MySQL Constraints

Constraints in MySQL are rules applied to table columns to ensure the **accuracy**, **validity**, and **integrity** of the data.

---

1. `UNIQUE` Constraint

Ensures that all values in a column are **different**.

Example (during table creation):

```
CREATE TABLE users (
    id INT PRIMARY KEY,
    email VARCHAR(100) UNIQUE
);
```

Add `UNIQUE` using `ALTER TABLE`:

```
ALTER TABLE users
ADD CONSTRAINT unique_email UNIQUE (email);
```

---

2. `NOT NULL` Constraint

Ensures that a column **cannot contain NULL** values.

Example:

```
CREATE TABLE users (
    id INT PRIMARY KEY,
    name VARCHAR(100) NOT NULL
);
```

Change an existing column to NOT NULL:

```
ALTER TABLE users
MODIFY COLUMN name VARCHAR(100) NOT NULL;
```

Make a column nullable again:

```
ALTER TABLE users
MODIFY COLUMN name VARCHAR(100) NULL;
```

---

3. `CHECK` Constraint

Ensures that values in a column satisfy a **specific condition**.

Example: Allow only dates of birth after Jan 1, 2000

```
ALTER TABLE users
ADD CONSTRAINT chk_dob CHECK (date_of_birth > '2000-01-01');
```

Naming the constraint (`chk_dob`) helps if you want to drop it later.

---

4. `DEFAULT` Constraint

Sets a **default value** for a column if none is provided during insert.

Example:

```
CREATE TABLE users (
    id INT PRIMARY KEY,
    is_active BOOLEAN DEFAULT TRUE
);
```

Add DEFAULT using `ALTER TABLE`:

```
ALTER TABLE users
ALTER COLUMN is_active SET DEFAULT TRUE;
```

---

5. `PRIMARY KEY` Constraint

Uniquely identifies each row. Must be **NOT NULL** and **UNIQUE**.

Example:

```
CREATE TABLE users (
    id INT PRIMARY KEY,
    name VARCHAR(100)
);
```

Add later with `ALTER TABLE`:

```
ALTER TABLE users
ADD PRIMARY KEY (id);
```

---

6. `AUTO_INCREMENT`

Used with `PRIMARY KEY` to automatically assign the next number.

Example:

```
CREATE TABLE users (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100)
);
```

Each new row gets the next available integer value in `id`.

Summary Table

Constraint	Purpose
`UNIQUE`	Prevents duplicate values
`NOT NULL`	Ensures value is not NULL
`CHECK`	Restricts values using a condition
`DEFAULT`	Sets a default value
`PRIMARY KEY`	Uniquely identifies each row
`AUTO_INCREMENT`	Automatically generates unique numbers

SQL Functions (MySQL)

SQL functions help you **analyze**, **transform**, or **summarize** data in your tables.

We'll use the `users` table which includes:

- `id`, `name`, `email`, `gender`, `date_of_birth`, `salary`, `created_at`

---

1. Aggregate Functions

These return a **single value** from a set of rows.

`COUNT()`

Count total number of users:

```
SELECT COUNT(*) FROM users;
```

Count users who are Female:

```
SELECT COUNT(*) FROM users WHERE gender = 'Female';
```

`MIN()` and `MAX()`

Get the minimum and maximum salary:

```
SELECT MIN(salary) AS min_salary, MAX(salary) AS max_salary FROM users;
```

`SUM()`

Calculate total salary payout:

```
SELECT SUM(salary) AS total_payroll FROM users;
```

`AVG()`

Find average salary:

```
SELECT AVG(salary) AS avg_salary FROM users;
```

Grouping with `GROUP BY`

Average salary by gender:

```
SELECT gender, AVG(salary) AS avg_salary
FROM users
GROUP BY gender;
```

---

2. String Functions

`LENGTH()`

Length of user names:

```
SELECT name, LENGTH(name) AS name_length FROM users;
```

`LOWER()` and `UPPER()`

Convert names to lowercase or uppercase:

```
SELECT name, LOWER(name) AS lowercase_name FROM users;
SELECT name, UPPER(name) AS uppercase_name FROM users;
```

`CONCAT()`

Combine name and email:

```
SELECT CONCAT(name, ' <', email, '>') AS user_contact FROM users;
```

---

3. Date Functions

`NOW()`

Current date and time:

```
SELECT NOW();
```

`YEAR()`, `MONTH()`, `DAY()`

Extract parts of `date_of_birth`:

```
SELECT name, YEAR(date_of_birth) AS birth_year FROM users;
```

`DATEDIFF()`

Find number of days between today and birthdate:

```
SELECT name, DATEDIFF(CURDATE(), date_of_birth) AS days_lived FROM users;
```

`TIMESTAMPDIFF()`

Calculate age in years:

```
SELECT name, TIMESTAMPDIFF(YEAR, date_of_birth, CURDATE()) AS age FROM users;
```

---

4. Mathematical Functions

`ROUND()`, `FLOOR()`, `CEIL()`

```
SELECT salary,
       ROUND(salary) AS rounded,
       FLOOR(salary) AS floored,
       CEIL(salary) AS ceiled
FROM users;
```

`MOD()`

Find even or odd user IDs:

```
SELECT id, MOD(id, 2) AS remainder FROM users;
```

---

5. Conditional Functions

`IF()`

```
SELECT name, gender,
       IF(gender = 'Female', 'Yes', 'No') AS is_female
FROM users;
```

Summary Table

Function	Purpose
`COUNT()`	Count rows
`SUM()`	Total of a column
`AVG()`	Average of values
`MIN()` / `MAX()`	Lowest / highest value
`LENGTH()`	String length
`CONCAT()`	Merge strings
`YEAR()` / `DATEDIFF()`	Date breakdown / age
`ROUND()`	Rounding numbers
`IF()`	Conditional logic