😏

Union & Cases

Suppliers.csv

Customers.csv

Data

Union in SQL

Understanding the `UNION` Keyword in SQL

The `UNION` keyword is used in SQL to combine the result sets of two or more `SELECT` queries into a single result set. The primary conditions for using `UNION` are:

- Each `SELECT` statement within the `UNION` must have the same number of columns in the result sets.
- The columns must also have similar data types.
- The columns in each `SELECT` statement must be in the same order.

How `UNION` Works

`UNION` performs two main actions:

1. **Combines the Result Sets:** It takes the results from multiple `SELECT` statements and merges them into one.

2. **Removes Duplicates:** By default, `UNION` eliminates duplicate rows from its result set, ensuring each row is unique.

Basic Syntax

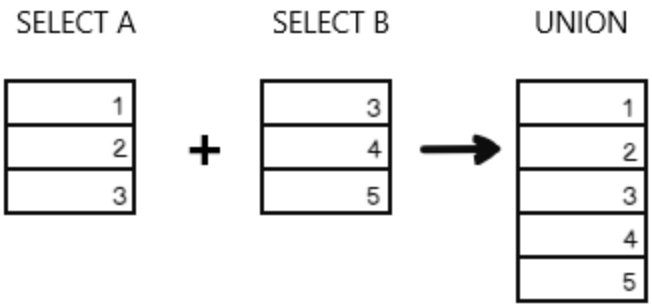
```
SELECT column_names FROM table1
UNION
SELECT column_names FROM table2;
```

Example Use Case

Imagine you have two tables, `Table_A` and `Table_B`, each containing a list of names. If you want to create a combined list of names from both tables without any duplicates, you would use the `UNION` keyword like this:

```
SELECT name FROM Table_A
UNION
SELECT name FROM Table_B;
```

This query retrieves all unique names from both `Table_A` and `Table_B`.



`UNION` vs. `UNION ALL`

While `UNION` removes duplicates, sometimes you might want to include all duplicates in your result set. In such cases, you use `UNION ALL` instead of `UNION`. `UNION ALL` will combine the result sets of the `SELECT` statements without filtering out duplicate rows.

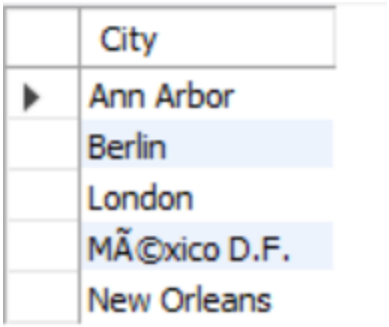
Why Use `UNION`?

`UNION` is particularly useful when you need to retrieve related information from different tables and present it as a coherent whole. For example, if you're aggregating data from different time periods, product types, or geographical regions stored in separate tables, `UNION` helps you create a unified view of the information for analysis or reporting purposes

Let’s take an example:-

```
SELECT City FROM Customers
UNION
SELECT City FROM Suppliers
ORDER BY City;
```

Output:-



	City
▶	Ann Arbor
	Berlin
	London
	MÃ©xico D.F.
	New Orleans

Case Statement

The CASE statement in SQL acts like a switch or if-else ladder in programming languages. It allows you to apply logic to choose between a series of conditions, resulting in different outputs for each condition within your SQL query. You can use it to create complex conditional logic directly in your SQL statements, making it incredibly powerful for transforming, adding conditional logic, or categorizing data directly within your database queries.

Syntax:

```
SELECT CASE Expression
When expression1 Then Result1
```

```
When expression2 Then Result2
ELSE Result
END
```

- **WHEN**: This part specifies the condition you're testing.
- **THEN**: This part specifies the result or action to take if the condition is true.
- **ELSE**: This (optional) part specifies what to do if none of the conditions are true. If the ELSE part is omitted and none of the conditions are met, the CASE statement will return NULL.
- **END**: This part closes the CASE statement.

Why Use CASE Statements?

- **Flexibility**: They allow for complex conditional logic in SQL queries.
- **Readability**: They can make queries more readable compared to using multiple nested functions or joins.
- **Efficiency**: They enable conditional logic to be processed directly in the database, often leading to more efficient data retrieval.

Consider the following table employees:

employee_id	first_name	last_name	job_title	salary	reports_to	office_id
33391	D'arcy	Nortunen	Account Executive	62871	37270	1
37270	Yovonnda	Magrannell	Executive Secretary	63996	NULL	10
37851	Sayer	Matterson	Statistician III	98926	37270	1
40448	Mindy	Crissil	Staff Scientist	94860	37270	1
56274	Keriann	Alloisi	VP Marketing	110150	37270	1
63196	Alaster	Scutchin	Assistant Professor	32179	37270	2
67009	North	de Clerc	VP Product Management	114257	37270	2
67370	Elladine	Rising	Social Worker	96767	37270	2
68249	Nisse	Voysey	Financial Advisor	52832	37270	2
72540	Guthrey	Iacopetti	Office Assistant I	117690	37270	3
72913	Kass	Hefferan	Computer Systems Anal...	96401	37270	3
75900	Virge	Goodrum	Information Systems M...	54578	37270	3
76196	Mirilla	Janowski	Cost Accountant	119241	37270	3
80529	Lynde	Aronson	Junior Executive	77182	37270	4
80679	Mildrid	Sokale	Geologist II	67987	37270	4

Case using Comparison Operator:

In the following SQL statement, we are using a comparison operator to evaluate an expression.

```
SELECT employee_id, job_title, salary,
CASE
    WHEN Salary >=50000 THEN 'The salary is greater than 50000'
    WHEN Salary =50000 THEN 'The salary is 50000'
    ELSE 'The salary is under 50000'
END AS SalaryText
FROM employees;
```

In the following image, we get the salary details as per the condition specified in the CASE statement.

employee_id	job_title	salary	SalaryText
33391	Account Executive	62871	The salary is greater than 50000
37270	Executive Secretary	63996	The salary is greater than 50000
37851	Statistician III	98926	The salary is greater than 50000
40448	Staff Scientist	94860	The salary is greater than 50000
56274	VP Marketing	110150	The salary is greater than 50000
63196	Assistant Professor	32179	The salary is under 50000
67009	VP Product Management	114257	The salary is greater than 50000
67370	Social Worker	96767	The salary is greater than 50000
68249	Financial Advisor	52832	The salary is greater than 50000

Case using Order By:

The CASE statement can be used in the ORDER BY clause as well. In SQL, the ORDER BY clause is used to sort the result in ascending or descending order.

In the following SQL statement, you can see the use of the ORDER BY and CASE statements together.

```
SELECT first_name, office_id, salary
 FROM employees
 ORDER BY  CASE office_id
WHEN '1' THEN Salary END DESC,
CASE WHEN '2' THEN Salary
END;
```

In the output, it sorts the salary in descending order where the office id is equal to 1 and in ascending order where the office id is equal to 2.

	first_name	office_id	salary
▶	Keriann	1	110150
	Sayer	1	98926
	Mindy	1	94860
	D'arcy	1	62871
	Alaster	2	32179
	Theresa	5	47354
	Nisse	2	52832
	Virge	3	54578
	Yovonnda	10	63996
	Mildrid	4	67987
	Estrellita	5	70187
	Lynde	4	77182
	Cole	4	86119
	Ivy	5	92710

Case using Group By:

Suppose we want to group the employees based on their salaries and calculate the maximum and minimum salary for a specified range of employee data.

In the following SQL statement, you can see the use of the GROUP BY clause with the required CASE statement.

```
3 •   SELECT
4      CASE
5     WHEN Salary >=80000 AND Salary <=100000 THEN 'MID-LEVEL'
6     WHEN Salary >=50000 AND Salary <80000 THEN 'EXECUTIVE LEVEL'
7     ELSE 'SENIOR LEVEL'
8     END AS Designation,
9     Min(salary) as MinimumSalary,
10    Max(Salary) as MaximumSalary
11    FROM EmployeeS
12    Group By
13    CASE
14    WHEN Salary >=80000 AND Salary <=100000 THEN 'MID-LEVEL'
15    WHEN Salary >=50000 AND Salary <80000 THEN 'EXECUTIVE-LEVEL'
16    ELSE 'SENIOR-LEVEL'
17    END
```

The statement will produce the following result.

	Designation	MinimumSalary	MaximumSalary
▶	EXECUTIVE LEVEL	52832	77182
	MID-LEVEL	86119	98926
	SENIOR LEVEL	32179	119241

▼ Question

https://leetcode.com/problems/big-countries/description/

https://leetcode.com/problems/rearrange-products-table/

https://leetcode.com/problems/calculate-special-bonus/description/

https://leetcode.com/problems/exchange-seats/description/