit_b;
```

2. View: Consider the 'Price' table given below:

```
Demo=# SELECT * FROM Price;
 id | price
----+-------
  1 |   200
  2 |   250
  4 |   190
  3 |   300
(4 rows)

Demo=#
```

Let us create a view from the above table:

```
CREATE VIEW Price_View AS
  SELECT id, price
  FROM Price
  WHERE price > 200;
```

The above command will create a view based on the SELECT statement. Only the records where the price is greater than 200 will be added to the view. The view has been given the name Price_View. Let us query it to see its contents:

```
SELECT *
FROM Price_View;
```

```
Demo=# SELECT *
Demo-# FROM Price_View;
 id | price
----+-------
  2 |   250
  3 |   300
(2 rows)

Demo=#
```

SUMMARY

- The PostgreSQL Joins clause is used to combine records from two or more tables in a database. A JOIN is a means for combining fields from two tables by using values common to each.
- A PostgreSQL view is a logical table that represents data of one or more underlying tables through a SELECT statement.
- A user defined PostgreSQL function is a group of arbitrary SQL statements designated to perform some task.

SELF-ASSESSMENT QUESTIONS

1. Which join refers to join records from the write table that have no matching key in the left table are include in the result set:

(a) Left Outer Join

(b) Right Outer Join

(c) Full Outer Join

(d) None of the above

SELF-ASSESSMENT QUESTIONS

1. Which join refers to join records from the write table that have no matching key in the left table are include in the result set:

(a) Left Outer Join
(b) Right Outer Join
(c) Full Outer Join
(d) None of the above

Answer: B) Right Outer Join

SELF-ASSESSMENT QUESTIONS

1. Which join refers to join records from the write table that have no matching key in the left table are include in the result set:

(a) Left Outer Join
(b) Right Outer Join
(c) Full Outer Join
(d) None of the above

Answer: B) Right Outer Join

2. Which view that contains more than one table in the top-level FROM clause of the SELECT statement:

(a) Join View
(b) Datable Join View
(c) Updatable Join View
(d) All of the mentioned

SELF-ASSESSMENT QUESTIONS

1. Which join refers to join records from the write table that have no matching key in the left table are include in the result set:

(a) Left Outer Join
(b) Right Outer Join
(c) Full Outer Join
(d) None of the above

Answer: B) Right Outer Join

2. Which view that contains more than one table in the top-level FROM clause of the SELECT statement:

(a) Join View
(b) Datable Join View
(c) Updatable Join View
(d) All of the mentioned

Answer: C) Updatable Join View

TERMINAL QUESTIONS

1. Describe types of joins in PostgresQL.

2. How can you a view in PostgresQL?

3. Analyze user defined function with suitable example.

REFERENCES FOR FURTHER LEARNING OF THE SESSION

Reference Books:

1. Database System Concepts, Sixth Edition, Abraham Silberschatz, Yale University Henry, F. Korth Lehigh University, S. Sudarshan Indian Institute of Technology, Bombay.

2. Fundamentals of Database Systems, 7th Edition, RamezElmasri, University of Texas at Arlington, Shamkant B. Navathe, University of Texasat Arlington.

3. An Introduction to Database Systems by Bipin C. Desai

Sites and Web links:

1. https://www.javatpoint.com/sql-tutorial
2. https://www.tutorialspoint.com/sql/index.htm
3. https://www.programiz.com/sql /

THANK YOU

Team - Database Management System