UNION and UNION ALL Demystified: Combining Data Effectively in SQL

Pooja Pawar

UNION and UNION ALL in SQL

In the world of data analysis and database management, there are often scenarios where you need to combine results from multiple datasets. For instance, you might need to merge sales records from different regions, consolidate customer information from various systems, or unify data from multiple time periods. SQL provides two essential tools for this purpose: **UNION** and **UNION ALL**.

At first glance, these operators seem similar—they both combine rows from two or more queries into a single result set. However, their behavior regarding duplicates, performance, and output structure makes them distinct. Choosing the right operator depends on your specific needs, such as whether you require unique rows or wish to preserve all data, including duplicates.

This guide dives into the intricacies of **UNION** and **UNION ALL**, offering a thorough exploration of their functionality, key differences, and practical applications. Through detailed explanations, queries, and examples, you'll gain the skills to leverage these operators effectively in real-world scenarios. Whether you're cleaning data, building reports, or analyzing trends, understanding these tools will empower you to work smarter with SQL.

What is UNION?

The **UNION** operator merges the results of two or more SELECT statements into a single result set, eliminating duplicate rows by default. It ensures that every row in the output is unique.

Key Characteristics:

- Removes duplicate rows.
- Combines data from multiple SELECT queries.
- Sorts the output in ascending order by default unless otherwise specified.

What is UNION ALL?

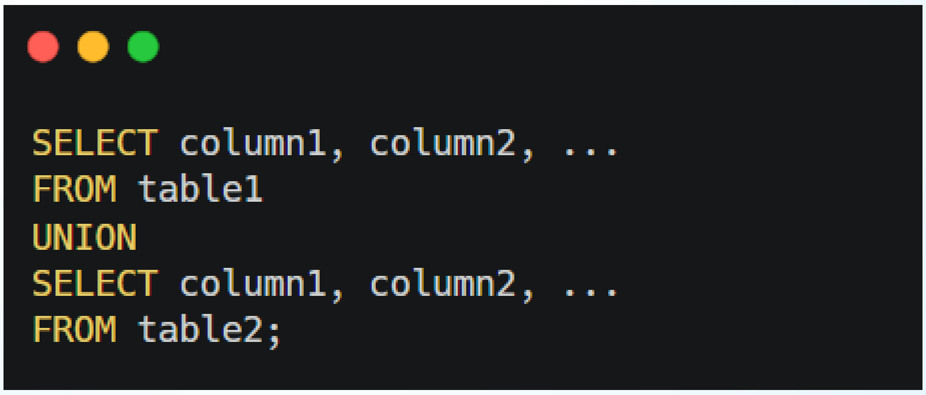
The **UNION ALL** operator also combines results from multiple SELECT queries but **retains duplicate rows**. It is faster than UNION because it skips the step of checking for duplicates.

Key Characteristics:

- Retains duplicates in the result set.
- Does not perform sorting or filtering, making it faster.
- Combines data exactly as retrieved from the queries.

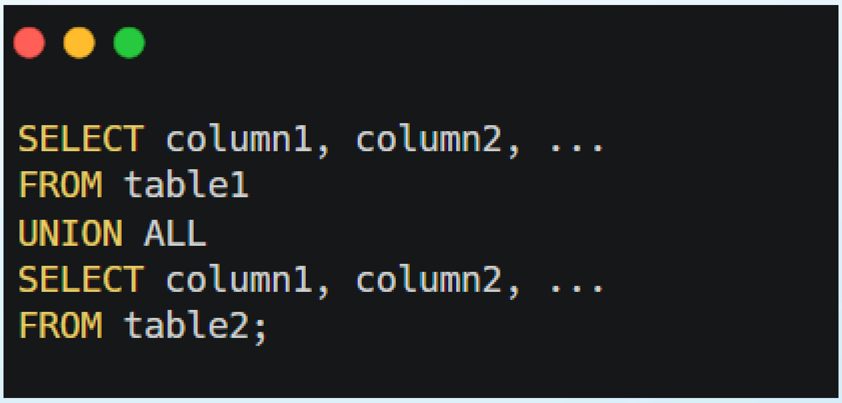
Syntax and Rules

Syntax for UNION



```sql
SELECT column1, column2, ...
FROM table1
UNION
SELECT column1, column2, ...
FROM table2;
```

Syntax for UNION ALL



```sql
SELECT column1, column2, ...
FROM table1
UNION ALL
SELECT column1, column2, ...
FROM table2;
```

Rules for Both UNION and UNION ALL

1. The number of columns in all SELECT queries must be **equal**.
2. The data types of corresponding columns must be **compatible** (e.g., integer with integer, string with string).
3. The column order in all SELECT queries must **match**.

Key Differences Between UNION and UNION ALL

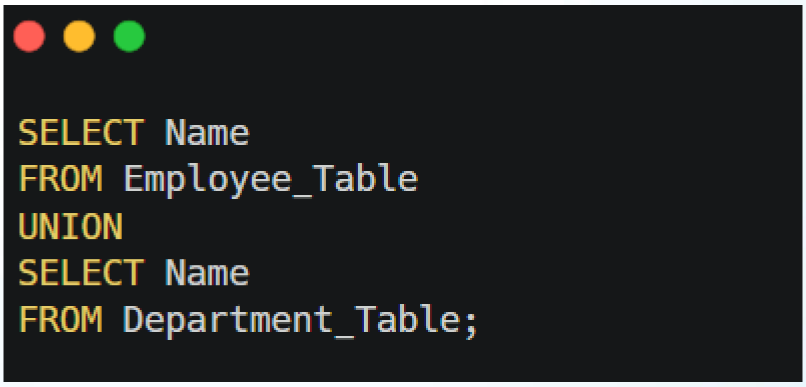
Aspect	**UNION**	**UNION ALL**
Duplicates	Removes duplicates	Retains duplicates
Performance	Slower due to duplicate elimination	Faster as no duplicate removal is performed
Sorting	Implicitly sorts the result	Does not sort by default
Use Case	When unique rows are needed	When duplicates have significance
Memory Usage	Higher due to sorting and filtering	Lower as duplicates are retained

Examples with Queries and Outputs

Let's explore UNION and UNION ALL with practical examples and tabular outputs.

Example 1: Basic Usage of UNION

Query Using UNION:



```sql
SELECT Name
FROM Employee_Table
UNION
SELECT Name
FROM Department_Table;
```

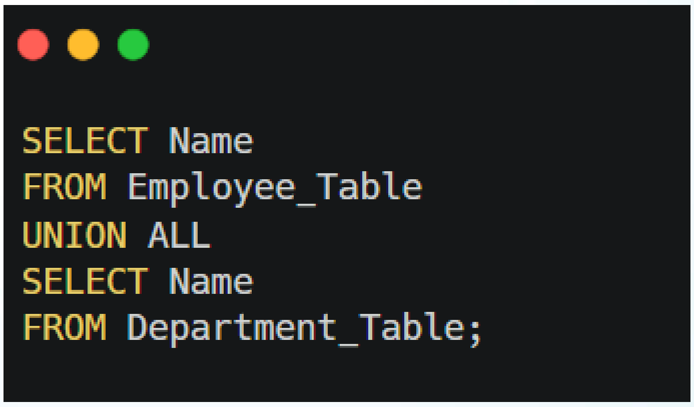
Output:

Name
Alice
Bob
Charlie
Diana
HR
Finance
IT
Marketing

Explanation: Duplicate rows (if any) are removed, and the result is sorted in ascending order by default.

Example 2: Basic Usage of UNION ALL

Query Using UNION ALL:



```sql
SELECT Name
FROM Employee_Table
UNION ALL
SELECT Name
FROM Department_Table;
```

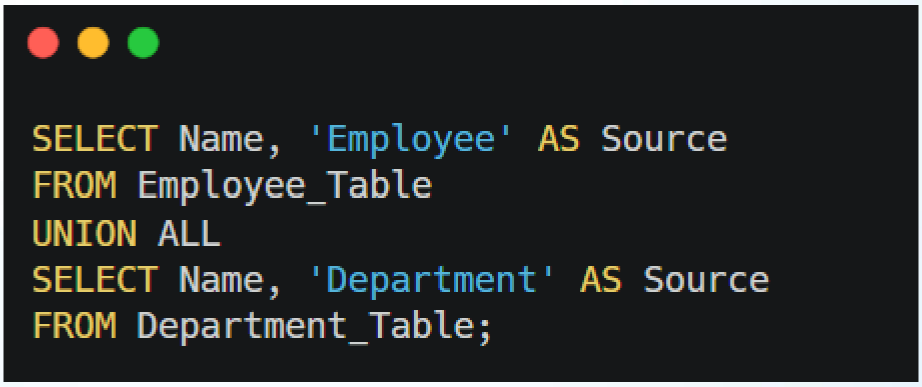
Output:

Name
Alice
Bob
Charlie
Diana
HR
Finance
IT
Marketing
Alice

Explanation: All rows, including duplicates, are retained in the output.

Example 3: Adding a Column to Identify Sources

Query Using UNION ALL:



```sql
SELECT Name, 'Employee' AS Source
FROM Employee_Table
UNION ALL
SELECT Name, 'Department' AS Source
FROM Department_Table;
```

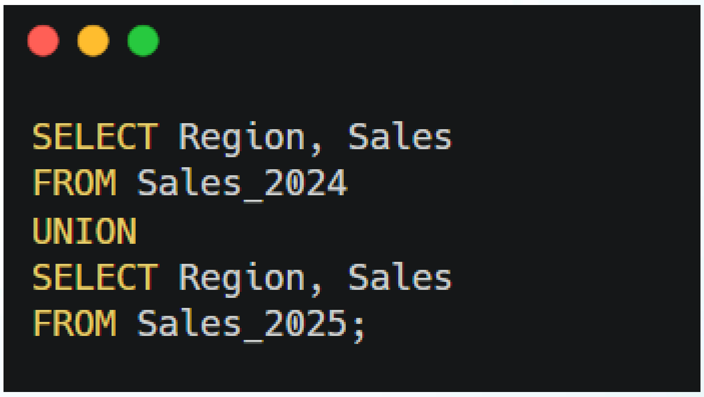
Output:

Name	Source
Alice	Employee
Bob	Employee
Charlie	Employee
Diana	Employee
HR	Department
Finance	Department
IT	Department
Marketing	Department

Explanation: The constant column Source identifies the origin of each row.

Example 4: Combining Data with Different Conditions

Query: Total Sales Data Across Years



```sql
SELECT Region, Sales
FROM Sales_2024
UNION
SELECT Region, Sales
FROM Sales_2025;
```

Output:

Region	Sales
North	1000
South	800
West	700
East	1500
North	1200

Performance Considerations

Why UNION is Slower:

1. Duplicate elimination involves comparing all rows in the result set.

2. Sorting is performed by default to aid duplicate elimination.

Why UNION ALL is Faster:

1. No duplicate elimination means no sorting or comparison is required.
2. Useful for large datasets where duplicates are acceptable.

Practical Use Cases

When to Use UNION

- To create a **unique list** of values (e.g., email lists from multiple sources).
- To remove redundancy in the data.
- Example: Generating a list of unique products sold across multiple stores.

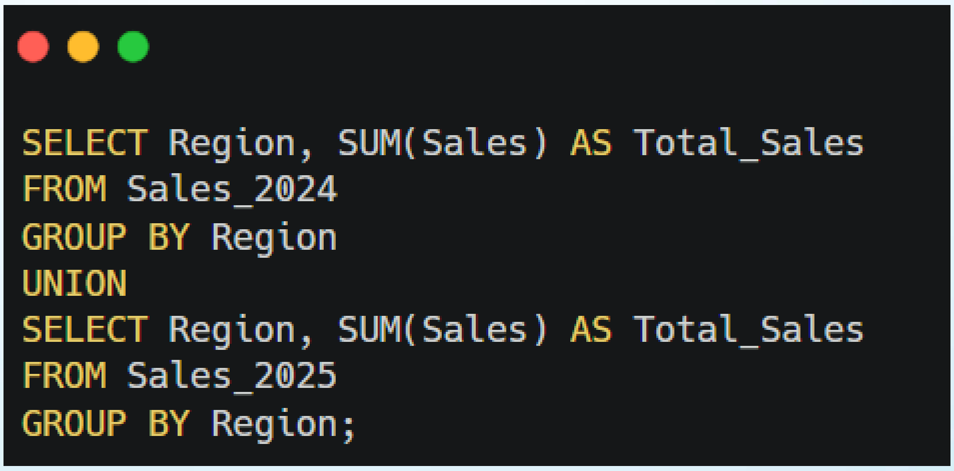
When to Use UNION ALL

- To **retain all data**, including duplicates (e.g., transaction logs from multiple branches).

- To analyze data where duplicates hold significance.
- Example: Combining sales data across regions for a detailed analysis.

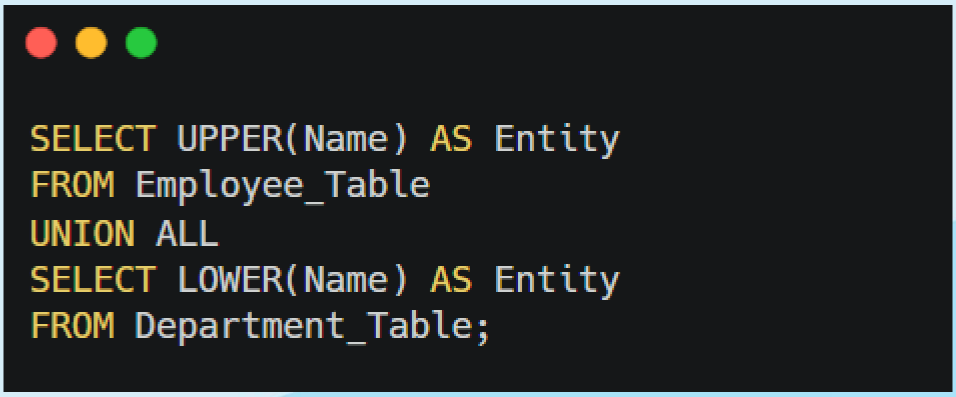
Advanced Examples

Example 1: UNION with Aggregation



```sql
SELECT Region, SUM(Sales) AS Total_Sales
FROM Sales_2024
GROUP BY Region
UNION
SELECT Region, SUM(Sales) AS Total_Sales
FROM Sales_2025
GROUP BY Region;
```

Example 2: UNION ALL with Data Transformation



```sql
SELECT UPPER(Name) AS Entity
FROM Employee_Table
UNION ALL
SELECT LOWER(Name) AS Entity
FROM Department_Table;
```

Common Errors and Solutions

Error	**Cause**	**Solution**
Column count does not match	Different number of columns in queries	Ensure both queries have the same number of columns
Data type mismatch	Incompatible column data types	Use type casting (e.g., CAST or CONVERT)
Unexpected duplicates in UNION	Hidden variations (e.g., trailing spaces)	Use TRIM() or clean data in preprocessing

Conclusion

Understanding **UNION** and **UNION ALL** is essential for efficient SQL querying. Whether you need unique data or wish to retain duplicates for detailed analysis, selecting the right operator can improve performance and clarity in your results. By mastering the differences, syntax, and use cases, you can effectively combine datasets to gain valuable insights.