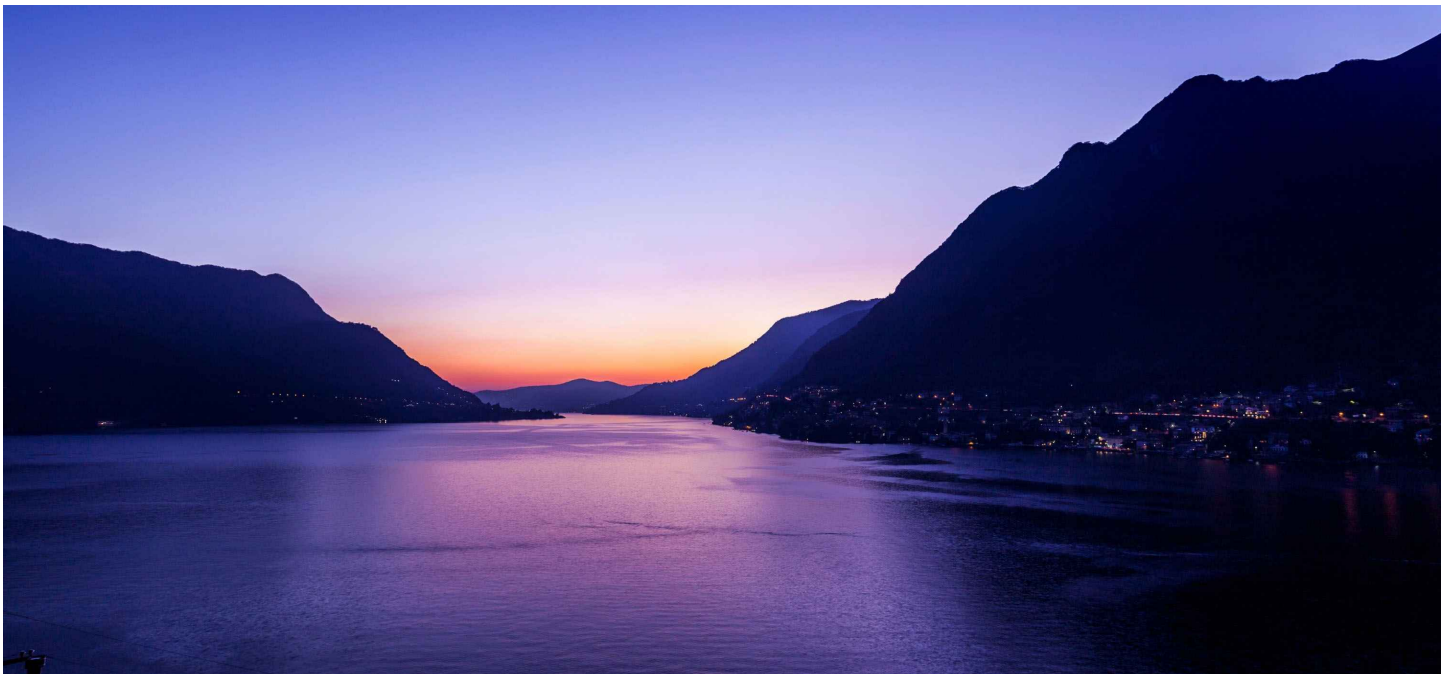
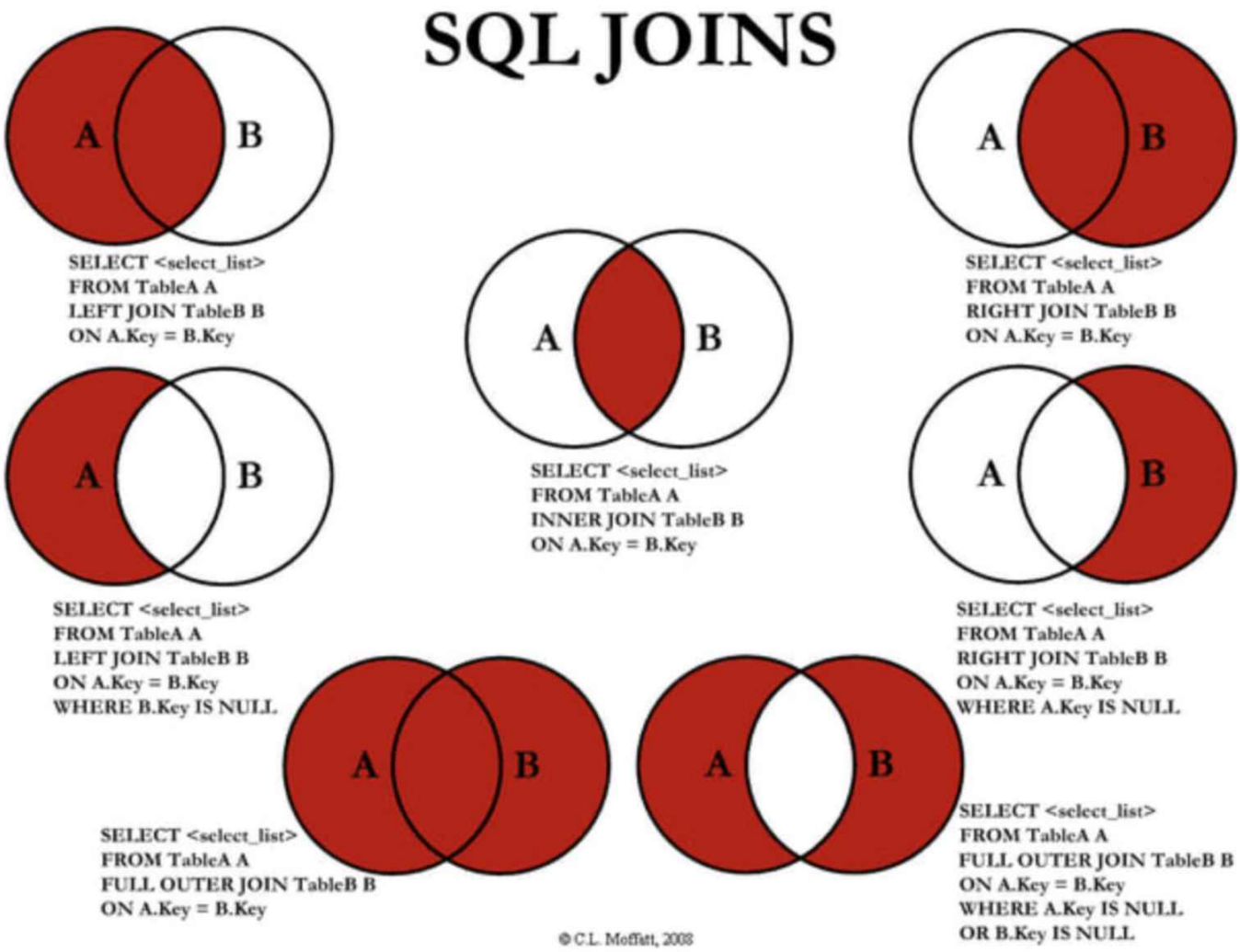


10-SQL Join & Union

Author: Vincent Lau

Learning Objectives

- Understand the purposes of JOINS
- Understand different types of JOIN
- Understand the difference between UNION and UNION ALL
- Understand the differences between JOIN and UNION



Introduction

The `JOIN` operation allows you to combine rows from two or more tables based on a related column between them. This is a fundamental feature of relational databases that enables you to retrieve data from multiple tables simultaneously. There are several types of `JOIN` operations: `INNER JOIN`, `LEFT JOIN` (or `LEFT OUTER JOIN`), `RIGHT JOIN` (or `RIGHT OUTER JOIN`), and `FULL JOIN` (or `FULL OUTER JOIN`).

Types of JOIN

Creating Table

```sql
CREATE TABLE Customers (
    customer_id INT PRIMARY KEY,
    first_name VARCHAR(50),
    last_name VARCHAR(50)
);

CREATE TABLE Orders (
```

```
    order_id INT PRIMARY KEY,
    customer_id INT,
    order_date DATE,
    total_amount DECIMAL(10, 2),
    FOREIGN KEY (customer_id) REFERENCES Customers(customer_id)
);

INSERT INTO Customers (customer_id, first_name, last_name)
VALUES
    (1, 'John', 'Doe'),
    (2, 'Jane', 'Smith'),
    (3, 'Alice', 'Johnson');

INSERT INTO Orders (order_id, customer_id, order_date, total_amount)
VALUES
    (101, 1, '2023-08-01', 150.00),
    (102, 1, '2023-08-05', 200.00),
    (103, 2, '2023-08-02', 120.00);

```

INNER JOIN

The `INNER JOIN` returns rows that have matching values in both tables.

1.1 By INNER JOIN keyword

```
SELECT Customers.customer_id,
Customers.first_name,
Orders.order_id,
Orders.order_date
FROM Customers
INNER JOIN Orders ON Customers.customer_id = Orders.customer_id;
```

1.2 By FROM with key provided in WHERE clause

```
SELECT Customers.customer_id,
Customers.first_name,
Orders.order_id,
Orders.order_date
FROM Customers, Orders
WHERE Customers.customer_id = Orders.customer_id;
```

LEFT JOIN - with intersect

The `LEFT JOIN` returns all rows from the left table and the matching rows from the right table. If no match is found, `NULL` values are returned for the right table's columns.

```
SELECT Customers.customer_id, Customers.first_name, Orders.order_id,
Orders.order_date
FROM Customers
LEFT JOIN Orders ON Customers.customer_id = Orders.customer_id;
```

LEFT JOIN - without intersect

```
SELECT Customers.customer_id, Customers.first_name, Orders.order_id,
Orders.order_date
FROM Customers
LEFT JOIN Orders ON Customers.customer_id = Orders.customer_id
WHERE Orders.customer_id IS NULL;
```

RIGHT JOIN - with intersect

The `RIGHT JOIN` returns all rows from the right table and the matching rows from the left table. If no match is found, `NULL` values are returned for the left table's columns.

```
SELECT Customers.customer_id, Customers.first_name, Orders.order_id,
Orders.order_date
FROM Customers
RIGHT JOIN Orders ON Customers.customer_id = Orders.customer_id;
```

RIGHT JOIN - without intersect

```
SELECT Customers.customer_id, Customers.first_name, Orders.order_id,
Orders.order_date
FROM Customers
RIGHT JOIN Orders ON Customers.customer_id = Orders.customer_id
WHERE Customers.customer_id IS NULL;
```

FULL OUTER JOIN - with intersect

The `FULL JOIN` returns all rows when there is a match in either the left or the right table. If no match is found, `NULL` values are returned for the columns of the table without a match.

```
-- Or you can use "FULL JOIN" in MySQL
SELECT Customers.customer_id, Customers.first_name, Orders.order_id,
Orders.order_date
FROM Customers
FULL OUTER JOIN Orders ON Customers.customer_id = Orders.customer_id;
```

FULL OUTER JOIN - without intersect

```
-- Or you can use "FULL JOIN" in MySQL
SELECT Customers.customer_id, Customers.first_name, Orders.order_id,
Orders.order_date
FROM Customers
FULL OUTER JOIN Orders ON Customers.customer_id = Orders.customer_id
WHERE Customers.customer_id IS NULL
OR Orders.customer_id IS NULL;
```

Notes when using JOIN

1. INNER JOIN

Use an `INNER JOIN` when you want to retrieve only the rows that have matching values in both tables. This is useful when you're interested in data that exists in both tables and you want to combine them based on a common condition.

2. LEFT JOIN & RIGHT JOIN

Use a `LEFT JOIN` when you want to retrieve all rows from the left table and any matching rows from the right table. Similarly, use a `RIGHT JOIN` when you want to retrieve all rows from the right table and any matching rows from the left table. These joins are useful when you want to include unmatched data from one table along with matching data from the other.

3. FULL OUTER JOIN

Use a `FULL JOIN` (or `FULL OUTER JOIN`) when you want to retrieve all rows from both tables, along with matching rows from both tables. This join is useful when you want to see all the data from both tables, including unmatched data.

4. Use LEFT JOIN or RIGHT JOIN

In most cases, you can achieve the same result by switching between `LEFT JOIN` and `RIGHT JOIN`, as they are essentially the same operation with the tables switched. Choose the one that makes your query more readable and easier to understand in the context of your application.

5. Performance

Each type of `JOIN` has different performance implications. Generally, `INNER JOIN` and `LEFT JOIN` are more commonly used and optimized, while `RIGHT JOIN` and `FULL JOIN` can be less efficient in some databases.

6. Aliases for Table Names

When using multiple joins, aliasing your table names can make your query more readable and concise. It also helps to differentiate columns from different tables.

7. Plan and Test Your Queries

Complex queries with multiple joins can become difficult to understand and troubleshoot. Plan your query carefully and test it with sample data to ensure the results are as expected.

Summary of JOIN

Each type of `JOIN` serves a specific purpose for retrieving data from related tables. You can use these operations to combine data from different tables based on common columns and perform complex data retrieval tasks efficiently.

Types of UNION

The `UNION` and `UNION ALL` operators are used to combine the result sets of two or more `SELECT` queries into a single result set. Both operators allow you to merge rows from different queries, but there is a key difference between them.

Creating Tables

```sql
CREATE TABLE Students (
    student_id INT,
    first_name VARCHAR(50),
    last_name VARCHAR(50)
);
```

```sql

CREATE TABLE Teachers (
    teacher_id INT,
    first_name VARCHAR(50),
    last_name VARCHAR(50)
);

INSERT INTO Students (student_id, first_name, last_name)
VALUES
    (1, 'John', 'Doe'),
    (2, 'Jane', 'Smith'),
    (3, 'Alice', 'Johnson');

INSERT INTO Teachers (teacher_id, first_name, last_name)
VALUES
    (101, 'Michael', 'Brown'),
    (102, 'Emily', 'Wilson'),
    (103, 'John', 'Doe');
```

UNION Operator

The `UNION` operator combines the result sets of multiple `SELECT` queries and removes duplicate rows from the combined result set. This means that if there are duplicate rows between the queries, only one copy of the duplicated row will appear in the final result set.

In the first example using `UNION`, duplicate rows are removed, so only unique combinations of first names and last names will appear in the result.

```sql
-- Using UNION (removes duplicate rows)
SELECT first_name, last_name
FROM Students
UNION
SELECT first_name, last_name
FROM Teachers;
```

UNION ALL Operator

The `UNION ALL` operator also combines the result sets of multiple `SELECT` queries, but it retains all rows from all queries, including duplicate rows. This means that if there are duplicate rows between the queries, all copies of the duplicated row will appear in the final result set.

In the second example, using `UNION ALL`, all rows from both queries are included, even if there are duplicates.

```sql
-- Using UNION ALL (retains all rows)
SELECT first_name, last_name
FROM Students
UNION ALL
SELECT first_name, last_name
FROM Teachers;
```

Summary of UNION

Choose between `UNION` and `UNION ALL` based on whether you want to eliminate duplicate rows or keep all rows from the combined queries. Keep in mind that `UNION` requires more processing to remove duplicates, which might impact performance.

JOIN (x) vs UNION (+)

Purpose

`JOIN` is used to combine rows from two or more tables based on a related column between them. It allows you to retrieve data from multiple tables and combine rows that have matching values in the specified columns.

`UNION` is used to combine the result sets of two or more `SELECT` queries into a single result set. It allows you to stack the rows from different queries on top of each other.

Result

The result of a `JOIN` operation is a single result set that includes columns from the combined tables, with matching rows from each table based on the specified condition.

The result of a `UNION` operation is a single result set that includes rows from each of the combined queries. Duplicate rows are typically removed, depending on whether you're using `UNION` or `UNION ALL`.

Duplicates

`JOIN` does not eliminate duplicate rows by default. If there are multiple matching rows, they will all be included in the result set.

By default, `UNION` removes duplicate rows from the result set, ensuring that each row is unique. However, you can use `UNION ALL` to retain all rows, including duplicates.

Types

There are various types of `JOIN` operations, including `INNER JOIN`, `LEFT JOIN`, `RIGHT JOIN`, and `FULL OUTER JOIN`. Each type determines how unmatched rows are handled and which rows are included in the result set.

`UNION` is often used when you want to combine the results of queries that have a similar structure and retrieve a unified set of rows.