SQL Conditions

Introduction:

SQL conditions are used to filter records and return only those that meet specific criteria. They are mainly used with the WHERE clause and help in forming more precise and powerful queries. These conditions act as logical checks to control the result set.

Common SQL Conditions:

SQL Conditions Covered

1. **AND** – All conditions must be true
2. **OR** – Any one condition must be true
3. **AND & OR Combined** – For complex logic using parentheses
4. **NOT** – Negates the condition (e.g., NOT IN, NOT LIKE)
5. **LIKE** – Pattern matching using `%` and `_`
6. **IN** – Checks if a value exists in a list or subquery
7. **NOT IN** – Excludes values in a list
8. **BETWEEN** – Filters values in a range
9. **EXISTS / NOT EXISTS** – Checks presence/absence of rows from a subquery

SQL AND Condition

Introduction

In **SQL**, the AND condition is used in the WHERE clause to filter rows **only when all specified conditions are true**. It helps return **more specific results** by applying multiple filters in a single query.

It is commonly used with the SQL statements:

SELECT,INSERT,UPDATE,DELETE

Syntax

```
WHERE condition1
AND condition2
AND condition3;
```

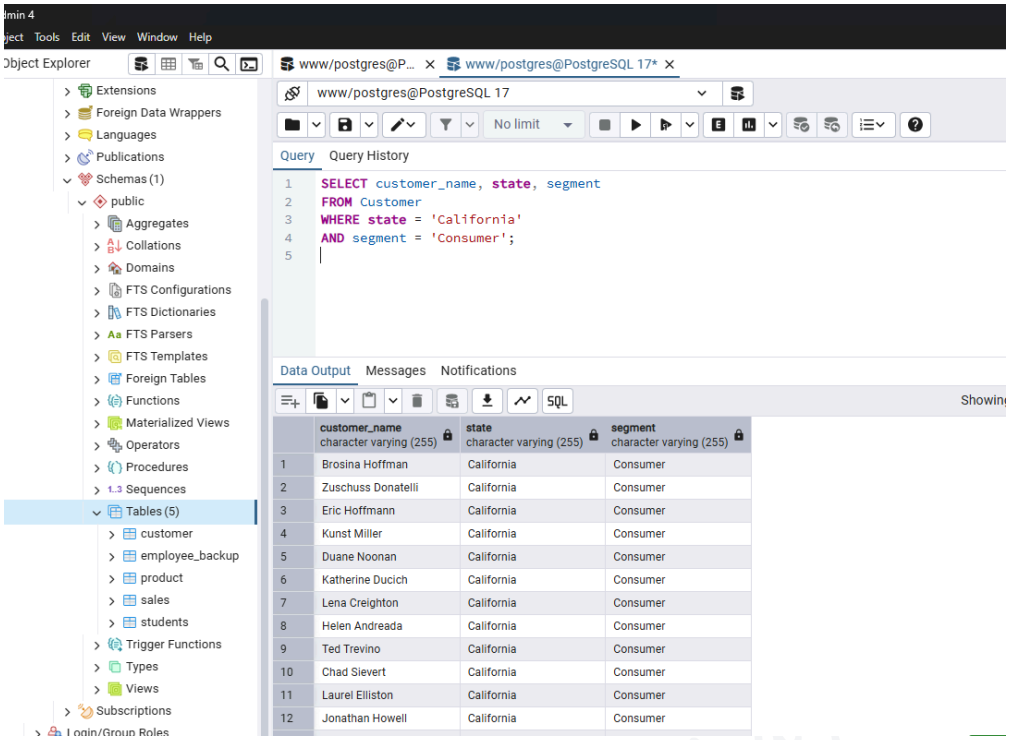

Example Parameters:

- condition1, condition2, ... → Logical expressions used to filter data (e.g., column comparisons).

1. SELECT with AND

Retrieve customer names from **California** who are in the **Consumer** segment.

```
SELECT customer_name, state, segment
FROM Customer
WHERE state = 'California'
AND segment = 'Consumer';
```



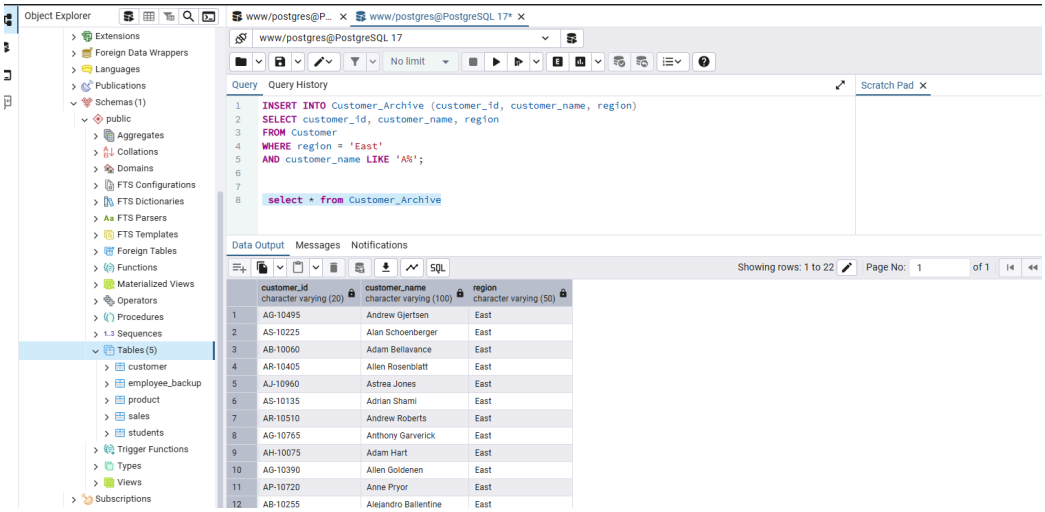
2. INSERT with AND

SQL to Create Customer_Archive Table

```
CREATE TABLE Customer_Archive (
    customer_id VARCHAR(20),
    customer_name VARCHAR(100),
    region VARCHAR(50)
);
```

Insert customers into an archive table where the **region is 'East'** and **customer name starts with 'A'**.

```
INSERT INTO Customer_Archive (customer_id, customer_name, region)
SELECT customer_id, customer_name, region
FROM Customer
WHERE region = 'East'
AND customer_name LIKE 'A%';
```



3. UPDATE with AND

Basic Orders Table Structure:

```
CREATE TABLE Orders (
    order_id VARCHAR(20) PRIMARY KEY,
    customer_id VARCHAR(20),
    order_date DATE,
    region VARCHAR(50),
    sales NUMERIC(10,2),
    discount NUMERIC(4,2),
    priority VARCHAR(20)
);
```

Insert sample records into Orders table

```
INSERT INTO Orders (order_id, customer_id, order_date, region, sales, discount, priority) VALUES
('O1001', 'C001', '2024-05-01', 'West', 1200.00, 0.10, 'Medium'),
('O1002', 'C002', '2024-05-05', 'West', 800.00, 0.15, 'Low'),
('O1003', 'C003', '2024-06-10', 'East', 950.00, 0.05, 'Medium'),
('O1004', 'C004', '2024-06-12', 'South', 2200.00, 0.20, 'High'),
('O1005', 'C005', '2024-06-20', 'West', 1300.00, 0.00, 'Low'),
('O1006', 'C006', '2024-07-01', 'Central', 500.00, 0.10, 'Medium'),
('O1007', 'C007', '2024-07-04', 'West', 300.00, 0.00, 'Low');
```

Example (Before Update):

Data Output Messages Notifications

Showing rows: 1 to 7 Page No:

	order_id [PK] character varying (20)	customer_id character varying (20)	order_date date	region character varying (50)	sales numeric (10,2)	discount numeric (4,2)	priority character varying (20)
1	O1001	C001	2024-05-01	West	1200.00	0.10	Medium
2	O1002	C002	2024-05-05	West	800.00	0.15	Low
3	O1003	C003	2024-06-10	East	950.00	0.05	Medium
4	O1004	C004	2024-06-12	South	2200.00	0.20	High
5	O1005	C005	2024-06-20	West	1300.00	0.00	Low
6	O1006	C006	2024-07-01	Central	500.00	0.10	Medium
7	O1007	C007	2024-07-04	West	300.00	0.00	Low

Update the Orders table to mark high-priority orders in the **West** region with **sales above 1000**.

```
UPDATE Orders
SET priority = 'High'
WHERE region = 'West'
AND sales > 1000;
```

Data Output Messages Notifications

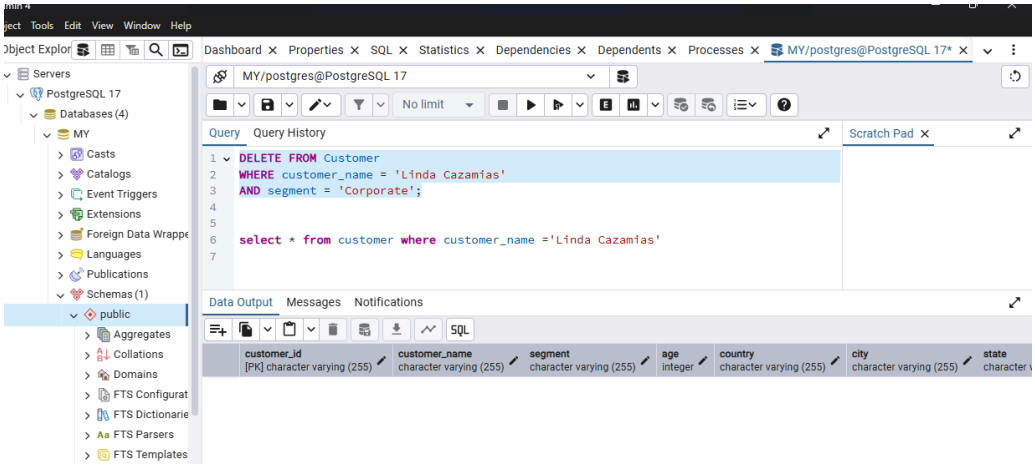
Showing rows: 1 to 7 Page No:

	order_id [PK] character varying (20)	customer_id character varying (20)	order_date date	region character varying (50)	sales numeric (10,2)	discount numeric (4,2)	priority character varying (20)
1	O1002	C002	2024-05-05	West	800.00	0.15	Low
2	O1003	C003	2024-06-10	East	950.00	0.05	Medium
3	O1004	C004	2024-06-12	South	2200.00	0.20	High
4	O1006	C006	2024-07-01	Central	500.00	0.10	Medium
5	O1007	C007	2024-07-04	West	300.00	0.00	Low
6	O1001	C001	2024-05-01	West	1200.00	0.10	High
7	O1005	C005	2024-06-20	West	1300.00	0.00	High

4. DELETE with AND

Delete customers from the database whose **name is 'John Doe'** and who belong to the **Corporate** segment.

```
DELETE FROM Customer
WHERE customer_name = 'Linda Cazamias'
AND segment = 'Corporate';
```



Summary

Use Case	Description
SELECT	Filters query results with multiple true conditions
INSERT	Inserts data based on multiple filters from another table
UPDATE	Updates records only when all conditions match
DELETE	Deletes records only when all conditions match

Practice Questions: SQL AND Condition

1. **SELECT**:Retrieve customer names from **New York** who belong to the **Home Office** segment.

2. **SELECT**:Find customers whose **state is 'Texas'** and **segment is 'Corporate'**.

3. **INSERT**: Insert records into a Customer_Backup table for customers from the **South** region whose names start with **'S'**.

4. **INSERT**:Create an Orders_Archive table and insert records where sales > 1000 and priority = 'High'.

5. **UPDATE**:Update the Orders table to set discount = 0.05 for orders in the **Central** region where the priority = 'Medium'.

SQL OR Condition

Introduction

The SQL OR condition is used in the WHERE clause to filter records when **any one of multiple conditions is true**. It's useful for retrieving rows that meet **at least one** of several criteria.

It can be used with:

SELECT,INSERT,UPDATE,DELETE

Syntax

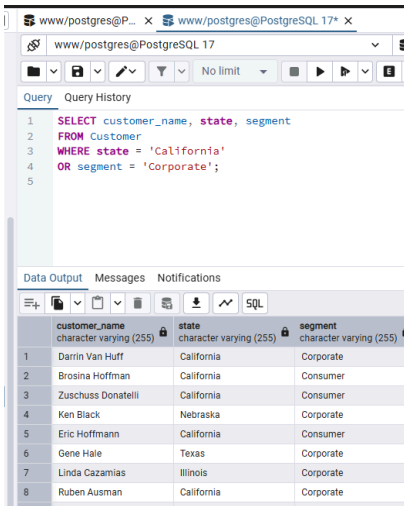
WHERE condition1
OR condition2
OR condition3;

- If **any one** of the conditions is true, the row is included in the result.

1. SELECT with OR

Get customers from California OR who belong to the Corporate segment:

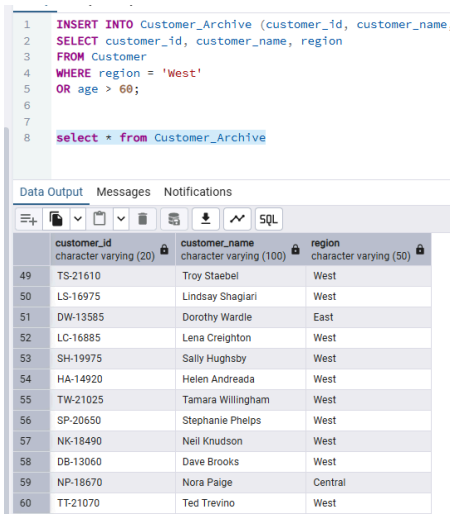
```
SELECT customer_name, state, segment
FROM Customer
WHERE state = 'California'
OR segment = 'Corporate';
```



2. INSERT with OR

Insert customers into archive table if they are from the 'West' region or aged above 60:

```
INSERT INTO Customer_Archive (customer_id, customer_name, region)
SELECT customer_id, customer_name, region
FROM Customer
WHERE region = 'West'
OR age > 60;
```



```
INSERT INTO Customer_Archive (customer_id, customer_name
SELECT customer_id, customer_name, region
FROM Customer
WHERE region = 'West'
OR age > 60;


select * from Customer_Archive
```

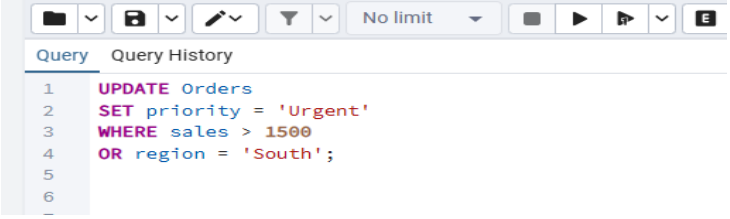
Data Output Messages Notifications

	customer_id character varying (20)	customer_name character varying (100)	region character varying (50)
49	TS-21610	Troy Staebel	West
50	LS-16975	Lindsay Shagiari	West
51	DW-13585	Dorothy Wardle	East
52	LC-16885	Lena Creighton	West
53	SH-19975	Sally Hughsby	West
54	HA-14920	Helen Andreada	West
55	TW-21025	Tamara Willingham	West
56	SP-20650	Stephanie Phelps	West
57	NK-18490	Neil Knudson	West
58	DB-13060	Dave Brooks	West
59	NP-18670	Nora Paige	Central
60	TT-21070	Ted Trevino	West

3. UPDATE with OR

Update order priority to 'Urgent' for orders with sales > 1500 or from the 'South' region:

```
UPDATE Orders
SET priority = 'Urgent'
WHERE sales > 1500
OR region = 'South';
```



No limit

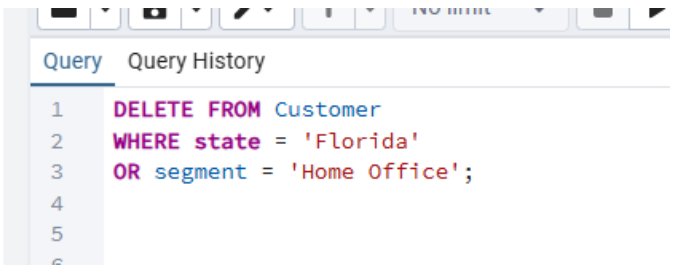
Query Query History

```
UPDATE Orders
SET priority = 'Urgent'
WHERE sales > 1500
OR region = 'South';
```

4. DELETE with OR

Delete customers from 'Florida' or whose segment is 'Home Office':

```
DELETE FROM Customer
WHERE state = 'Florida'
OR segment = 'Home Office';
```



Summary

- The OR condition allows broader searches.
- It is inclusive — **one or more** conditions being true is enough.
- Useful for applying **multi-condition filters** without requiring all to be true.
- Works great when paired with AND, LIKE, or subqueries for powerful filtering.

Practice Questions: SQL OR Condition

1. **SELECT**:Get all customers who are from **'Texas'** or belong to the **'Consumer'** segment.
2. **SELECT**: Retrieve orders where the **region is 'Central'** or **sales are less than 500**.
3. **INSERT**: Insert records into Customer_Archive for customers who are from **'East'** region or whose names start with **'M'**.
4. **INSERT**:Create a High_Value_Customers table and insert customers whose **age > 55** or who are from **'South'** region.
5. **UPDATE**:Set priority = 'High' for orders where **sales > 1200** or **region = 'North'**.

SQL AND & OR Condition

Introduction

The SQL AND & OR condition is used to combine multiple logical expressions in a WHERE clause, enabling complex filtering in a single query.

- AND ensures **all** specified conditions are true.
- OR requires **any one** of the conditions to be true.
- You can **combine both** to build advanced filtering logic.
- Use **parentheses ()** to control the **order of evaluation**, as AND has higher precedence than OR.

Syntax

```
WHERE (condition1 AND condition2)
  OR (condition3 AND condition4)
  ...
```

- Conditions inside parentheses are grouped and evaluated first.
- Use this format in SELECT, INSERT, UPDATE, and DELETE statements.

1. SELECT with AND & OR

Get customers from California and in the 'Consumer' segment, or anyone aged over 60:

```
SELECT customer_id, customer_name, state, segment, age
FROM Customer
WHERE (state = 'California' AND segment = 'Consumer')
  OR age > 60;
```

```
SELECT customer_id, customer_name, state, segment, age
FROM Customer
WHERE (state = 'California' AND segment = 'Consumer')
   OR age > 60;
```

Output Messages Notifications

Showing rows: 1 to 196 Page No:

customer_id [PK] character varying (255)	customer_name character varying (255)	state character varying (255)	segment character varying (255)	age integer
CG-12520	Claire Gute	Kentucky	Consumer	67
BH-11710	Brosina Hoffman	California	Consumer	20
IM-15070	Irene Maddox	Washington	Consumer	66
ZD-21925	Zuschuss Donatelli	California	Consumer	66
KB-16585	Ken Black	Nebraska	Corporate	67
EH-13945	Eric Hoffmann	California	Consumer	21
PO-18865	Patrick O'Donnell	Michigan	Consumer	64
LH-16900	Lena Hernandez	Delaware	Consumer	66
KM-16720	Kunst Miller	California	Consumer	69

2. INSERT with AND & OR

Insert customers into archive who are from the West and named 'John', or anyone from the East:

```
INSERT INTO Customer_Archive (customer_id, customer_name, region)
SELECT customer_id, customer_name, region
FROM Customer
WHERE (region = 'West' AND customer_name = 'John')
   OR region = 'East';
```

No limit

Query Query History

```
INSERT INTO Customer_Archive (customer_id, customer_name, region)
SELECT customer_id, customer_name, region
FROM Customer
WHERE (region = 'West' AND customer_name = 'John')
   OR region = 'East';
```

Data Output Messages Notifications

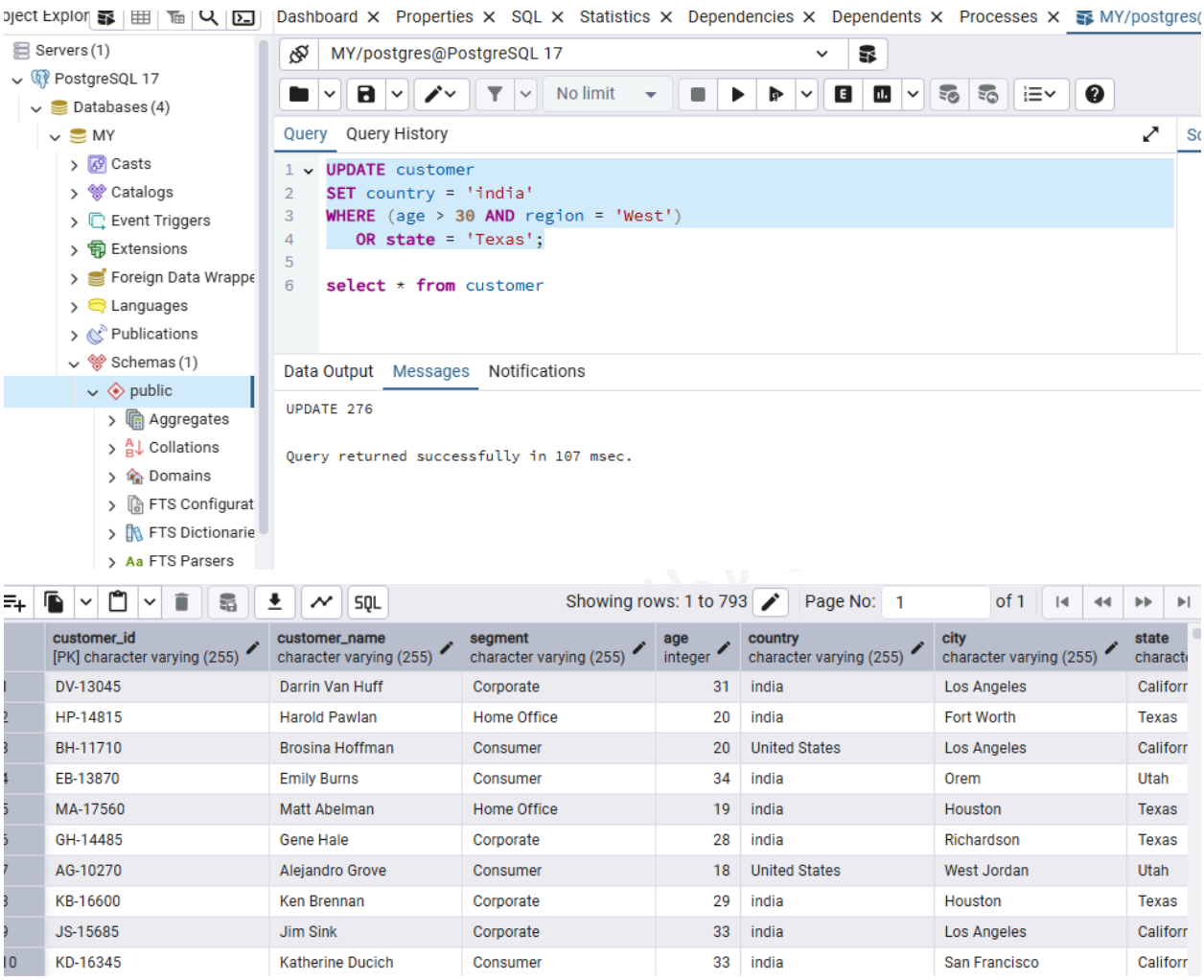
```
INSERT 0 177

Query returned successfully in 123 msec.
```

3. UPDATE with AND & OR

Mark country as 'india' where age> 30 and region is 'West', or state is 'Texas':

```
UPDATE customer
```

```
SET country = 'india'
WHERE (age > 30 AND region = 'West')
   OR state = 'Texas';
```

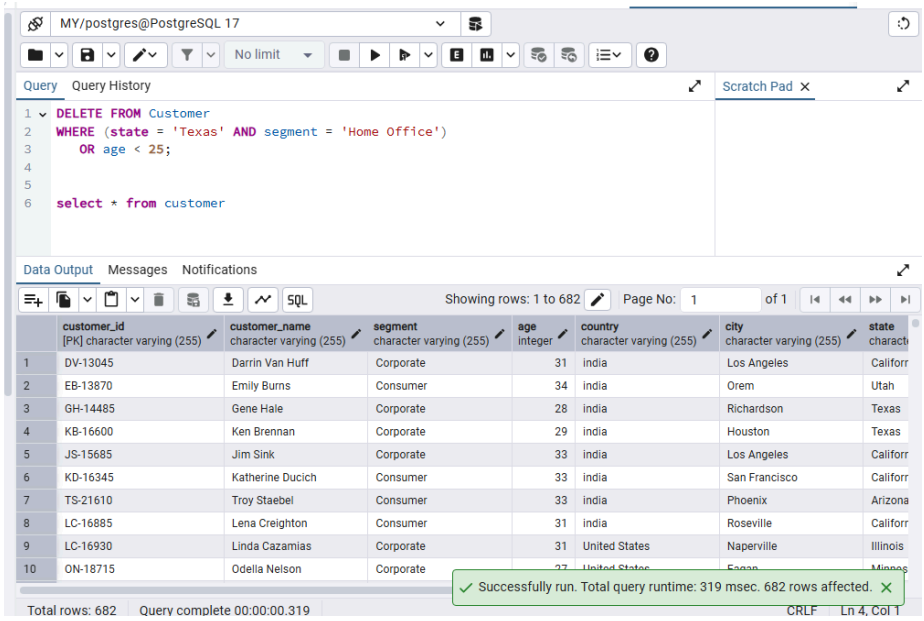


	customer_id [PK] character varying (255)	customer_name character varying (255)	segment character varying (255)	age integer	country character varying (255)	city character varying (255)	state charact
1	DV-13045	Darrin Van Huff	Corporate	31	india	Los Angeles	Californ
2	HP-14815	Harold Pawlan	Home Office	20	india	Fort Worth	Texas
3	BH-11710	Brosina Hoffman	Consumer	20	United States	Los Angeles	Californ
4	EB-13870	Emily Burns	Consumer	34	india	Orem	Utah
5	MA-17560	Matt Abelman	Home Office	19	india	Houston	Texas
6	GH-14485	Gene Hale	Corporate	28	india	Richardson	Texas
7	AG-10270	Alejandro Grove	Consumer	18	United States	West Jordan	Utah
8	KB-16600	Ken Brennan	Corporate	29	india	Houston	Texas
9	JS-15685	Jim Sink	Corporate	33	india	Los Angeles	Californ
10	KD-16345	Katherine Ducich	Consumer	33	india	San Francisco	Californ

4. DELETE with AND & OR

Delete customers from Texas and in 'Home Office' segment, or anyone below age 25:

```
DELETE FROM Customer
WHERE (state = 'Texas' AND segment = 'Home Office')
   OR age < 25;
```



Summary

Keyword Combination	Use Case Example
AND only	All conditions must be true
OR only	At least one condition must be true
AND + OR	Use parentheses to group and control logic

Practice Questions: AND / OR Condition

1. Select all customers from 'California' in the 'Corporate' segment, or customers older than 50.

2. Find customers whose region is 'East' and age is less than 40, or whose state is 'Florida'.

3. Insert customers into customer_archive where age < 25 and region is 'South', or state is 'Nevada'.

4. Update customers and set category = 'Young West' where age < 30 and region = 'West', or state is 'Arizona'.

5. Update customers to category = 'Senior East' where age > 60 and region = 'East', or state is 'New York'.

SQL NOT Condition

Introduction

The **SQL NOT condition** is used with the WHERE clause to **negate a condition** in SQL statements.
It returns rows **only when the condition is false**.

Syntax

```
SELECT column1, column2
FROM table_name
WHERE NOT condition;
```

The NOT condition can be used in:

SELECT,INSERT,UPDATE,DELETE

Parameter

Parameter	Description
condition	Any valid condition that needs to be negated

Examples of NOT with Other Conditions

1. NOT with IN

Description: Returns rows where a column value is **not** in a given list.

```
SELECT customer_id, customer_name, age
FROM customer
WHERE age NOT IN (28)
ORDER BY customer_id;
```

Query Query History

```
SELECT customer_id, customer_name, age
FROM customer
WHERE age NOT IN (28)
ORDER BY customer_id;


select * from customer
```

Data Output Messages Notifications

Showing rows: 1 to 672 Page No: 1

	customer_id [PK] character varying (255)	customer_name character varying (255)	age integer
1	AA-10315	Alex Avila	66
2	AA-10480	Andrew Allen	50
3	AA-10645	Anna Andreadi	32
4	AB-10015	Aaron Bergman	66
5	AB-10060	Adam Bellavance	25
6	AB-10105	Adrian Barton	63
7	AB-10150	Aimee Bixby	65
8	AB-10255	Alejandro Ballentine	34
9	AB-10600	Ann Blume	34
10	AC-10420	Alyssa Crouse	69
11	AC-10450	Amy Cox	46

2. NOT with LIKE

Description: Filters rows where a column **does not match** a pattern.

```
SELECT customer_id, customer_name, age
FROM customer
WHERE customer_name NOT LIKE 'Smi%'
ORDER BY customer_id;
```

Query Query History

```
SELECT customer_id, customer_name, age
FROM customer
WHERE customer_name NOT LIKE 'Smi%'
ORDER BY customer_id;


select * from customer
```

Data Output Messages Notifications

Showing rows: 1 to 682 Page No: 1

	customer_id [PK] character varying (255)	customer_name character varying (255)	age integer
1	AA-10315	Alex Avila	66
2	AA-10480	Andrew Allen	50
3	AA-10645	Anna Andreadi	32
4	AB-10015	Aaron Bergman	66
5	AB-10060	Adam Bellavance	25
6	AB-10105	Adrian Barton	63
7	AB-10150	Aimee Bixby	65
8	AB-10255	Alejandro Ballentine	34
9	AB-10600	Ann Blume	34
10	AC-10420	Alyssa Crouse	69

3. NOT with BETWEEN

Description: Returns rows where a value is **not between** two values.

SELECT order_id, customer_id, sales, discount
FROM sales
WHERE sales NOT BETWEEN 261 AND 731
ORDER BY customer_id DESC;

```
SELECT order_id, customer_id, sales, discount
FROM sales
WHERE sales NOT BETWEEN 261 AND 731
ORDER BY customer_id DESC;



select * from sales
```

Data Output Messages Notifications

Showing rows: 1 to 1000 Page No: 1

	order_id character varying (255)	customer_id character varying (255)	sales double precision	discount double precision
1	CA-2016-152471	ZD-21925	15.984	0.2
2	CA-2016-167682	ZD-21925	71.12	0
3	CA-2016-152471	ZD-21925	823.96	0.2
4	CA-2017-141481	ZD-21925	61.44	0
5	CA-2014-143336	ZD-21925	8.56	0
6	CA-2014-143336	ZD-21925	213.48	0.2
7	CA-2014-143336	ZD-21925	22.72	0.2
8	CA-2016-167682	ZD-21925	259.96	0
9	US-2016-147991	ZD-21925	16.72	0.2
10	CA-2016-130946	ZC-21910	1088.792	0.8
11	CA-2015-130365	ZC-21910	128.058	0.3

Equivalent Alternative:

SELECT customer_id, sales, product_id
FROM sales
WHERE sales < 731
  OR sales > 230
ORDER BY customer_id DESC;

Query Query History

```
SELECT customer_id, sales, product_id
FROM sales
WHERE sales < 731
   OR sales > 230
ORDER BY customer_id DESC;


select * from sales
```

Data Output Messages Notifications

Showing rows: 1 to 1000 Page No: 1

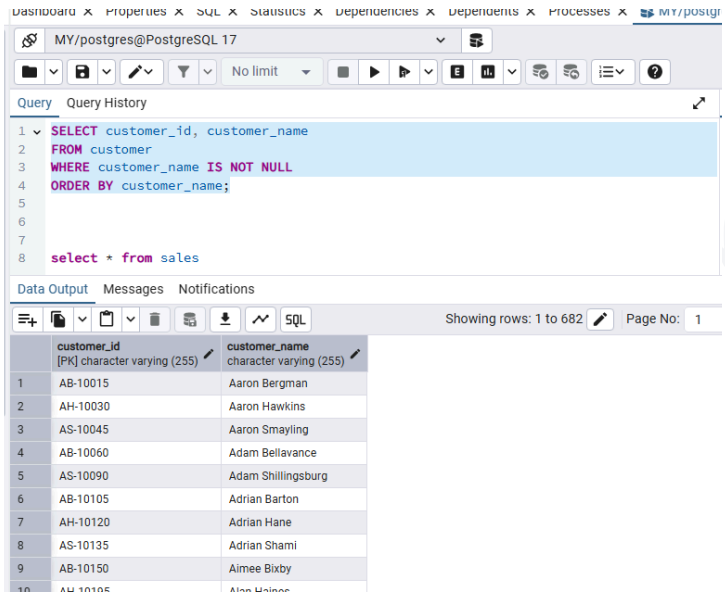
	customer_id character varying (255)	sales double precision	product_id character varying (255)
1	ZD-21925	16.72	FUR-FU-10004270
2	ZD-21925	823.96	TEC-PH-10002824
3	ZD-21925	61.44	OFF-AP-10004532
4	ZD-21925	8.56	OFF-AR-10003056
5	ZD-21925	213.48	TEC-PH-10001949
6	ZD-21925	22.72	OFF-BI-10002215
7	ZD-21925	15.984	OFF-PA-10004965
8	ZD-21925	259.96	TEC-PH-10000673
9	ZD-21925	71.12	FUR-FU-10003799
10	ZC-21910	1.08	OFF-BI-10002012
11	ZC-21910	4.512	OFF-FA-10003021

Successfully run. Total query runtime: 258 m

4. NOT with IS NULL

Description: Finds rows where a column **is not null**.

```
SELECT customer_id, customer_name
FROM customer
WHERE customer_name IS NOT NULL
ORDER BY customer_name;
```

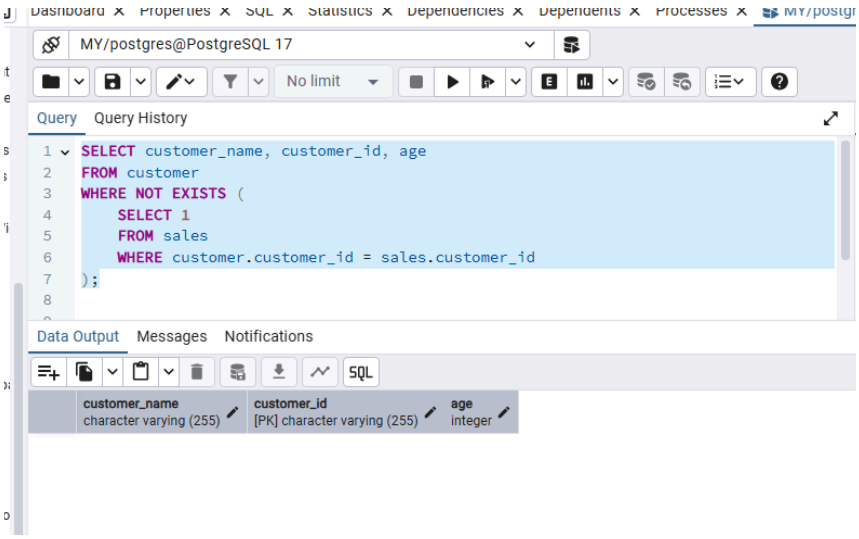


5. NOT with EXISTS

Description: Finds rows where a **subquery does not return** any rows.

Query:

```
SELECT customer_name, customer_id, age
FROM customer
WHERE NOT EXISTS (
    SELECT 1
    FROM sales
    WHERE customer.customer_id = sales.customer_id  );
```



Summary

Use Case	Example Operator Used
NOT with list of values	NOT IN
NOT with text pattern	NOT LIKE
NOT with value range	NOT BETWEEN
NOT with null check	IS NOT NULL
NOT with subquery check	NOT EXISTS

Practice Questions: SQL NOT Condition

1. **Find customers** whose age is **not 30**.
2. **List customer names** that do **not start with "Jo"**.
3. **Get all orders** where the sales amount is **not between 100 and 500**.
4. **Show customers** whose names are **not null**.
5. **Find all customers** who **do not** have a matching record in the orders table

SQL LIKE Condition*(PostgreSQL)*

What is LIKE in SQL?

The LIKE condition in SQL is used to **search for a specific pattern** in a column (typically a string column).

Syntax:

expression LIKE pattern [ESCAPE 'escape_character']

expression NOT LIKE pattern [ESCAPE 'escape_character']

- **expression** – The column to be searched.
- **pattern** – The string pattern to match.
- **escape_character** (optional) – Used to treat % or _ as literal characters.

What is the _ Wildcard?

- The **underscore (_)** in PostgreSQL's LIKE condition is used to match **exactly one character** at a specific position.
- It is often used with the LIKE operator to filter results using **pattern matching**.

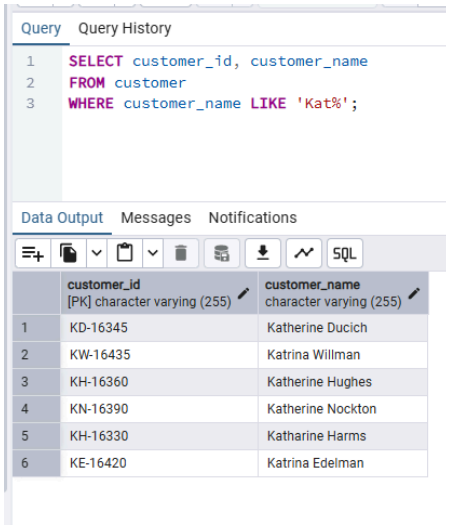
Wildcards in LIKE

Wildcard	Description
%	Matches **zero or more characters**
_	Matches **exactly one character**

Examples with % Wildcard

Example 1: Name starts with Kat

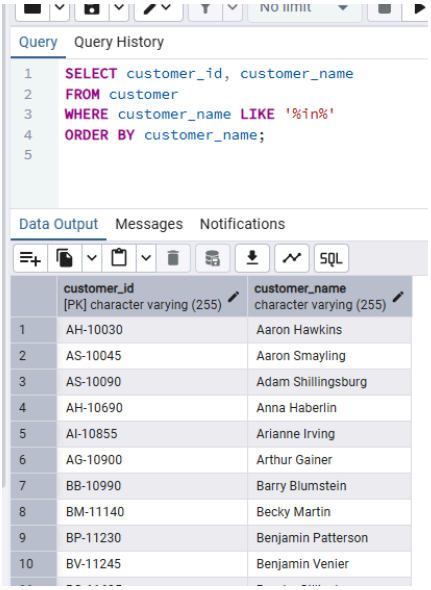
```
SELECT customer_id, customer_name
FROM customer
WHERE customer_name LIKE 'Kat%';
```



	customer_id [PK] character varying (255)	customer_name character varying (255)
1	KD-16345	Katherine Ducich
2	KW-16435	Katrina Willman
3	KH-16360	Katherine Hughes
4	KN-16390	Katherine Nockton
5	KH-16330	Katharine Harms
6	KE-16420	Katrina Edelman

Example 2: Name contains in

```
SELECT customer_id, customer_name
FROM customer
WHERE customer_name LIKE '%in%'
ORDER BY customer_name;
```



Examples with _ Wildcard

Example 3: Last name like _tah

SELECT customer_name, city, state

FROM customer

WHERE state LIKE '_tah';

Explanation of the Pattern _tah:

- _ → matches **exactly one character**
- tah → must appear **after** that character

So, _tah matches:

- A **4-letter word** where:
    - First character can be **any letter**
    - Next characters must be **t**, **a**, **h**

Query Query History

```
SELECT customer_name, city, state
FROM customer
WHERE state LIKE '_tah';


 select * from customer
```

Data Output Messages Notifications

	customer_name character varying (255)	city character varying (255)	state character varying (255)
1	Alejandro Grove	West Jordan	Utah
2	Emily Burns	Orem	Utah
3	Karen Seio	Lehi	Utah

Example 4: First name like _at%

SELECT customer_id, customer_name

FROM customer

WHERE customer_name LIKE '_at%'

ORDER BY customer_name;

Query Query History

```
SELECT customer_id, customer_name
FROM customer
WHERE customer_name LIKE '_at%'
ORDER BY customer_name;


 select * from customer
```

Data Output Messages Notifications

	customer_id [PK] character varying (255)	customer_name character varying (255)
1	CP-12085	Cathy Prescott
2	KH-16330	Katharine Harms
3	KD-16345	Katherine Ducich
4	KH-16360	Katherine Hughes
5	KN-16390	Katherine Nockton
6	KE-16420	Katrina Edelman
7	KW-16435	Katrina Willman
8	MC-17575	Matt Collins

Notes:

- % matches any sequence of characters (including none).

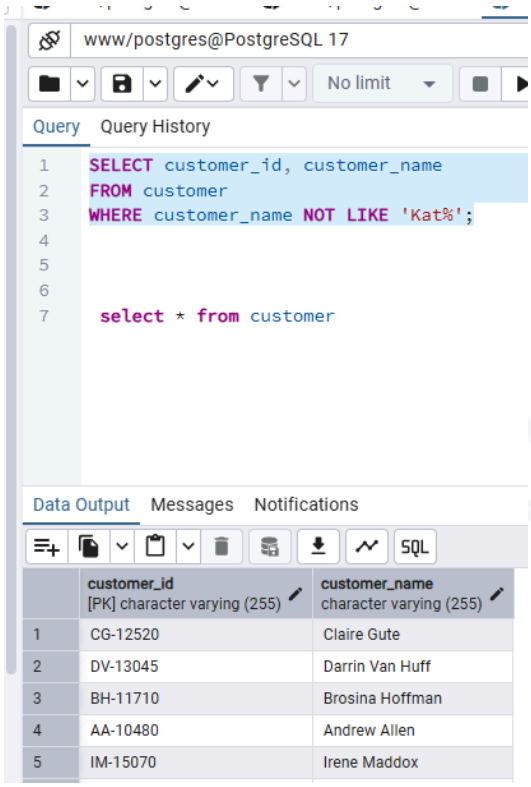
- _ matches any **one** character.
- Pattern 'abc' behaves like = 'abc' if no wildcard is used.

NOT LIKE Example

SELECT customer_id, customer_name

FROM customer

WHERE customer_name NOT LIKE 'Kat%';

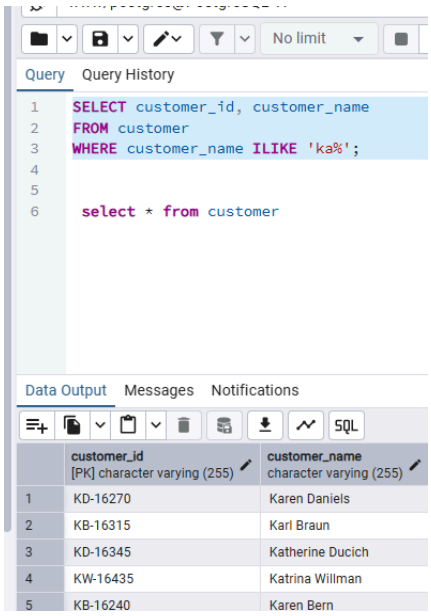


ILIKE Operator (Case-Insensitive LIKE)

SELECT customer_id, customer_name

FROM customer

WHERE customer_name ILIKE 'ka%';



LIKE Operator Equivalents in PostgreSQL

Operator	Meaning
~~	LIKE
!~~	NOT LIKE
~~*	ILIKE
!~~*	NOT ILIKE

Summary

- Used for **pattern matching** with % and _.
- LIKE is **case-sensitive**.
- Use ILIKE for **case-insensitive** search.
- Use NOT LIKE / !~~ to exclude patterns.
- Combine % and _ for complex matches.

Practice Questions: SQL LIKE Condition

1.Write a query to find all customers whose names start with **"Jo"**.

2.Write a query to list customers whose names contain the letters **"an"** anywhere.

3.Find customers where the **state** has exactly 4 letters and ends with **"ex"**.

4.Display customers whose **names do not start with "Ka"**.

5.Retrieve customers whose names start with **"al"**, case-insensitively.

IN Condition

Introduction

The IN condition in PostgreSQL is used in the WHERE clause to check if a value matches **any value in a list**. It is commonly used to **replace multiple OR conditions** for cleaner, shorter queries.

Syntax

```
-- With value list
expression IN (value1, value2, ..., valueN);

-- With subquery
expression IN (SELECT column_name FROM table_name);
```

Parameters:

Parameter	Description
expression	A column or field being compared
value1, value2...	List of values to match against
subquery	A SELECT statement that returns values to match

Note:
If any of the values in the list match the expression, the condition returns TRUE.

Example 1: IN Condition with Character Values

Fetch all customers whose state is either **'Kentucky'** or **'Utah'**.

```
SELECT *
FROM customer
WHERE state IN ('Kentucky', 'Utah')
ORDER BY customer_name DESC;
```

Query Query History Scratch Pad

```sql
SELECT *
FROM customer
WHERE state IN ('Kentucky', 'Utah')
ORDER BY customer_name DESC;
```

Data Output Messages Notifications

Showing rows: 1 to 15 Page No: 1 of

	varying (255)	customer_name character varying (255)	segment character varying (255)	age integer	country character varying (255)	city character varying (255)	state character varying (255)	postal_code bigint
1		Rob Dowd	Consumer	60	United States	Henderson	Kentucky	42420
2		Pauline Webber	Corporate	47	United States	Richmond	Kentucky	40475
3		Natalie DeCherney	Consumer	41	United States	Louisville	Kentucky	40214
4		Liz Preis	Consumer	58	United States	Murray	Kentucky	42071
5		Katharine Harms	Corporate	66	United States	Bowling Green	Kentucky	42104
6		Karen Seio	Corporate	53	United States	Lehi	Utah	84043
7		Emily Burns	Consumer	34	United States	Orem	Utah	84057
8		Dave Hallsten	Corporate	58	United States	Richmond	Kentucky	40475
9		Darren Koutras	Consumer	58	United States	Henderson	Kentucky	42420
10		Claire Gute	Consumer	67	United States	Henderson	Kentucky	42420
11		Bruce Degenhardt	Consumer	36	United States	Richmond	Kentucky	40475
12		Brian Derr	Consumer	42	United States	Bowling Green	Kentucky	42104
13		Bobby Odegard	Consumer	63	United States	Richmond	Kentucky	40475
14		Art Foster	Consumer	40	United States	Louisville	Kentucky	40214
15		Alejandro Grove	Consumer	18	United States	West Jordan	Utah	84084

Equivalent using OR:

```sql
SELECT *
FROM customer
WHERE state = 'Kentucky' OR state = 'Utah'
ORDER BY customer_name DESC;
```


Why IN is better:
More readable and shorter than using multiple ORs.

Example 2: IN Condition with Numeric Values

Get customer info for customers aged 50, 25, or 30.

```sql
SELECT customer_id,customer_name, age, state
FROM customer
WHERE age IN (50, 25, 30)
ORDER BY customer_name DESC;
```

Query Query History

```sql
SELECT customer_id,customer_name, age, state
FROM customer
WHERE age IN (50, 25, 30)
ORDER BY customer_name DESC;
```

Data Output Messages Notifications

	customer_id [PK] character varying (255)	customer_name character varying (255)	age integer	state character varying (255)
1	TB-21595	Troy Blackwell	50	Indiana
2	TP-21565	Tracy Poddar	25	Colorado
3	SC-20305	Sean Christensen	30	Michigan
4	SJ-20215	Sarah Jordon	50	Texas
5	RP-19855	Roy Phan	50	Wisconsin
6	RO-19780	Rose O'Brian	25	Tennessee
7	RE-19450	Richard Eichhorn	30	Texas
8	PF-19165	Philip Fox	30	Illinois
9	PB-19150	Philip Brown	30	California
10	PC-18745	Pamela Coakley	25	California
11	NG-18430	Nathan Gelder	50	Ohio
12	MY-18295	Muhammed Yedwab	25	Texas
13	MW-18235	Mitch Willingham	30	Virginia
14	MC-17845	Michael Chen	50	Mississippi
15	ME-17725	Max Engle	50	Colorado
16	MC-17575	Matt Collins	25	Ohio
17	MO-17500	Mary O'Rourke	25	New York
18	MC-17425	Mark Cousins	50	Alabama

Total rows: 41 Query complete 00:00:00.119

Example 3: IN Condition with Subquery

Get all customer details whose state has an **average customer age above 40**

Step 1: Subquery to find such states

```sql
SELECT state
FROM customer
GROUP BY state
HAVING AVG(age) > 40;
```

Query Query History

```
SELECT state
FROM customer
GROUP BY state
HAVING AVG(age) > 40;
```

Data Output Messages Notifications

	state character varying (255)
1	Oklahoma
2	Colorado
3	North Carolina
4	Delaware
5	Louisiana
6	New York
7	Missouri
8	South Dakota
9	Iowa
10	Massachusetts
11	Rhode Island

Step 2: Use the subquery in the IN condition

```
SELECT customer_id, customer_name, age, state
FROM customer
WHERE state IN (
    SELECT state
    FROM customer
    GROUP BY state
    HAVING AVG(age) > 40
)
ORDER BY customer_name;
```


Query Query History

```
SELECT customer_id, customer_name, age, state
FROM customer
WHERE state IN (
    SELECT state
    FROM customer
    GROUP BY state
    HAVING AVG(age) > 40
)
ORDER BY customer_name;
```

Data Output Messages Notifications

	customer_id [PK] character varying (255)	customer_name character varying (255)	age integer	state character varying (255)
1	AB-10015	Aaron Bergman	66	Washington
2	AH-10030	Aaron Hawkins	60	Pennsylvania
3	AS-10045	Aaron Smayling	42	North Carolina
4	AH-10075	Adam Hart	21	New York
5	AS-10090	Adam Shillingsburg	46	Virginia
6	AB-10105	Adrian Barton	63	Arizona
7	AB-10150	Aimee Bixby	65	New York
8	AB-10165	Alan Barnes	22	California
9	AH-10210	Alan Hwang	58	California

Summary

The IN condition simplifies filtering by checking if a value exists in a list of values or a subquery result.

Used in the WHERE clause to **replace multiple OR conditions** for better readability and cleaner SQL code.

Supports:

- **Character values** – e.g., customers from 'Kentucky', 'Utah'
- **Numeric values** – e.g., customers aged 25, 30, 50
- **Subqueries** – e.g., customers from states with **average age > 40**

Works with SQL statements: SELECT, UPDATE, DELETE, and more.

Improves performance and logic clarity when dealing with multiple values or dynamic result sets.

Practice Questions: SQL IN Condition

1. Fetch all customers from the states 'California', 'Texas', and 'Florida'.
2. Get the customer IDs and names of customers aged either 25, 35, or 45.
3. Find all customers who belong to the segment 'Consumer' or 'Corporate'.
4. List customers who live in cities where the average customer age is less than 30.
5. Display customer details whose postal codes are among the top 3 highest in the table.

SQL NOT IN Condition

Introduction

The NOT IN condition in PostgreSQL is used in the WHERE clause to return records that do not match any value in a specified list.
It is commonly used to exclude specific values from query results and is a cleaner alternative to multiple <> (not equal) comparisons.

Syntax

```
-- With value list

expression NOT IN (value1, value2, ..., valueN);
```

The NOT IN condition can be used with SELECT, INSERT, UPDATE, and DELETE statements.

Examples

Example 1: NOT IN with Numeric Values

Get customer info where age is not 25 or 30:

```
SELECT customer_id, customer_name, age, state

FROM customer

WHERE age NOT IN (25, 30)

ORDER BY customer_name DESC;
```

Query Query History Scratch P

```
SELECT customer_id, customer_name, age, state
FROM customer
WHERE age NOT IN (25, 30)
ORDER BY customer_name DESC;
```

Data Output Messages Notifications

Showing rows: 1 to 600 Page No: 1 of 1

	customer_id [PK] character varying (255)	customer_name character varying (255)	age integer	state character varying (255)
1	ZD-21925	Zuschuss Donatelli	66	California
2	ZC-21910	Zuschuss Carroll	61	Oregon
3	YC-21895	Yoseph Carroll	42	California
4	YS-21880	Yana Sorensen	31	North Carolina
5	XP-21865	Xylona Preis	32	California
6	WB-21850	William Brown	38	Pennsylvania
7	VM-21835	Vivian Mathis	60	Delaware
8	VS-21820	Vivek Sundaresam	22	California
9	VG-21805	Vivek Grady	54	Virginia
10	VG-21790	Vivek Gonzalez	44	California
11	VW-21775	Victoria Wilson	62	Ohio
12	VP-21760	Victoria Pisteka	35	California

Equivalent using AND and <>:

SELECT customer_id, customer_name, age, state

FROM customer

WHERE age <> 25 AND age <> 30

ORDER BY customer_name DESC;

This query returns customers whose age is neither 25 nor 30.

Example 2: NOT IN with Character Values

Get customer details where state is not 'California' or 'Texas':

SELECT customer_id, customer_name, state

FROM customer

WHERE state NOT IN ('California', 'Texas')

ORDER BY customer_name;

Query Query History

```sql
SELECT customer_id, customer_name, state
FROM customer
WHERE state NOT IN ('California', 'Texas')
ORDER BY customer_name;
```

Data Output Messages Notifications

Showing rows: 1 to 423 Page No: 1 of 1

	customer_id [PK] character varying (255)	customer_name character varying (255)	state character varying (255)
1	AB-10015	Aaron Bergman	Washington
2	AH-10030	Aaron Hawkins	Pennsylvania
3	AS-10045	Aaron Smayling	North Carolina
4	AH-10075	Adam Hart	New York
5	AS-10090	Adam Shillingsburg	Virginia
6	AB-10105	Adrian Barton	Arizona
7	AB-10150	Aimee Bixby	New York
8	AS-10225	Alan Schoenberger	New York
9	AG-10270	Alejandro Grove	Utah
10	AA-10315	Alex Avila	Minnesota
11	AR-10345	Alex Russell	Massachusetts
12	AA-10375	Alle	

Successfully run. Total query runtime: 134

Summary

- NOT IN is used to exclude specific values from the result set.
- It can be used with both character and numeric columns.
- It simplifies queries compared to using multiple <> and AND operators.
- Suitable for filtering unwanted values cleanly and efficiently.

Practice Questions: SQL NOT IN

1. Retrieve customer records for customers who are not from the states 'California', 'Texas', or 'Florida'.
2. List customer names and ages for those who are not aged 25, 30, or 35.
3. Get details of customers who are not in the 'Consumer' or 'Home Office' segments.
4. Find all customers whose city is not 'New York' or 'San Francisco'.
5. Select customers whose postal codes are not 10001, 94105, or 77001.

SQL BETWEEN Condition

Purpose

The BETWEEN condition is used in PostgreSQL to filter records within a specified **range of values**. It can be used with:

- Numbers
- Dates
- Text (lexical range)
- Combined with NOT to exclude ranges

Syntax

```
expression BETWEEN value1 AND value2;

-- OR

expression BETWEEN low AND high;
```

Equivalent syntax using comparison operators:

```
expression >= value1 AND expression <= value2;
```

To exclude a range:

```
expression NOT BETWEEN value1 AND value2;

-- Equivalent to:

expression < value1 OR expression > value2;
```

Use with SQL Commands

The BETWEEN condition can be used with:

SELECT,INSERT,UPDATE,DELETE

Examples

1. Between with Numeric Values

Get customers aged **between 23 and 26**:

SELECT customer_id, customer_name, age

FROM customer

WHERE age BETWEEN 23 AND 26

ORDER BY age DESC

Query Query History

```
SELECT customer_id, customer_name, age
FROM customer
WHERE age BETWEEN 23 AND 26
ORDER BY age DESC
```

Data Output Messages Notifications

Showing rows: 1 to 36 Page No: 1 of 1

	customer_id [PK] character varying (255)	customer_name character varying (255)	age integer
1	DM-12955	Dario Medina	26
2	JD-15790	John Dryer	26
3	DL-12865	Dan Lawera	26
4	ST-20530	Shui Tom	26
5	BP-11290	Beth Paige	26
6	JF-15190	Jamie Frazer	26
7	FH-14275	Frank Hawley	26
8	JG-15805	John Grady	26
9	RD-19900	Ruben Dartt	26
10	TP-21565	Tracy Poddar	25
11	JE-15745	Joel	

Successfully run. Total qu

Total rows: 36 Query complete 00:00:00.115

2. Equivalent using >= and <=

SELECT customer_id, customer_name, age

FROM customer

WHERE age >= 23 AND age <= 26

ORDER BY age DESC;

3. Between with Date Values

Get orders placed between 2014-06-01 and 2016-06-30:

SELECT order_id,order_date,ship_date,customer_id,product_id,sales, quantity

FROM Sales WHERE order_date BETWEEN '2014-06-01' AND '2016-06-30'

ORDER BY order_date DESC;

Query | Query History | Scratch Pad

```
SELECT
    order_id,order_date,ship_date,customer_id,product_id,sales, quantity
FROM
    sales
WHERE
    order_date BETWEEN '2014-06-01' AND '2016-06-30'
ORDER BY
    order_date DESC;
```

Data Output | Messages | Notifications

Showing rows: 1 to 1000 | Page No: 1 of 5

	order_id character varying (255)	order_date date	ship_date date	customer_id character varying (255)	product_id character varying (255)	sales double precision	quantity integer
1	CA-2016-135594	2016-06-30	2016-07-03	AH-10120	TEC-AC-10003038	50.12	7
2	CA-2016-162355	2016-06-30	2016-07-02	PF-19165	FUR-BO-10004695	1266.86	7
3	CA-2016-162355	2016-06-30	2016-07-02	PF-19165	OFF-AR-10004344	35.97	3
4	CA-2016-140543	2016-06-29	2016-07-03	Co-12640	OFF-ST-10000563	191.88	6
5	CA-2016-161095	2016-06-28	2016-06-28	CS-12250	OFF-BI-10001524	33.568	2
6	CA-2016-110044	2016-06-28	2016-07-02	RF-19735	TEC-PH-10001299	359.976	3
7	CA-2016-154767	2016-06-28	2016-06-30	BP-11155	OFF-AP-10002311	275.24	4
8	CA-2016-154767	2016-06-28	2016-06-30	BP-11155	OFF-BI-10002003	7.96	2
9	CA-2016-161095	2016-06-28	2016-06-28	CS-12250	OFF-BI-10002764	7.752	3
10	CA-2016-154767	2016-06-28	2016-06-30	BP-11155	OFF-PA-10003039	61.96	2
11	CA-2016-154767	2016-06-28	2016-06-30	BP-11155	OFF-AP-10000595	8.74	2

Note: Always specify the **lower value first**, or you'll get an **empty result set**.

4. Using NOT BETWEEN

Get customers **not aged between 23 and 26**:

SELECT customer_id, customer_name, age

FROM customer

WHERE age NOT BETWEEN 23 AND 26

ORDER BY age DESC;

Query Query History

```sql
SELECT customer_id, customer_name, age
FROM customer
WHERE age NOT BETWEEN 23 AND 26
ORDER BY age DESC;
```

Data Output Messages Notifications

	customer_id [PK] character varying (255)	customer_name character varying (255)	age integer
1	RB-19360	Raymond Buch	70
2	BT-11680	Brian Thompson	70
3	NP-18325	Naresj Patel	70
4	KB-16315	Karl Braun	70
5	JL-15850	John Lucas	70
6	KM-16375	Katherine Murray	70
7	SD-20485	Shirley Daniels	70
8	JK-15730	Joe Kamberova	69
9	SC-20845	Sung Chung	69
10	AG-10525	Andy Gerbode	69
11	JR-15700	Jocasta Rupert	69

5. Equivalent using < and >

SELECT customer_id, customer_name,age

FROM customer

WHERE customer_age < 23 OR customer_age > 26

ORDER BY customer_age DESC;

Summary

- BETWEEN simplifies range-based filtering.
- It’s inclusive of the boundary values.
- More readable and often more efficient than using >= and <=.
- Can be combined with NOT to exclude ranges.

Practice Questions: Between Conditions

1. List all orders with sales between 100 and 500.
2. Find all orders placed between '2015-01-01' and '2016-12-31'.
3. Get orders where the quantity is not between 2 and 5.
4. Display all orders where the ship_date is between '2014-06-01' and '2014-06-30'.
5. Find products that were sold in quantities between 4 and 7 with a discount of 0.

SQL EXISTS Condition

Introduction

The EXISTS condition in PostgreSQL is used with a **subquery**.
It checks whether the subquery returns any rows.
Commonly used in SELECT, INSERT, UPDATE, and DELETE statements.

Syntax

```
SELECT column1, column2

FROM table1

WHERE EXISTS (

    SELECT 1

    FROM table2

    WHERE table2.column = table1.column

);
```


Result

- Returns **TRUE** if the subquery returns **at least one row**.
- Returns **FALSE** if the subquery returns **no rows**.

Key Points

- The SELECT 1 part inside the subquery is a common convention. It can be replaced by any constant or even *.
- Often used to test the **existence of related rows** in another table.
- **NOT EXISTS** is used to find rows that **do not** have matching records in the subquery.

Example 1: Customers who have placed orders

```
SELECT customer_name
FROM customer c
WHERE EXISTS (
    SELECT 1
    FROM sales s
    WHERE s.customer_id = c.customer_id
);
```

Query Query History

```
1  SELECT customer_name
2  FROM customer c
3  WHERE EXISTS (
4      SELECT 1
5      FROM sales s
6      WHERE s.customer_id = c.customer_id
7  );
8
```

Data Output Messages Notifications

	customer_name character varying (255)
1	Odella Nelson
2	Ted Butterfield
3	Paul Stevenson
4	Henry MacAllister
5	Joel Eaton
6	Christopher Schild
7	Parhena Norris
8	Pete Armstrong
9	Ryan Crowe
10	Steven Cartwright
11	Philip Fox

Example 2: Customers who have NOT placed orders

```
SELECT customer_name
FROM customer c
WHERE NOT EXISTS (
    SELECT 1
    FROM sales s
```

```
    WHERE c.customer_id = s.customer_id);
```

Query Query History

```sql
SELECT customer_name
FROM customer c
WHERE NOT EXISTS (
    SELECT 1
    FROM sales s
    WHERE c.customer_id = s.customer_id
);
```

Data Output Messages Notifications

customer_name character varying (255)

No match found

Alternative to EXISTS:

You can use IN or JOIN, but EXISTS is often more efficient, especially when checking for presence only.

Practice Questions: EXISTS Conditions

1. List all customers who have placed at least one order.
2. Find all products that have been sold at least once.
3. Show all orders where the customer exists in the customer table.
4. List customers who have not placed any orders.
5. Display all product IDs that were never ordered (do not exist in the sales table).

Download Complete SQL & PostgreSQL Notes + Practice Files

You can access the full set of **SQL/PostgreSQL notes** along with practice datasets and queries from this GitHub repository:

👉 **[SQL-resources-and-tutorials by akshay-dhage]**