SQL Joins

LESSON

Generate a quiz about this unit

Summarize this unit

Have any doubt about this content? Ask!

Learning Objectives

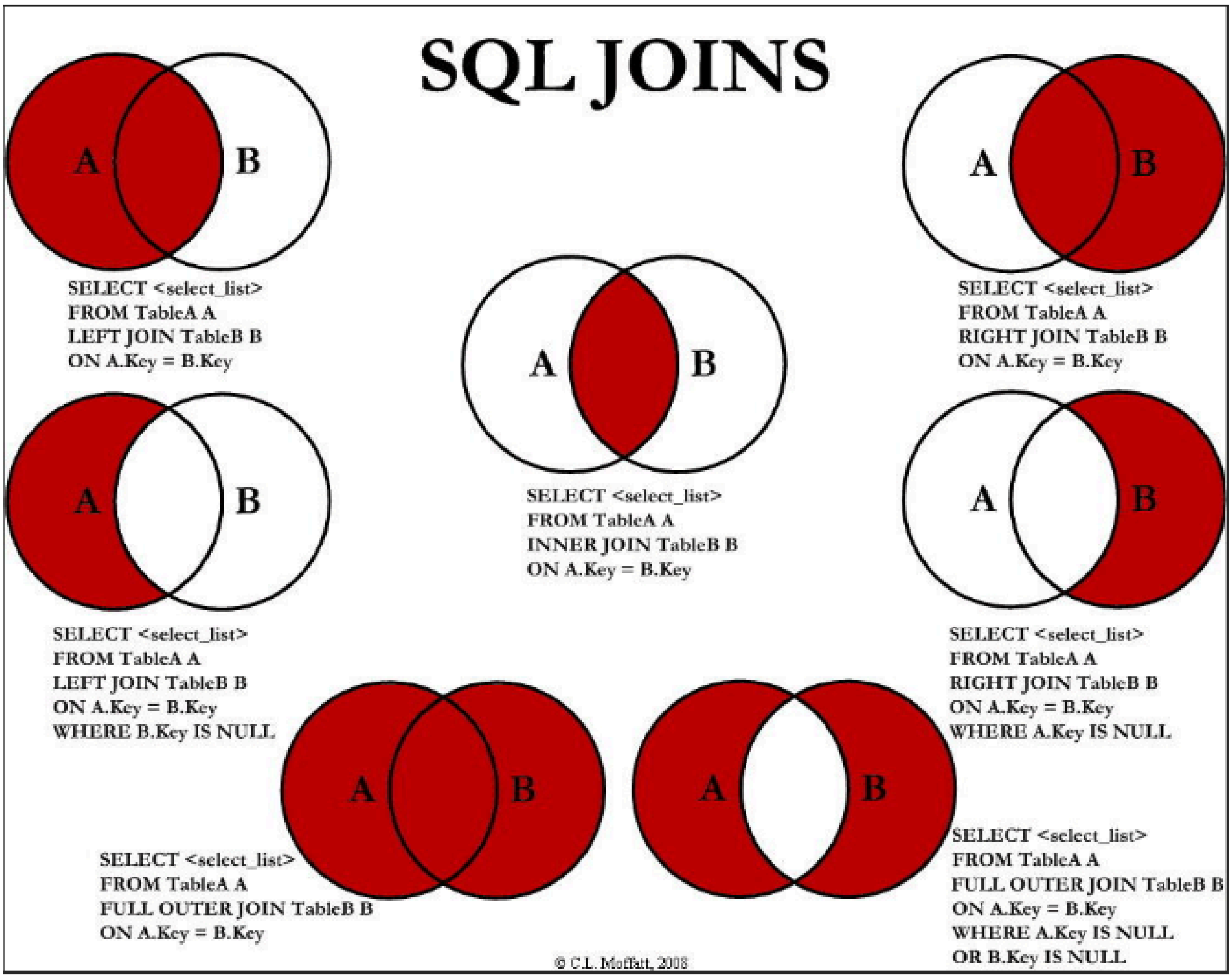
By the end of this lesson, you will be able to:

- Identify the different types of joins in SQL: `INNER JOIN`, `LEFT JOIN (or LEFT OUTER JOIN)`, `RIGHT JOIN (or RIGHT OUTER JOIN)`, and `FULL JOIN (or FULL OUTER JOIN)`.
- Retrieve data from multiple tables using appropriate join operations, utilizing the `ON` clause to specify conditions for the join.
- Apply aliases to tables and columns to enhance the readability of queries.
- Execute set operations, including `UNION`, `INTERSECT`, and `EXCEPT`, to combine and compare data across different tables.

Introduction to Joins

SQL joins are used to combine rows from two or more tables based on a related column. This allows us to perform queries that retrieve information from multiple tables in a single query.

A `JOIN` operation merges rows from two tables based on matching values in specified columns. Typically, one table possesses a `primary key`, a unique identifier for its rows. The counterpart table contains a column (or columns) that it references. This referencing column is termed a `foreign key`. The crux of the `JOIN` lies in the equality between the primary key of one table and the corresponding foreign key in the other.



INNER JOIN

The `INNER JOIN` keyword selects records that have matching values in both tables.

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

Copy

Explain this code

Example

Given two tables, `orders` with columns `order_id`, `product_id`, and `quantity`, and `products` with columns `product_id`, `product_name`, and `price`.

Table `products`:

product_id	product_name	price
1	Laptop	1000
2	Mouse	20
3	Keyboard	50
4	Monitor	200

Table `orders`:

order_id	product_id	quantity
101	1	3
102	3	5
103	4	2

To retrieve the name of the product and the quantity ordered:

```sql
SELECT products.product_name, orders.quantity
FROM orders
INNER JOIN products
ON orders.product_id = products.product_id;
```

Copy

Explain this code

Resulting Set:

product_name	quantity
Laptop	3
Keyboard	5
Monitor	2

LEFT JOIN (or LEFT OUTER JOIN)

The `LEFT JOIN` keyword returns all records from the left table (table1), and the matched records from the right table (table2). The result is NULL for the right side when there is no match.

```sql
SELECT columns
FROM table1
LEFT JOIN table2
ON table1.column_name = table2.column_name;
```

Copy

Explain this code

Example

We'll continue using the same two tables. Here they are again for better understanding of the lesson.

Table `products`:

product_id	product_name	price
1	Laptop	1000
2	Mouse	20
3	Keyboard	50
4	Monitor	200

Table `orders`:

order_id	product_id	quantity
101	1	3
102	3	5
103	4	2

To retrieve all products and their corresponding order quantities (including products that haven't been ordered):

```sql
SELECT products.product_name, orders.quantity
FROM products
LEFT JOIN orders
ON products.product_id = orders.product_id;
```

Copy

Explain this code

Resulting Set:

product_name	quantity
Laptop	3
Mouse	NULL
Keyboard	5
Monitor	2

RIGHT JOIN (or RIGHT OUTER JOIN)

The `RIGHT JOIN` keyword returns all records from the right table (table2), and the matched records from the left table (table1). The result is NULL for the left side when there is no match. It is not as commonly used as other join types.

```
SELECT columns
FROM table1
RIGHT JOIN table2
ON table1.column_name = table2.column_name;
```

Copy

Explain this code

Example

We'll continue using the same two tables. Here they are again for better understanding of the lesson.

Table `products`:

product_id	product_name	price
1	Laptop	1000
2	Mouse	20
3	Keyboard	50
4	Monitor	200

Table `orders`:

order_id	product_id	quantity
101	1	3
102	3	5
103	4	2

Note: The RIGHT JOIN is less commonly used since it's essentially the same as a LEFT JOIN but with the tables reversed.

Here is how to retrieve the name of the product and the quantity ordered, including orders that do not have corresponding products:

```
SELECT products.product_name, orders.quantity
FROM products
RIGHT JOIN orders
ON products.product_id = orders.product_id;
```

Copy

Explain this code

Resulting Set:

product_name	quantity
Laptop	3
Keyboard	5
Monitor	2

FULL JOIN (or FULL OUTER JOIN)

The `FULL JOIN` keyword returns all records when there's a match in one of the tables. This means it returns all rows from both tables and puts NULL in the columns from the table that doesn't have a match.

```
SELECT columns
FROM table1
FULL JOIN table2
ON table1.column_name = table2.column_name;
```

Copy

Explain this code

Note: Not all database systems support `FULL JOIN`. For instance, MySQL does not support it, but you can achieve similar results using a combination of LEFT and RIGHT JOINs.

Example

We'll continue using the same two tables. Here they are again for better understanding of the lesson.

Table `products`:

product_id	product_name	price
1	Laptop	1000
2	Mouse	20
3	Keyboard	50
4	Monitor	200

Table `orders`:

order_id	product_id	quantity
101	1	3
102	3	5
103	4	2

Here is how to retrieve all products and their corresponding order quantities, regardless of whether the product has been ordered, or an order does not have a corresponding product:

```
SELECT products.product_name, orders.quantity
FROM products
FULL JOIN orders
ON products.product_id = orders.product_id;
```

Copy

Explain this code

Resulting Set:

product_name	quantity
Laptop	3
Mouse	NULL
Keyboard	5
Monitor	2

Using Aliases with Joins

Aliases can be used to give a table or a column a temporary name, making columns and table names more readable.

Example:

Given our previous tables, `products` and `orders`:

```sql
SELECT p.product_name, o.quantity
FROM products AS p
INNER JOIN orders AS o
ON p.product_id = o.product_id;
```

Copy

Explain this code

Here, `p` is an alias for the `products` table and `o` is an alias for the `orders` table.

Combining Joins with Other SQL Commands

Joins can be combined with `WHERE`, `GROUP BY`, and `ORDER BY` clauses to create more complex queries.

Example

Here is how to retrieve the total quantity ordered for each product, with results sorted by product name:

Copy

```sql
SELECT p.product_name, SUM(o.quantity) as total_quantity
FROM products AS p
LEFT JOIN orders AS o
ON p.product_id = o.product_id
GROUP BY p.product_name
ORDER BY p.product_name;
```

Explain this code

Resulting Set:

product_name	total_quantity
Keyboard	5
Laptop	3
Monitor	2
Mouse	NULL

Tips

Remember! In SQL syntax, the `left table` refers to the table that is listed immediately after the `FROM` keyword, and the `right table` refers to the table listed immediately after the `JOIN` keyword. The type of join you use, in conjunction with these table positions, determines the information that will be retrieved in the query result.

As a guide to create a join, follow the steps below:

- Identify the tables that contain the columns you need.
- Ensure there is a common column between the two tables, which will be used in the ON section of the join. If not, find another table that has a common column with both.
- Decide which table will be the left table (placed after FROM) and which one will be the right table (placed after JOIN).
- Determine which table should return all records; this decision will inform the type of JOIN you should use.
- Construct the query.

Note: To get better computational performance (beyond the scope of this bootcamp), when you specify the common column in the query, it is better to place the `left table` directly after `ON` and then, the common column of the `right table`:

```
ON left_table_alias.common_column = right_table_alias.common
```

Copy

Explain this code

Set Operations

Set operations allow you to compare, combine, and retrieve data from two or more tables. The main set operations in SQL are `UNION`, `INTERSECT`, and `EXCEPT` (or `MINUS` in some databases).

Prerequisite:

For set operations to work, the queries involved must meet the following conditions:

1. They must retrieve the same number of columns.
2. The columns must have compatible data types.

UNION

The `UNION` operator combines the result sets of two or more `SELECT` statements into a single result set, removing duplicates by default.

```
SELECT column_name(s) FROM table1
UNION
SELECT column_name(s) FROM table2;
```

Copy

Explain this code

Example:

Imagine we have two tables:

- `students`: Contains names of students.
- `teachers`: Contains names of teachers.

students

id	name
1	Alice
2	Bob

teachers

id	name
2	Bob
3	Charlie

We want to create a list of names of both students and teachers. We could do:

```sql
SELECT name FROM students
UNION
SELECT name FROM teachers;
```

Copy

Explain this code

This query retrieves names from both the `students` and `teachers` tables and combines them into a single list. If the same name appears in both tables, `UNION` will display it only once because it eliminates duplicates.

The resulting table would be:

name
Alice
Bob
Charlie

INTERSECT

The `INTERSECT` operator returns the rows that two `SELECT` statements have in common.

Copy

```sql
SELECT column_name(s) FROM table1
INTERSECT
SELECT column_name(s) FROM table2;
```

Explain this code

Example:

Using the same `students` and `teachers` tables, suppose we want to find names that are common in both tables (maybe some students are also teaching assistants).

```sql
SELECT name FROM students
INTERSECT
SELECT name FROM teachers;
```

Copy

Explain this code

This query retrieves names that appear in both the `students` and `teachers` tables. Only the names common to both tables will be included in the result set.

The resulting table would be:

name
Bob

EXCEPT (or MINUS)

The `EXCEPT` operator returns the rows from the first `SELECT` statement that are not returned by the second `SELECT` statement.

```sql
SELECT column_name(s) FROM table1
EXCEPT
SELECT column_name(s) FROM table2;
```

Copy

Explain this code

Example:

Using the `students` and `teachers` tables, suppose we want a list of names that are in the `students` table but not in the `teachers` table.

```
SELECT name FROM students
EXCEPT
SELECT name FROM teachers;
```

Copy

Explain this code

This query retrieves names that appear in the `students` table but not in the `teachers` table. So, if a student is also a teacher (appears in the `teachers` table), they won't be included in the result set.

The result would be:

name
Alice

Other sources

Check this cheat sheet for more detail on how to use SQL JOINs.

Summary

In this lesson, we explored various SQL join types, including `INNER JOIN`, `LEFT (OUTER) JOIN`, `RIGHT (OUTER) JOIN`, and `FULL (OUTER) JOIN`, each serving different data retrieval needs. We learned how to use these joins to combine rows from two or more tables based on related columns, enhancing the flexibility and depth of our queries. The lesson also introduced the importance of table aliases for readability and the process of planning and constructing join queries efficiently. Lastly, we discussed set operations like `UNION`, `INTERSECT`, and `EXCEPT`, which allow for the combination and comparison of data across different tables.

Any doubt? Ask our AI Chatbot!

Lesson Completed