The `WHERE` **clause** is used to filter records from a table. It returns only the rows that satisfy the specified condition.

We will use the **PostgreSQL DVD Rental (`dvdrental`) database** examples.

---

## 1. Basic Syntax

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

### Example

```
SELECT first_name, last_name
FROM customer
WHERE first_name = 'Mary';
```

This returns customers whose first name is **Mary**.

---

# 2. Comparison Operators

PostgreSQL supports the following common comparison operators:

| Operator | Meaning |
|---|---|
| = | Equal to |
| <> | Not equal to |
| != | Not equal to |
| > | Greater than |
| < | Less than |
| >= | Greater than or equal to |
| <= | Less than or equal to |

### = Equal To

```
SELECT *
FROM customer
WHERE store_id = 1;
```

Returns customers belonging to store 1.

### <> Not Equal To

```
SELECT *
FROM customer
WHERE store_id <> 1;
```

### != Not Equal To

```
SELECT *
FROM customer
WHERE store_id != 1;
```

<> and != both mean **not equal** in PostgreSQL.

### > Greater Than

```
SELECT title, rental_rate
FROM film
WHERE rental_rate > 3;
```

### < Less Than

```
SELECT title, rental_rate
FROM film
WHERE rental_rate < 2;
```

### >= Greater Than or Equal To

```
SELECT title, length
FROM film
WHERE length >= 130;
```

### <= Less Than or Equal To

```
SELECT payment_id, customer_id, amount
FROM payment
WHERE amount <= 4.99;
```

---

# 3. Logical Operators

Logical operators allow us to combine multiple conditions.

| Operator | Purpose |
|---|---|
| AND | All conditions must be true |
| OR | At least one condition must be true |
| NOT | Reverses a condition |

---

### AND

Both conditions must be true.

```
SELECT title, rental_rate, length
FROM film
WHERE rental_rate > 2
AND length > 120;
```

**Meaning:** Find films with a rental rate greater than 2 **and** length greater than 120 minutes.

---

### OR

At least one condition must be true.

```
SELECT title, length
FROM film
WHERE length < 60
   OR length > 120;
```

**Meaning:** Display films that are **less than 60 minutes OR greater than 120 minutes**.

---

### NOT

Reverses the condition.

```
SELECT title, rental_rate
FROM film
WHERE NOT rental_rate = 2.99;
```

This returns films whose rental rate is not 2.99.

---

## 4. BETWEEN Operator

BETWEEN checks whether a value falls within a range.

```sql
SELECT title, length
FROM film
WHERE length BETWEEN 90 AND 120;
```

`BETWEEN` is **inclusive**, so values `90` and `120` are included.

It is equivalent to:

```sql
WHERE length >= 90
  AND length <= 120;
```

`NOT BETWEEN`

```sql
SELECT title, length
FROM film
WHERE length NOT BETWEEN 90 AND 120;
```

---

# 5. `IN` Operator

`IN` checks whether a value matches any value in a list.

```sql
SELECT first_name, last_name, store_id
FROM customer
WHERE store_id IN (1, 2);
```

This is easier than writing:

```sql
WHERE store_id = 1
OR store_id = 2;
```

`NOT IN`

```sql
SELECT first_name, last_name, store_id
FROM customer
WHERE store_id NOT IN (1, 2);
```

---

# 6. `LIKE` Operator

`LIKE` is used for **pattern matching**.

### % Wildcard

% represents zero or more characters.

```
SELECT first_name, last_name
FROM customer
WHERE first_name LIKE 'A%';
```

Finds names beginning with **A**.

Examples:

```
Adam
Alice
Andrew
Amanda
```

### Ends With

```
SELECT first_name, last_name
FROM customer
WHERE first_name LIKE '%a';
```

Finds names ending with a.

### Contains

```
SELECT first_name, last_name
FROM customer
WHERE first_name LIKE '%an%';
```

Finds names containing an.

---

## 7. _ Wildcard

The underscore _ represents **exactly one character**.

```
SELECT first_name,last_name,email
FROM customer
WHERE first_name LIKE 'A____';
```

This searches for names beginning with A followed by exactly four characters.

For example:

```
Alice
Aaron
```

---

# 8. `ILIKE` Operator

PostgreSQL provides `ILIKE` for **case-insensitive pattern matching**.

```sql
SELECT first_name, last_name
FROM customer
WHERE first_name ILIKE 'a%';
```

It can match:

```
Adam
alice
AMANDA
Andrew
```

Compare:

```
LIKE
```

with:

```
ILIKE
```

`ILIKE` ignores differences between uppercase and lowercase letters.

---

# 9. `NOT LIKE`

```sql
SELECT first_name, last_name
FROM customer
WHERE first_name NOT LIKE 'A%';
```

Returns customers whose first name does not start with `A`.

---

# 10. `IS NULL`

NULL means that a value is **missing or unknown**.

We cannot use:

```
WHERE column = NULL
```

Instead, use:

```
IS NULL
```

Example:

```
SELECT *
FROM address
WHERE address2 IS NULL;
```

---

## 11. IS NOT NULL

```
SELECT rental_id, rental_date, return_date
FROM rental
WHERE return_date IS NOT NULL;
```

**Meaning:** Display rentals where the movie has been returned.

---

## 12. ANY

ANY compares a value with values returned by an array or subquery.

Example:

```
SELECT title, rental_rate
FROM film
WHERE rental_rate = ANY (ARRAY[0.99, 2.99, 4.99]);
```

This means the rental rate matches **any** value in the array. ANY works with an array or subquery and can be combined with comparison operators such as >, <, >=, etc.

---

## 14. ALL

`ALL` requires the comparison to be true for **every value** returned by the expression.

Example:

```
SELECT title, rental_rate
FROM film
WHERE rental_rate > ALL (ARRAY[0.99, 1.99]);
```

The rental rate must be greater than **both** 0.99 and 1.99.

---

# 15. Regular Expression Operators

PostgreSQL also supports regular-expression matching.

`~`

Case-sensitive regular expression matching.

```
SELECT first_name
FROM customer
WHERE first_name ~ '^A';
```

Names beginning with `A`.

`~*`

Case-insensitive regular expression matching.

```
SELECT first_name
FROM customer
WHERE first_name ~* '^a';
```

`!~`

Does not match a regular expression.

```
SELECT first_name
FROM customer
WHERE first_name !~ '^A';
```

`!~*`

Case-insensitive negative regular expression matching.

```
SELECT first_name
FROM customer
WHERE first_name !~* '^a';
```

---

# 16. Combining Operators

You can combine several operators in one `WHERE` clause.

```
SELECT title, rental_rate, length
FROM film
WHERE rental_rate >= 2
AND rental_rate <= 4
AND length > 100;
```

Another example:

```
SELECT first_name, last_name, store_id
FROM customer
WHERE store_id IN (1, 2)
AND first_name ILIKE 'a%';
```

---

# 17. Using Parentheses

Parentheses are important when using `AND` and `OR`.

```
SELECT title, rental_rate
FROM film
WHERE (rental_rate = 2.99 OR rental_rate = 4.99)
AND length > 100;
```

Without parentheses, the result may not be what you expect because PostgreSQL evaluates `AND` before `OR`.

---

# 18. `WHERE` with Dates

The DVD Rental database contains date/time columns.

Example:

```
SELECT payment_id, customer_id, amount, payment_date
FROM payment
```

```
WHERE payment_date >= '2007-02-15';
```

Date Range

```
SELECT payment_id, customer_id, amount, payment_date
FROM payment
WHERE payment_date BETWEEN '2007-02-15' AND '2007-02-20';
```

---

# 19. WHERE with Numeric Values

```
SELECT payment_id, customer_id, amount
FROM payment
WHERE amount > 5;
```

Multiple conditions:

```
SELECT payment_id, customer_id, amount
FROM payment
WHERE amount >= 5
AND amount <= 10;
```

Or:

```
SELECT payment_id, customer_id, amount
FROM payment
WHERE amount BETWEEN 5 AND 10;
```

---

# 20. WHERE with Text

```
SELECT first_name, last_name
FROM customer
WHERE last_name = 'Smith';
```

Case-insensitive search:

```
SELECT first_name, last_name
FROM customer
WHERE last_name ILIKE 'smith';
```

Partial search:

```
SELECT first_name, last_name
FROM customer
WHERE last_name ILIKE '%son%';
```

---

# 21. Important Difference: `WHERE` vs `HAVING`

### `WHERE`

Filters **individual rows before grouping**.

```
SELECT *
FROM payment
WHERE amount > 5;
```

### `HAVING`

Filters **groups after `GROUP BY`**.

```
SELECT customer_id, COUNT(*)
FROM payment
GROUP BY customer_id
HAVING COUNT(*) > 10;
```

A simple rule for students:

> **WHERE → filter rows HAVING → filter groups**

---

# 22. Complete Example

Suppose we want to find customers:

- from store 1 or 2
- whose first name starts with `A`
- and whose customer ID is greater than 10.

```
SELECT customer_id, first_name, last_name, store_id
FROM customer
WHERE store_id IN (1, 2)
AND first_name ILIKE 'A%'
AND customer_id > 10;
```

This demonstrates:

- `IN`
- `ILIKE`
- `>`
- `AND`

---

# 23. Operator Summary

| Category | Operators |
|---|---|
| Equality | `=` |
| Not equal | `<>`, `!=` |
| Comparison | `>`, `<`, `>=`, `<=` |
| Logical | `AND`, `OR`, `NOT` |
| Range | `BETWEEN`, `NOT BETWEEN` |
| List | `IN`, `NOT IN` |
| Pattern | `LIKE`, `NOT LIKE` |
| Case-insensitive pattern | `ILIKE` |
| NULL | `IS NULL`, `IS NOT NULL` |
| Array/Subquery comparison | `ANY`, `ALL` |
| Regular expression | `~`, `~*`, `!~`, `!~*` |

---

## Practice Questions

Using the **DVD Rental database**, write SQL queries to:

1. Find all customers from `store_id = 1`.
2. Find films with a rental rate greater than `3`.
3. Find films with a rental rate between `2` and `4`.
4. Find customers whose first name starts with `M`.
5. Find customers whose last name contains `son`.
6. Find customers belonging to store 1 or 2 using `IN`.
7. Find payments greater than `5`.
8. Find payments between `2` and `6`.
9. Find addresses where `address2` is `NULL`.
10. Find addresses where `address2` is not `NULL`.
11. Find films whose title contains the word `love`, ignoring case.
12. Find customers whose first name does not start with `A`.
13. Find customers with `customer_id > 100` and `store_id = 1`.
14. Find customers whose first name starts with `A` **or** `B`.

15. Find customers who have at least one payment using `EXISTS`.

## Key idea to remember

```
SELECT columns
FROM table
WHERE condition;
```

Think of `WHERE` as a **filter**:

**Table → WHERE condition → Only matching rows → SELECT output**.