From Confusion to Clarity

MYSQL SET OPERATORS JOURNEY

Presented by Gagandeep Kaur Bhatti — Data Analyst

UNION..UNION ALL..
INTERSECT..
EXCEPT / MINUS

Why are SQL Set Operators so confusing? Why does nothing make sense?

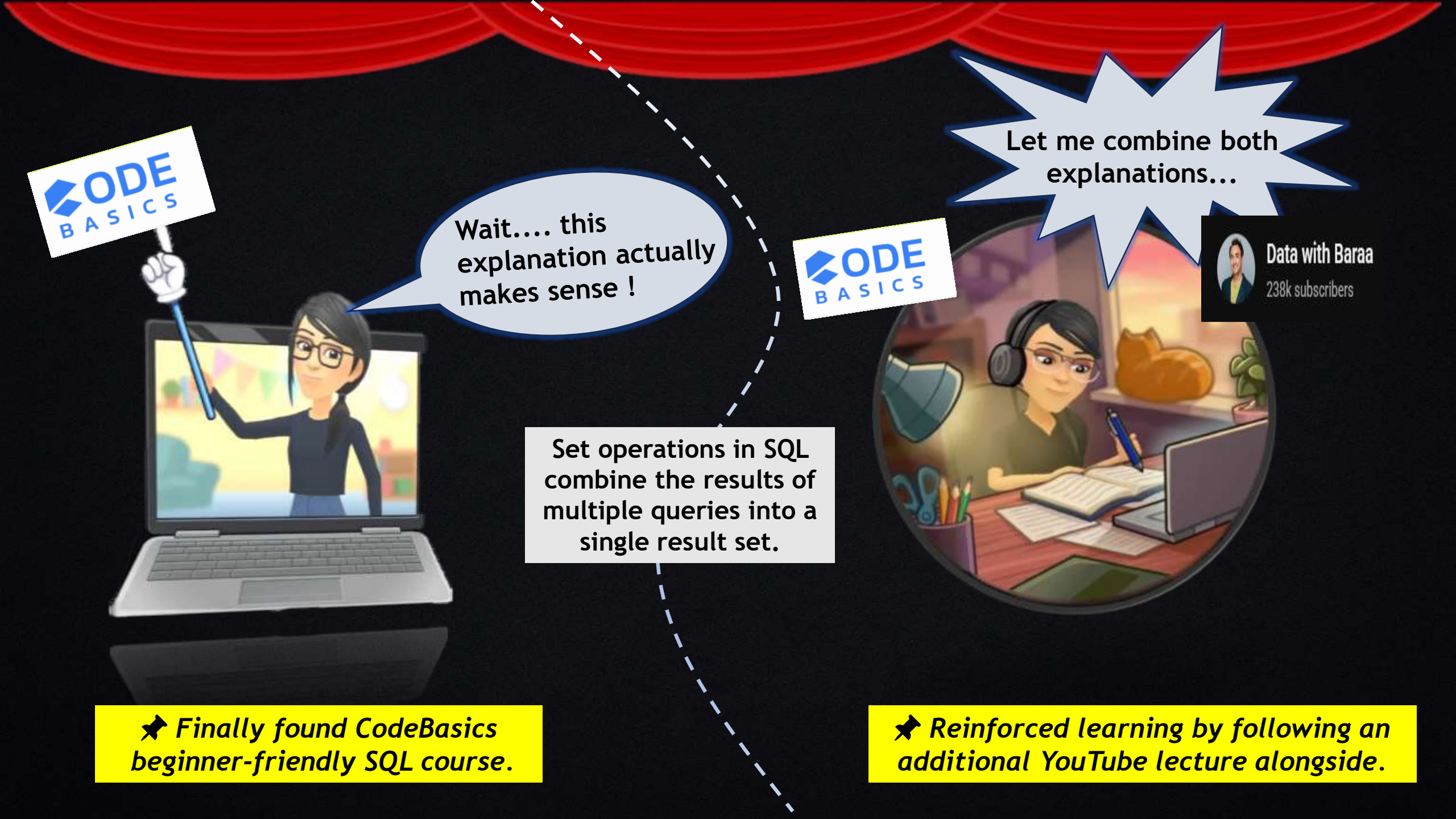
But don't worry — it will make sense.

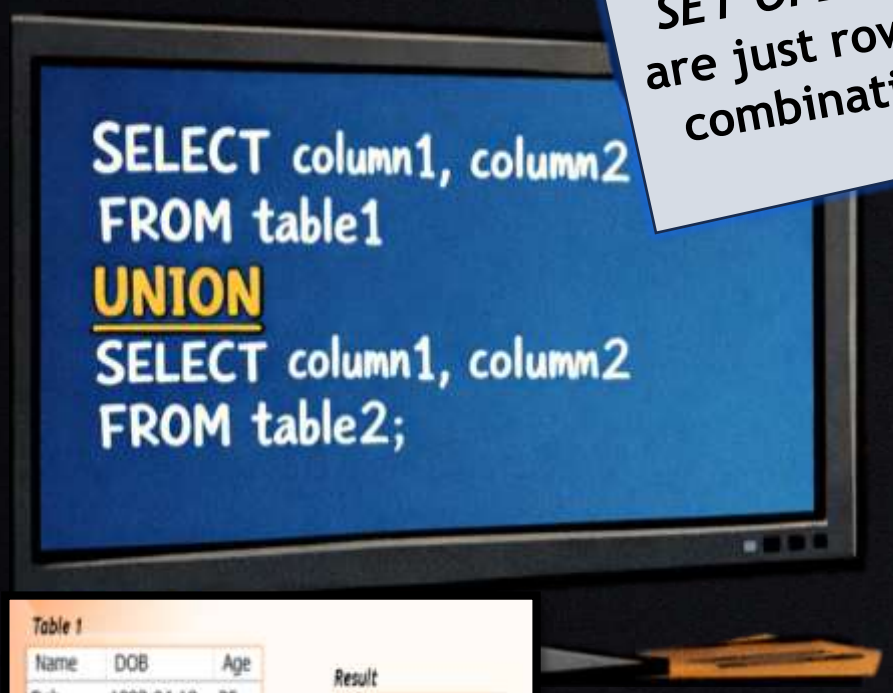
📌 *Started learning SQL Set Operators… feeling completely stuck.*

Another tutorial… maybe this one will help ?

Still searching…

📌 *Jumped between random tutorials - but the concepts still felt unclear.*


CODE BASICS
Wait.... this explanation actually makes sense !
Set operations in SQL combine the results of multiple queries into a single result set.
CODE BASICS
Let me combine both explanations...
Data with Baraa
238k subscribers
📌 Finally found CodeBasics beginner-friendly SQL course.
📌 Reinforced learning by following an additional YouTube lecture alongside.



Ohhh...
SET OPERATORS
are just row-wise
combinations !

Appending means **adding rows from one table to another**, as long **as both tables share the same column structure**

APPEND → Vertical

- Same number of columns
- Same column purpose (same meaning)
- Data types must be compatible

❑ ***This is done using Set Operators:***

- UNION
- UNION ALL
- INTERSECT
- EXCEPT / MINUS

NOTE :- INTERSECT & EXCEPT are **not directly supported in MySQL.**

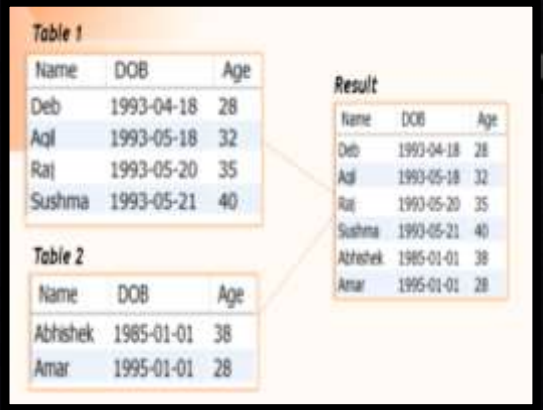


Table 1

Name	DOB	Age
Deb	1993-04-18	28
Aqil	1993-05-18	32
Rat	1993-05-20	35
Sushma	1993-05-21	40

Table 2

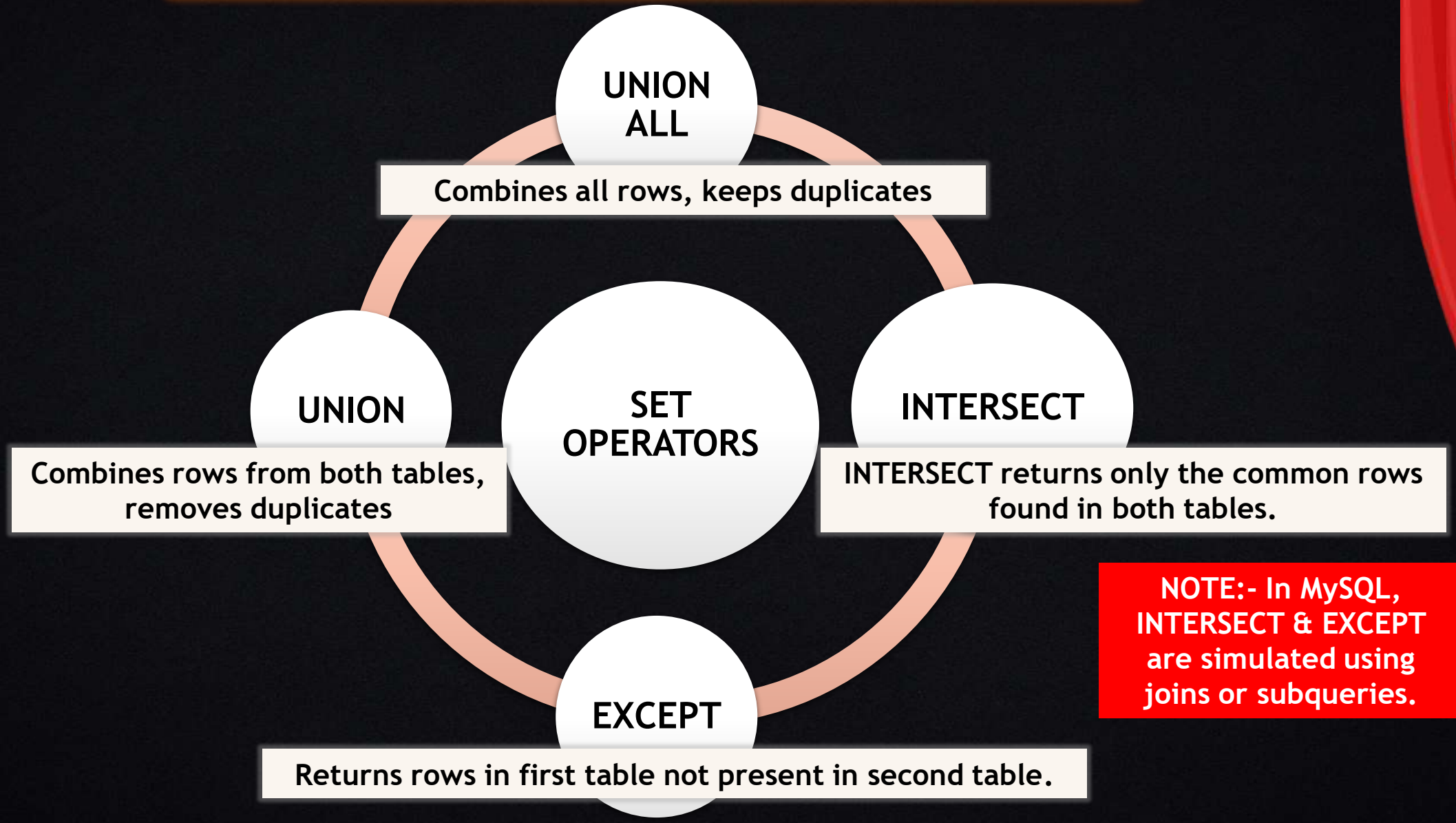
Name	DOB	Age
Abhishek	1985-01-01	38
Amar	1995-01-01	28

Result

Name	DOB	Age
Deb	1993-04-18	28
Aqil	1993-05-18	32
Rat	1993-05-20	35
Sushma	1993-05-21	40
Abhishek	1985-01-01	38
Amar	1995-01-01	28

📌 **Everything started connecting - clarity at last.**

Row-wise Data Combination Using Set Operators



NOTE:- In MySQL, INTERSECT & EXCEPT are simulated using joins or subqueries.

Once structure is the same, set operators decide which rows stay.

Rules Before Using Set Operators

📌 *Rule: Same Number of Columns*

When using **set operators** (UNION, UNION ALL, INTERSECT, EXCEPT), **each SELECT statement must return the same number of columns.**

☞ SQL combines **rows position-by-position**, not by column name.

```
SELECT id, name
FROM table1
UNION
SELECT id, name
FROM table2;
```

✓ Both SELECT statements return **2 columns** → works.

Set operators stack rows like blocks. If the blocks are not the same width, stacking is impossible.

```
SELECT id, name
FROM table1
UNION
SELECT id, name, salary
FROM table2;
```

✗ First SELECT has 2 columns, second has 3 columns → error.

📌 *All SELECT queries must return the same number of columns for set operators to work.*

📌 *Rule: Same Column Order*

When using set operators,
the order of columns in each SELECT statement must be the same.

☞ SQL matches columns by position, not by column name.

```
SELECT id, name
FROM table1
UNION
SELECT id, name
FROM table2;
```

✓ Column order matches → works correctly.

Column 1 joins Column 1,
Column 2 joins Column 2.
Names don't matter — position does.

```
SELECT id, name
FROM table1
UNION
SELECT name, id
FROM table2;
```

✗ Same columns, but **different order** → wrong result or data mismatch.

📌 *All SELECT statements must follow the same column order when using set operators.*

📌 *Rule: Compatible Data Types*

For set operators to work,
each column position must have compatible data types in all SELECT statements.
☞ SQL cannot combine values of different data types in the same column position.

```
SELECT id, name
FROM table1
UNION
SELECT emp_id, emp_name
FROM table2;
```

✓ id and emp_id → INT
✓ name and emp_name → VARCHAR
✓ Data types match → works.

Column position = same type of data.
Numbers with numbers, text with text,
dates with dates.

```
SELECT id, name
FROM table1
UNION
SELECT salary, joining_date
FROM table2;
```

✗ id (INT) ≠ salary (DECIMAL)
✗ name (VARCHAR) ≠ joining_date (DATE)
→ Error or unexpected result.

📌 *Each column position must have compatible data types when using set operators.*

📌 *Rule: Same Column Meaning*

Even if:
- number of columns is same
- order is same
- data types are compatible

👉 the columns should represent the same kind of data.

SQL will run the query, but the result may not make sense logically.

```
SELECT id, name
FROM students
UNION
SELECT emp_id, emp_name
FROM employees;
```

✓ Both first columns → IDs
✓ Both second columns → Names
✓ Result makes sense.

SQL checks structure, not meaning — logic is your responsibility.

```
SELECT id, name
FROM students
UNION
SELECT salary, department
FROM employees;
```

✗ id mixed with salary
✗ name mixed with department
→ Query runs, but output is confusing.

📌 *Columns should have the same meaning when using set operators.*

📌 *Rule: ORDER BY with Set Operators*

When using set operators,
ORDER BY can be applied only at the very end of the entire query.
☞ You cannot use ORDER BY inside individual SELECT statements.

```
SELECT id, name
FROM table1
UNION
SELECT id, name
FROM table2
ORDER BY name;
```

✓ ORDER BY at the end → works.

Column position = same type of data.
Numbers with numbers, text with text, dates with dates.

```
SELECT id, name
FROM table1
ORDER BY name
UNION
SELECT id, name
FROM table2;
```

✗ ORDER BY before UNION → error.

📌 *ORDER BY is allowed only once, after all set operations are completed.*

📌 *SQL Set Operators with GROUP BY, HAVING & ORDER BY*

```
SELECT
    department,
    SUM(salary) AS total_salary
FROM employees_2023
GROUP BY department
HAVING SUM(salary) > 500000

UNION

SELECT
    department,
    SUM(salary) AS total_salary
FROM employees_2024
GROUP BY department
HAVING SUM(salary) > 500000

ORDER BY total_salary DESC
LIMIT 5;
```

- ***Execution Flow***

1) FROM
2) GROUP BY
3) AGGREGATION (SUM)
4) HAVING
5) **SET OPERATOR (UNION / UNION ALL / INTERSECT / EXCEPT)**
6) ORDER BY *(applies to final combined result)*
7) LIMIT

Key Rules to Remember 📌

1) Each SELECT in a **set operator** must have:
    a. Same number of columns
    b. Compatible data types
2) ORDER BY and LIMIT are written **only once at the end**
3) Column names in ORDER BY come from the **first SELECT**

📌 *Rule: Column Alias*

Column aliases determine the final column names when using set operators.
☞ Use aliases only in the first SELECT statement. The final result will take column names from the first query.

```
SELECT staff_name AS Employee,
 service AS ServiceName
FROM staff_schedule
UNION
SELECT staff_name, service
FROM staff_schedule;
```

✓ Aliases in the first query → final result shows Employee and ServiceName.

The first query "names the columns," and all combined results follow those names.

```
SELECT staff_name, service
FROM staff_schedule
UNION
SELECT staff_name AS Employee,
service AS ServiceName
FROM staff_schedule;
```

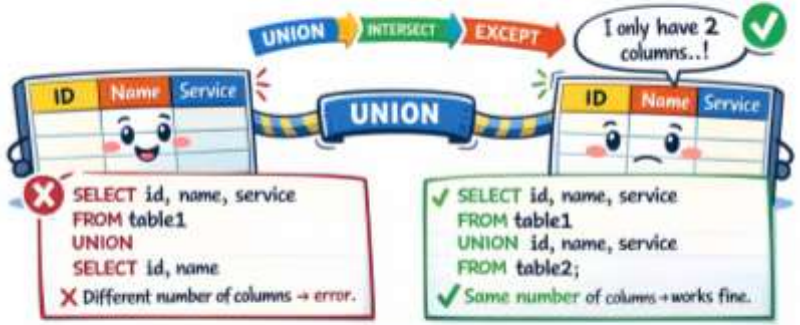
❌ Aliases in the second query → ignored.

📌 *Column aliases in set operators must be in the first query to affect the final result.*

Set Operator Rules — Quick Recap

Rule: Same Number of Columns

All queries in a set operator (UNION, INTERSECT, EXCEPT) must return the same number of columns.

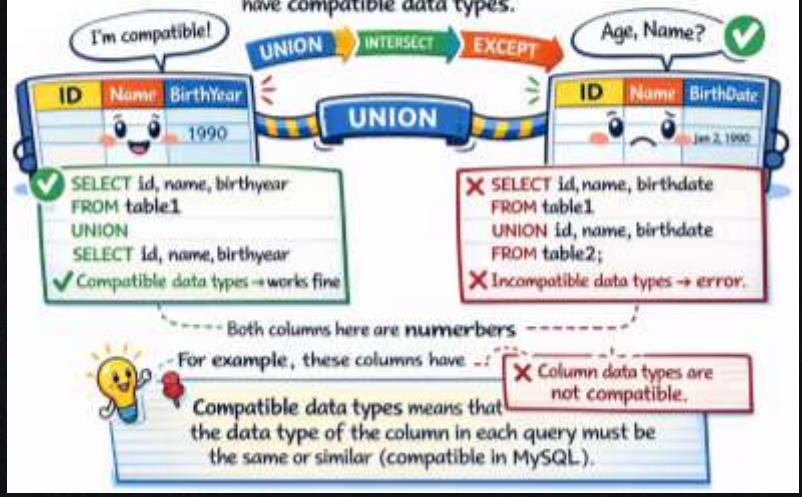


Each column in one query lines up with the column in the same position from the other query.

All queries in a set operator must have the same number of columns, in the same order.

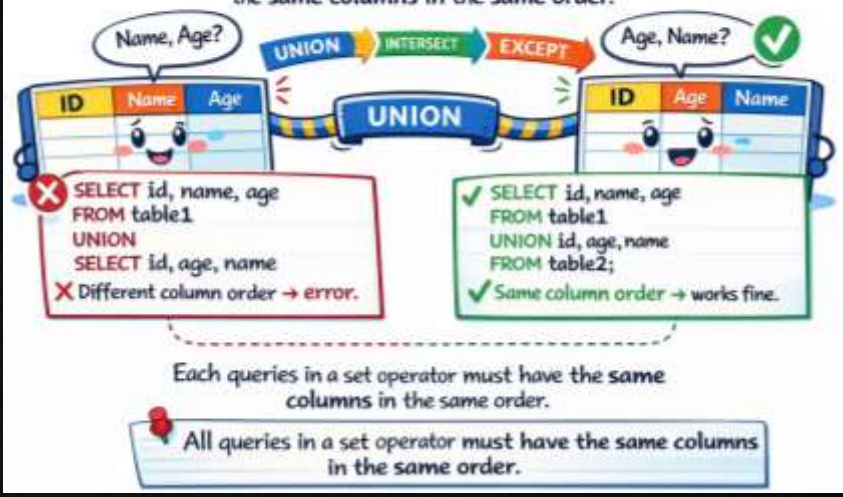
Rule: Compatible Data Types

All queries in a set operator (UNION, INTERSECT, EXCEPT) must have compatible data types.



Rule: Same Column Order

All queries in a set operator (UNION, INTERSECT, EXCEPT) must return the same columns in the same order.

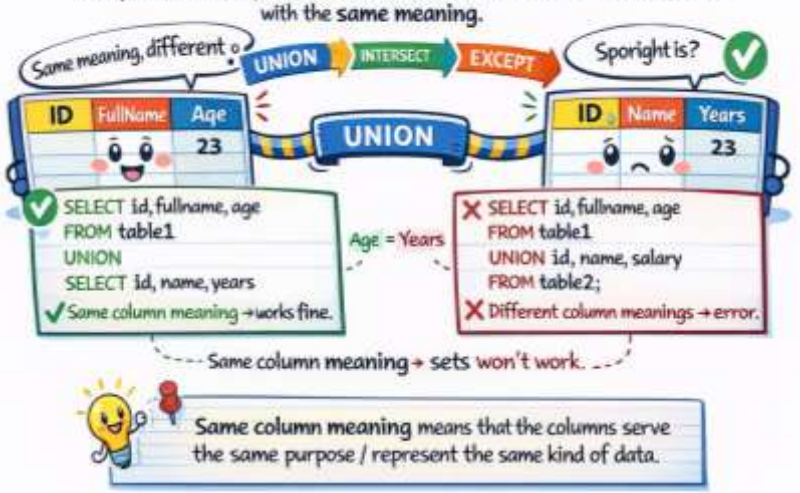


Each queries in a set operator must have the same columns in the same order.

All queries in a set operator must have the same columns in the same order.

Rule: Same Column Meaning

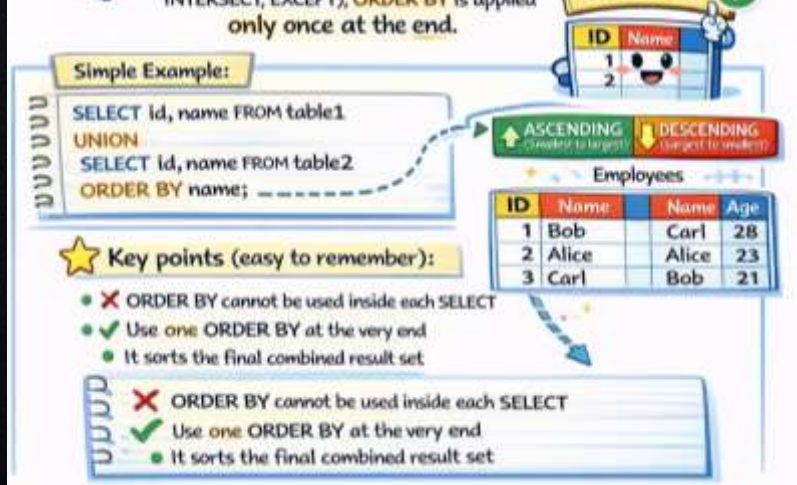
All queries in a set operator (UNION, INTERSECT, EXCEPT) should have columns with the same meaning.



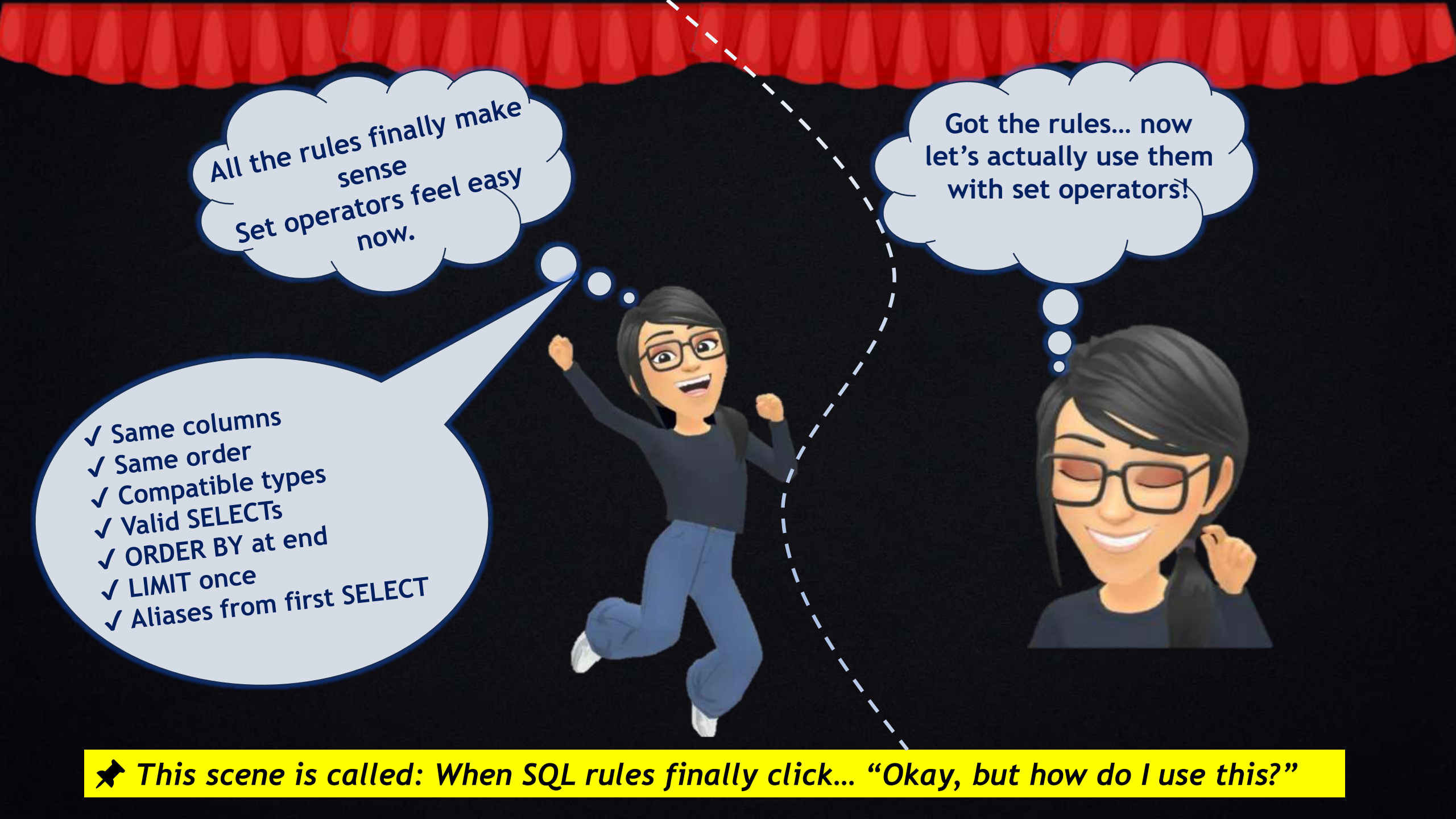
Same column meaning means that the columns serve the same purpose / represent the same kind of data.

ORDER BY Rule with Set Operators:

When using set operators (UNION, UNION ALL, INTERSECT, EXCEPT), ORDER BY is applied only once at the end.



Column aliases in set operators must be in the first query to affect the final result.



📌 *This scene is called: When SQL rules finally click… “Okay, but how do I use this?”*

TechNova – Innovating technology solutions for a smarter tomorrow
Empowering businesses with cutting-edge tech and seamless solutions

Marketing Team

emp_id	name	department
1	Angad	Marketing
2	Rohan	Marketing
3	Priya	Marketing
NULL	NULL	NULL

Sales Team

emp_id	name	department
2	Rohan	Sales
4	Meera	Sales
5	Tanvi	Sales
6	Vikram	Sales
NULL	NULL	NULL

Angad Rohan Priya

What if we want one combined employee list?

Let's see how SQL combines these two team tables.

Rohan Meera Tanvi Vikram

UNION Operator in SQL

UNION in SQL is used to combine rows from two or more queries into a single result set.

- **Removes duplicate rows by default.**

Syntax

```
SELECT column1, column2, ...
FROM table1
WHERE condition

UNION

SELECT column1, column2, ...
FROM table2
WHERE condition;
```

In TechNova , we have two teams:
Marketing and Sales.
Rohan (Emp ID 2) is part of both teams.
Let’s see what happens when we use the UNION operator to generate the employee list.

What employee list do we get (Emp ID & Name) when we use UNION on Marketing and Sales teams in TechNova?

Query

```
SELECT emp_id, name
FROM team_marketing
UNION
SELECT emp_id, name
FROM team_sales
ORDER BY emp_id;
```

We can see **Rohan appears twice (in both Marketing and Sales)**, but when we use UNION, it returns Rohan only once because **UNION removes duplicate records.**

UNION = DISTINCT rows

Result

emp_id	name
1	Angad
2	Rohan
3	Priya
4	Meera
5	Tanvi
6	Vikram

What happens when we include the Department along with Emp ID and Name using UNION?

Query

```
SELECT emp_id, name, department
FROM team_marketing
UNION
SELECT emp_id, name, department
FROM team_sales
ORDER BY emp_id;
```

We see Rohan twice because the department is different (Marketing and Sales), so UNION treats them as separate records.

UNION checks all selected columns

Result

emp_id	name	department
1	Angad	Marketing
2	Rohan	Marketing
2	Rohan	Sales
3	Priya	Marketing
4	Meera	Sales
5	Tanvi	Sales
6	Vikram	Sales

UNION ALL Operator in SQL

UNION ALL is used to combine the result of two or more SELECT queries and keeps all rows, including duplicates.

- **Use UNION ALL if you want to include duplicates.**

Syntax

```
SELECT column1, column2, ...
FROM table1
WHERE condition

UNION ALL

SELECT column1, column2, ...
FROM table2
WHERE condition;
```

In TechNova , we have two teams:
Marketing and Sales.
Rohan (Emp ID 2) is part of both teams.
Let’s see what happens when we use the
UNION ALL operator to generate the employee list.

What employee list do we get (Emp ID & Name) when we use UNION ALL on Marketing and Sales teams in TechNova?

Query

```
SELECT emp_id, name
FROM team_marketing
UNION ALL
SELECT emp_id, name
FROM team_sales
ORDER BY emp_id ;
```

We see Rohan twice because **UNION ALL** keeps all records and does not remove duplicates.

UNION ALL keeps duplicates

Result

emp_id	name
1	Angad
2	Rohan
2	Rohan
3	Priya
4	Meera
5	Tanvi
6	Vikram

What happens when we use UNION ALL with Emp ID, Name, and Department?

Query

```
SELECT emp_id, name, department
FROM team_marketing
UNION ALL
SELECT emp_id, name, department
FROM team_sales
ORDER BY emp_id;
```

UNION ALL returns all rows from both queries without removing duplicates.

Result

emp_id	name	department
1	Angad	Marketing
2	Rohan	Marketing
2	Rohan	Sales
3	Priya	Marketing
4	Meera	Sales
5	Tanvi	Sales
6	Vikram	Sales

UNION vs UNION ALL — Know the Difference

UNION

- Removes duplicate rows
- Slower (extra step to eliminate duplicates)
- Acts like DISTINCT
- Returns unique records only

UNION ALL

- Keeps all rows
- Faster performance
- Does not remove duplicates
- Includes repeated records

📌 *UNION removes duplicates — UNION ALL keeps everything.*

INTERSECT Operator in SQL

INTERSECT returns only the common rows that appear in both queries.

☞ Think of it as “common data in both tables”

Syntax

```
SELECT column1, column2
FROM table1

INTERSECT

SELECT column1, column2
FROM table2;
```

QUERY

```
SELECT DISTINCT s.name
FROM team_sales s
INNER JOIN team_marketing m
ON s.name = m.name;
```

O
R

QUERY

```
SELECT name
FROM team_sales
WHERE name IN (SELECT name
FROM team_marketing);
```

MySQL

MySQL Workbench

This syntax works in:
MySQL, MySQL Workbench

Both queries return the same structure.

name
Rohan

- **INTERSECT returns only the records that appear in both queries.**
- In this case, **Rohan** is common to both teams, so he is returned.

📌 ***INTERSECT returns only rows that exist in both result sets.***

EXCEPT / MINUS Operator in SQL

EXCEPT (called MINUS in Oracle) returns rows from the first query that do NOT appear in the second query.

☞ Think of it as “left table minus right table”

Syntax

```
SELECT column1, column2
FROM table1

EXCEPT

SELECT column1, column2
FROM table2;
```

EXCEPT SQL operator works in

Syntax

```
SELECT column1, column2
FROM table1

MINUS

SELECT column1, column2
FROM table2;
```

MINUS SQL operator works in Oracle

QUERY

```
SELECT m.name
FROM team_marketing m
LEFT JOIN team_sales s
ON m.name = s.name
WHERE s.name IS NULL;
```

MySQL

MySQL Workbench

This syntax works in:
MySQL, MySQL Workbench

name
Angad
Priya

- LEFT JOIN keeps **all people from Marketing**
- SQL checks if each name exists in **Sales**
- If a name **does not exist in Sales**, Sales columns become NULL
- WHERE s.name IS NULL keeps only those names

Result :

- **Angad** → not in Sales ✓
- **Priya** → not in Sales ✓
- **Rohan** → in Sales ✗ (removed)

📌 *This query returns Marketing team members who are not present in the Sales team — same result as EXCEPT.*

QUERY

```
SELECT name
FROM team_marketing
EXCEPT
SELECT name
FROM team_sales;
```

name
Angad
Priya

- EXCEPT returns all rows from the **first table** that are **not in the second table**.
- MySQL does not natively support EXCEPT.
- If it's working in your Workbench, it might be:
  1) You are connected to a different database (like MariaDB, PostgreSQL, or SQL Server).
  2) Or MySQL version/plugin allows it.

📌 ***MySQL Workbench can execute EXCEPT when connected to MariaDB, even though MySQL itself does not support it.***

INTERSECT vs EXCEPT

- Table A 👥 + Table B 👥
- **Common rows appear in both tables.**
- **INTERSECT → Only matching rows from both tables**

- Table A 👥 → Table B ❌
- Characters that exist only in the first query
- EXCEPT → Rows in first table NOT in second

📌 *INTERSECT = Common | EXCEPT = Difference*

Finally! All set operators are clear — no more confusion about when to use which one!

UNION → Combine, remove duplicates
UNION ALL → Combine, keep duplicates
INTERSECT → Only common rows
EXCEPT → First minus second
MINUS → Oracle version of EXCEPT

Time to recap and put what we've learned into practice!

📌 *Clarity achieved… now let's try it with real queries!*

Practice Exercise

TechNova: Project Alpha & Project Beta Dataset

Project Alpha Dataset

emp_id	name	department	city
101	Amit	IT	Delhi
102	Riya	HR	Mumbai
103	Karan	Finance	Pune
104	Sneha	IT	Bangalore
105	Pooja	Marketing	Delhi
106	Arjun	Sales	Chennai
107	Nikhil	IT	Hyderabad
108	Isha	HR	Pune

Project Beta Dataset

emp_id	name	department	city
103	Karan	Finance	Pune
104	Sneha	IT	Bangalore
106	Arjun	Sales	Chennai
108	Isha	HR	Pune
109	Rahul	Marketing	Mumbai
110	Mehul	IT	Delhi
111	Ananya	HR	Kolkata
112	Sonal	Finance	Mumbai
113	Varun	Sales	Jaipur
114	Kavya	Marketing	Bangalore

Query the datasets to find the information TechNova needs.

-- Get a unique list of all employee names working in either Project Alpha or Project Beta.

QUERY

```
SELECT
    name AS Employee_Name
FROM project_alpha
UNION
SELECT
    name
FROM project_beta;
```

RESULT

Employee_Name
Amit
Riya
Karan
Sneha
Pooja
Arjun
Nikhil
Isha
Rahul
Mehul
Ananya
Sonal
Varun
Kavya

-- Get all employee records from both projects including duplicates.

QUERY

```sql
SELECT
    "project_alpha",
    emp_id,
    name AS Employee_Name
FROM project_alpha
UNION ALL
SELECT
    "project_beta",
    emp_id,
    name
FROM project_beta;
```

RESULT

project_alpha	emp_id	Employee_Name
project_alpha	101	Amit
project_alpha	102	Riya
project_alpha	103	Karan
project_alpha	104	Sneha
project_alpha	105	Pooja
project_alpha	106	Arjun
project_alpha	107	Nikhil
project_alpha	108	Isha
project_beta	103	Karan
project_beta	104	Sneha
project_beta	106	Arjun
project_beta	108	Isha
project_beta	109	Rahul
project_beta	110	Mehul
project_beta	111	Ananya
project_beta	112	Sonal
project_beta	113	Varun
project_beta	114	Kavya

-- Find employees who are working on both Project Alpha and Project Beta.

QUERY

```
SELECT
    DISTINCT pa.name AS Employee_name
FROM project_alpha pa
INNER JOIN project_beta pb
ON pa.name = pb.name;
```

RESULT

Employee_name
Karan
Sneha
Arjun
Isha

-- Find employees who are working only in Project Alpha and not in Project Beta. (Alpha − Beta)

QUERY

```
SELECT
    name AS Employee_Name
FROM project_alpha
EXCEPT
SELECT
    name
FROM project_beta ;
```

RESULT

Employee_Name
Amit
Riya
Pooja
Nikhil

-- Find employees who are working only in Project Beta and not in Project Alpha. (Beta – Alpha)

QUERY

```
SELECT
    name AS Employee_Name
FROM project_beta
EXCEPT
SELECT
    name
FROM project_alpha ;
```

RESULT

Employee_Name
Rahul
Mehul
Ananya
Sonal
Varun
Kavya

-- Combine both project tables and find the total number of employees per department.

QUERY

```sql
SELECT
    department,
    COUNT(*) AS total_employees
FROM (
    SELECT department FROM project_alpha
    UNION ALL
    SELECT department FROM project_beta
) combined_projects
GROUP BY department;
```

RESULT

department	total_employees
IT	5
HR	4
Finance	3
Marketing	3
Sales	3

-- From the combined data, show only those departments that have more than 2 employees.

QUERY

```sql
SELECT
    department,
    COUNT(*) AS total_employees
FROM (
    SELECT department FROM project_alpha
    UNION ALL
    SELECT department FROM project_beta
) combined_projects
GROUP BY department
HAVING total_employees > 2 ;
```

RESULT

department	total_employees
IT	5
HR	4
Finance	3
Marketing	3
Sales	3

-- From both projects together, find the top 3 cities with the highest employee count.

QUERY

```sql
SELECT
    city,
    COUNT(*) AS total_employees
FROM(
    SELECT city FROM project_alpha
    UNION ALL
    SELECT city FROM project_beta
)combined_project
GROUP BY city
ORDER BY total_employees DESC
LIMIT 3;
```

RESULT

city	total_employees
Pune	4
Delhi	3
Mumbai	3

-- Find employees who are present in both projects, then show the count of such employees by department, and display only departments with more than 1 common employee.

QUERY

```sql
SELECT
    department,
    COUNT(*) AS total_employees
FROM (
    SELECT DISTINCT
        pa.emp_id,
        pa.department
    FROM project_alpha pa
    INNER JOIN project_beta pb
        ON pa.emp_id = pb.emp_id
) common_employees
GROUP BY department
HAVING total_employees > 1;
```

RESULT

department	total_employees

-- From both projects combined, list departments sorted by employee count (descending) and return only the top 2 departments.

QUERY

```sql
SELECT
    department,
    COUNT(*) AS total_employees
FROM (
    SELECT department FROM project_alpha
    UNION ALL
    SELECT department FROM project_beta
) combined_projects
GROUP BY department
ORDER BY  total_employees DESC
LIMIT 2 ;
```

RESULT

department	total_employees
IT	5
HR	4

✨ From confusion to confidence.
✨ Structured learning + real practice made it click.
✨ No shortcuts — just consistency and the right resources.

Learning SQL Set Operators became easy once I stopped rushing and learned step by step.
With the right resources, real datasets, and consistent practice — everything finally made sense.

Clarity comes from practice.

(From learner → mentor vibe)


THANK YOU
Happy learning. Keep practicing.