Database Fundamentals

Cybersecurity Department

Course Code: ***CBS 213***

Practical Lecture 3: MySQL Data Types and Constraints

Halal Abdulrahman Ahmed

Lecture Outlines

- Introduction to MySQL data types
- String, numeric, and date/time data types
- Choosing the right data type for columns
- Introduction to constraints (rules for valid data)
- Applying **PRIMARY KEY**, **FOREIGN KEY**, **NOT NULL**, **UNIQUE**, **CHECK**, and **DEFAULT**
- Practice tasks and lab exercises

Learning Outcomes

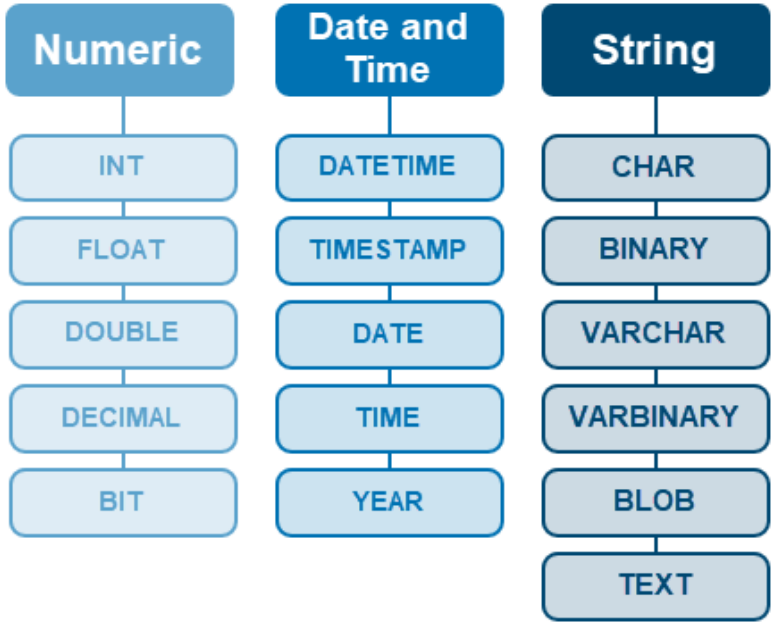
By the end of this lab, students will be able to:

- Identify and use appropriate **MySQL data types**.
- Apply constraints to ensure **data integrity**.
- Create and modify tables using SQL commands.
- Validate and test constraints through insert and update operations.

MySQL Data Types

MySQL Data Types

- Each column in a database table is required to have a name and a data type.
- MySQL uses data types to optimize storage, enforce consistency, and ensure data integrity.
- Each column in a MySQL table is assigned a specific data type.



In MySQL there are three main data types:

1. String Data Types
2. Numeric Data Types
3. Date and Time Data Types

String Data Types:

1. CHAR(n):

- Fixed-length string. It is efficient for storing data that is always the same length.
- Example: country codes (e.g., 'US’)

```
-- CHAR(n): fixed-length string, pads shorter values with spaces
CREATE TABLE test_char (
  country_code CHAR(3)
);
INSERT INTO test_char VALUES ('USA'), ('UK'), ('IRQ');
SELECT * FROM test_char;
```

String Data Types:

2. VARCHAR(n), (variable character):

- Variable-length string. It is efficient for most text data.
- Example: names, email addresses.

```sql
-- VARCHAR(n): variable-length string, best for names and general text
CREATE TABLE test_varchar (
  name VARCHAR(50),
  email VARCHAR(100)
);
INSERT INTO test_varchar VALUES ('Ali Ahmed', 'ali@example.com'), ('Sara Omer', 'sara@tiu.edu');
SELECT * FROM test_varchar;
```

3. TEXT:

- Used for large text fields. Cannot be indexed or used in GROUP BY or ORDER BY clauses easily.
- Types: TINYTEXT, TEXT, MEDIUMTEXT, LONGTEXT

```sql
-- TEXT: used for long text, such as comments or articles
CREATE TABLE test_text (
    description TEXT
);
INSERT INTO test_text VALUES ('This course covers MySQL basics and database design.');
SELECT * FROM test_text;
```

Attribute

Entity

Tuple

Types of TEXT

Type	Maximum Length	Used For
TINYTEXT	255 characters	Short notes, brief comments
TEXT	65,535 characters (~64 KB)	Descriptions, messages
MEDIUMTEXT	16,777,215 characters (~16 MB)	Articles, blog posts, long reports
LONGTEXT	4,294,967,295 characters (~4 GB)	Books, research papers, large documents

4. ENUM:

- Accepts one value from a predefined list.
- Example: ENUM('low', 'medium', 'high')
- Used for storing controlled options like status or categories.

```sql
-- ENUM: allows one value from a predefined list
CREATE TABLE test_enum (
  level ENUM('Low', 'Medium', 'High')
);
INSERT INTO test_enum VALUES ('Low'), ('High');
SELECT * FROM test_enum;
```

Maximum Length

Data Type	Maximum Length
CHAR	255 characters
VARCHAR	65,535 characters
TINYTEXT	255 characters
TEXT	65,535 characters
MEDIUMTEXT	16,777,215 characters
LONGTEXT	4,294,967,295 characters
ENUM	65,535 possible values

Security Note:

- ENUM ensures valid input. Helps prevent injection via invalid values.
- Avoid TEXT for sensitive user input unless properly sanitized and validated.

Integer Types (Exact Value):

1. INT (INTEGER):

- Stores whole numbers.
- Example: age, number of items.

Range: -2,147,483,648 to 2,147,483,647. Use UNSIGNED if only positive values are allowed.

```sql
-- INT: stores whole numbers (positive or negative)
CREATE TABLE test_int (
  id INT,
  quantity INT
);
INSERT INTO test_int VALUES (1, 10), (2, 25), (3, -5);
SELECT * FROM test_int;
```

Integer Types (Exact Value):

2. BIGINT:

- Used for very large integers.
- Example: storing large user IDs or counters.
- Range: -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807

```
-- BIGINT: used for very large integer values
CREATE TABLE test_bigint (
  population BIGINT
);
INSERT INTO test_bigint VALUES (7500000000), (12000000000);
SELECT * FROM test_bigint;
```

3. DECIMAL (M, D):

- Used for storing exact numeric values.
- Example: prices, financial data (e.g., DECIMAL(10,2)).
- Avoids rounding errors found in floating point types.

```sql
-- DECIMAL(M, D): stores exact numeric values (for financial or precise data)
CREATE TABLE test_decimal (
  price DECIMAL(8,2)
);
INSERT INTO test_decimal VALUES (12.50), (199.99);
SELECT * FROM test_decimal;
```

4. FLOAT/DOUBLE:

- Used for approximate values.
- Example: scientific data, temperature, or measurements.

```sql
-- FLOAT / DOUBLE: stores approximate numeric values (scientific or measurement data)
CREATE TABLE test_float (
  temperature FLOAT,
  distance DOUBLE
);
INSERT INTO test_float VALUES (36.6, 12500.45), (25.4, 8750.22);
SELECT * FROM test_float;
```

Data Type	Number of Digits (Precision)	Notes
FLOAT	About **6–7 digits** total	May round long numbers
DOUBLE	About **15–16 digits** total	More accurate

Security Tip:

- Use DECIMAL for financial data to maintain precision.
- Avoid using FLOAT or DOUBLE for IDs or sensitive calculations to prevent rounding attacks.

Date and Time Data Types:

1. DATE:

- Stores calendar dates in format: YYYY-MM-DD
- Use case: birthdates, deadlines.

```sql
-- DATE: stores date only in YYYY-MM-DD format
CREATE TABLE test_date (
  birth_date DATE
);
INSERT INTO test_date VALUES ('2000-05-14'), ('1998-11-21');
SELECT * FROM test_date;
```

Date and Time Data Types:

2. TIME:

- Stores time values in format: HH:MM:SS
- Use case: event start/end times.

```
-- TIME: stores time only in HH:MM:SS format
CREATE TABLE test_time (
  start_time TIME,
  end_time TIME
);
INSERT INTO test_time VALUES ('09:00:00', '11:30:00');
SELECT * FROM test_time;
```

3. DATETIME:

- Stores both date and time. Format: YYYY-MM-DD HH:MM:SS
- Use case: user login timestamps, transactions.

```
-- DATETIME: stores both date and time (local time zone)
CREATE TABLE test_datetime (
  login_time DATETIME
);
INSERT INTO test_datetime VALUES ('2025-10-16 14:30:00');
SELECT * FROM test_datetime;
```

4. TIMESTAMP:

- Similar to DATETIME but stored in UTC.
- Auto-updates on row update if configured.

```
-- TIMESTAMP: stores both date and time in UTC, auto-updates on changes
CREATE TABLE test_timestamp (
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
);
INSERT INTO test_timestamp VALUES ();
SELECT * FROM test_timestamp;
```

Security/Operational Tip:

- Prefer TIMESTAMP for logs and audits (consistent timezone).
- DATETIME is preferred for application-defined time (local time).

Common Data Types

Data Type	Use Case	Format/Example	Notes
INT	User ID, age	101	Basic whole number
VARCHAR(255)	Emails, names	'alice@uni.edu'	Flexible text
DATE	Birthdays	'2003-05-12'	Stores date only
DATETIME	Audit log	'2025-10-16 09:45:00'	Application time
DECIMAL(10,2)	Money	1000.50	Accurate, no rounding
ENUM('open','closed')	Status fields	'open'	Safe list enforcement

Why Choosing the Right Data Type Matters

- **Performance:** Efficient data types reduce memory and improve query speed.
- **Security:** Proper types prevent unexpected input (e.g., text in a number field). Helps reduce SQL injection risks.
- **Storage:** Smaller, specific types reduce database size and increase cache efficien

Best Practices

- Always specify length (e.g., VARCHAR(100), DECIMAL(10,2)).
- Use ENUM or CHECK constraints to enforce valid inputs.
- Prefer CHAR for fixed-length codes (e.g., country codes).
- Avoid TEXT/BLOB for searchable fields.
- Validate and sanitize all user inputs even if you use safe data types.
- Use TIMESTAMP for consistent, timezone-safe logging.
- Normalize your schema to reduce redundancy and errors.
- Understand the tradeoffs between JSON (flexibility) and structured columns (reliability).

Practice Task 1

- **Task:**
  Create a table called **student_info** with these columns:
- **id** → INT
- **name** → VARCHAR(50)
- **age** → TINYINT
- **gpa** → DECIMAL(3,2)
- **birth_date** → DATE
- **email** → VARCHAR(100)

Step 1

```
CREATE TABLE student_info (
  id INT,
  name VARCHAR(50),
  age TINYINT,
  gpa DECIMAL(3,2),
  birth_date DATE,
  email VARCHAR(100)
);
```

Step 2

```
DESCRIBE student_info;
```

MySQL Constraints

What Are Constraints?

- **Constraints** = Rules that restrict what data can go into a column.
- They protect the **accuracy and reliability** of data.
- Common types:
  - PRIMARY KEY
  - FOREIGN KEY
  - NOT NULL
  - UNIQUE
  - CHECK
  - DEFAULT

PRIMARY KEY

- Uniquely identifies each record in a table.
- Automatically NOT NULL and indexed.
- Each table should have **one** primary key.

```
CREATE TABLE student (
  id INT PRIMARY KEY,
  name VARCHAR(50)
);
```

FOREIGN KEY

- Links data between two tables.
- Ensures the value exists in the parent table.

```sql
CREATE TABLE department (
  dept_id INT PRIMARY KEY,
  dept_name VARCHAR(50)
);

CREATE TABLE employee (
  emp_id INT PRIMARY KEY,
  emp_name VARCHAR(50),
  dept_id INT,
  FOREIGN KEY (dept_id) REFERENCES department(dept_id)
);
```

NOT NULL, UNIQUE, CHECK, DEFAULT

- **NOT NULL**
  Value must be provided:

```
name VARCHAR(50) NOT NULL
```

- **UNIQUE**
  Prevents duplicates:

```
email VARCHAR(100) UNIQUE
```

CHECK
Ensures valid range or rule:

```
age INT CHECK (age BETWEEN 18 AND 30)
```

DEFAULT
Sets automatic value:

```
shift VARCHAR(10) DEFAULT 'Morning'
```

Full Example: All Constraints

```sql
CREATE TABLE student (
  id INT AUTO_INCREMENT PRIMARY KEY,
  name VARCHAR(50) NOT NULL,
  email VARCHAR(100) UNIQUE,
  age INT CHECK (age >= 18),
  major VARCHAR(30) DEFAULT 'IT',
  dept_id INT,
  FOREIGN KEY (dept_id) REFERENCES department(dept_id)
);
```

Homework 1

Create a table courses:

- course_id → INT PRIMARY KEY AUTO_INCREMENT
- title → VARCHAR(50) NOT NULL
- credits → INT CHECK (credits BETWEEN 1 AND 5)
- dept_id → INT FOREIGN KEY → department(dept_id)
- Insert some valid and invalid rows to see the constraints in action.

Homework 2

- Create database university_lab:
- Add tables: student, course, instructor.
- Use all major constraints.
- Insert and test data.

References

- Smirnova, S., & Tezuyal, A. (2022). *MySQL cookbook: Solutions for database developers and administrators* (4th ed.). O'Reilly Media.
- Silva, B. (2021). *MySQL crash course: A hands-on introduction to database development*. No Starch Press.


Any
Question