S34: SQL JOINS

WHAT ARE SQL JOINS?

- In **SQL**, a join is a way to combine data from two or more database tables based on a related column between them.
- Joins are used when we want to query information that is distributed across multiple tables in a database, and the information we need is not contained in a single table. By joining tables together, we can create a virtual table that contains all of the information we need for our query.
- **But why have data in multiple tables?**
  - ✓ **Data Redundancy:** repeated the data
  - ✓ **Reduce anomaly**: like Updating/ delete the info etc.
  - ✓ **Maintaining Data Integrity:** Relationships between tables ensure that the data remains consistent.
  - ✓ **Database normalization:** It is the process of organizing a database into tables to minimize redundancy and dependency. The main objectives of normalization are to reduce data redundancy, improve data integrity, and simplify data management.

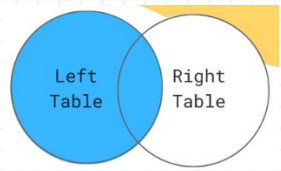
ON CLAUSE

- In **MySQL**, the **ON** clause is used in **JOIN** operations to specify the condition on which the join should be based.
- This condition determines how rows from one table will be matched with rows from another table.
- The **ON** clause typically includes a comparison between columns from the tables being joined.

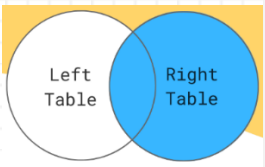
TYPES OF SQL JOINS

- Here are several types of SQL joins, each serving a specific purpose:

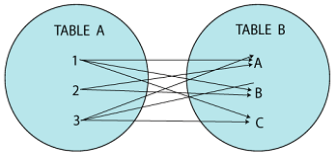
LEFT JOIN



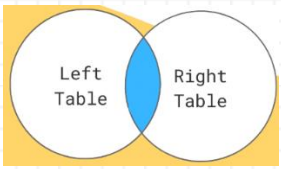
RIGHT JOIN



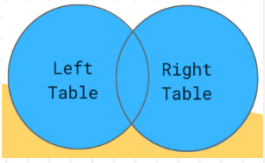
CROSS JOIN



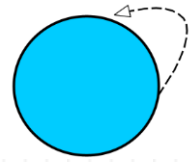
INNER JOIN



FULL JOIN



SELF JOIN



CROSS JOIN

- In SQL, a cross join (also known as a Cartesian product) is a type of join that returns the Cartesian product of the two tables being joined.
- **In other words,** it returns all possible combinations of rows from the two tables.
- **Practical scenarios:** Cross joins are not commonly used in practice, but they can be useful in certain scenarios, such as generating test data or exploring all possible combinations of items in a product catalogue. However, it's important to be cautious when using cross joins with large tables, as they can generate a very large result set, which can be resource-intensive and slow to process.

CROSS JOIN KEYWORD

- The **CROSS JOIN** keyword returns all records from both tables (table1 and table2).
- **Syntax:**

```
SELECT col1, col2,... FROM table1
CROSS JOIN table2;
```

- **Example:**

Table 1 (Left Table)

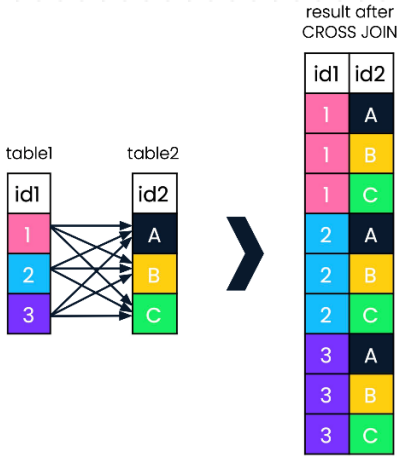
Column1	Column2
A	1
B	2
C	3

Table 2 (Right Table)

Column3	Column4
X	Y
Z	W

Result

Column 1	Column 2	Column 3	Column 4
A	1	X	Y
A	1	Z	W
B	2	X	Y
B	2	Z	W
C	3	X	Y
C	3	Z	W



NOTE: **CROSS JOIN** can potentially return very large result-sets

INNER JOIN

- In **SQL**, an inner join is a type of join operation that combines data from two or more tables based on a specified condition.
- The inner join returns only the rows from both tables that satisfy the specified condition, i.e., the matching rows.
- When you perform an inner join on two tables, the result set will only contain rows where there is a match between the joining columns in both tables. If there is no match, then the row will not be included in the result set.

INNER JOIN KEYWORD

- The **INNER JOIN** keyword selects records that have matching values in both tables.
- **Syntax:**

```
SELECT col1, col2,...
FROM table1 [AS alias_name]
INNER JOIN table2 [AS alias_name]
ON table1.column_name = table2.column_name;
```

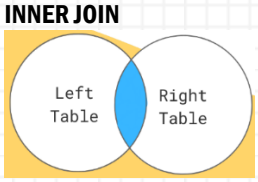
- **Example:**

Emp (Left Table)

Emp_ID	Name	Dept_ID	Salary
1	John Smith	1	100000
2	Jane Doe	2	50000
3	Bob Johnson	3	75000
4	Lisa Wong	1	90000
5	Mike Lee	2	120000
6	Time Davis	4	60000
7	Sarah Chen	NULL	80000

Dept (Right Table)

Dept_ID	Dept_name
1	Engineering
2	Sales
3	Finance
4	Marketing
5	Operations



Result:

Emp_ID	Name	Dept_ID	Salary	Dept_ID	Dept_name
1	John Smith	1	100000	1	Engineering
2	Jane Doe	2	50000	2	Sales
3	Bob Johnson	3	75000	3	Finance
4	Lisa Wong	1	90000	1	Engineering
5	Mike Lee	2	120000	2	Sales
6	Time Davis	4	60000	4	Marketing

LEFT JOIN / LEFT OUTER JOIN

- A **left join** (or **left outer join**) in SQL that returns all the rows from the left table (also known as the "first" table) and matching rows from the right table (also known as the "second" table).
- If there are no matching rows in the right table, the result will contain NULL values in the columns that come from the right table.

- **In other words,** a left join combines the rows from both tables based on a common column, but it also includes all the rows from the left table, even if there are no matches in the right table.
- This is useful when you want to include all the records from the first table, but only some records from the second table.

LEFT JOIN KEYWORD

- The **LEFT JOIN** keyword returns all records from the left table (table1), and the matching records (if any) from the right table (table2).
- **Syntax:**

```
SELECT col1, col2,...
FROM table1 [AS alias_name]
LEFT JOIN table2 [AS alias_name]
ON table1.column_name = table2.column_name;
```

- **Example:**

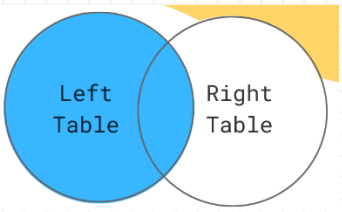
Emp (Left Table)

Emp_ID	Name	Dept_ID	Salary
1	John Smith	1	100000
2	Jane Doe	2	50000
3	Bob Johnson	3	75000
4	Lisa Wong	1	90000
5	Mike Lee	2	120000
6	Time Davis	4	60000
7	Sarah Brown	5	80000
8	Mak Wilson	7	95000

Dept (Right Table)

Dept_ID	Dept_name
1	Engineering
2	Sales
3	Finance

LEFT JOIN



Result:

Emp_ID	Name	Dept_ID	Salary	Dept_ID	Dept_name
1	John Smith	1	100000	1	Engineering
2	Jane Doe	2	50000	2	Sales
3	Bob Johnson	3	75000	3	Finance
4	Lisa Wong	1	90000	1	Engineering
5	Mike Lee	2	120000	2	Sales
6	Time Davis	4	60000	NULL	NULL
7	Sarah Brown	5	80000	NULL	NULL
8	Mak Wilson	7	95000	2	Sales

RIGHT JOIN

- A **right join** (or **right outer join**) in **SQL** that returns all the rows from the right table and matching rows from the left table.
- If there are no matches in the left table, the result will still contain all the rows from the right table, with **NULL** values for the columns from the left table.

RIGHT JOIN KEYWORD

- The **RIGHT JOIN** keyword returns all records from the right table (table2), and the matching records (if any) from the left table (table1).
- **Syntax:**

```
SELECT col1, col2,... FROM table1 [AS alias_name]
RIGHT JOIN table2 [AS alias_name]
ON table1.column = table2.column;
```

- **Example:**

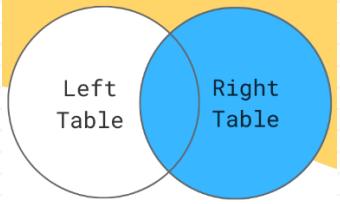
Emp (Left Table)

Emp_ID	Name	Dept_ID	Salary
1	John Smith	1	100000
2	Jane Doe	2	50000
3	Bob Johnson	3	75000
4	Lisa Wong	1	90000
5	Mike Lee	2	120000
6	Sarah Brown	NULL	80000
7	Mak Wilson	2	95000

Dept (Right Table)

Dept_ID	Dept_name
1	Engineering
2	Sales
3	Finance
4	Marketing
5	HR

RIGHT JOIN



Result:

Emp_ID	Name	Dept_ID	Salary	Dept_name
1	John Smith	1	100000	Engineering
4	Lisa Wong	1	50000	Engineering
2	Jane Doe	2	75000	Sales
5	Mike Lee	2	90000	Sales
7	Mike Lee	2	120000	Sales
3	Bob Johnson	3	60000	Finance
NULL	NULL	4	NULL	Marketing
NULL	NULL	5	NULL	HR

FULL JOIN / FULL OUTER JOIN

- A **full join** (or **full outer join**) is a type of join operation in **SQL** that returns all matching rows from both the left and right tables, as well as any non- matching rows from either table.
- **In other words,** a full outer join returns all the rows from both tables and matches rows with common values in the specified columns, and fills in **NULL** values for columns where there is no match.

  NOTE: **MySQL** does not support the **FULL OUTER JOIN** directly. However, you can achieve the equivalent result using a combination of **LEFT JOIN, RIGHT JOIN,** and **UNION.**

- **Syntax:**

```
SELECT col1, col2,... FROM table1 [AS alias_name]
LEFT JOIN table2 [AS alias_name]
ON table1.column = table2.column;
UNION
SELECT col1, col2,... FROM table1 [AS alias_name]
RIGHT JOIN table2 [AS alias_name]
ON table1.column = table2.column;
```

- **Example:**

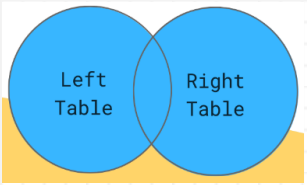
Emp (Left Table)

Emp_ID	Emp_Name	Dept_ID
1	John Smith	1
2	Jane Doe	1
3	Bob Johnson	2
4	Lisa Wong	NULL
5	Mike Lee	3

Dept (Right Table)

Dept_ID	Dept_name
1	Sales
2	Marketing
3	Finance
4	IT
5	Operations

FULL OUTER JOIN / FULL JOIN



Result:

Emp_ID	Name	Dept_ID	Dept_ID	Dept_name
1	John Smith	1	1	Sales
2	Jane Doe	1	1	Sales
3	Bob Johnson	2	2	Marketing
4	Lisa Wong	NULL	NULL	NULL
5	Mike Lee	3	3	Finance
NULL	NULL	NULL	4	IT
NULL	NULL	NULL	5	Marketing

SQL SET OPERATIONS

- Two tables to perform the set operations:

person1

id	name
1	Alice
2	Bob
3	Charlie

person2

id	name
3	Charlie
4	David
5	Emily

- **UNION**
  - ✓ The **UNION** operator is used to combine the results of two or more **SELECT** statements into a single result set. It removes duplicate rows between the various **SELECT** statements.
  - ✓ **Syntax:**

```
SELECT col1, col2,... FROM table1
UNION
SELECT col1, col2,... FROM table2;
```

  - ✓ **Example:**

query:

```
SELECT * FROM sql_cx_live.person1
UNION
SELECT * FROM sql_cx_live.person2;
```

output:

id	name
1	Alice
2	Bob
3	Charlie
4	David
5	Emily

- **UNION ALL**
  - ✓ The **UNION ALL** operator is similar to the **UNION** operator, but it does not remove duplicate rows from the result set.
  - ✓ **Syntax:**

```
SELECT col1, col2,... FROM table1
UNION ALL
SELECT col1, col2,... FROM table2;
```

  - ✓ **Example:**

query:

```
SELECT * FROM sql_cx_live.person1
UNION ALL
SELECT * FROM sql_cx_live.person2;
```

output:

id	name
1	Alice
2	Bob
3	Charlie
3	Charlie
4	David
5	Emily

- **INTERSECT**
  - ✓ The **INTERSECT** operator returns only the rows that appear in both result sets of two **SELECT** statements.
  - ✓ **Syntax:**

```
SELECT col1, col2,... FROM table1
INTERSECT
SELECT col1, col2,... FROM table2;
```

  - ✓ **Example:**

query:

```
SELECT * FROM sql_cx_live.person1
INTERSECT
SELECT * FROM sql_cx_live.person2;
```

output:

id	name
3	Charlie

- **EXCEPT**
  - ✓ The **EXCEPT** or **MINUS** operator returns only the distinct rows that appear in the first result set but not in the second result set of two **SELECT** statements.
  - ✓ **Syntax:**

```
SELECT col1, col2,... FROM table1,
EXCEPT
SELECT col1, col2,... FROM table2;
```

  - ✓ **Example:**

query:

```
SELECT * FROM sql_cx_live.person1
EXCEPT
SELECT * FROM sql_cx_live.person2;
```

output:

id	name
1	Alice
2	Bob

SELF JOIN

- A **SELF JOIN** is a type of join in which a table is joined with itself. This means that the table is treated as two separate tables, with each row in the table being compared to every other row in the same table.
- Self joins are used when you want to compare the values of two different rows within the same table.
- **Practical scenarios**: For example, you might use a self join to compare the salaries of two employees who work in the same department, or to find all pairs of customers who have the same billing address.
- **Syntax:** T1 and T2 are different table aliases for the same table.

```
SELECT col1, col2,... FROM table1 [AS alias_name]
JOIN table2 [AS alias_name]
ON table1.column = table2.column;
```

- **Example: To find each employee's name along with their emergency contact's name**

id	name	age	emergaency_contact
1	Nitish	34	11
2	Ankit	32	1
3	Neha	23	1
4	Radhika	34	3
8	Abhinav	31	11
11	Rahul	29	8

query:

```
SELECT T1.user_id, T1.name AS emp_name, T1.emergency_contact,  T2.name AS contact_name
FROM sql_cx_live.users1 AS T1
JOIN sql_cx_live.users1 AS T2;
ON T1.emergency_contact = T2.user_id;
```

output:

user_id	emp_name	emergency_contact	contact_name
3	Neha	1	Nitish
2	Ankit	1	Nitish
4	Radhika	3	Neha
11	Rahul	8	Abhinav
8	Abhinav	11	Rahul
1	Nitish	11	Rahul

FILTER COLUMNS / JOINING ON MORE THAN ONE COLUMN / JOIN MORE THAN 2 TABLES

- **PRACTICE QUESTIONS**
  1. **Find all profitable orders.**
  2. **Find the customer who has placed max number of orders.**
  3. **Which is the most profitable category.**
  4. **Which is the most profitable state.**
  5. **Find all categories worth profit higher than 5000.**

Solutions: https://github.com/akashpagi/SQL-FOR-DATA-SCIENCE/blob/main/SQL_FILES/S34_SQL_JOINS.sql