Day28_SQL_Wildcard_Operators_&_Joins

June 25, 2025

SQL Wildcard Operators and Joins

Today, we learned two important concepts in SQL:

- Wildcard Operators – used for flexible pattern matching
- Joins – used to combine data from multiple tables based on related columns

In this notebook, we will:

- Create required tables and insert data
- Practice different wildcard patterns using the LIKE operator
- Learn and apply different types of SQL JOINs with examples

1 SQL Wildcard Operators

What are Wildcard Operators?

Wildcard operators in SQL are used with the `LIKE` clause to **search for patterns** in string data. They let you filter values based on partial matches — instead of exact matches — using special characters.

Common Wildcard Operators

Wildcard	Description
`%`	Matches **zero, one, or many characters**
`_`	Matches **exactly one character**

Note: Wildcards only work with the `LIKE` operator.

Basic Syntax

```sql
SELECT * FROM table_name
WHERE column_name LIKE 'pattern';
```

Examples

1. `%` – Match any number of characters

```
-- Finds: All names starting with 'A'
SELECT * FROM CUSTOMERS WHERE NAME LIKE 'A%';
```

```
-- Finds: All names ending with `sh'
SELECT * FROM CUSTOMERS WHERE NAME LIKE '%sh';
```

```
-- Finds: All names containing `a' anywhere
SELECT * FROM CUSTOMERS WHERE NAME LIKE '%a%';
```

---

2. `_` – Match exactly one character

```
-- Finds: Names where the second letter is 'a'
SELECT * FROM CUSTOMERS WHERE NAME LIKE '_a%';
```

```
-- Finds: Names that are exactly 4 characters long
SELECT * FROM CUSTOMERS WHERE NAME LIKE '____';
```

Use Cases

- Partial name searches: `'R%h'` → "Ramesh", "Rohit"
- Email matching: `'%.com'` → ends with `.com`
- Pattern matching in product codes or IDs

When Not to Use

- When you want exact matches, use `=` instead.
- Avoid leading `%` (e.g., `LIKE '%abc'`) on large tables — it's slow.

Summary

- `%` → Matches **0 or more characters**
- `_` → Matches **exactly 1 character**
- Use with `LIKE` for pattern-based filtering

"'

1.1 Create Database and Tables

```
-- Creating a new database
mysql> CREATE DATABASE Day28SQLDemo;
Query OK, 1 row affected (0.78 sec)

-- Switching to the new database
mysql> USE Day28SQLDemo;
Database changed
```

```
-- Creating a 'CUSTOMERS' table
mysql> CREATE TABLE CUSTOMERS (
    -> ID INT,
    -> NAME VARCHAR(50),
    -> AGE INT,
    -> ADDRESS VARCHAR(100),
    -> SALARY DECIMAL(10, 2)
    -> );
Query OK, 0 rows affected (2.50 sec)
```

1.2 Insert Records into CUSTOMERS

```
mysql> INSERT INTO CUSTOMERS VALUES
    -> (1, 'Ramesh', 32, 'Ahmedabad', 2000.00),
    -> (2, 'Khilan', 25, 'Delhi', 1500.00),
    -> (3, 'Kaushik', 23, 'Kota', 2000.00),
    -> (4, 'Chaitali', 25, 'Mumbai', 6500.00),
    -> (5, 'Hardik', 27, 'Bhopal', 8500.00),
    -> (6, 'Komal', 22, 'MP', 4500.00),
    -> (7, 'Muffy', 24, 'Indore', 10000.00);
Query OK, 7 rows affected (0.38 sec)
Records: 7  Duplicates: 0  Warnings: 0

mysql> select * from customers;
+------+----------+------+-----------+----------+
| ID   | NAME     | AGE  | ADDRESS   | SALARY   |
+------+----------+------+-----------+----------+
|    1 | Ramesh   |   32 | Ahmedabad |  2000.00 |
|    2 | Khilan   |   25 | Delhi     |  1500.00 |
|    3 | Kaushik  |   23 | Kota      |  2000.00 |
|    4 | Chaitali |   25 | Mumbai    |  6500.00 |
|    5 | Hardik   |   27 | Bhopal    |  8500.00 |
|    6 | Komal    |   22 | MP        |  4500.00 |
|    7 | Muffy    |   24 | Indore    | 10000.00 |
+------+----------+------+-----------+----------+
7 rows in set (0.00 sec)
```

1.3 Wildcard Queries Using LIKE

```
-- Salaries starting with 200
mysql> SELECT * FROM CUSTOMERS WHERE SALARY LIKE '200%';
+------+---------+------+-----------+---------+
| ID   | NAME    | AGE  | ADDRESS   | SALARY  |
+------+---------+------+-----------+---------+
|    1 | Ramesh  |   32 | Ahmedabad | 2000.00 |
|    3 | Kaushik |   23 | Kota      | 2000.00 |
+------+---------+------+-----------+---------+
2 rows in set (0.00 sec)

-- Salaries containing 200 anywhere
mysql> SELECT * FROM CUSTOMERS WHERE SALARY LIKE '%200%';
+------+---------+------+-----------+---------+
| ID   | NAME    | AGE  | ADDRESS   | SALARY  |
+------+---------+------+-----------+---------+
|    1 | Ramesh  |   32 | Ahmedabad | 2000.00 |
|    3 | Kaushik |   23 | Kota      | 2000.00 |
+------+---------+------+-----------+---------+
2 rows in set (0.00 sec)

-- Salaries with 00 in second and third positions
mysql> SELECT * FROM CUSTOMERS WHERE SALARY LIKE '_00%';
+------+---------+------+-----------+----------+
| ID   | NAME    | AGE  | ADDRESS   | SALARY   |
+------+---------+------+-----------+----------+
|    1 | Ramesh  |   32 | Ahmedabad |  2000.00 |
|    3 | Kaushik |   23 | Kota      |  2000.00 |
|    7 | Muffy   |   24 | Indore    | 10000.00 |
+------+---------+------+-----------+----------+
3 rows in set (0.00 sec)

-- Salaries starting with 2 and at least 3 characters
mysql> SELECT * FROM CUSTOMERS WHERE SALARY LIKE '2_%_%';
+------+---------+------+-----------+---------+
| ID   | NAME    | AGE  | ADDRESS   | SALARY  |
+------+---------+------+-----------+---------+
|    1 | Ramesh  |   32 | Ahmedabad | 2000.00 |
|    3 | Kaushik |   23 | Kota      | 2000.00 |
+------+---------+------+-----------+---------+
2 rows in set (0.00 sec)

-- Salaries ending with 2
mysql> SELECT * FROM CUSTOMERS WHERE SALARY LIKE '%2';
Empty set (0.00 sec)

-- Second character is 2 and ends with 3
mysql> SELECT * FROM CUSTOMERS WHERE SALARY LIKE '_2%3';
Empty set (0.00 sec)

-- 5-digit salary starting with 2 and ending with 3
mysql> SELECT * FROM CUSTOMERS WHERE SALARY LIKE '2___3';
Empty set (0.00 sec)
```

2 SQL Joins

What are Joins?

SQL **JOINs** are used to **combine rows** from two or more tables based on a related column (like a foreign key).

Types of Joins

2.1 INNER JOIN

Returns only the **matching rows** from both tables.

```sql
SELECT *
FROM Students s
INNER JOIN Courses c ON s.course_id = c.course_id;
```

Use when: You want data that exists in **both** tables.

2.2 LEFT JOIN

Returns **all rows from the left table** and matched rows from the right. If no match, shows NULL.

```sql
SELECT *
FROM Students s
LEFT JOIN Courses c ON s.course_id = c.course_id;
```

Use when: You want **all students**, even if they're not enrolled in a course.

2.3 RIGHT JOIN

Returns **all rows from the right table** and matched rows from the left. If no match, shows NULL.

```sql
SELECT *
FROM Students s
RIGHT JOIN Courses c ON s.course_id = c.course_id;
```

Use when: You want **all courses**, even if no one is enrolled.

2.4 4. FULL OUTER JOIN (via UNION in MySQL)

Returns **all rows** from both tables, with NULLs where there is no match.

```sql
SELECT *
FROM Students s
LEFT JOIN Courses c ON s.course_id = c.course_id
```

```
UNION
SELECT *
FROM Students s
RIGHT JOIN Courses c ON s.course_id = c.course_id;
```

Use when: You want **everything**, matched and unmatched.

Summary

Join Type	Returns
INNER JOIN	Only matched rows in both tables
LEFT JOIN	All from left + matched from right
RIGHT JOIN	All from right + matched from left
FULL OUTER	All from both tables (UNION of LEFT + RIGHT JOIN)

3 Joins Examples

Create STUDENTS and COURSES tables

```
-- STUDENTS table
mysql> CREATE TABLE STUDENTS (
    -> ID INT PRIMARY KEY,
    -> NAME VARCHAR(50),
    -> COURSE_ID INT
    -> );
Query OK, 0 rows affected (1.25 sec)

-- COURSES table
mysql> CREATE TABLE COURSES (
    -> COURSE_ID INT PRIMARY KEY,
    -> COURSE_NAME VARCHAR(50)
    -> );
Query OK, 0 rows affected (0.76 sec)
```

Insert Records

```
-- STUDENTS data
mysql> INSERT INTO STUDENTS VALUES
    -> (1, 'Akshay', 101),
    -> (2, 'Swara', 102),
    -> (3, 'Ravi', 103),
    -> (4, 'Neha', NULL);
Query OK, 4 rows affected (0.27 sec)

-- COURSES data
mysql> INSERT INTO COURSES VALUES
    -> (101, 'Math'),
    -> (102, 'Science'),
    -> (104, 'English');
Query OK, 3 rows affected (0.34 sec)

mysql> select * from courses;
+-----------+-------------+
| COURSE_ID | COURSE_NAME |
+-----------+-------------+
|       101 | Math        |
|       102 | Science     |
|       104 | English     |
+-----------+-------------+
3 rows in set (0.00 sec)

mysql> select * from students;
+----+--------+-----------+
| ID | NAME   | COURSE_ID |
+----+--------+-----------+
|  1 | Akshay |       101 |
|  2 | Swara  |       102 |
|  3 | Ravi   |       103 |
|  4 | Neha   |      NULL |
+----+--------+-----------+
4 rows in set (0.00 sec)
```

3.1 INNER JOIN – Matching records only

```
mysql> SELECT s.ID, s.NAME, c.COURSE_NAME
    -> FROM STUDENTS s
    -> INNER JOIN COURSES c ON s.COURSE_ID = c.COURSE_ID;
+----+--------+-------------+
| ID | NAME   | COURSE_NAME |
+----+--------+-------------+
|  1 | Akshay | Math        |
|  2 | Swara  | Science     |
+----+--------+-------------+
2 rows in set (0.06 sec)
```

3.2 LEFT JOIN – All students, even if course is missing

```
mysql> SELECT s.ID, s.NAME, c.COURSE_NAME
    -> FROM STUDENTS s
    -> LEFT JOIN COURSES c ON s.COURSE_ID = c.COURSE_ID;
+----+--------+-------------+
| ID | NAME   | COURSE_NAME |
+----+--------+-------------+
|  1 | Akshay | Math        |
|  2 | Swara  | Science     |
|  3 | Ravi    | NULL         |
|  4 | Neha    | NULL         |
+----+---------+-------------+
4 rows in set (0.17 sec)
```

3.3 RIGHT JOIN – All courses, even if no student enrolled

```
mysql> SELECT s.ID, s.NAME, c.COURSE_NAME
    -> FROM STUDENTS s
    -> RIGHT JOIN COURSES c ON s.COURSE_ID = c.COURSE_ID;
+------+---------+-------------+
| ID   | NAME    | COURSE_NAME |
+------+---------+-------------+
|    1 | Akshay  | Math        |
|    2 | Swara   | Science     |
| NULL | NULL    | English     |
+------+---------+-------------+
3 rows in set (0.00 sec)
```

3.4 FULL OUTER JOIN – Simulated using UNION

```
mysql> SELECT s.ID, s.NAME, c.COURSE_NAME
    -> FROM STUDENTS s
    -> LEFT JOIN COURSES c ON s.COURSE_ID = c.COURSE_ID
    -> UNION
    -> SELECT s.ID, s.NAME, c.COURSE_NAME
    -> FROM STUDENTS s
    -> RIGHT JOIN COURSES c ON s.COURSE_ID = c.COURSE_ID;
+------+---------+-------------+
| ID   | NAME    | COURSE_NAME |
+------+---------+-------------+
|    1 | Akshay  | Math        |
|    2 | Swara   | Science     |
|    3 | Ravi    | NULL        |
|    4 | Neha    | NULL        |
| NULL | NULL    | English     |
+------+---------+-------------+
5 rows in set (0.07 sec)
```

4 Summary

- % matches 0 or more characters; _ matches exactly 1 character.
- LIKE helps pattern-match in string queries.
- Joins help combine data across tables:
  - INNER JOIN: only matching rows
  - LEFT JOIN: all from left + matches

  - RIGHT JOIN: all from right + matches
  - FULL OUTER JOIN: all from both using UNION