# Database Management Systems.

Lecture 4

# Content:

## Joining Multiple Tables

1. **Inner Join**
2. **Left Join**
3. **Right Join**
4. **Outer Join**
5. **Self Join**
6. **Cross Join**
7. **Natural Join**

# JOINS

- PostgreSQL **JOIN** is used to combine columns from one or more tables based on the values of the common columns between related tables.
- The common columns are typically the primary key columns of the first table and foreign key columns of the second table.
- PostgreSQL supports **inner join, left join, right join, full outer join, cross join, natural join**, and a special kind of join called **self-join**.

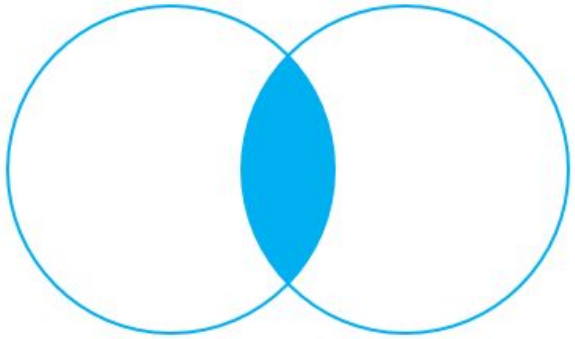
# INNER JOIN

- The **INNER JOIN** keyword selects all rows from both the tables if the condition satisfies.
- This keyword will create the result-set by combining all rows from both the tables where the condition satisfies i.e value of the common field will be same.

- **Basic syntax:**

```
SELECT table1.column1,table1.column2,table2.column1,....
FROM table1 INNER JOIN table2
ON table1.matching_column = table2.matching_column;
```

The following Venn diagram illustrates how INNER JOIN clause works:



INNER JOIN

# Example:

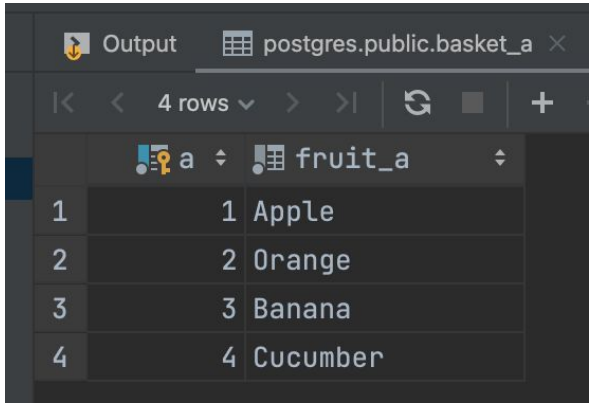
- Suppose you have two tables called basket_a and basket_b and that store fruits:

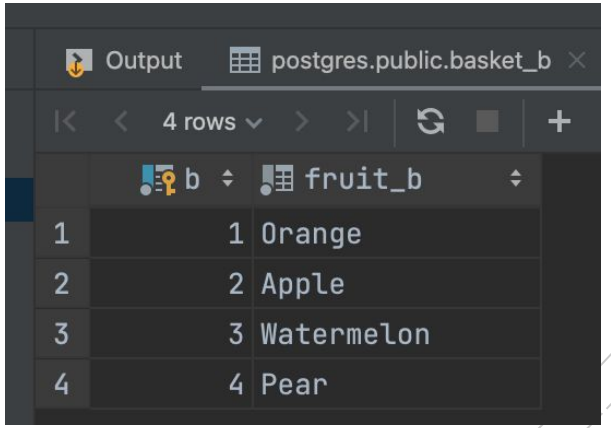
```
CREATE TABLE basket_a (
    a INT PRIMARY KEY,
    fruit_a VARCHAR (100) NOT NULL
);

CREATE TABLE basket_b (
    b INT PRIMARY KEY,
    fruit_b VARCHAR (100) NOT NULL
);

INSERT INTO basket_a (a, fruit_a)
VALUES
    (1, 'Apple'),
    (2, 'Orange'),
    (3, 'Banana'),
    (4, 'Cucumber');

INSERT INTO basket_b (b, fruit_b)
VALUES
    (1, 'Orange'),
    (2, 'Apple'),
    (3, 'Watermelon'),
    (4, 'Pear');
```




- The tables have some common fruits such as apple and orange.

# Example:

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

2 rows

| | a | fruit_a | b | fruit_b |
|---|---|---|---|---|
| 1 | 1 | Apple | 2 | Apple |
| 2 | 2 | Orange | 1 | Orange |

The inner join examines each row in the first table (basket_a).
It compares the value in the fruit_a column with the value
in the fruit_b column of each row in the second table (basket_b).
If these values are equal, the inner join creates a new row
that contains columns from both tables and adds this new row the result set.

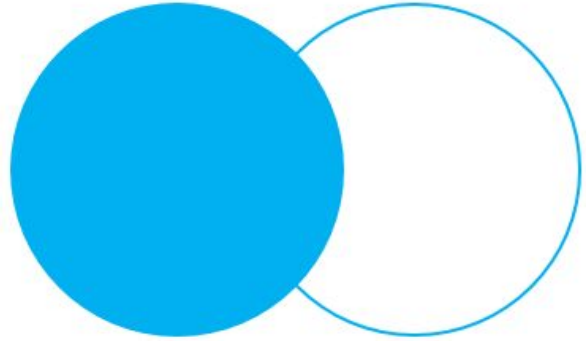
# LEFT JOIN

- This join returns all the rows of the table on the left side of the join and matching rows for the table on the right side of join.
- The rows for which there is no matching row on right side, the result-set will contain *null*.
- **LEFT JOIN** is also known as **LEFT OUTER JOIN**

- **Basic syntax:**

```
SELECT table1.column1,table1.column2,table2.column1,....
FROM table1 LEFT JOIN table2
ON table1.matching_column = table2.matching_column;
```

The following Venn diagram illustrates how LEFT JOIN clause works:



LEFT OUTER JOIN

# Example:

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

Output | Result 24

4 rows

| | a | fruit_a | b | fruit_b |
|---|---|---|---|---|
| 1 | 1 | Apple | 2 | Apple |
| 2 | 2 | Orange | 1 | Orange |
| 3 | 3 | Banana | <null> | <null> |
| 4 | 4 | Cucumber | <null> | <null> |

- The left join starts selecting data from the left table. It compares values in the fruit_a column with the values in the fruit_b column in the basket_b table.
- If these values are equal, the left join creates a new row that contains columns of both tables and adds this new row to the result set. (see the row #1 and #2 in the result set).
- In case the values do not equal, the left join also creates a new row that contains columns from both tables and adds it to the result set. However, it fills the columns of the right table (basket_b) with null. (see the row #3 and #4 in the result set).

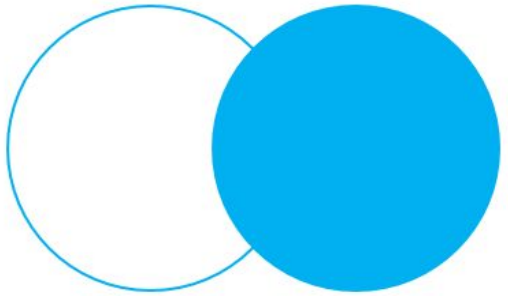
# RIGHT JOIN

- **RIGHT JOIN** is similar to LEFT JOIN.
- This join returns all the rows of the table on the right side of the join and matching rows for the table on the left side of join.
- The rows for which there is no matching row on left side, the result-set will contain *null*.
- **RIGHT JOIN** is also known as **RIGHT OUTER JOIN**

- **Basic syntax:**

```
SELECT table1.column1,table1.column2,table2.column1,....
FROM table1 RIGHT JOIN table2
ON table1.matching_column = table2.matching_column;
```

The following Venn diagram illustrates how RIGHT JOIN clause works:



RIGHT OUTER JOIN

# Example:

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

4 rows

| | a | fruit_a | b | fruit_b |
|---|---|---|---|---|
| 1 | 2 | Orange | 1 | Orange |
| 2 | 1 | Apple | 2 | Apple |
| 3 | <null> | <null> | 3 | Watermelon |
| 4 | <null> | <null> | 4 | Pear |

- The right join is a reversed version of the left join. The right join starts selecting data from the right table. It compares each value in the fruit_b column of every row in the right table with each value in the fruit_a column of every row in the fruit_a table.
- If these values are equal, the right join creates a new row that contains columns from both tables.
- In case these values are not equal, the right join also creates a new row that contains columns from both tables. However, it fills the columns in the left table with NULL.

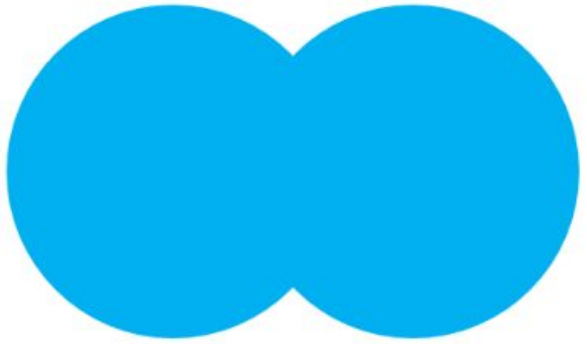
# FULL JOIN

- **FULL JOIN** creates the result-set by combining result of both LEFT JOIN and RIGHT JOIN.
- The result-set will contain all the rows from both the tables.
- The rows for which there is no matching, the result-set will contain *NULL* values

- **Basic syntax:**

```
SELECT table1.column1,table1.column2,table2.column1,....
FROM table1 FULL JOIN table2
ON table1.matching_column = table2.matching_column;
```

The following Venn diagram illustrates how FULL JOIN clause works:



FULL OUTER JOIN

# Example:

```sql
SELECT
    a,
    fruit_a,
    b,
    fruit_b
FROM
    basket_a
FULL OUTER JOIN basket_b
    ON fruit_a = fruit_b;
```

6 rows

| | a | fruit_a | b | fruit_b |
|---|---|---|---|---|
| 1 | 1 | Apple | 2 | Apple |
| 2 | 2 | Orange | 1 | Orange |
| 3 | 3 | Banana | <null> | <null> |
| 4 | 4 | Cucumber | <null> | <null> |
| 5 | <null> | <null> | 3 | Watermelon |
| 6 | <null> | <null> | 4 | Pear |

- The full outer join or full join returns a result set that contains all rows from both left and right tables, with the matching rows from both sides if available.
- In case there is no match, the columns of the table will be filled with NULL.

# CROSS JOIN

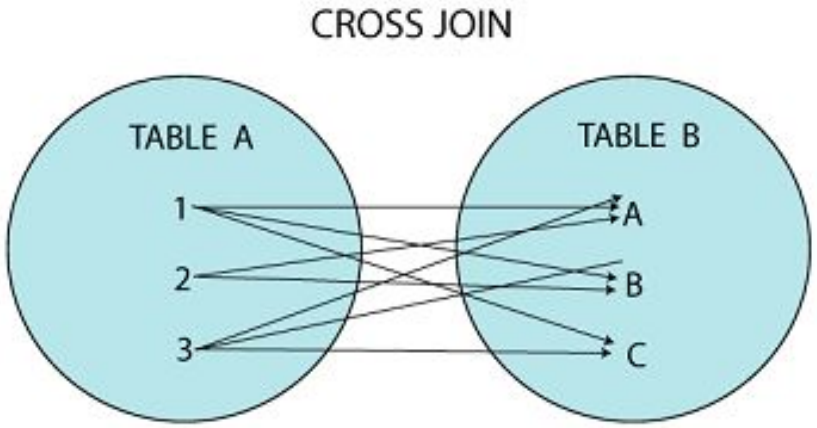
- A **CROSS JOIN** clause allows you to produce a Cartesian Product of rows in two or more tables.
- Different from other join clauses such as LEFT JOIN or INNER JOIN, the CROSS JOIN clause does not have a join predicate.

- **Basic syntax:**

```
SELECT select_list
FROM T1 CROSS JOIN T2;
```

OR

```
SELECT select_list
FROM T1, T2;
```



# Example:

```
select * from basket_a cross join basket_b
where basket_a.fruit_a = basket_b.fruit_b;
```

2 rows

| | a | fruit_a | b | fruit_b |
|---|---|---|---|---|
| 1 | 1 | Apple | 2 | Apple |
| 2 | 2 | Orange | 1 | Orange |

- In this case CROSS JOIN works like INNER JOIN

# NATURAL JOIN

- A **NATURAL JOIN** is a join that creates an implicit join based on the same column names in the joined tables.
- A **NATURAL JOIN** can be an inner join or 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.
- If you use the asterisk (*) in the select list, the result will contain the following columns:
  - All the common columns, which are the columns from both tables that have the same name.
  - Every column from both tables, which is not a common column.

- **Basic syntax:**

```
SELECT select_list
FROM T1 NATURAL [INNER, LEFT, RIGHT] JOIN T2;
```

**equivalent to:**

```
SELECT  select_list FROM T1
INNER JOIN T2 USING (matching_column);
```

# Example:

```sql
DROP TABLE IF EXISTS categories;
CREATE TABLE categories (
    category_id serial PRIMARY KEY,
    category_name VARCHAR (255) NOT NULL
);

DROP TABLE IF EXISTS products;
CREATE TABLE products (
    product_id serial PRIMARY KEY,
    product_name VARCHAR (255) NOT NULL,
    category_id INT NOT NULL,
    FOREIGN KEY (category_id) REFERENCES categories (category_id)
);
```

```sql
INSERT INTO categories (category_name)
VALUES
    ('Smart Phone'),
    ('Laptop'),
    ('Tablet');

INSERT INTO products (product_name, category_id)
VALUES
    ('iPhone', 1),
    ('Samsung Galaxy', 1),
    ('HP Elite', 2),
    ('Lenovo Thinkpad', 2),
    ('iPad', 3),
    ('Kindle Fire', 3);
```

3 rows Tx:

| | category_id | category_name |
|---|---|---|
| 1 | 1 | Smart Phone |
| 2 | 2 | Laptop |
| 3 | 3 | Tablet |

6 rows Tx: Auto DDL

| | product_id | product_name | category_id |
|---|---|---|---|
| 1 | 1 | iPhone | 1 |
| 2 | 2 | Samsung Galaxy | 1 |
| 3 | 3 | HP Elite | 2 |
| 4 | 4 | Lenovo Thinkpad | 2 |
| 5 | 5 | iPad | 3 |
| 6 | 6 | Kindle Fire | 3 |

# Example:

```
select * from products natural join categories;
```

| | category_id | product_id | product_name | category_name |
|---|---|---|---|---|
| 1 | 1 | 1 | iPhone | Smart Phone |
| 2 | 1 | 2 | Samsung Galaxy | Smart Phone |
| 3 | 2 | 3 | HP Elite | Laptop |
| 4 | 2 | 4 | Lenovo Thinkpad | Laptop |
| 5 | 3 | 5 | iPad | Tablet |
| 6 | 3 | 6 | Kindle Fire | Tablet |

```
select * from products inner join categories
1..n<->1: using (category_id);
```

# SELF JOIN

- A **self-join** is a regular join that joins a table to itself.
- In practice, you typically use a self-join to query hierarchical data or to compare rows within the same table.
- To form a self-join, you specify the same table twice with different table aliases and provide the join predicate after the ON keyword.
- The following query uses an INNER JOIN that joins the table to itself:

```
SELECT select_list

FROM table_name t1 INNER JOIN table_name t2
ON join_predicate;
```

- Also, you can use the LEFT JOIN or RIGHT JOIN clause to join table to itself like this:

```
SELECT select_list

FROM table_name t1 LEFT JOIN table_name t2
ON join_predicate;
```