CS232L – Database Management System

LAB#06

Postgres Joins

Ghulam Ishaq Khan Institute of Engineering Sciences

Faculty of Computer Science & Engineering

CS232L – Database Management system Lab

Lab 06 – Postgres Joins

Objective

The objective of this session is to

1. Introduction of Joins
2. Types of Joins
3. Examples

Instructions

- Open the handout/lab manual in front of a computer in the lab during the session.
- Practice each new command by completing the examples and exercise.
- Turn-in the answers for all the exercise problems as your lab report.
- When answering problems, indicate the commands you entered and the output displayed.

POSTGRES JOINS

A PostgreSQL Join statement is used to combine data or rows from one(self-join) or more tables based on a common field(or column) between them. These common fields are generally the **Primary key** of the first table and **Foreign key** of other tables.
There are 4 basic types of joins supported by PostgreSQL, namely:

1. Inner Join (or sometimes called simple join)
2. Outer Join
   - Left outer Join (or sometimes called LEFT JOIN)
   - Right outer Join (or sometimes called RIGHT JOIN)
   - Full Outer Join (or sometimes called FULL JOIN)

Some special PostgreSQL joins are below:

- Natural Join
- Cross Join (Cartesian Product)
- Self-Join
- Equijoins
- Antijoins
- Semijoins

1. INNER JOIN

PostgreSQL inner join is also called as self-join. It is the most common type of join in PostgreSQL. This join returns all matching rows from multiple tables when the join condition is satisfied.The ON clause is used with join condition.

Syntax:
The syntax for the INNER JOIN in PostgreSQL is:

```
SELECT columns
FROM table1
INNER JOIN table2
ON table1.column = table2.column;
```

In this visual diagram, the PostgreSQL INNER JOIN returns the shaded area

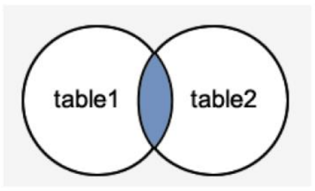
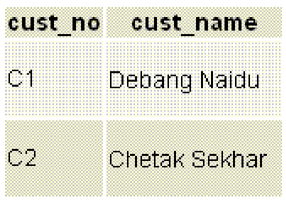


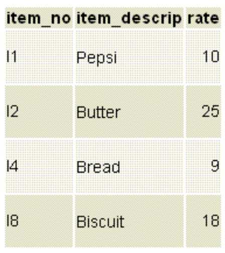
Image representation of Inner Join

EXAMPLE:

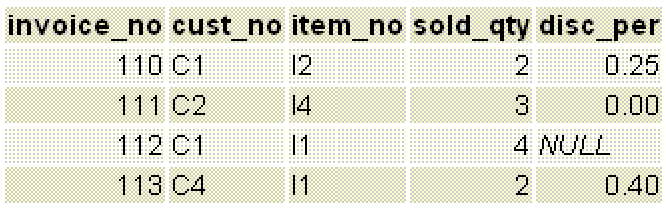
Sample table: **Customer** Sample table**: Item** Sample table**: Invoice**



cust_no	cust_name
C1	Debang Naidu
C2	Chetak Sekhar



item_no	item_descrip	rate
I1	Pepsi	10
I2	Butter	25
I4	Bread	9
I8	Biscuit	18

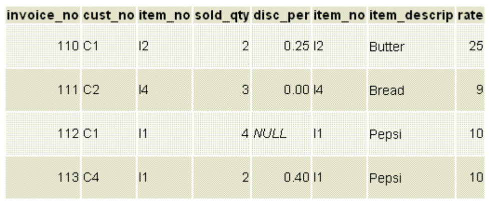


invoice_no	cust_no	item_no	sold_qty	disc_per
110	C1	I2	2	0.25
111	C2	I4	3	0.00
112	C1	I1	4	*NULL*
113	C4	I1	2	0.40

Example of Inner Join with ON clause

```
SELECT *

FROM invoice

INNER JOIN item

ON invoice.item_no=item.item_no;
```

Output:



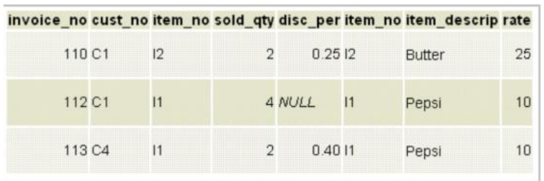
invoice_no	cust_no	item_no	sold_qty	disc_per	item_no	item_descrip	rate
110	C1	I2	2	0.25	I2	Butter	25
111	C2	I4	3	0.00	I4	Bread	9
112	C1	I1	4	*NULL*	I1	Pepsi	10
113	C4	I1	2	0.40	I1	Pepsi	10

4 row(s)

Example of Inner Join with ON and WHERE clause

```
SELECT *

FROM invoice

INNER JOIN item

ON invoice.item_no=item.item_no

WHERE

item.rate>=10;
```

Output:



invoice_no	cust_no	item_no	sold_qty	disc_per	item_no	item_descrip	rate
110	C1	I2	2	0.25	I2	Butter	25
112	C1	I1	4	*NULL*	I1	Pepsi	10
113	C4	I1	2	0.40	I1	Pepsi	10

3 row(s)

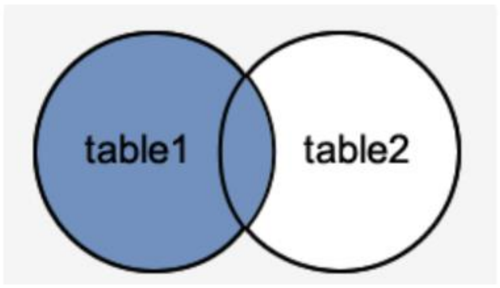
Outer Join

LEFT OUTER JOIN or LEFT JOIN

While joining the table using the left outer join, PostgreSQL first does normal join and then it starts scanning from the left table. **PostgreSQL left join retrieves** all rows from the left table and all matching rows from the right table. If there is no match in both tables, the right tables have null values.

Syntax:

```
Select *
FROM table1
LEFT [ OUTER ] JOIN table2
ON table1.column_name=table2.column_name
```



The PostgreSQL LEFT OUTER JOIN would return the all records from *table1* and only those records from *table2* that intersect with *table1*.

Example:

Code:

```
SELECT item.item_no,item_descrip,

invoice.invoice_no,invoice.sold_qty

FROM item

LEFT JOIN invoice

ON item.item_no=invoice.item_no;
```

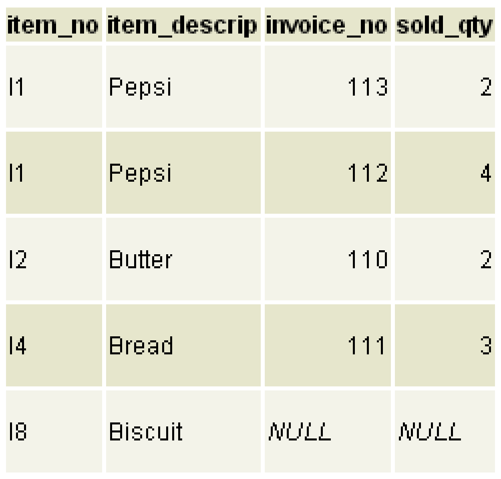
OR

Code:

```
SELECT item.item_no,item_descrip,

invoice.invoice_no,invoice.sold_qty
```

```
FROM item

LEFT OUTER JOIN invoice

ON item.item_no=invoice.item_no;
```

Output:



item_no	item_descrip	invoice_no	sold_qty
I1	Pepsi	113	2
I1	Pepsi	112	4
I2	Butter	110	2
I4	Bread	111	3
I8	Biscuit	*NULL*	*NULL*

Explanation :

In the above example, the item_no I8 of item table not exists in the invoice table, and for this rows of item table a new row in invoice table have generated and sets the NULL for this rows.

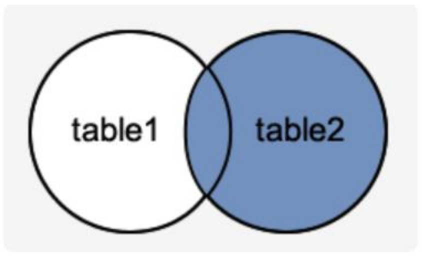
RIGHT OUTER JOIN OR RIGHT JOIN

While joining the table using the right outer join, PostgreSQL first does normal join and then it starts scanning from the right table. PostgreSQL right join retrieves all rows from the right table and all matching rows from the left table. If there is no match in both tables, the left tables have null values.

Syntax:

```
Select *
FROM table1
RIGHT [ OUTER ] JOIN table2
ON table1.column_name=table2.column_name;
```

In this visual diagram, the PostgreSQL RIGHT OUTER JOIN returns the shaded area:



The PostgreSQL RIGHT OUTER JOIN would return the all records from *table2* and only those records from *table1* that intersect with *table2*.

Example:

Code:

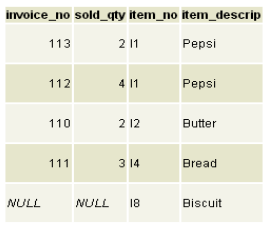
```
SELECT invoice.invoice_no,invoice.sold_qty,
item.item_no,item_descrip
FROM invoice
RIGHT JOIN item
ON item.item_no=invoice.item_no;
```

OR

Code:

```
SELECT invoice.invoice_no,invoice.sold_qty,
item.item_no,item_descrip
FROM invoice
RIGHT OUTER JOIN item
ON item.item_no=invoice.item_no;
```

Output:



invoice_no	sold_qty	item_no	item_descrip
113	2	I1	Pepsi
112	4	I1	Pepsi
110	2	I2	Butter
111	3	I4	Bread
NULL	*NULL*	I8	Biscuit

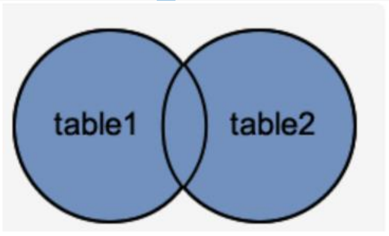
In the above example, the item_no I8 of item table not exists in the invoice table, and for this row in the item table, a new row have been generated in the invoice table and sets the value NULL for this row.

FULL OUTER JOIN OR FULL JOIN

PostgreSQL full outer join returns all rows from the left table as well as the right table. It will put null when the full outer join condition was not satisfied. While joining the table using FULL OUTER JOIN first, it will join be using an inner join. The combination of left and right join is known as a full outer join.

Syntax:

```
SELECT * | column_name(s)
FROM table_name1
FULL [OUTER] JOIN table_name2
ON table_name1.column_name=table_name2.column_name
```



The PostgreSQL FULL OUTER JOIN would return the all records from both *table1* and *table2*.

Example:

Code:

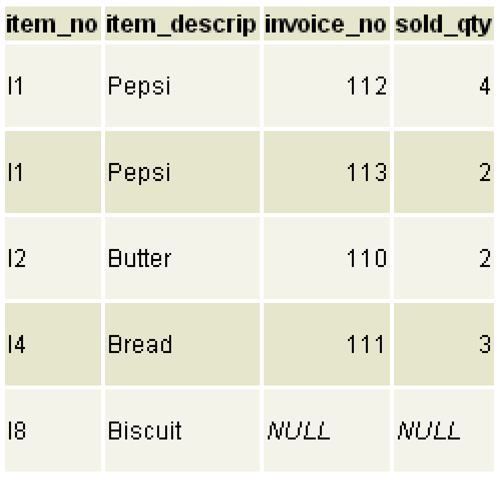
```
SELECT item.item_no,item_descrip,

invoice.invoice_no,invoice.sold_qty

FROM invoice

FULL JOIN item

ON invoice.item_no=item.item_no

ORDER BY item_no;
```

OR

Code:

```
SELECT item.item_no,item_descrip,
invoice.invoice_no,invoice.sold_qty
FROM invoice
FULL OUTER JOIN item
ON invoice.item_no=item.item_no
ORDER BY item_no;
```

Output:



item_no	item_descrip	invoice_no	sold_qty
I1	Pepsi	112	4
I1	Pepsi	113	2
I2	Butter	110	2
I4	Bread	111	3
I8	Biscuit	*NULL*	*NULL*

Explanation

In the above example, the matching rows from both the table item and invoice have appeared, as well as the unmatched row i.e. I8 of item table which is not exists in the invoice table have also appeared, and for this row of item table a new row in invoice table have generated and sets the value NULL .

PostgreSQL Cross Join

The Cross Join creates a cartesian product between two sets of data. This type of join does not maintain any relationship between the sets; instead returns the result, which is the number of rows in the first table multiplied by the number of rows in the second table. It is called a product because it returns every possible combination of rows between the joined sets.

Syntax:

```
SELECT [* | column_list]
FROM table1
CROSS JOIN table2;
```

OR

```
SELECT [* | column_list]
FROM table1,table2;
```

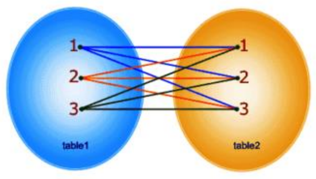


Image representation of cross join

EXAMPLE:

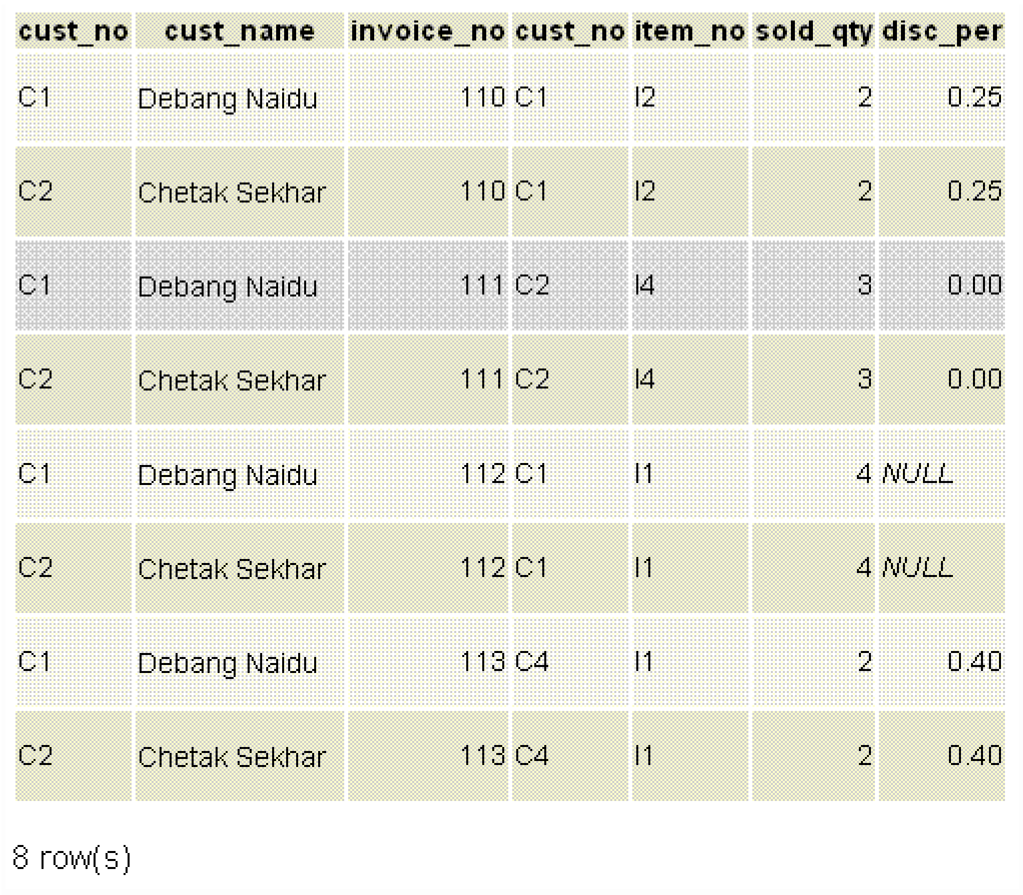
```
SELECT * FROM customer
CROSS JOIN
invoice;
```

OR

Code:

```
SELECT customer.cust_no, customer.cust_name,

invoice.invoice_no,invoice.cust_no,invoice.item_no,

invoice.sold_qty,invoice.disc_per

FROM customer,invoice;
```

Output:



cust_no	cust_name	invoice_no	cust_no	item_no	sold_qty	disc_per
C1	Debang Naidu	110	C1	I2	2	0.25
C2	Chetak Sekhar	110	C1	I2	2	0.25
C1	Debang Naidu	111	C2	I4	3	0.00
C2	Chetak Sekhar	111	C2	I4	3	0.00
C1	Debang Naidu	112	C1	I1	4	*NULL*
C2	Chetak Sekhar	112	C1	I1	4	*NULL*
C1	Debang Naidu	113	C4	I1	2	0.40
C2	Chetak Sekhar	113	C4	I1	2	0.40

8 row(s)

Explanation

In the above example, the 'customer' table and 'invoice' table join together to return a cartesian product. Here in the above example the two rows of 'customer' table joining with 4 rows of 'invoice' table and makes a product of 4*2 rows.

ANTI-JOINS

Even after learning the principles of inner, outer, left, and right joins, you might still have a nagging question: how do I find values from one table that are NOT present in another table?

That's where anti joins come in. They can be helpful in a variety of business situations when you're trying to find something that hasn't happened, such as:

- Customers who did not place an order
- Customers who have not visited your website
- Salespeople who did not close a deal

Anti-join will return rows, when no matching records are found in the second table. It is quite opposite to the semi join, since it is returning records from the first table, when there is no match in the second table.

Types of anti joins

There are two types of anti joins:

- A **left anti join** : This join returns rows in the left table that have no matching rows in the right table.
- A **right anti join** : This join returns rows in the right table that have no matching rows in the left table.

Syntax:

An anti join doesn't have its own syntax - meaning one actually performs an anti join using a combination of other postgres sql queries. To find all the values from Table_1 that are not in Table_2, you'll need to use a combination of LEFT JOIN and WHERE.

1. Select every column from Table_1. Assign Table_1 an alias: t1.
2. LEFT JOIN Table_2 to Table_1 and assign Table_2 an alias: t2. By using a LEFT JOIN, your query will return all values from Table_1 (whether or not they are also present in Table_2).
3. SELECT * FROM Table_1 t1 LEFT JOIN Table_2 t2 ON t1.id = t2.id
4. Use a WHERE clause to filter out values that *are* present in Table_2.

   SELECT * FROM Table_1 t1 LEFT JOIN Table_2 t2 ON t1.id = t2.id WHERE t2.id IS NULL

   The above entire query will return only values that are in Table_1 but not in Table_2.

   Example:

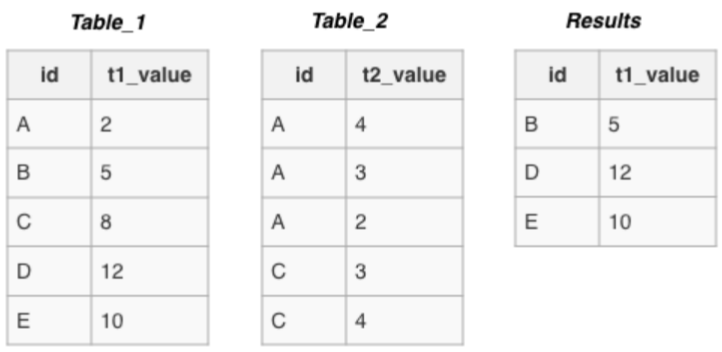


Table_1

id	t1_value
A	2
B	5
C	8
D	12
E	10

Table_2

id	t2_value
A	4
A	3
A	2
C	3
C	4

Results

id	t1_value
B	5
D	12
E	10

:

PostgreSQL optimizes both `LEFT JOIN / IS NULL` and `NOT EXISTS` in the same way.

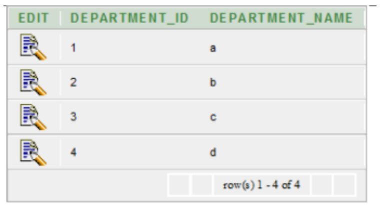
Semi-join:

Semi join will return a single value for all the matching records from the other table. That is, if the second table has multiple matching entries for the first table's record, then it will return only one copy from the first table. However, a normal join it will return multiple copies from the first table.

Semijoins are used for calculating the EXISTS predicate.

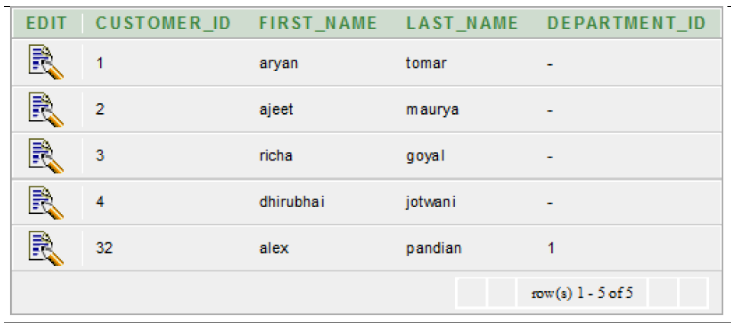
EXAMPLE:

Consider table depart and customer table



EDIT	DEPARTMENT_ID	DEPARTMENT_NAME
	1	a
	2	b
	3	c
	4	d

row(s) 1 - 4 of 4

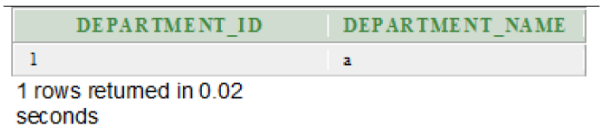


EDIT	CUSTOMER_ID	FIRST_NAME	LAST_NAME	DEPARTMENT_ID
	1	aryan	tomar	-
	2	ajeet	maurya	-
	3	richa	goyal	-
	4	dhirubhai	jotwani	-
	32	alex	pandian	1

row(s) 1 - 5 of 5

After applying semi-join

```
SELECT departments.department_id, departments.department_name
FROM departments
WHERE EXISTS
(
SELECT 1
FROM customer
WHERE customer.department_id = departments.department_id
)


ORDER BY departments.department_id;


Output:
```



DEPARTMENT_ID	DEPARTMENT_NAME
1	a

1 rows returned in 0.02 seconds

Semi or anti joins are kind of sub join types to the joining methods such as hash, merge, and nested loop, where the optimizer prefers to use them for EXISTS/IN or NOT EXISTS/NOT IN operators.

Difference between anti-join and semi-join

While a semi-join returns one copy of each row in the first table for which at least one match is found, an anti-join returns one copy of each row in the first table for which no match is found.

EQUI-JOINS:

Oracle Equi join returns the matching column values of the associated tables. It uses a comparison operator in the WHERE clause to refer equality.

Syntax:

```
SELECT column_list
FROM table1, table2....
WHERE table1.column_name
=
table2.column_name;
```

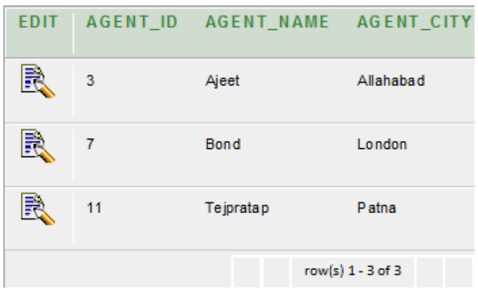
Equijoin also can be performed by using JOIN keyword followed by ON keyword and then specifying names of the columns along with their associated tables to check equality.

Syntax:

```
SELECT *
FROM table1
JOIN table2
[ON
(join_condition)]
```

EQUI JOIN Example

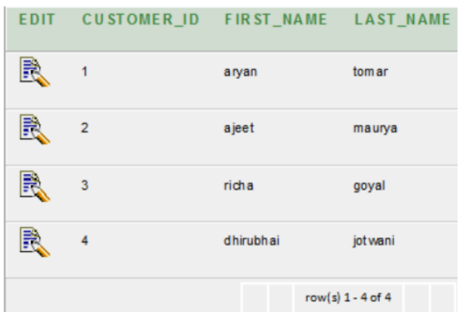
Consider Agent table



EDIT	AGENT_ID	AGENT_NAME	AGENT_CITY
	3	Ajeet	Allahabad
	7	Bond	London
	11	Tejpratap	Patna

row(s) 1 - 3 of 3

Consider customer table:



EDIT	CUSTOMER_ID	FIRST_NAME	LAST_NAME
	1	aryan	tomar
	2	ajeet	maurya
	3	richa	goyal
	4	dhirubhai	jotwani

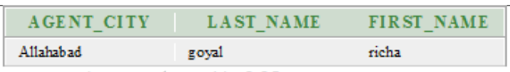
row(s) 1 - 4 of 4

Execute this query

Syntax:

```
SELECT agents.agent_city,customer.last_name,
customer.first_name
FROM agents,customer
WHERE agents.agent_id=customer.customer_id;
```

Output



AGENT_CITY	LAST_NAME	FIRST_NAME
Allahabad	goyal	richa

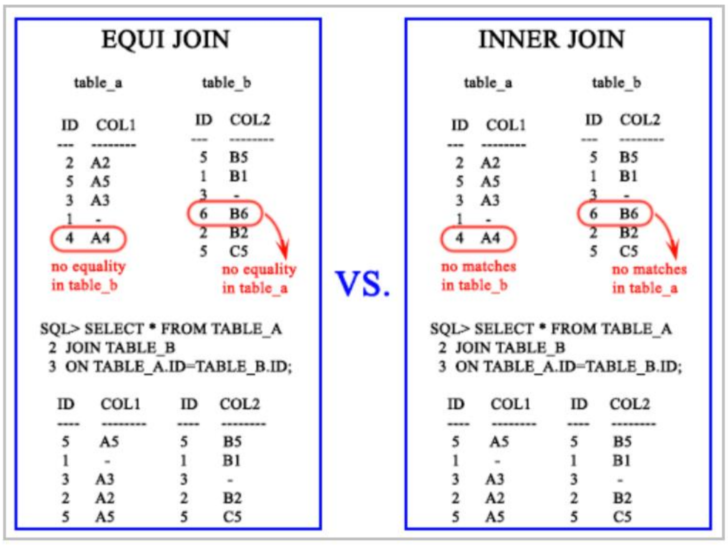
1 rows returned in 0.02 seconds

What is the difference between Equi Join and Inner Join in SQL?

An equijoin is a join with a join condition containing an equality operator. An equijoin returns only the rows that have equivalent values for the specified columns.

An inner join is a join of two or more tables that returns only those rows (compared using a comparison operator) that satisfy the join condition.

Pictorial presentation : SQL Equi Join Vs. SQL Inner Join



EQUI JOIN

```
  table_a        table_b

 ID  COL1       ID  COL2
 --- --------   --- --------
  2  A2          5  B5
  5  A5          1  B1
  3  A3          3  -
  1  -           6  B6
  4  A4          2  B2
                 5  C5

SQL> SELECT * FROM TABLE_A
  2  JOIN TABLE_B
  3  ON TABLE_A.ID=TABLE_B.ID;

 ID    COL1      ID    COL2
 ----  --------  ----  --------
 5     A5        5     B5
 1     -         1     B1
 3     A3        3     -
 2     A2        2     B2
 5     A5        5     C5
```

INNER JOIN

```
  table_a        table_b

 ID  COL1       ID  COL2
 --- --------   --- --------
  2  A2          5  B5
  5  A5          1  B1
  3  A3          3  -
  1  -           6  B6
  4  A4          2  B2
                 5  C5

SQL> SELECT * FROM TABLE_A
  2  JOIN TABLE_B
  3  ON TABLE_A.ID=TABLE_B.ID;

 ID    COL1      ID    COL2
 ----  --------  ----  --------
 5     A5        5     B5
 1     -         1     B1
 3     A3        3     -
 2     A2        2     B2
 5     A5        5     C5
```

SELF JOIN

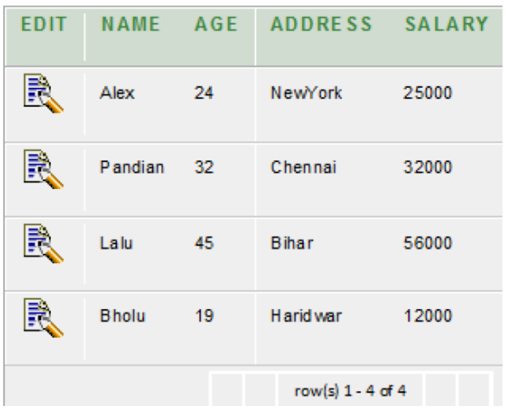
Self Join is a specific type of Join. In Self Join, a table is joined with itself (Unary relationship). A self join simply specifies that each rows of a table is combined with itself and every other row of the table.

Syntax:

```
SELECT a.column_name,
b.column_name...
FROM table1 a, table1 b
WHERE a.common_filed =
b.common_field;
```

SELF JOIN Example

Let's take a table "customers".



EDIT	NAME	AGE	ADDRESS	SALARY
✎	Alex	24	NewYork	25000
✎	Pandian	32	Chennai	32000
✎	Lalu	45	Bihar	56000
✎	Bholu	19	Haridwar	12000

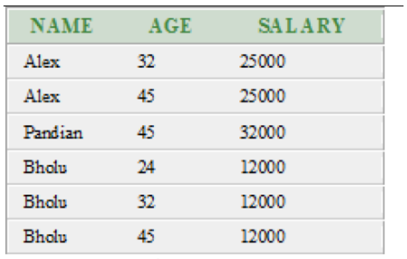
row(s) 1 - 4 of 4

Join this table using SELF JOIN as follows:

Syntax:

```
SELECT a.name, b.age, a.SALARY
FROM CUSTOMERS a, CUSTOMERS b
WHERE a.SALARY < b.SALARY;
```

Output:



NAME	AGE	SALARY
Alex	32	25000
Alex	45	25000
Pandian	45	32000
Bholu	24	12000
Bholu	32	12000
Bholu	45	12000

6 rows returned in 0.02 seconds

Natural Join:

A natural join is a join that creates an implicit join based on the same column names in the joined tables.

The following shows the syntax of the PostgreSQL natural join:

```
SELECT select_list
FROM T1
```

```sql
NATURAL [INNER, LEFT, RIGHT] JOIN T2;
```
Code language: SQL (Structured Query Language) (sql)

A natural join can be an inner join, left join, or right join. If you do not specify a join explicitly e.g., INNER JOIN, LEFT JOIN, RIGHT JOIN, PostgreSQL will use the INNER JOIN by default.