SQL Joins

Agenda

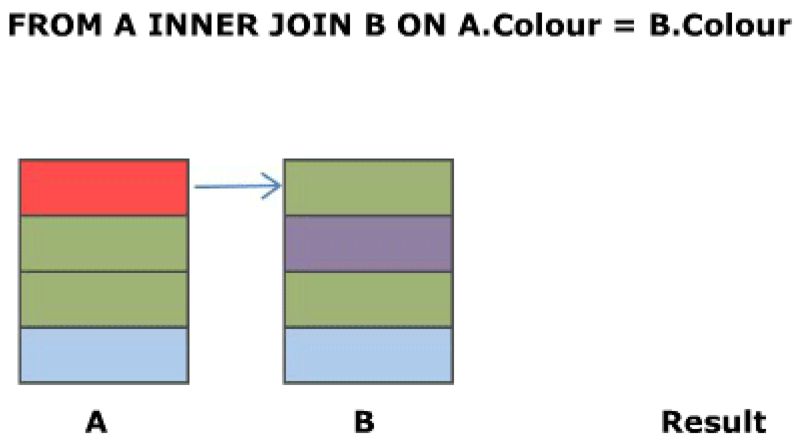
- SQL Joins
- Aggregation Functions (SUM, AVG, COUNT etc)
- Sub-queries
- Built-in Functions

Joins

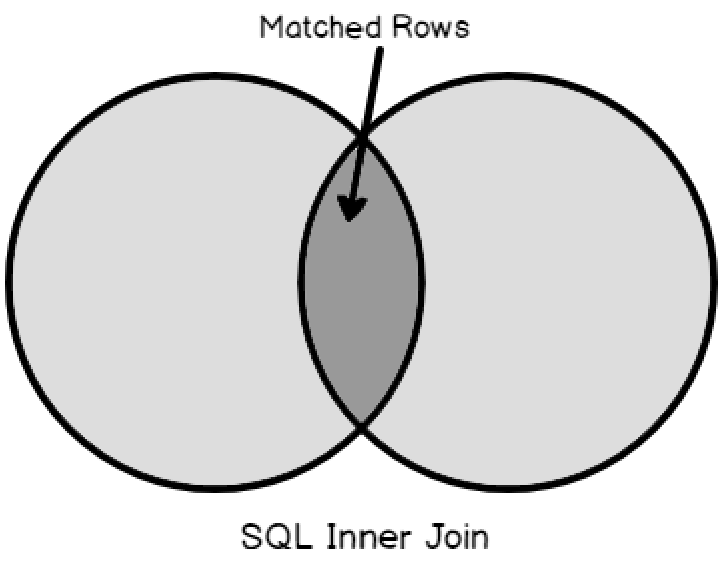
Join is the widely-used clause in the SQL Server essentially to combine and retrieve data from two or more tables. In a real-world relational database, data is structured in many tables and which is why, there is a constant need to join these multiple tables based on logical relationships between them.

Inner join

Inner Join clause in SQL Server creates a new table (not physical) by combining rows that have matching values in two or more tables. This join is based on a logical relationship (or a common field) between the tables and is used to retrieve data that appears in both tables.



Assume, we have two tables, Table A and Table B, that we would like to join using SQL Inner Join. The result of this join will be a new result set that returns matching rows in both these tables. The intersection part in black below shows the data retrieved using Inner Join.



Keyword: `INNER JOIN` or simply `JOIN` **Syntax**: `SELECT column1, column2, ... FROM table_name1 JOIN table_name2 ON condition`

For example, we want to get the batch names of all the students along with their names.

id	first_name	last_name	batch_name
1	John	Watson	Sherlock
2	Mycroft	Holmes	Sherlock

This can be achieved by using the following SQL query:

```
SELECT s.first_name, s.last_name, b.batch_name FROM students s JOIN
batches b ON s.batch_id = b.id;
```

Left Outer Join

Let us consider the students and batches tables below:

id	first_name	last_name	batch_id
1	John	Watson	1
2	Mycroft	Holmes	1
3	Moriarty	Patel	NULL

id	batch_name
1	Sherlock Academy
2	Crime Academy

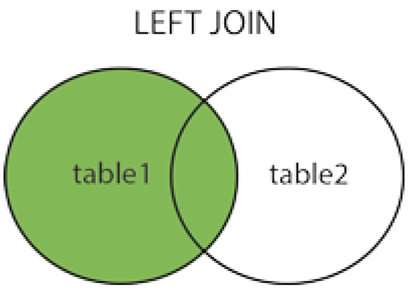
Now if we decide to get the batch names of all the students along with their names with an `inner join` we will get the following result:

id	first_name	last_name	batch_name
1	John	Watson	Sherlock Academy
2	Mycroft	Holmes	Sherlock Academy

Since both John and Mycroft had a valid batch_id, the respective rows were merged to produce the result. But Moriarty did not have a valid batch_id, so the result set will not contain him.

But if we want to run a query to get all students along with their batch names or NULL if they do not have a batch_id, we can use the **Left Outer 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.



Keyword: `LEFT OUTER JOIN` or simply `LEFT JOIN` **Syntax**: `SELECT column1, column2, ... FROM table_name1 LEFT JOIN table_name2 ON condition`

The query for fetching students with their batch names now will be:

```
SELECT
    s.first_name, s.last_name, b.name
FROM
    students s
        LEFT JOIN
    batches b ON s.batch_id = b.id;
```

And the result will be:

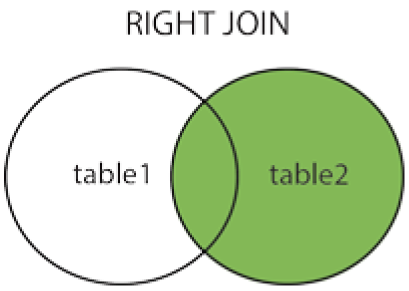
id	first_name	last_name	batch_name
1	John	Watson	Sherlock Academy
2	Mycroft	Holmes	Sherlock Academy
3	Moriarty	Patel	NULL

Right Outer Join

Similarly, `RIGHT OUTER JOIN` 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.

Now if we want to get all the batches along with their students if present else `NULL`, we can use `RIGHT OUTER JOIN`.

Keyword: `RIGHT OUTER JOIN` or simply `RIGHT JOIN` **Syntax**: `SELECT column1, column2, ... FROM table_name1 RIGHT JOIN table_name2 ON condition`



The query for fetching batches with their students now will be:

```
SELECT
    s.first_name, s.last_name, b.name
FROM
    students s
        RIGHT JOIN
    batches b ON s.batch_id = b.id;
```

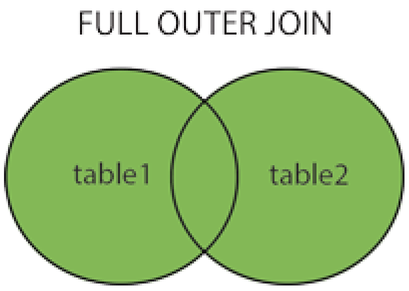
This will return the following result:

id	first_name	last_name	batch_name
1	John	Watson	Sherlock Academy
2	Mycroft	Holmes	Sherlock Academy
NULL	NULL	NULL	Crime Academy

Full Outer Join

The FULL OUTER JOIN clause returns a result set that includes rows from both left and right tables.

When no matching rows exist for the row in the left table, the columns of the right table will contain NULL. Likewise, when no matching rows exist for the row in the right table, the column of the left table will contain NULL.



Keyword: `FULL OUTER JOIN` or simply `FULL JOIN` **Syntax**: `SELECT column1, column2, ... FROM table_name1 FULL JOIN table_name2 ON condition`

The query for fetching batches with their students now will be:

```
SELECT
    s.first_name, s.last_name, b.name
FROM
    students s
        FULL JOIN
    batches b ON s.batch_id = b.id;
```

This will return the following result:

id	first_name	last_name	batch_name
1	John	Watson	Sherlock Academy
2	Mycroft	Holmes	Sherlock Academy
3	Moriarty	Patel	NULL
NULL	NULL	NULL	Crime Academy

> **Warning** MySQL does not support the FULL OUTER JOIN clause.

Cross Join

The result set from a `cross join` will include all rows from both tables, where each row is the combination of the row in the first table with the row in the second table. In general, if each table has `n` and `m` rows respectively, the result set will have `n` x `m` rows.

In other words, the CROSS JOIN clause returns a Cartesian product of rows from the joined tables.

Keyword: `CROSS JOIN` **Syntax**: `SELECT column1, column2, ... FROM table_name1 CROSS JOIN table_name2` **Syntax 2**: `SELECT column1, column2, ... FROM table_name1, table_name2`

An example query for fetching from the students and batches tables will be:

```
SELECT
    s.first_name, s.last_name, b.name
FROM
    students s
        CROSS JOIN
    batches b;
```

Note that different from the INNER JOIN, LEFT JOIN , and RIGHT JOIN clauses, the CROSS JOIN clause does not have a join predicate. In other words, it does not have the ON or USING clause.

If you add a WHERE clause, in case table t1 and t2 has a relationship, the CROSS JOIN works like the INNER JOIN.

Update rows

Keyword: `UPDATE` **Syntax**: `UPDATE table_name SET column1 = value1, column2 = value2, ... WHERE condition`

Examples

1. Update a row

   ```
   UPDATE students SET first_name = 'Tantia' WHERE id = 1;
   ```

2. Update a row with a condition

   ```
   UPDATE students SET first_name = 'Tantia' WHERE id = 1 AND first_name
   = 'John';
   ```

3. Update multiple columns

   ```
   UPDATE students SET first_name = 'Tantia', last_name = 'Tope' WHERE id
   = 1 AND first_name = 'John';
   ```

Delete rows

Keyword: `DELETE` **Syntax**: `DELETE FROM table_name WHERE condition`

Examples

1. Delete a row with a condition

   ```
   DELETE FROM students WHERE id = 1 AND first_name = 'John';
   ```

2. Delete a multiple rows

   ```
   DELETE FROM students WHERE id IN (1, 2, 3);
   ```

Problems

1. Joins - I
2. Joins - II
3. Join - III
4. Join - IV
5. Join - V
6. Join - VI
7. Join - VII
8. Join - VIII