```sql
INSERT INTO customers (id, first_name, country, score)
VALUES
    (8, 'USA', 'Max', NULL)

SELECT * FROM customers
```

Messages

```
(1 row affected)

Completion time: 2025-02-16T14:54:39.507605
```

Due to data type rules and its constraints

```sql
INSERT INTO customers (id, first_name, country, score)
VALUES
    (8, 'USA', 'Max', NULL)

RULES FROM customers
```

Matching Data Types, Column Count & Constraints

| | id | first_name | country | score |
|---|---|---|---|---|
| 1 | 1 | Maria | Germany | 350 |
| 2 | 2 | John | USA | 900 |
| 3 | 3 | Georg | UK | 750 |
| 4 | 4 | Martin | Germany | 500 |
| 5 | 5 | Peter | USA | 0 |
| 6 | 6 | Anna | USA | NULL |
| 7 | 7 | Sam | NULL | 100 |
| 8 | 8 | USA | Max | NULL |

```sql
INSERT INTO customers (id, first_name, country, score)
VALUES
    ('Max', 9, 'Max', NULL)

RULES FROM customers
```

Matching Data Types, Column Count & Constraints

```
Msg 245, Level 16, State 1, Line 1
Conversion failed when converting the varch

Completion time: 2025-02-16T14:55:38.832552
```

```
INSERT INTO customers (id, first_name, country, score)
VALUES
    ('Max', 9, 'Max', NULL)
```

NOTE

You can skip the columns if you insert values for every column

| | id | first_name | country | score |
|---|---|---|---|---|
| 1 | 1 | Maria | Germany | 350 |
| 2 | 2 | John | USA | 900 |
| 3 | 3 | Georg | UK | 750 |
| 4 | 4 | Martin | Germany | 500 |
| 5 | 5 | Peter | USA | 0 |
| 6 | 6 | Anna | USA | NULL |
| 7 | 7 | Sam | NULL | 100 |
| 8 | 8 | USA | Max | NULL |

```
INSERT INTO customers
VALUES
    (9, 'Andreas', 'Germany', NULL)
```

TIP

Always list columns explicitly for clarity and maintainability

```
(1 row affected)

Completion time: 2025-02-16T14:58:04.3921015
```

```
INSERT INTO customers
VALUES
    (9, 'Andreas', 'Germany', NULL)
```

TIP

Always list columns explicitly for clarity and maintainability

```
(1 row affected)

Completion time: 2025-02-16T14:58:04.3921015
```

Column becomes null if we not enter value in by we need not to explicitly mention Null by using this way
But always insert column which is not null

```sql
INSERT INTO customers (id, first_name)
VALUES
    (10, 'Sahra')

NOTE * FROM customers
```

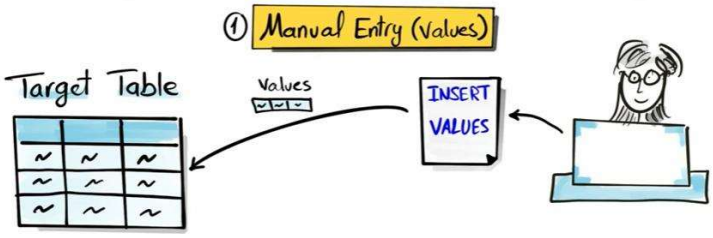
Columns not included in INSERT become NULL (unless a default or constraint exists)

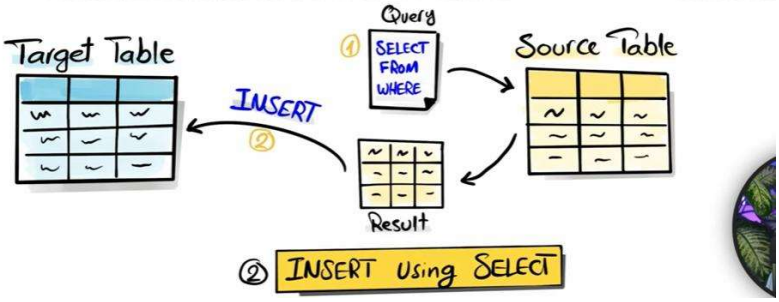
```
(1 row affected)

Completion time: 2025-02-16T15:01:04.335700
```

```sql
INSERT INTO customers (id, first_name)
VALUES
    (10, 'Sahra')

NOTE * FROM customers
```

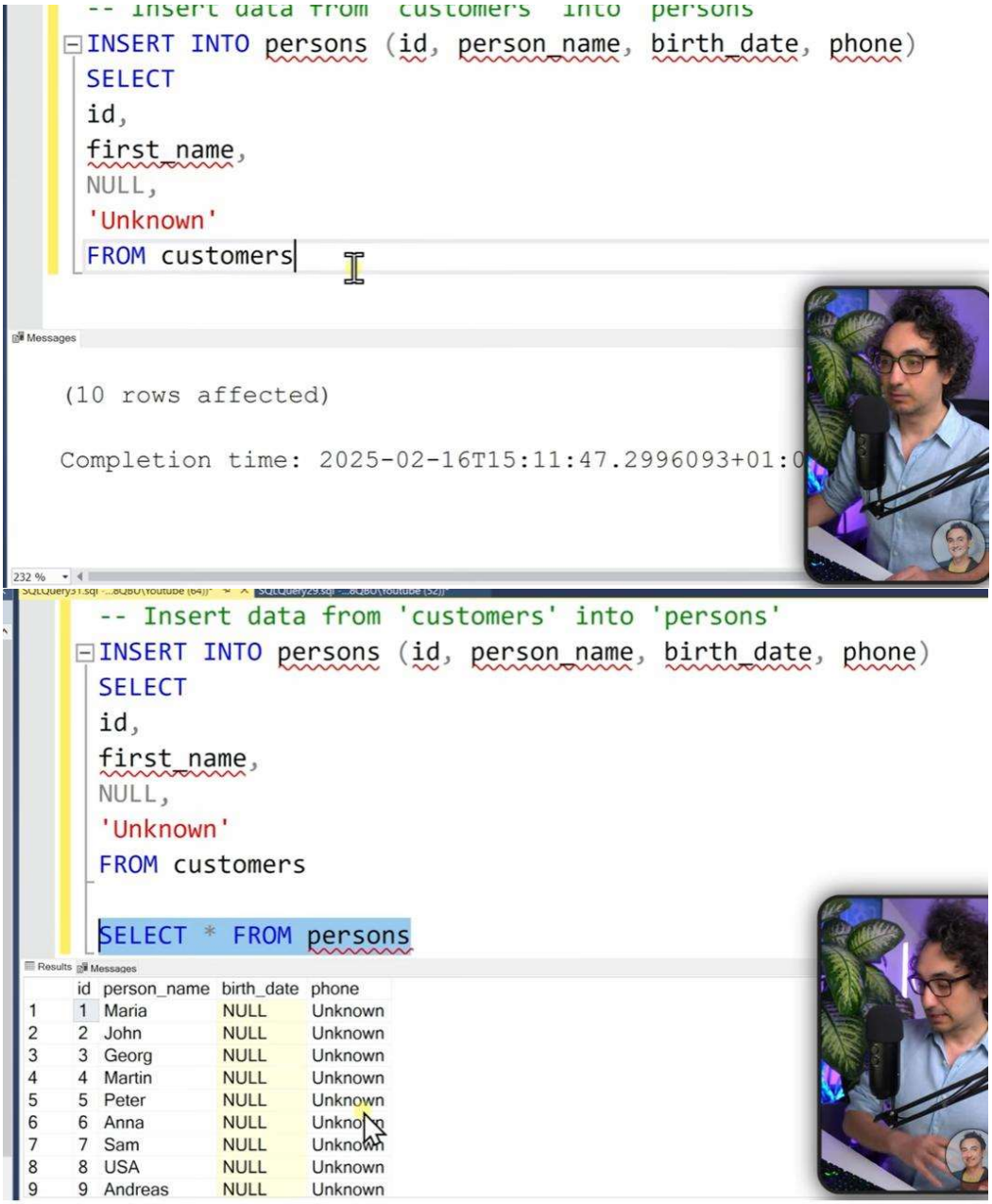
Columns not included in INSERT become NULL (unless a default or constraint exists)

| | id | first_name | country | score |
|---|---|---|---|---|
| 1 | 1 | Maria | Germany | 350 |
| 2 | 2 | John | USA | 900 |
| 3 | 3 | Georg | UK | 750 |
| 4 | 4 | Martin | Germany | 500 |
| 5 | 5 | Peter | USA | 0 |
| 6 | 6 | Anna | USA | NULL |
| 7 | 7 | Sam | NULL | 100 |
| 8 | 8 | USA | Max | NULL |
| 9 | 9 | Andreas | Germany | NULL |
| 10 | 10 | Sahra | NULL | NULL |



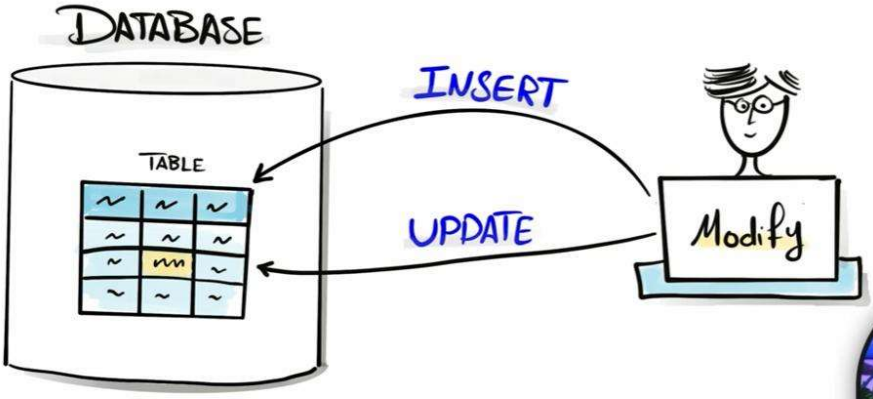
Insert data from one table to another table




DML Command Update:

Update the data of already existing rows

# Modify (Manipulate) Your Data




UPDATE
Syntax

```
UPDATE table_name
  SET  column1 = value1,
       column2 = value2
 WHERE <condition>
```

**NOTE**

- Always use WHERE to avoid UPDATING all rows unintentionally

```sql
/* Change the score of customer 6 to 0 */
UPDATE customers
SET score = 0
```

**CAUTION**

Without a WHERE, all rows will be updated!

Results | Messages

| | id | first_name | country | score |
|---|---|---|---|---|
| 1 | 1 | Maria | Germany | 350 |
| 2 | 2 | John | USA | 900 |
| 3 | 3 | Georg | UK | 750 |
| 4 | 4 | Martin | Germany | 500 |
| 5 | 5 | Peter | USA | 0 |
| 6 | 6 | Anna | USA | NULL |
| 7 | 7 | Sam | NULL | 100 |
| 8 | 8 | USA | Max | NULL |
| 9 | 9 | Andreas | Germany | NULL |
| 10 | 10 | Sahra | NULL | NULL |

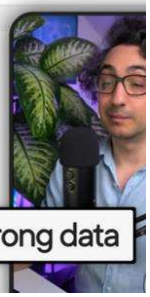
```sql
/* Change the score of customer 6 to 0 */
UPDATE customers
SET score = 0
WHERE id = 6

SELECT *
FROM customers
WHERE id = 6
```

Messages

| id | first_name | country | score |
|---|---|---|---|
| 6 | Anna | USA | NULL |

**BEST PRACTICE**

heck with SELECT before running UPDATE to avoid updating the wrong data



```sql
/* Change the score of customer 6 to 0 */
UPDATE customers
SET score = 0
WHERE id = 6

SELECT *
FROM customers
```

Results Messages

| | id | first_name | country | score |
|---|---|---|---|---|
| 1 | 1 | Maria | Germany | 350 |
| 2 | 2 | John | USA | 900 |
| 3 | 3 | Georg | UK | 750 |
| 4 | 4 | Martin | Germany | 500 |
| 5 | 5 | Peter | USA | 0 |
| 6 | 6 | Anna | USA | 0 |
| 7 | 7 | Sam | NULL | 100 |
| 8 | 8 | USA | Max | NULL |
| 9 | 9 | Andreas | Germany | NULL |
| 10 | 10 | Sahra | NULL | NULL |

17:18 / 25:06

```sql
/* Change the score of customer 10 to 0
   and update the country to UK */
UPDATE customers
SET score = 0,
    country = 'UK'
WHERE id = 10

SELECT *
FROM customers
```

32 %

| | | | | |
|---|---|---|---|---|
| 1 | 1 | Maria | Germany | 350 |
| 2 | 2 | John | USA | 900 |
| 3 | 3 | Georg | UK | 750 |
| 4 | 4 | Martin | Germany | 500 |
| 5 | 5 | Peter | USA | 0 |
| 6 | 6 | Anna | USA | 0 |
| 7 | 7 | Sam | NULL | 100 |
| 8 | 8 | USA | Max | NULL |
| 9 | 9 | Andreas | Germany | NULL |
| 10 | 1.. | Sahra | UK | 0 |

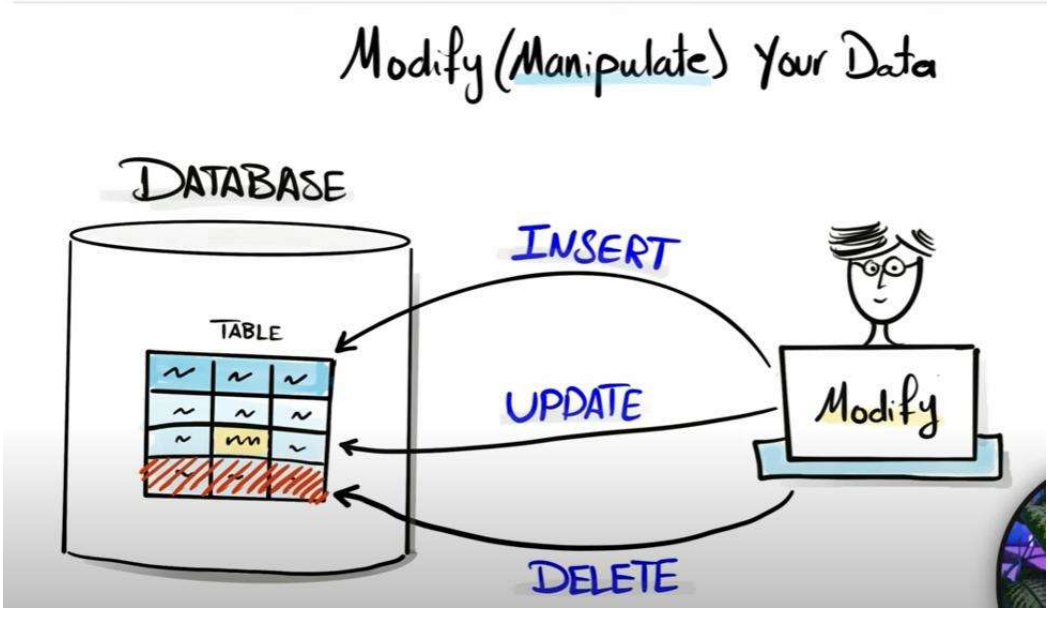
18:19 / 25:06

```sql
/* Update all customers with a NULL score
   by setting their score to 0 */

UPDATE customers
SET score = 0
WHERE score IS NULL

SELECT *
FROM customers
WHERE score IS NULL
```

Results Messages

| | id | first_name | country | score |
|---|---|---|---|---|
| 1 | 1 | Maria | Germany | 350 |
| 2 | 2 | John | USA | 900 |
| 3 | 3 | Georg | UK | 750 |
| 4 | 4 | Martin | Germany | 500 |
| 5 | 5 | Peter | USA | 0 |
| 6 | 6 | Anna | USA | 0 |
| 7 | 7 | Sam | NULL | 100 |
| 8 | 8 | USA | Max | 0 |
| 9 | 9 | Andreas | Germany | 0 |
| 10 | 10 | Sahra | UK | 0 |

Click to select the whole column



# DELETE
## Syntax

```
DELETE FROM table_name
WHERE <condition>
```

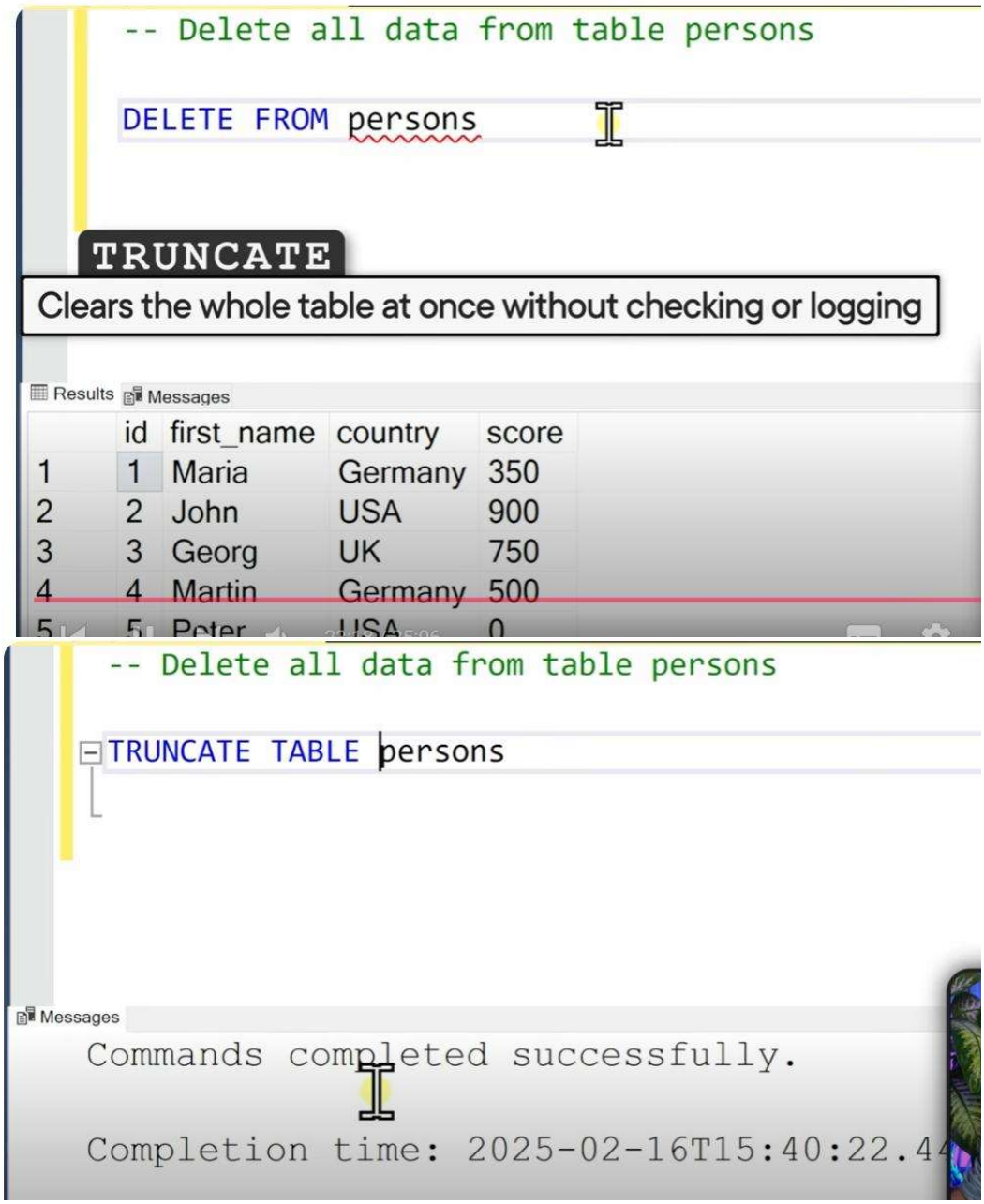
**NOTE**

- Always use WHERE to avoid DELETING all rows unintentionally

```
-- Delete all customers with an ID greater than 5.

DELETE FROM customers
WHERE id > 5

SELECT *
FROM customers
```

Results | Messages

| id | first_name | country | score |
|---|---|---|---|
| 1 | Maria | Germany | 350 |
| 2 | John | USA | 900 |
| 3 | Georg | UK | 750 |
| 4 | Martin | Germany | 500 |
| 5 | Pet[illegible] | USA | 0 |




Delete row-by row deletion

1. DELETE

✓ What it does:

Deletes **specific rows** from a table using a `WHERE` clause.

Can delete **all rows** if `WHERE` is omitted.

**Can be rolled back** (if inside a transaction).

2. TRUNCATE

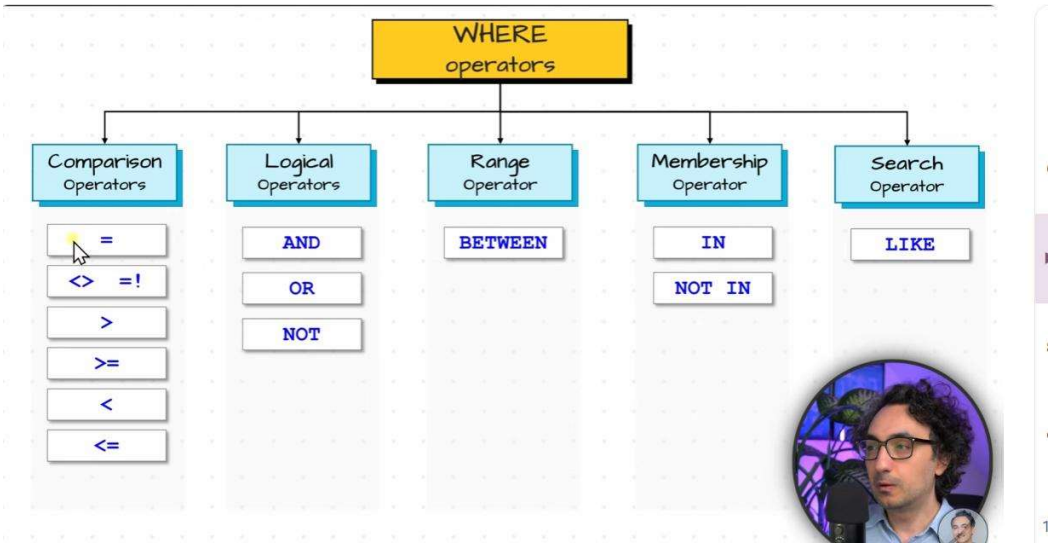
✔ What it does:

Deletes all rows from the table. Faster than **DELETE** (no row-by-row deletion). Cannot be rolled back (in most databases). Resets auto-increment counters.

3. DROP

✔ What it does:

**Completely deletes the table** from the database. Removes both **data and structure**. Cannot be rolled back.




Comparision Operator:

Compare two Things! **Comparison Operators**

Condition → Expression | Operator | Expression

| | | |
|---|---|---|
| Column1 = Column2 | first_name = | Last_name |
| Column1 = Value | first_name = | 'John' |
| Function = Value | UPPER (first_name) = | 'JOHN' |
| Expression = Value | Price * Quantity = | 1000 |
| Subquery = value | (SELECT AVG(sales) FROM orders ) = | 1000 |

Advanced

WHERE

Condition

Country = 'USA'

| Name | Country | Score | |
|---|---|---|---|
| Maria | Germany | 350 | ✗ |
| John | USA | 900 | ✓ |
| Georg | UK | 750 | ✗ |
| Martin | Germany | 500 | ✗ |
| Peter | USA | 0 | ✓ |

SQLQuery44.sql -...8QBU\Youtube (62))*

```sql
-- Retrieve all customers from Germany.
SELECT *
FROM customers
WHERE country = 'Germany'
```

Results Messages

| | id | first_name | country | score |
|---|---|---|---|---|
| 1 | 1 | Maria | Germany | 350 |
| 2 | 4 | Martin | Germany | 500 |

```sql
-- Retrieve all customers who are not from Germany.
SELECT *
FROM customers
WHERE country != 'Germany'
```

sults Messages

| | st_name | country | score |
|---|---|---|---|
| 2 | John | USA | 900 |
| 3 | Georg | UK | 750 |
| 5 | Peter | USA | 0 |

6:16 / 40:37

# Comparison Operators

| | |
|---|---|
| = | Checks if two values are equal |
| <> != | Checks if two values are not equal |


| | id | first_name | country | score |
|---|---|---|---|---|
| 1 | 2 | John | USA | 900 |
| 2 | 3 | Georg | UK | 750 |
| 3 | 5 | Peter | USA | 0 |


| | id | first_name | country | score |
|---|---|---|---|---|
| 1 | 2 | John | USA | 900 |
| 2 | 3 | Georg | UK | 750 |

```sql
-- Retrieve all customers with a score of 500 or more.
SELECT *
FROM customers
WHERE score >= 500
```

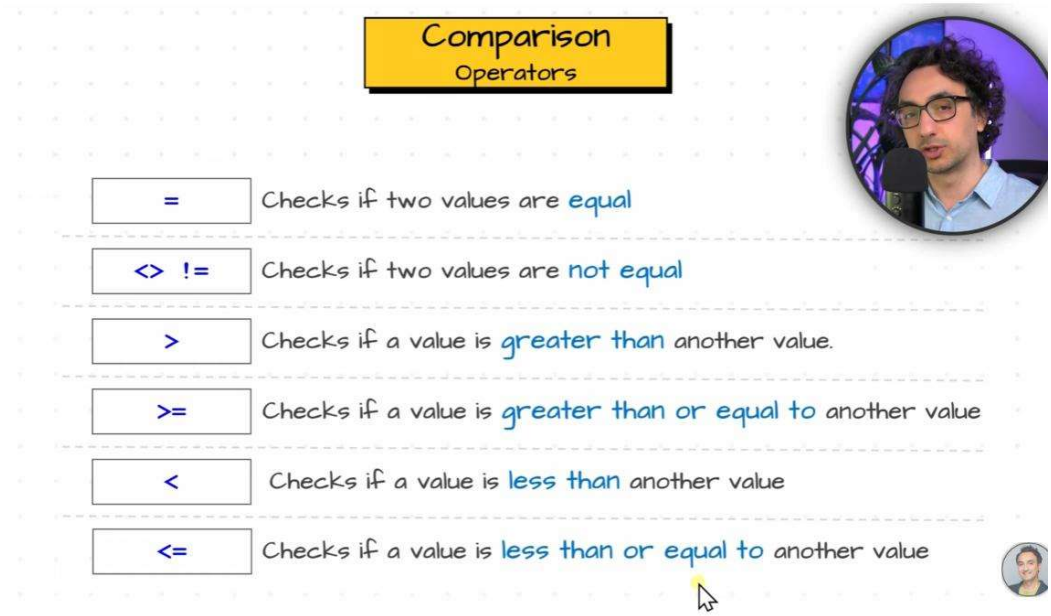
Results Messages

| | id | first_name | country | score |
|---|---|---|---|---|
| 1 | 2 | John | USA | 900 |
| 2 | 3 | Georg | UK | 750 |
| 3 | 4 | Martin | Germany | 500 |

```sql
-- Retrieve all customers with a score less than 500.
SELECT *
FROM customers
WHERE score < 500
```

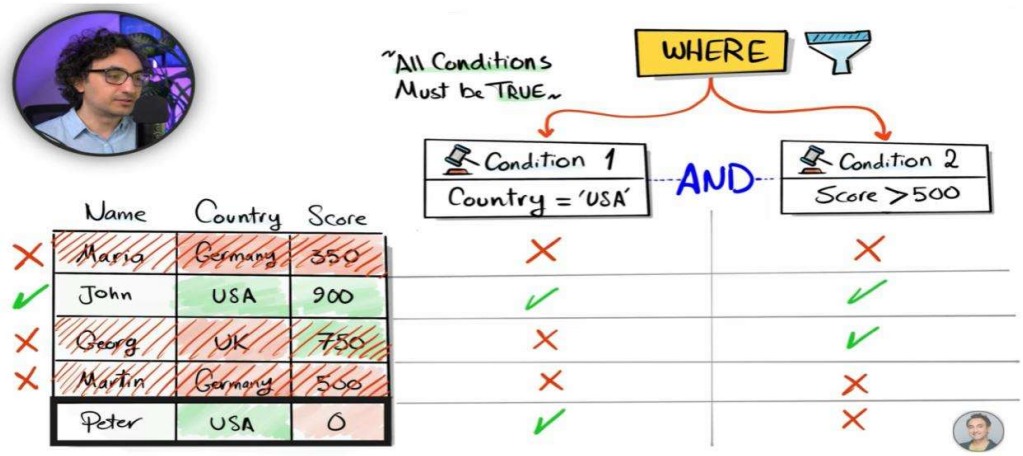
Results Messages

| | id | first_name | country | score |
|---|---|---|---|---|
| 1 | 1 | Maria | Germany | 350 |
| 2 | 5 | Peter | USA | 0 |





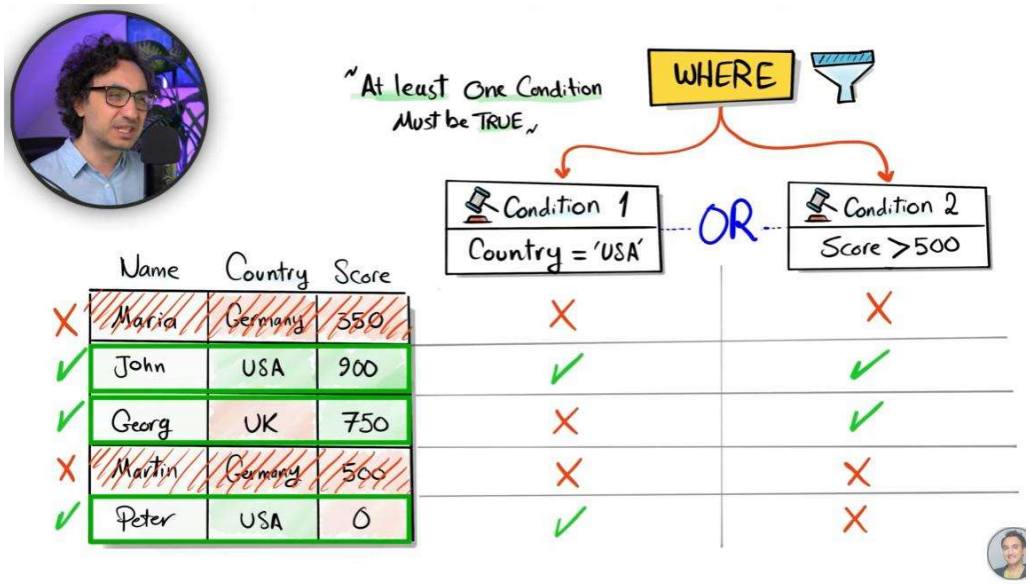
Logical Operators:

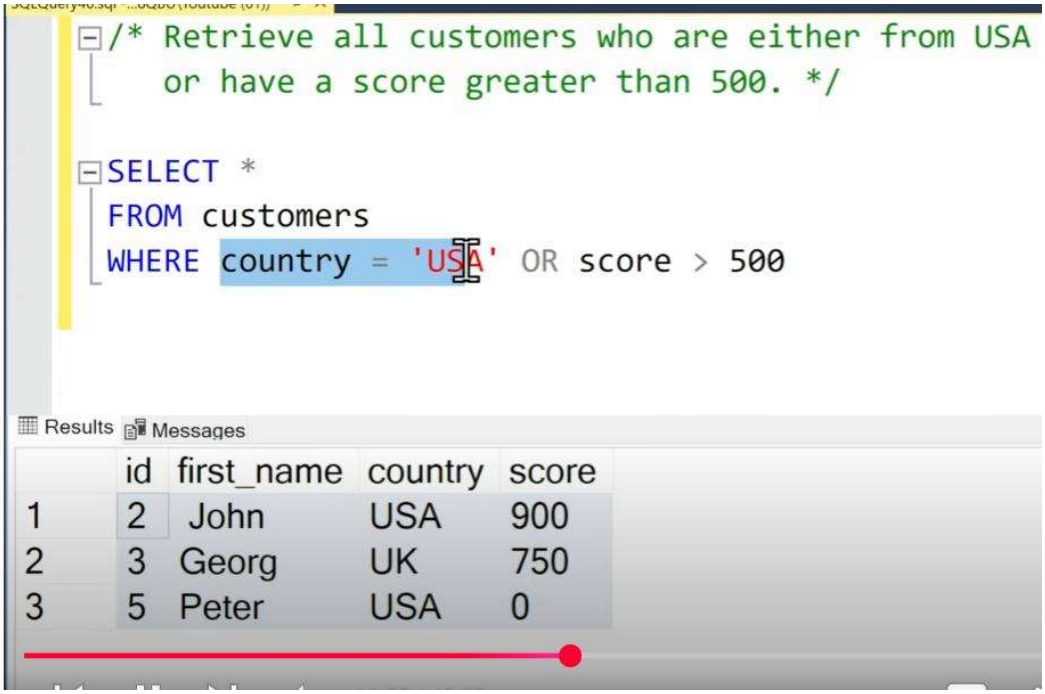



```sql
/* Retrieve all customers who are from USA
   and have a score greater than 500. */

SELECT *
FROM customers
WHERE country = 'USA' AND score > 500
```

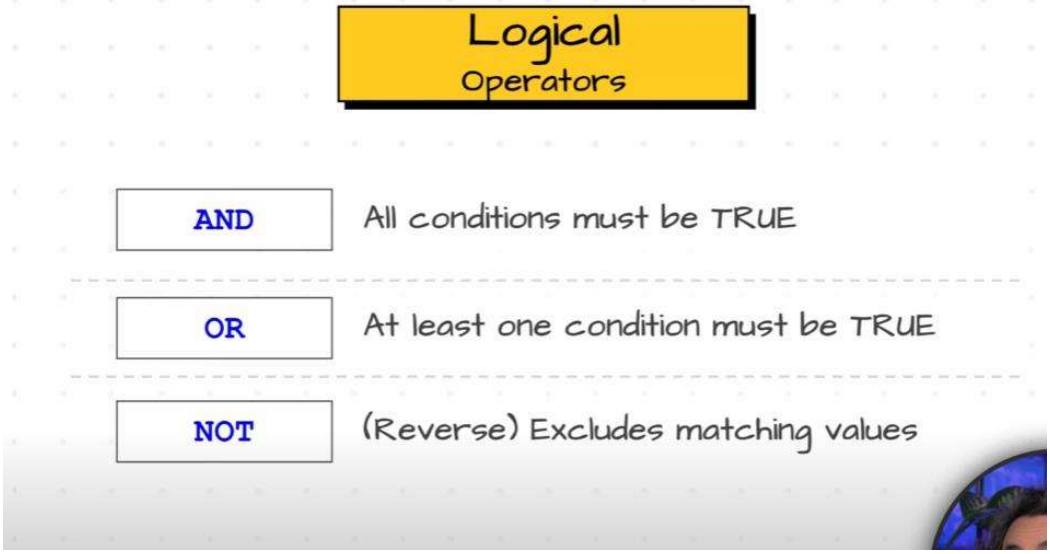
Results Messages

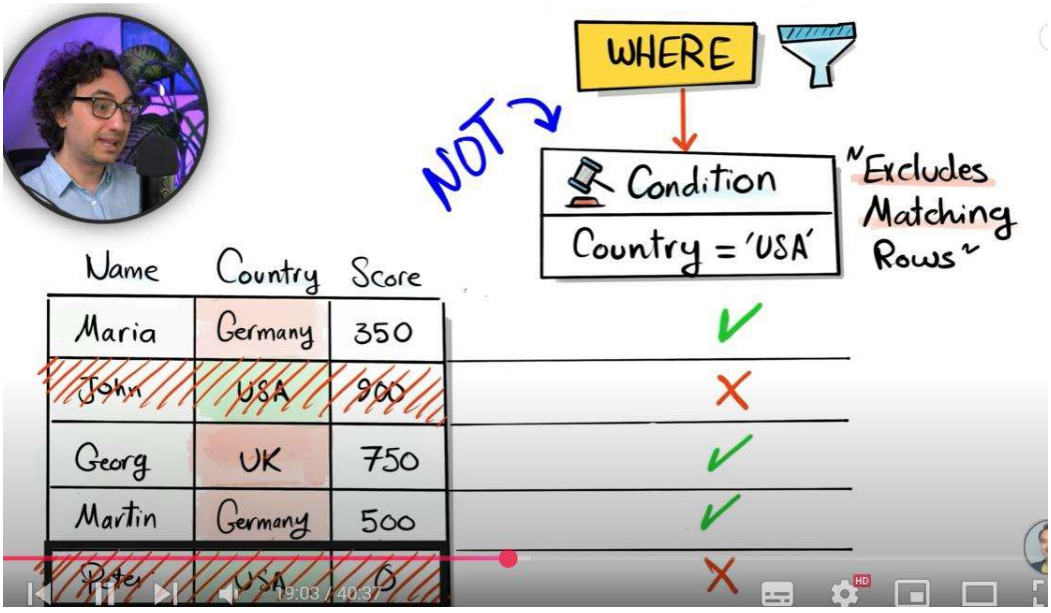
| | id | first_name | country | score |
|---|---|---|---|---|
| 1 | 2 | John | USA | 900 |




Not Operator reverse the matching values or Excludes.
If condition fulfilled it removes the fire result or exclude the data from result
It makes true - false
False - true


Logical
Operators
AND
All conditions must be TRUE
OR
At least one condition must be TRUE
NOT
(Reverse) Excludes matching values


WHERE
NOT
Condition
Country = 'USA'
Excludes Matching Rows
Name
Country
Score
Maria
Germany
350
John
USA
900
Georg
UK
750
Martin
Germany
500
Pete
USA
0
19:03 / 40:37

```sql
-- Retrieve all customers with a score not less than 500
SELECT
*
FROM customers
WHERE score >= 500
```

Results | Messages

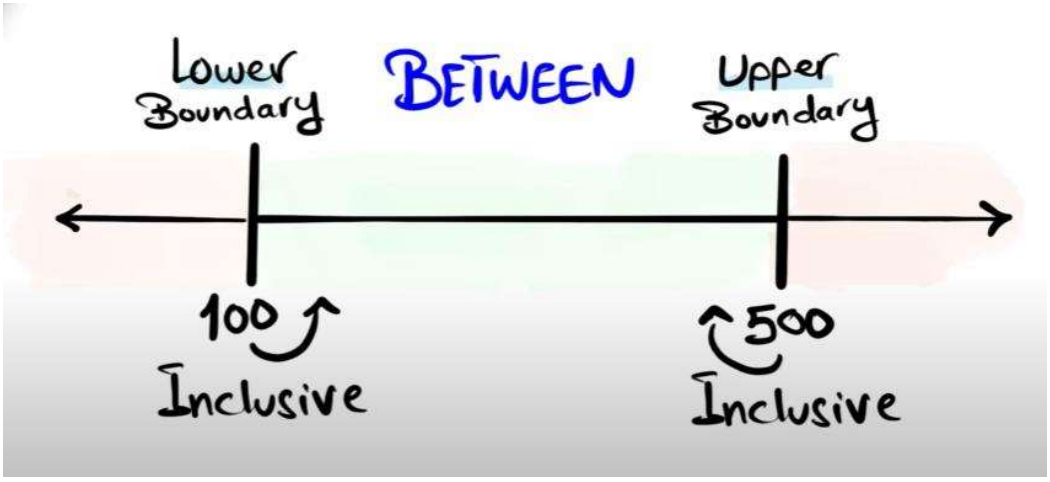
| | id | first_name | country | score |
|---|---|---|---|---|
| 1 | 2 | John | USA | 900 |
| 2 | 3 | Georg | UK | 750 |
| 3 | 4 | Martin | Germany | 500 |

```sql
-- Retrieve all customers with a score not less than 500
SELECT
*
FROM customers
WHERE NOT score < 500
```

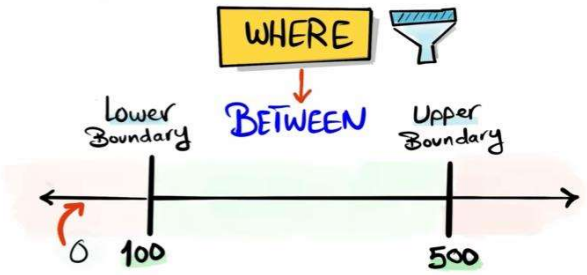
Results | Messages

| | id | first_name | country | score |
|---|---|---|---|---|
| 1 | 2 | John | USA | 900 |
| 2 | 3 | Georg | UK | 750 |
| 3 | 4 | Martin | Germany | 500 |

Range Operators:




If the value is 100 and 500 - true



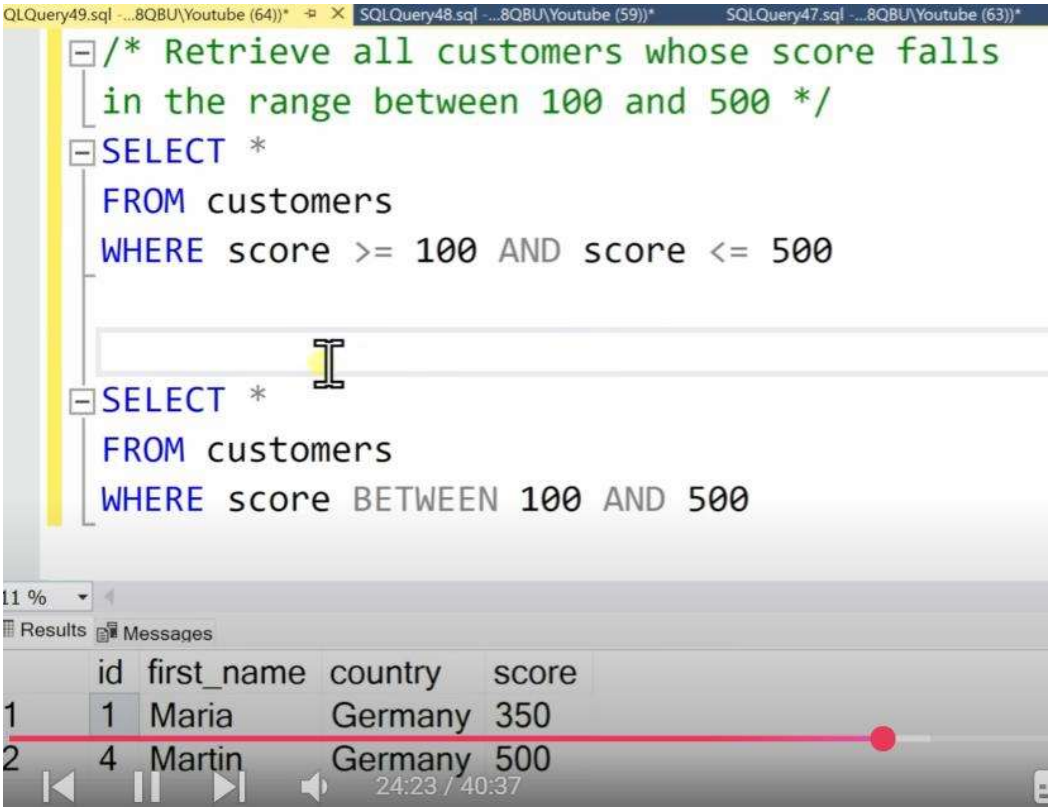

| Name | Country | Score | |
|---|---|---|---|
| Maria | Germany | 350 | ✔ |
| John | USA | 900 | ✘ |
| Georg | UK | 750 | ✘ |
| Martin | Germany | 500 | ✔ |
| Peter | USA | 0 | ✘ |

SQLQuery49.sql -...8QBU\Youtube (64))* SQLQuery48.sql -...8QBU\Youtube (59))* SQLQuery47.sql -...8QBU\Youtube (63))*

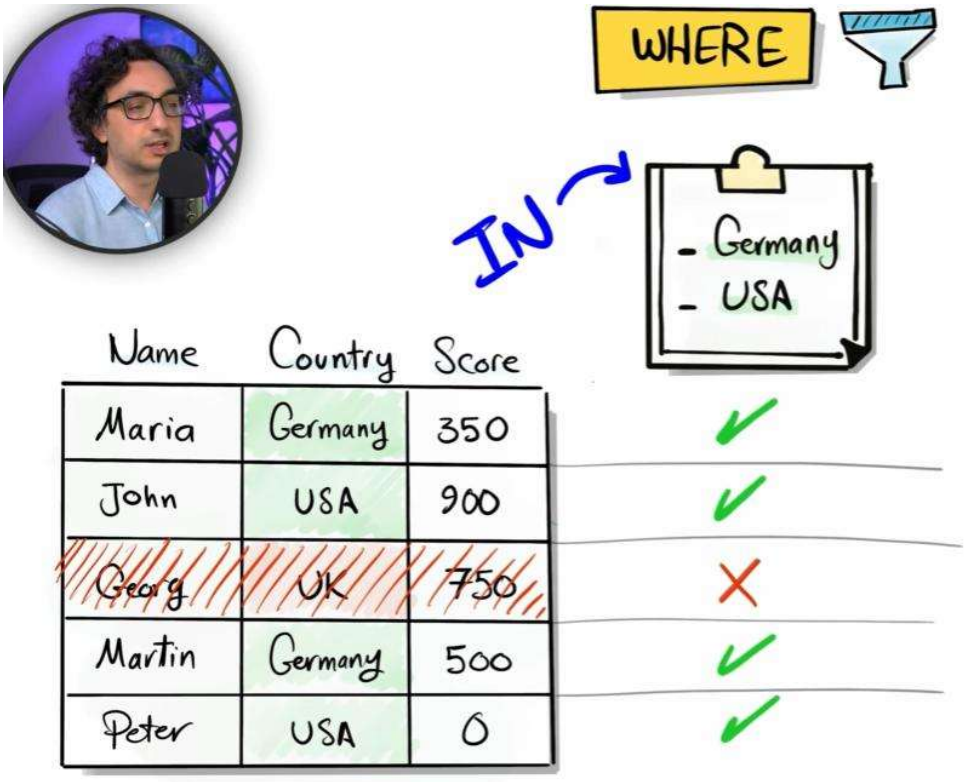
```sql
/* Retrieve all customers whose score falls
in the range between 100 and 500 */

SELECT *
FROM customers
WHERE score BETWEEN 100 AND 500
```

Results Messages

| | id | first_name | country | score |
|---|---|---|---|---|
| 1 | 1 | Maria | Germany | 350 |
| 2 | 4 | Martin | Germany | 500 |




Membership Operators



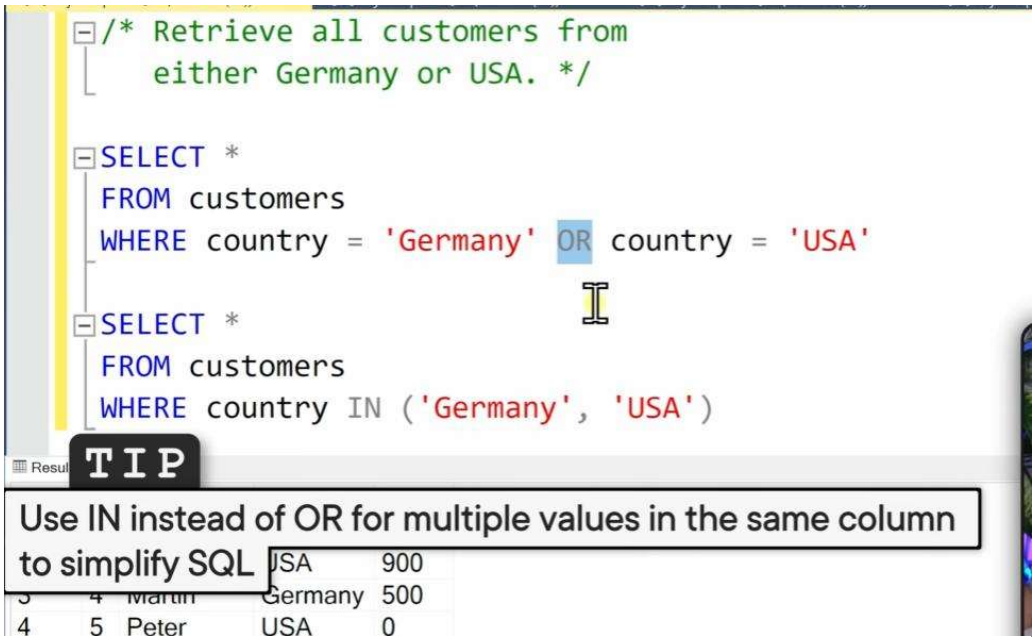

| Name | Country | Score | IN | NOT IN |
|---|---|---|---|---|
| Maria | Germany | 350 | ✓ | ✗ |
| John | USA | 900 | ✓ | ✗ |
| Georg | UK | 750 | ✗ | ✓ |
| Martin | Germany | 500 | ✓ | ✗ |
| Peter | USA | 0 | ✓ | ✗ |

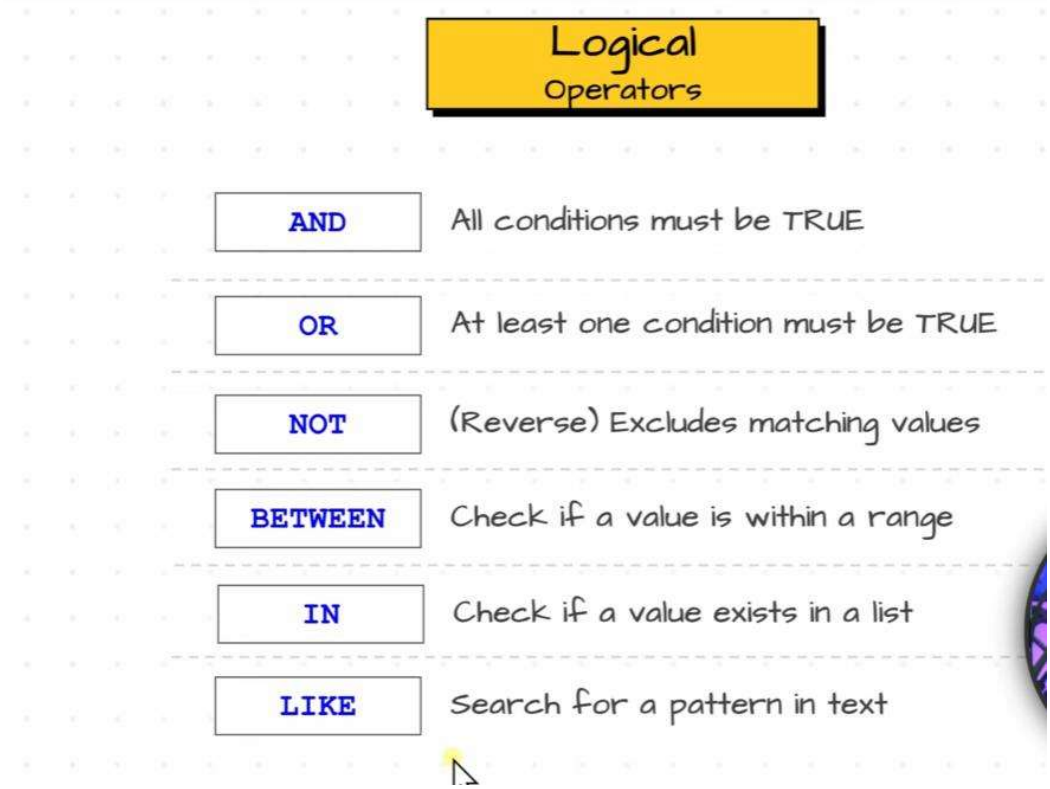
SQLQuery50.sql -...8QBU\Youtube (62))* | SQLQuery49.sql -...8QBU\Youtube (64))* | SQLQuery48.sql -...8QBU\Youtube (59))*

```sql
/* Retrieve all customers from
   either Germany or USA. */

SELECT *
FROM customers
WHERE country = 'Germany' OR country = 'USA'

SELECT *
FROM customers
WHERE country IN ('Germany', 'USA')
```

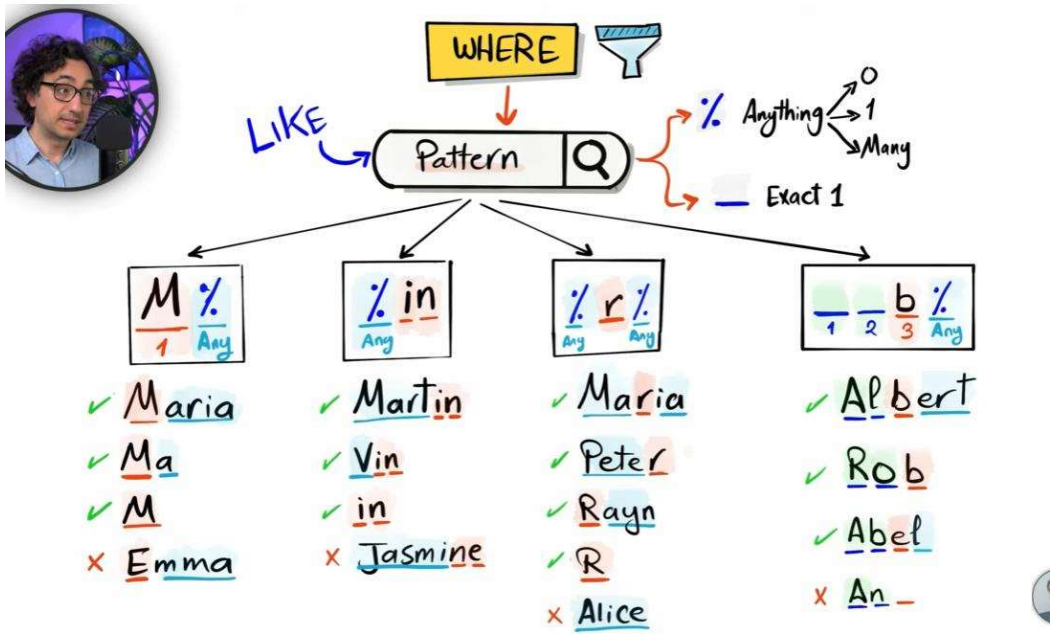
Results | Messages

| | id | first_name | country | score |
|---|---|---|---|---|
| 1 | 1 | Maria | Germany | 350 |
| 2 | 2 | John | USA | 900 |
| 3 | 4 | Martin | Germany | 500 |
| 4 | 5 | Peter | USA | 0 |




Like Operator - Used to search Pattern




```sql
-- Find all customers whose first name starts with 'M'

SELECT *
FROM customers
WHERE first_name LIKE 'M%'
```

Results Messages

| | id | first_name | country | score |
|---|---|---|---|---|
| 1 | 1 | Maria | Germany | 350 |
| 2 | 4 | Mart | Germany | 500 |

```sql
-- Find all customers whose first name ends with 'n'

SELECT *
FROM customers
WHERE first_name LIKE '%n'
```

Results Messages

| | id | first_name | country | score |
|---|---|---|---|---|
| 1 | 2 | John | USA | 900 |
| 2 | 4 | Martin | Germany | 500 |

```sql
-- Find all customers whose first name contains 'r'

SELECT *
FROM customers
WHERE first_name LIKE '%r%'
```

Results | Messages

| | id | first_name | country | score |
|---|---|---|---|---|
| 1 | 1 | Maria | Germany | 350 |
| 2 | 3 | Georg | UK | 750 |
| 3 | 4 | Martin | Germany | 500 |
| 4 | 5 | Peter | USA | 0 |

```sql
/* Find all customers whose first name
   has 'r' in the 3rd position */

SELECT *
FROM customers
WHERE first_name LIKE '__r%'
```

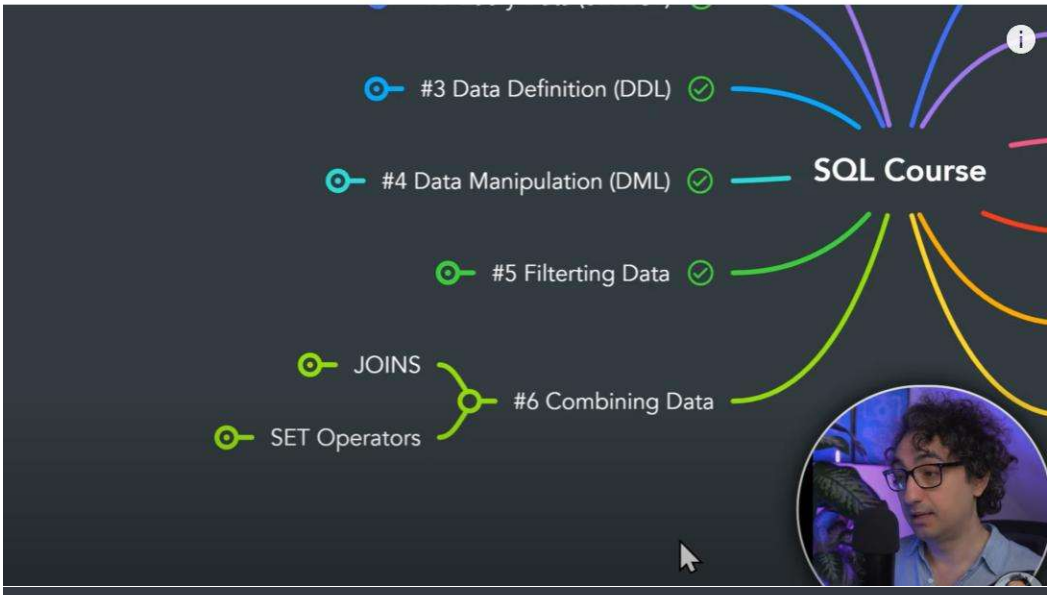
Results | Messages

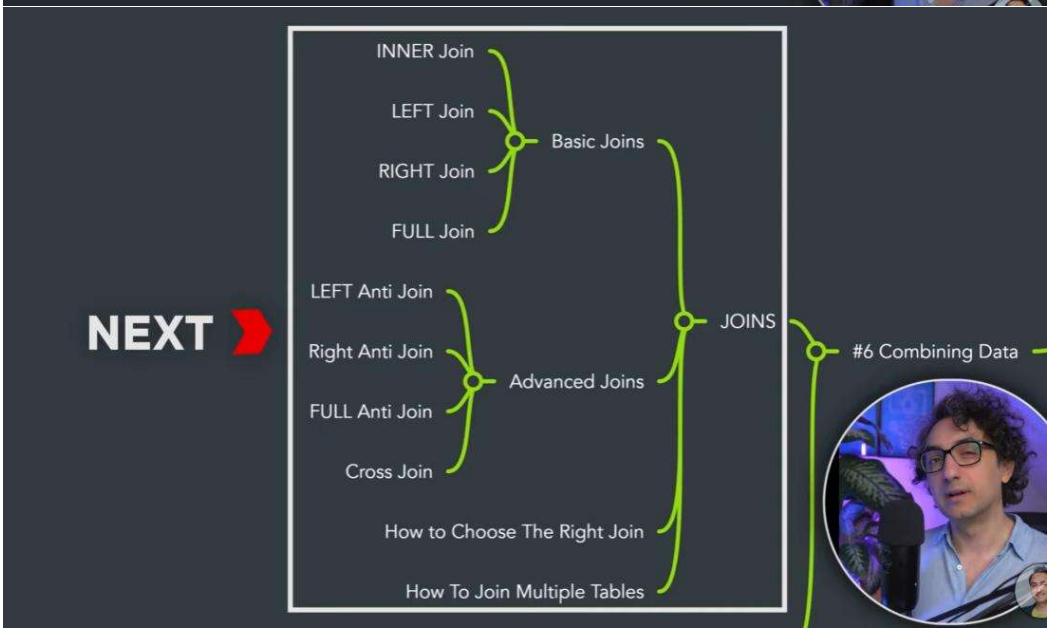
| | id | first_name | country | score |
|---|---|---|---|---|
| 1 | 1 | Maria | Germany | 350 |
| 2 | 4 | Martin | Germany | 500 |

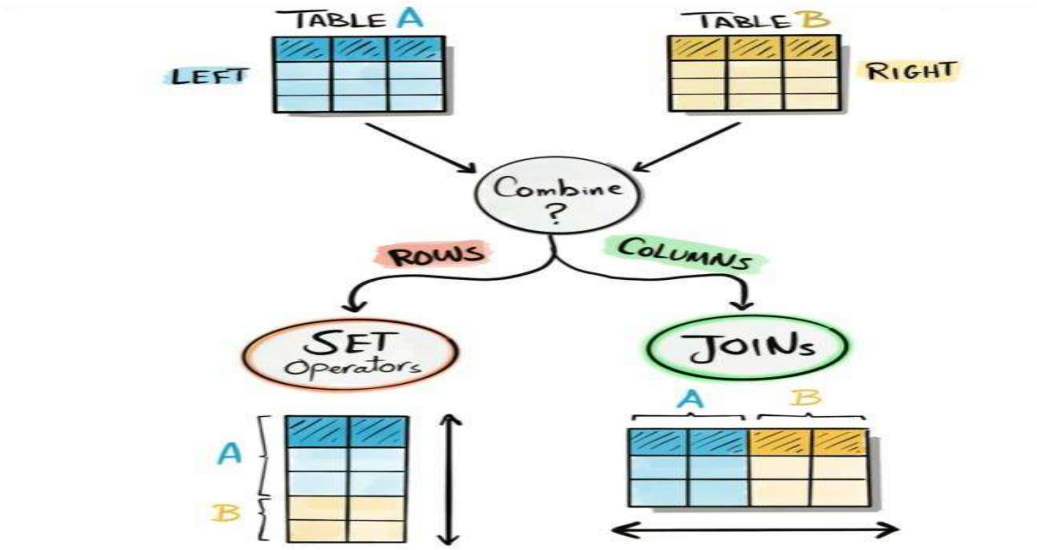
```sql
/* Find all customers whose first name
   has 'r' in the 3rd position */

SELECT *
FROM customers
WHERE first_name LIKE '_r%'
```

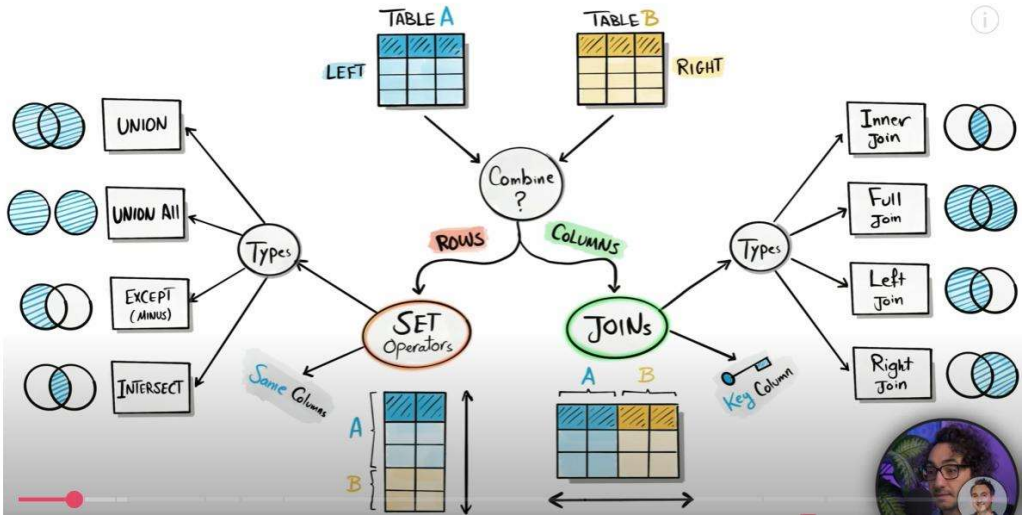
Results Messages

| id | first_name | country | score |
|---|---|---|---|

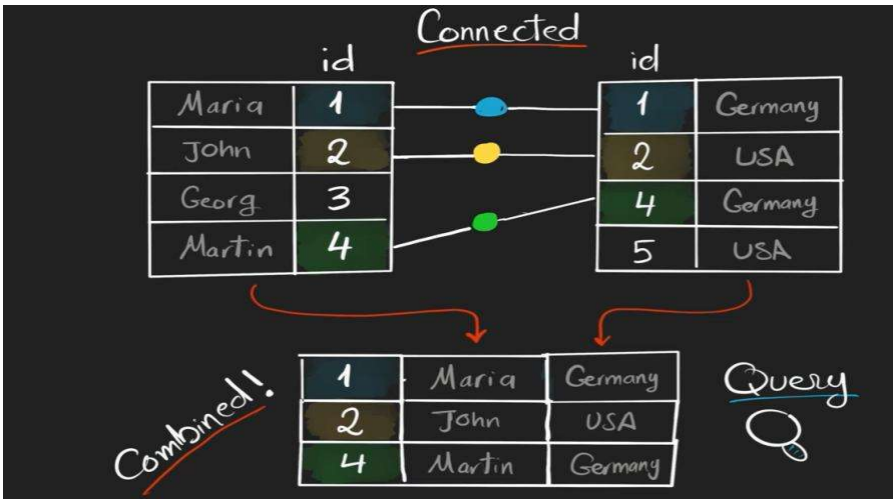

#3 Data Definition (DDL)
#4 Data Manipulation (DML)
#5 Filterting Data
JOINS
SET Operators
#6 Combining Data
SQL Course

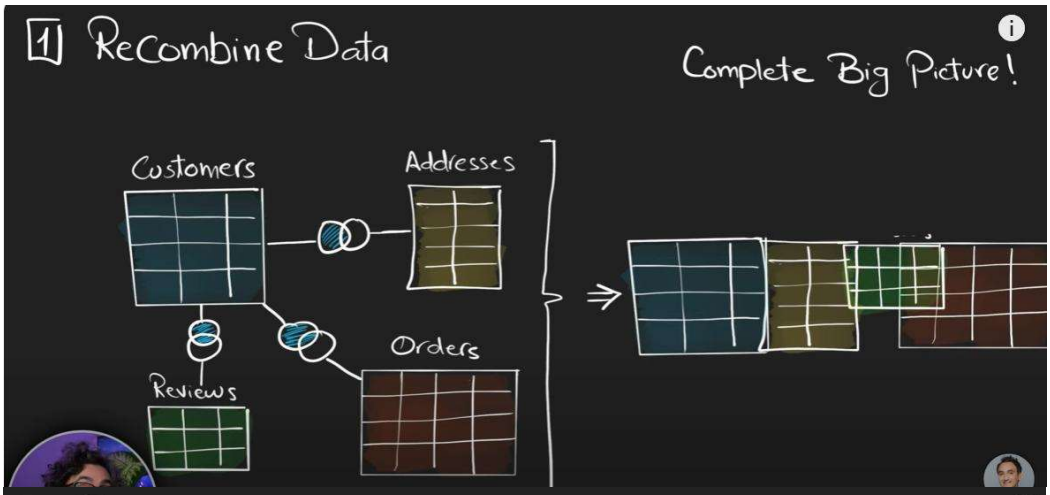

NEXT
INNER Join
LEFT Join
RIGHT Join
FULL Join
Basic Joins
LEFT Anti Join
Right Anti Join
FULL Anti Join
Cross Join
Advanced Joins
How to Choose The Right Join
How To Join Multiple Tables
JOINS
#6 Combining Data

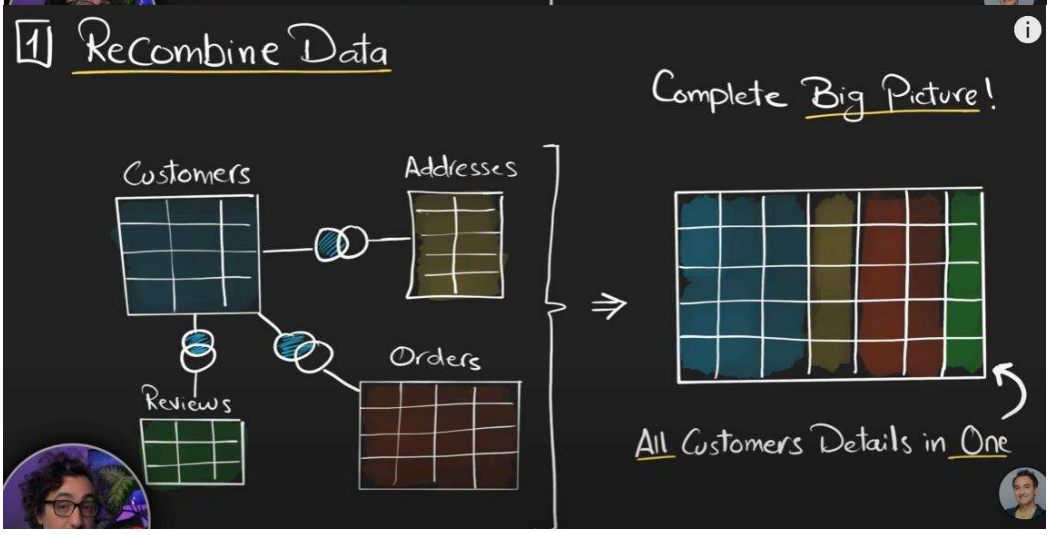


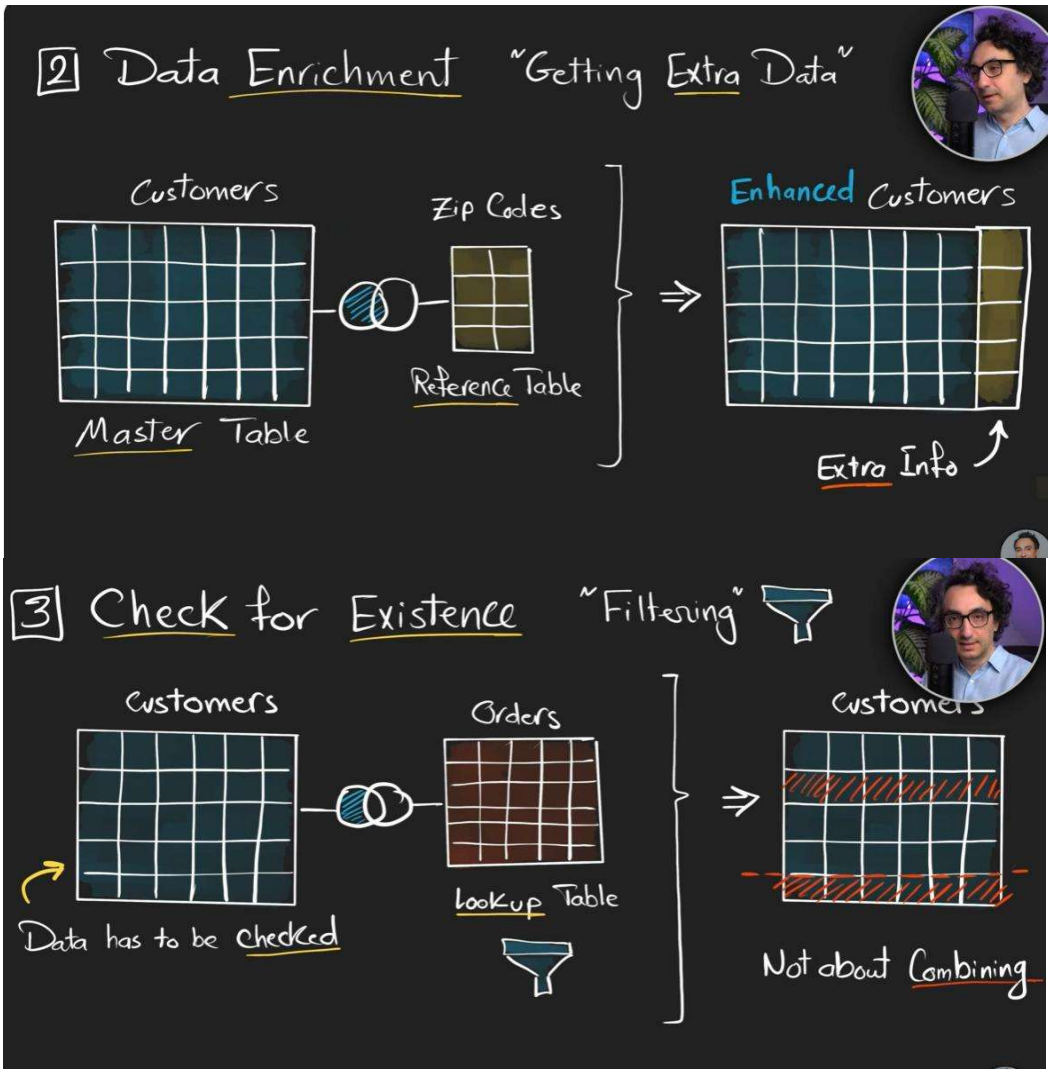




Joins to combine your data

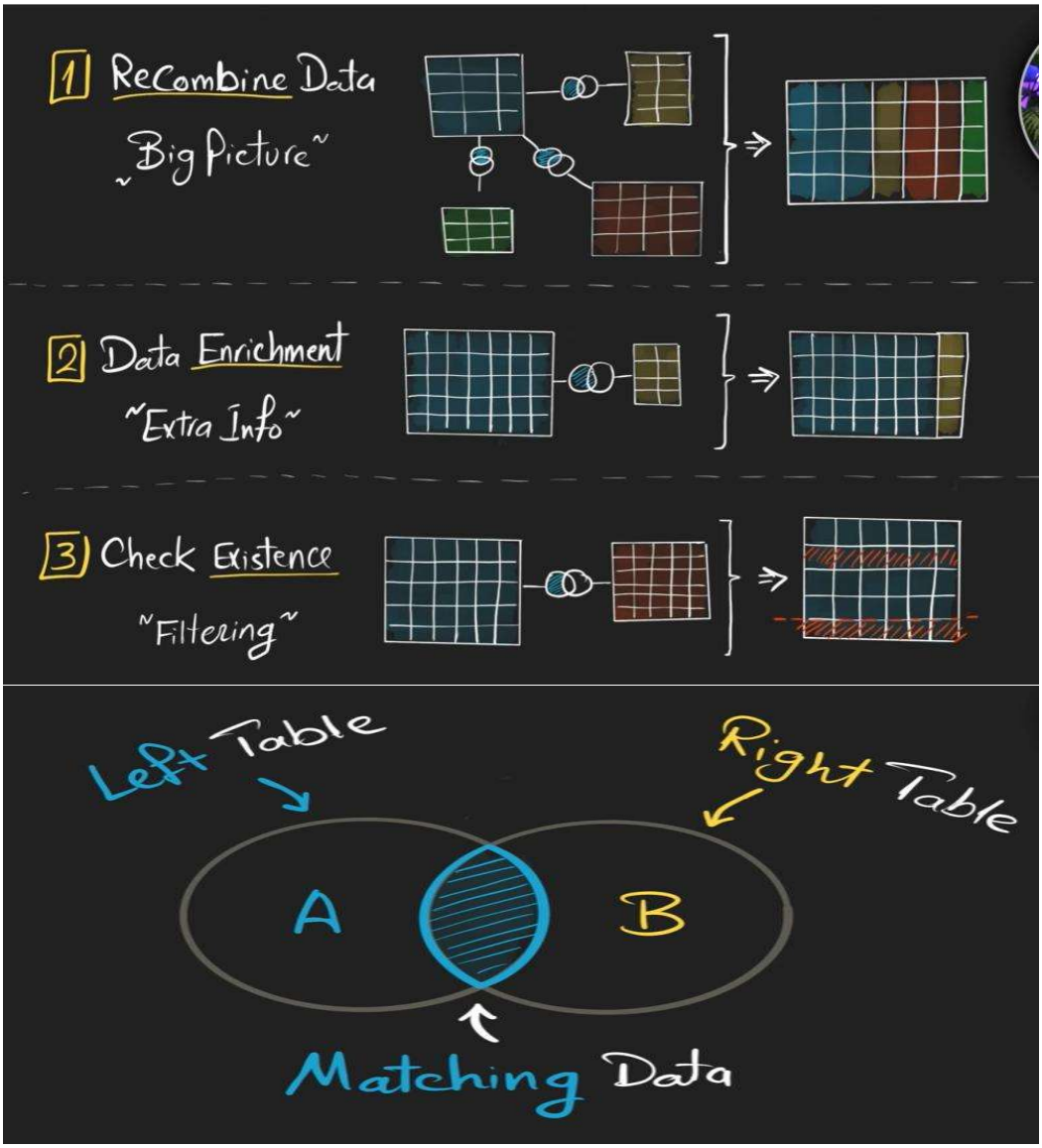

Connected
id
Maria
John
Georg
Martin
1
2
3
4
id
1
Germany
2
USA
4
Germany
5
USA
Combined!
1
Maria
Germany
2
John
USA
4
Martin
Germany
Query

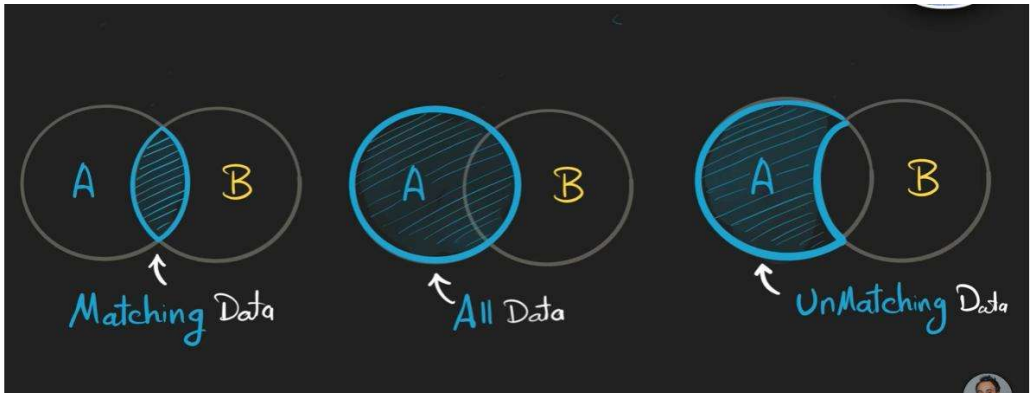

1 Recombine Data
Complete Big Picture!
Customers
Addresses
Orders
Reviews

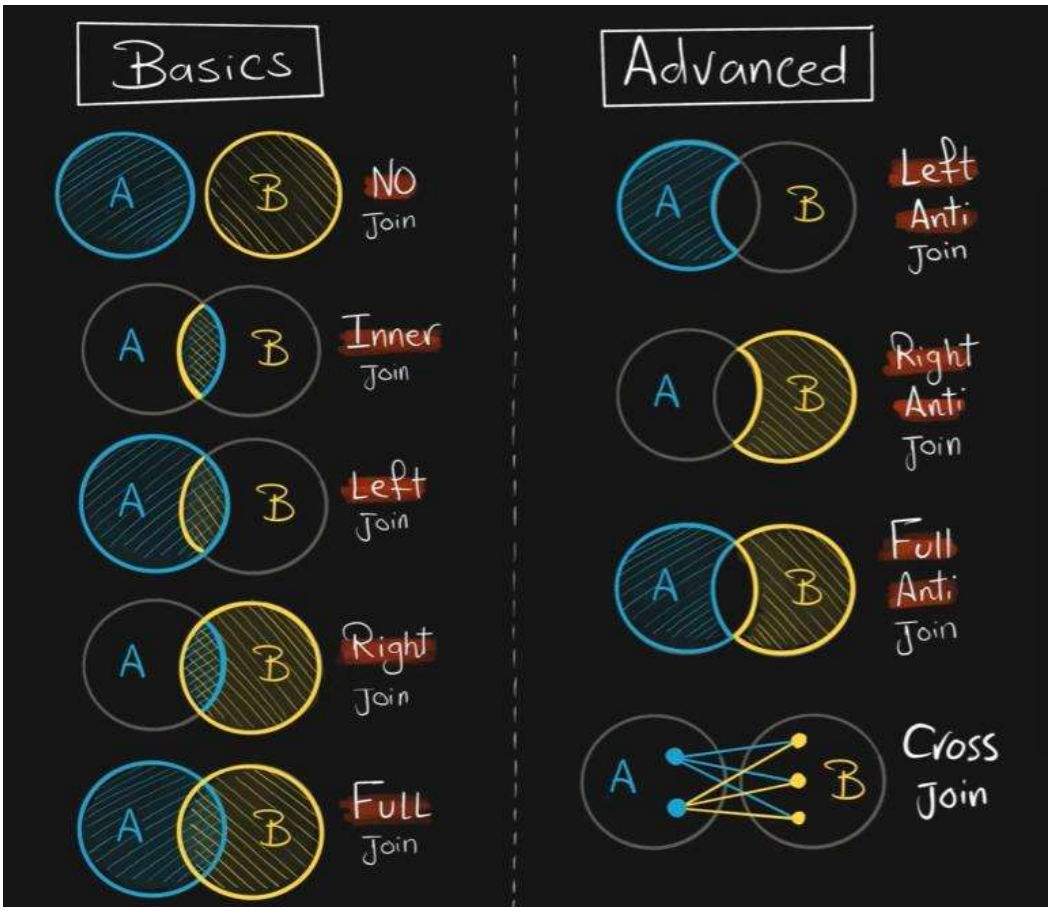

1 Recombine Data
Complete Big Picture!
Customers
Addresses
Orders
Reviews
All Customers Details in One



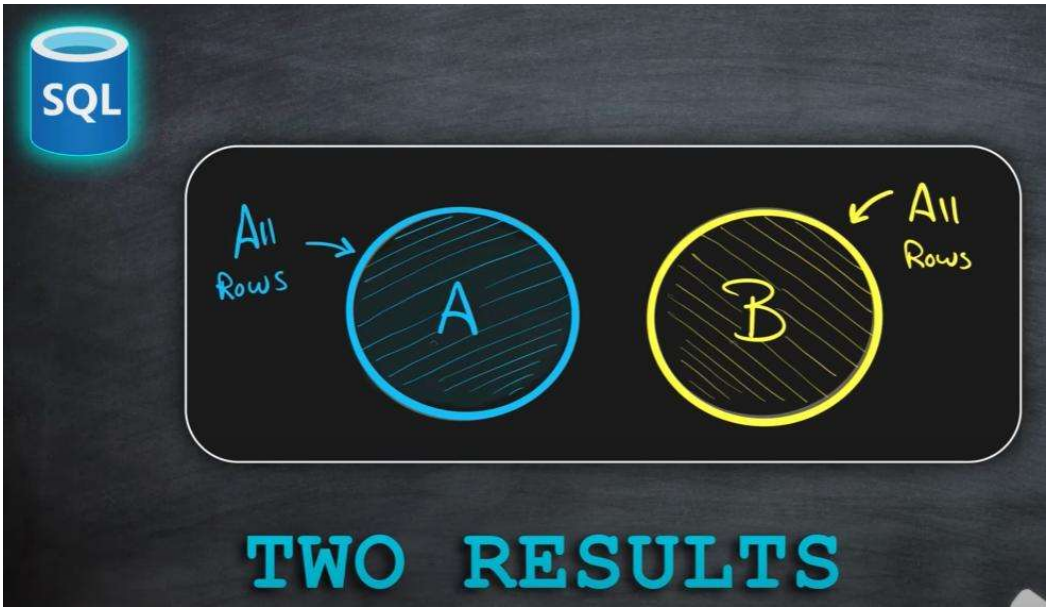

Why we needs joins in the  sql

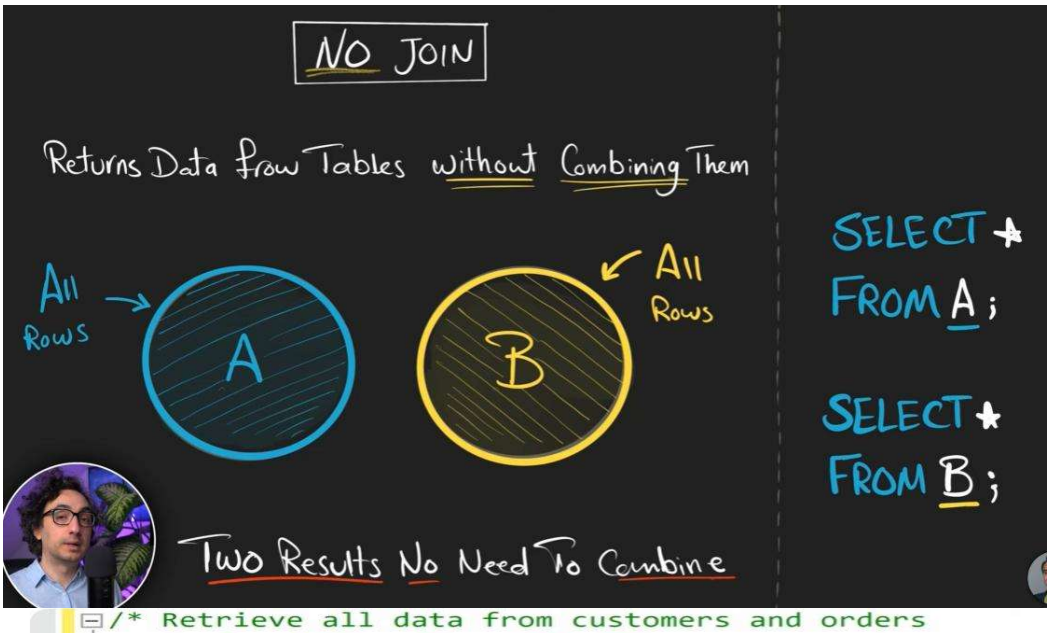

1 Recombine Data
"Big Picture"
2 Data Enrichment
"Extra Info"
3 Check Existence
"Filtering"
Left Table
Right Table
A
B
Matching Data


A
B
Matching Data
A
B
All Data
A
B
UnMatching Data


Basics
Advanced
A
B
NO Join
Inner Join
Left Join
Right Join
FULL Join
Left Anti Join
Right Anti Join
Full Anti Join
Cross Join

No Joins


SQL
All Rows
A
B
All Rows
TWO RESULTS



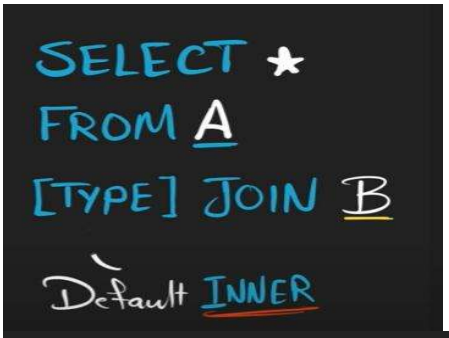

```sql
/* Retrieve all data from customers and orders
   in two different results */

SELECT *
FROM customers;

SELECT *
FROM orders;
```

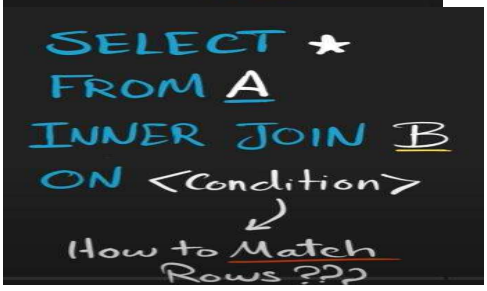
Results Messages

| | id | first_name | country | score |
|---|---|---|---|---|
| 1 | 1 | Maria | Germany | 350 |
| 2 | 2 | John | USA | 900 |
| 3 | 3 | Georg | UK | 750 |
| 4 | 4 | Martin | Germany | 500 |
| 5 | 5 | Peter | USA | 0 |

| | order_id | customer_id | order_date | sales |
|---|---|---|---|---|
| 1 | 1001 | 1 | 2021-01-11 | 35 |
| 2 | 1002 | 2 | 2021-04-05 | 15 |
| 3 | 1003 | 3 | 2021-06-18 | 20 |

10:13 / 40:18 · No Joins >



How to deal with rows between two tables sql know by using condition Inner Join.
ON Keyword  and Join condition tells how to combine the tables.
In order to join two tables we have to find out common column in order to match the data and usually in sql there are the keys and ID's so condition can be like that the key from the table a must be equal to the key from the table b so this is join condition using the join condition sql can start matching the data from the left table and  the right table.

One more important thing while you are joining the table you have to undersatand the order of the table in your query.

But in inner join the order of the join doesn't really matter.
So wheather you starts from A or starts from B it does not matter because you will get the same result both of the table has same priority and it does not matter from which table you start.wheather we say A join B or we can say B join A