# 1. MySQL Data Types — Definitions & Theory

## What is a Data Type in MySQL?

A data type in MySQL defines what kind of data can be stored in a table column.
It tells the database whether the value is a number, text, date, time, file, or structured data.

Using correct data types helps:

- Save storage space
- Improve performance
- Increase security
- Maintain data accuracy
- Build scalable applications

**Main Data Type Categories**

**1. Numeric Data Types : Numeric data types are used to store numbers. These can be whole numbers or decimal values.**

**2. String Data Types : String data types store text values.**

**3. Date and Time Data Types : These data types store date and time information.**

**4. Boolean Data Type : Boolean stores true or false values.**

**5. JSON Data Type : JSON stores structured data in JSON format.**

**6. Binary Data Types : Binary types store files or binary content.**

**7. ENUM Data Type : ENUM allows selecting one value from predefined options.**

## Datatype - information:

| Category | Data Type | Description | Size / Range | Example Use Case | Column Example | Insert Value Example |
|---|---|---|---|---|---|---|
| 1. Numeric Data Types | TINYINT | Small whole numbers | 1 Byte | Status flag | status TINYINT | 1 |
| | SMALLINT | Small integer values | 2 Bytes | Counters | count SMALLINT | 250 |
| | MEDIUMINT | Medium-size integers | 3 Bytes | Views | views MEDIUMINT | 500000 |
| | INT | Normal whole numbers | 4 Bytes | User ID | id INT | 1001 |
| | BIGINT | Very large numbers | 8 Bytes | Large counters | views BIGINT | 99999999 |
| | DECIMAL(10,2) | Exact decimal values | Precision based | Money | price DECIMAL(10,2) | 1999.99 |
| | FLOAT | Approximate decimal values | 4 Bytes | Ratings | rating FLOAT | 4.5 |
| | DOUBLE | High precision decimal | 8 Bytes | Analytics | score DOUBLE | 99.87654 |
| | BIT(1) | Binary true/false | 1 Bit | Boolean flag | is_active BIT(1) | 1 |
| 2. String Data Types | CHAR(2) | Fixed-length text | Fixed size | Country code | country CHAR(2) | IN' |
| | VARCHAR(50) | Small variable text | Up to 65KB | Username | username VARCHAR(50) | rahul123' |
| | VARCHAR(255) | Standard variable text | Up to 65KB | Email | email VARCHAR(255) | user@gmail.com' |

| | | | | | | |
|---|---|---|---|---|---|---|
| | TEXT | Large text content | 65,535 chars | Description | description TEXT | Sample description' |
| | MEDIUMTEXT | Medium large text | 16MB | Blog content | article MEDIUMTEXT | Long content...' |
| | LONGTEXT | Very large text | 4GB | News articles | content LONGTEXT | Huge article content...' |
| | ENUM | One value from list | 1–2 Bytes | Status | status ENUM('active','inactive') | active' |
| | SET | Multiple values from list | Up to 64 values | Roles | roles SET('admin','editor') | admin,editor' |
| 3. Date and Time Data Types | DATE | Stores only date | 3 Bytes | Birthday | dob DATE | 2000-05-10' |
| | TIME | Stores only time | 3 Bytes | Login time | login_time TIME | 10:30:00' |
| | DATETIME | Stores date and time | 8 Bytes | Event schedule | event_at DATETIME | 2026-05-18 10:30:00' |
| | TIMESTAMP | Auto current timestamp | 4 Bytes | Created time | created_at TIMESTAMP | CURRENT_TIMESTAMP |
| | YEAR | Stores year only | 1 Byte | Manufacturing year | year YEAR | 2026 |
| 4. Boolean Data Type | BOOLEAN | True or False value | 1 Byte | Active status | is_verified BOOLEAN | TRUE |
| 5. JSON Data Type | JSON | Structured JSON data | Variable | Settings/API data | config JSON | {"theme":"dark"}' |
| 6. Binary Data Types | BLOB | Binary/file data | 65KB | Images/files | image BLOB | binary data |
| | MEDIUMBLOB | Medium binary data | 16MB | Videos | video MEDIUMBLOB | binary data |
| | LONGBLOB | Very large binary data | 4GB | Backups/files | backup LONGBLOB | binary data |
| 7. ENUM Data Type | ENUM | Predefined values | 1–2 Bytes | News status | status ENUM('draft','published','archived') | published' |

# 2. Database - Create, Delete & More

## What is Database

A database is an organized collection of data stored electronically.
It is used to store, manage, retrieve, and manipulate information efficiently.

Databases help applications:

- Store user data
- Manage products
- Save news/articles
- Handle payments
- Maintain records securely

In MySQL, a database contains multiple tables.

Example: `Database → Tables → Rows → Columns`

## Database Concepts

| Concept | Meaning |
|---|---|
| Database | Collection of related tables |
| Table | Stores data in rows and columns |
| Row | Single record |
| Column | Single field/data type |
| Cell | Single value |

## Database Operations Table

| Category | SQL Command | Description | Example Use Case | Example Query |
|---|---|---|---|---|
| Database | CREATE DATABASE | Creates a new database | Start new project | CREATE DATABASE news_app; |
| | SHOW DATABASES | Displays all databases | View existing databases | SHOW DATABASES; |
| | USE | Selects a database | Work inside database | USE news_app; |
| | ALTER DATABASE | Modifies database settings | Change charset/collation | ALTER DATABASE news_app CHARACTER SET utf8mb4; |
| | DROP DATABASE | Deletes database permanently | Remove unused project | DROP DATABASE news_app; |

## Simple Database Flow

Create Database
↓
Use Database
↓
Create Tables
↓
Store Data(INSERT)

# 3. CREATE TABLE — Basic Theory

`CREATE TABLE` is used to create a new table inside a database.

A table stores data in:

- Rows
- Columns

Each column has:

- Column name
- Data type
- Constraints

Example:

Table → users
Columns → id, name, email
Rows → actual user data

## CREATE TABLE Syntax

```
CREATE TABLE table_name (
   column_name datatype constraints);
```

## CREATE TABLE Example

```
CREATE TABLE users (
   id INT PRIMARY KEY AUTO_INCREMENT,
   name VARCHAR(100),
   email VARCHAR(255),
   password VARCHAR(255),
   created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

## CREATE TABLE Components

| Component | Meaning | Example |
|---|---|---|
| Table Name | Name of table | users |
| Column Name | Field name | email |
| Data Type | Type of data | VARCHAR(255) |
| Constraint | Rules for data | PRIMARY KEY |

## Frequently Used Column Constraints

| Constraint | Description | Example |
|---|---|---|
| PRIMARY KEY | Unique row identifier | id INT PRIMARY KEY |
| AUTO_INCREMENT | Automatically increases value | id INT AUTO_INCREMENT |
| NOT NULL | Value required | ame VARCHAR(100) NOT NUL |
| UNIQUE | Prevent duplicate values | email VARCHAR(255) UNIQUE |
| DEFAULT | Default value if none provided | status TINYINT DEFAULT 1 |

## CREATE TABLE Operations Table

| Category | SQL Syntax | Description | Example Use Case | Example Query |
|---|---|---|---|---|
| Table | CREATE TABLE | Creates new table | Store users | CREATE TABLE users (...); |
| | SHOW TABLES | Shows all tables | View tables | SHOW TABLES; |
| | DESC table_name | Shows table structure | Check columns | DESC users; |
| | ALTER TABLE | Modify table | Add/remove columns | ALTER TABLE users ADD mobile VARCHAR(15); |
| | DROP TABLE | Deletes table permanently | Remove table | DROP TABLE users; |
| | TRUNCATE TABLE | Deletes all rows only | Clear table data | TRUNCATE TABLE users; |

## Example Enterprise-Level Table

```
CREATE TABLE news (
    id BIGINT UNSIGNED PRIMARY KEY AUTO_INCREMENT,
    title VARCHAR(255) NOT NULL,
    slug VARCHAR(255) UNIQUE,
    short_description TEXT,
    content LONGTEXT,
    status ENUM('draft','published') DEFAULT 'draft',
    views BIGINT DEFAULT 0,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
        ON UPDATE CURRENT_TIMESTAMP
);
```

## PRIMARY KEY

- Unique value
- Cannot be duplicate
- Identifies each row

---

## AUTO_INCREMENT

Automatically generates:

1
2
3
4

---

## CONSTRAINTS

SQL Constraints are rules applied to table columns to control the type of data that can be stored.

Constraints help:

- Maintain data accuracy
- Prevent invalid data
- Improve database integrity
- Enforce business rules

Example:

```
email VARCHAR(255) UNIQUE
```

Here:

UNIQUE constraint prevents duplicate emails

## Most Frequently Used SQL Constraints

| Constraint | Description | Example Use Case | Example Column | Insert Example |
|---|---|---|---|---|
| PRIMARY KEY | Unique identifier for each row | User ID | id INT PRIMARY KEY | 1 |
| AUTO_INCREMENT | Automatically increases value | Auto ID generation | id INT AUTO_INCREMENT | 1,2,3... |
| NOT NULL | Value cannot be empty | Required fields | name VARCHAR(100) NOT NULL | Rahul' |
| UNIQUE | Prevents duplicate values | Unique email | email VARCHAR(2 | user@gmail.com' |

| | | | | |
|---|---|---|---|---|
| | | | 55) UNIQUE | |
| DEFAULT | Sets default value | Default status | status TINYINT DEFAULT 1 | 1 |
| FOREIGN KEY | Creates relationship between tables | category_id relation | FOREIGN KEY(category_id) | 2 |
| CHECK | Validates condition | Age limit | CHECK(age >= 18) | 25 |
| ENUM | Allows only predefined values | Status/Role | status ENUM('active','inactive') | active' |

## Most Frequently Used CREATE TABLE Fields

| Field Name | Recommended Data Type | Description | Example Value | Example Column |
|---|---|---|---|---|
| id | BIGINT UNSIGNED AUTO_INCREMENT PRIMARY KEY | Unique row ID | 1 | id BIGINT UNSIGNED AUTO_INCREMENT PRIMARY KEY |
| uuid | CHAR(36) | Public unique ID | 550e8400-e29b... | uuid CHAR(36) |
| name | VARCHAR(100) | Store names | Rahul | name VARCHAR(100) |
| username | VARCHAR(50) | Login username | rahul123 | username VARCHAR(50) |
| email | VARCHAR(255) UNIQUE | Store email | user@gmail.com | email VARCHAR(255) UNIQUE |

| | | | | |
|---|---|---|---|---|
| password | VARCHAR(255) | Password hash | hashedpassword | password VARCHAR(255) |
| mobile | VARCHAR(15) | Phone number | 9876543210 | mobile VARCHAR(15) |
| title | VARCHAR(255) | Titles/headings | Breaking News | title VARCHAR(255) |
| slug | VARCHAR(255) UNIQUE | SEO URL | breaking-news | slug VARCHAR(255) UNIQUE |
| description | TEXT | Medium text content | Sample description | description TEXT |
| content | LONGTEXT | Large article/content | Long article... | content LONGTEXT |
| image | VARCHAR(255) | Image path/URL | image.jpg | image VARCHAR(255) |
| price | DECIMAL(10,2) | Money values | 1999.99 | price DECIMAL(10,2) |
| quantity | INT | Item count | 100 | quantity INT |
| views | BIGINT DEFAULT 0 | View counter | 5000 | views BIGINT DEFAULT 0 |
| status | TINYINT DEFAULT 1 | Active/inactive | 1 | status TINYINT DEFAULT 1 |
| role | ENUM('admin','user') | User role | admin | role ENUM('admin','user') |
| is_verified | BOOLEAN DEFAULT FALSE | True/False status | TRUE | is_verified BOOLEAN DEFAULT FALSE |
| meta | JSON | Flexible metadata | {"theme":"dark"} | meta JSON |
| created_at | TIMESTAMP DEFAULT CURRENT_TIMESTAMP | Creation time | auto generated | created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP |
| updated_at | TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP | Last update time | auto updated | updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP |
| deleted_at | TIMESTAMP NULL | Soft delete time | NULL | deleted_at TIMESTAMP NULL |

# 4. INSERT INTO — Basic Theory

`INSERT INTO` is used to add new records (rows) into a table.

It is used for:

- Adding users
- Storing products
- Saving news/articles
- Inserting orders/payments

## Basic INSERT Syntax

```
INSERT INTO table_name (column1, column2)
VALUES (value1, value2);
```

---

## INSERT INTO Example

```
INSERT INTO users (name, email)
VALUES ('Rahul', 'rahul@gmail.com');
```

# INSERT INTO Operations Table

| Category | Description | Example Query | Example Use Case | Example Output |
|---|---|---|---|---|
| INSERT | Inserts single row | INSERT INTO users VALUES (1,'Rahul','rahul@gmail.com',25); | Add new user | 1 row inserted |
| | Inserts complete row | INSERT INTO users VALUES (2,'Raj','raj@gmail.com',30); | Full row insert | Data inserted |
| | Inserts multiple rows | INSERT INTO users VALUES (3,'Kumar','kumar@gmail.com',28), (4,'Priya','priya@gmail.com',24), (5,'Arun','arun@gmail.com',22); | Bulk insert | Multiple rows inserted |
| | Insert using SET syntax | INSERT INTO users SET id=6, name='Sneha', email='sneha@gmail.com', age=29; | Alternative syntax | Data inserted |
| | Ignores duplicate errors | INSERT IGNORE INTO users VALUES (1,'Rahul','rahul@gmail.com',25); | Prevent failure | Row skipped |
| | Inserts data from another table | INSERT INTO backup_users SELECT * FROM users; | Backup/migration | Data copied |

# 5. SQL SELECT — Basic Theory

## What is Select

`SELECT` is used to retrieve/read data from a database table.

It is one of the most frequently used SQL commands.

Used for:

- Viewing records
- Searching data
- Filtering rows
- Sorting results
- Getting reports

## Basic SELECT Syntax

```
SELECT column_name FROM table_name;
```

# SQL SELECT Operations Table

| Category | Category | Example Query | Description | Example Use Case | Example Output |
|---|---|---|---|---|---|
| SELECT | SELECT ALL | SELECT * FROM users; | Retrieves all columns and rows | View complete users table | All user records |
| | SELECT COLUMNS | SELECT name,email FROM users; | Retrieves specific columns only | View names and emails | Rahul, rahul@gmail.com |
| | WHERE | SELECT * FROM users WHERE status=1; | Filters rows using condition | Active users | Only active users |
| | ORDER BY DESC | SELECT * FROM users ORDER BY salary DESC; | Sorts highest to lowest | Highest salary first | 95000, 75000 |
| | ORDER BY ASC | SELECT * FROM users ORDER BY age ASC; | Sorts lowest to highest | Youngest users first | 22, 24, 25 |
| | LIMIT | SELECT * FROM users LIMIT 5; | Restricts number of rows | Top 5 users | First 5 rows |
| | DISTINCT | SELECT DISTINCT city FROM users; | Removes duplicate values | Unique cities | Chennai, Delhi |
| | COUNT | SELECT COUNT(*) FROM users; | Counts total rows | Total users count | 10 |
| | LIKE | SELECT * FROM users WHERE name LIKE 'R%'; | Searches matching text | Names starting with R | Rahul, Raj |
| | IN | SELECT * FROM users WHERE role IN ('admin','editor'); | Matches multiple values | Admin and editor users | Matching rows |
| | BETWEEN | SELECT * FROM users WHERE salary BETWEEN 30000 AND 60000; | Filters within range | Medium salary users | 35000, 45000 |
| | GROUP BY | SELECT city, COUNT(*) FROM users GROUP BY city; | Groups similar rows | Users count by city | Chennai=3 |
| | MAX | SELECT MAX(salary) FROM users; | Finds highest value | Highest salary | 95000 |

| | | | | | |
|---|---|---|---|---|---|
| | MIN | SELECT MIN(salary) FROM users; | Finds lowest value | Lowest salary | 20000 |
| | AVG | SELECT AVG(rating) FROM users; | Finds average value | Average rating | 4.2 |
| | SUM | SELECT SUM(salary) FROM users; | Calculates total sum | Total salaries | 500000 |
| | AS | SELECT name AS username FROM users; | Renames output column | Better display name | username |
| | IS NULL | SELECT * FROM users WHERE mobile IS NULL; | Finds NULL values | Users without mobile | Matching rows |
| | IS NOT NULL | SELECT * FROM users WHERE mobile IS NOT NULL; | Finds non-NULL values | Users with mobile | Matching rows |
| | AND | SELECT * FROM users WHERE status=1 AND role='admin'; | Multiple conditions must match | Active admins | Matching rows |
| | OR | SELECT * FROM users WHERE role='admin' OR role='editor'; | Either condition matches | Admin or editor users | Matching rows |

# 6. LAB

## Create, Use Database, Create Table, Insert Table

```
-- Create Database
CREATE DATABASE company_db;

-- Select Database
USE company_db;

-- Create Users Table
CREATE TABLE users (
    id BIGINT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    email VARCHAR(255) UNIQUE NOT NULL,
    mobile VARCHAR(15),
    age INT,
    salary DECIMAL(10,2),
    role ENUM('admin','editor','user') DEFAULT 'user',
    status TINYINT DEFAULT 1,
    city VARCHAR(100),
    rating FLOAT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Insert Sample Data
INSERT INTO users
(name, email, mobile, age, salary, role, status, city, rating)
```

```
VALUES
('Rahul', 'rahul@gmail.com', '9876543210', 25, 45000.50, 'admin', 1, 'Chennai', 4.5),

('Raj', 'raj@gmail.com', '9876500000', 30, 55000.00, 'editor', 1, 'Mumbai', 4.2),

('Kumar', 'kumar@gmail.com', NULL, 28, 35000.75, 'user', 0, 'Delhi', 3.9),

('Ankit', 'ankit@gmail.com', '9123456789', 35, 75000.00, 'admin', 1, 'Bangalore', 4.8),

('Priya', 'priya@gmail.com', '9988776655', 24, 25000.00, 'user', 1, 'Chennai', 4.1),

('Sneha', 'sneha@gmail.com', NULL, 29, 48000.00, 'editor', 0, 'Hyderabad', 3.8),

('Vikram', 'vikram@gmail.com', '9000011111', 40, 95000.00, 'admin', 1, 'Pune', 4.9),

('Arun', 'arun@gmail.com', '9888899999', 22, 20000.00, 'user', 1, 'Chennai', 3.5),

('Deepak', 'deepak@gmail.com', '9777766666', 31, 60000.00, 'editor', 1, 'Delhi', 4.4),

('Meena', 'meena@gmail.com', '9666655555', 27, 42000.00, 'user', 0, 'Mumbai', 4.0);
```

## SELECT Queries

```
-- Select All
SELECT * FROM users;

-- Select Specific Columns
```

```sql
SELECT name, email FROM users;

-- WHERE
SELECT * FROM users WHERE status = 1;

-- ORDER BY DESC
SELECT * FROM users ORDER BY salary DESC;

-- ORDER BY ASC
SELECT * FROM users ORDER BY age ASC;

-- LIMIT
SELECT * FROM users LIMIT 5;

-- DISTINCT
SELECT DISTINCT city FROM users;

-- COUNT
SELECT COUNT(*) FROM users;

-- LIKE
SELECT * FROM users WHERE name LIKE 'R%';

-- IN
SELECT * FROM users
WHERE role IN ('admin','editor');

-- BETWEEN
SELECT * FROM users
WHERE salary BETWEEN 30000 AND 60000;
```

```sql
-- GROUP BY
SELECT city, COUNT(*)
FROM users
GROUP BY city;

-- MAX
SELECT MAX(salary) FROM users;

-- MIN
SELECT MIN(salary) FROM users;

-- AVG
SELECT AVG(rating) FROM users;

-- SUM
SELECT SUM(salary) FROM users;

-- IS NULL
SELECT * FROM users
WHERE mobile IS NULL;

-- IS NOT NULL
SELECT * FROM users
WHERE mobile IS NOT NULL;

-- AND
SELECT * FROM users
WHERE status = 1 AND role = 'admin';

-- OR
SELECT * FROM users
```

```
WHERE role = 'admin'
OR role = 'editor';
```