DATABASES

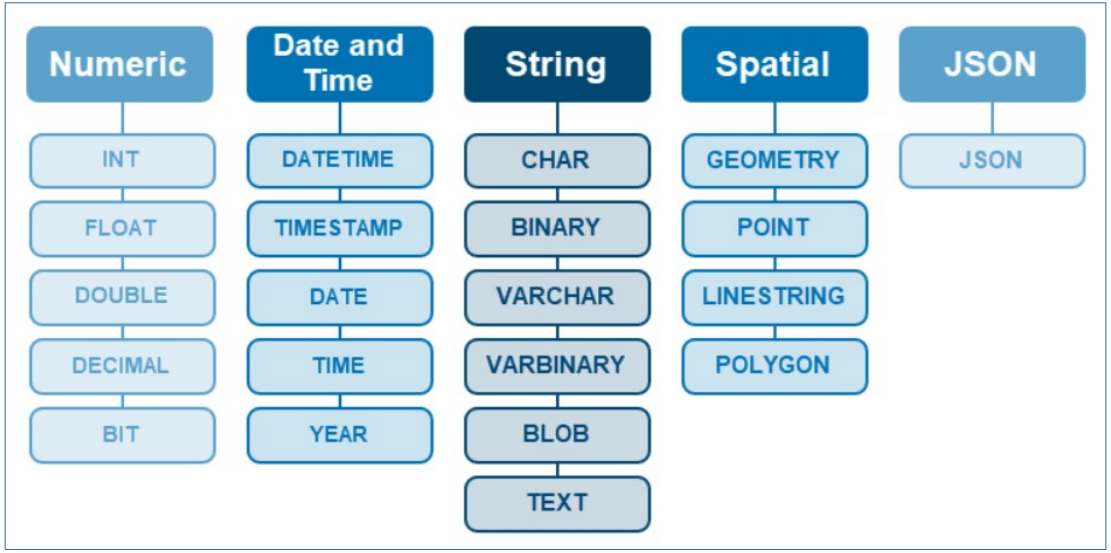
GS WEB APPLICATION DEVELOPMENT

Teacher: Vanesa Maldonado Guerrero
Curs 2025/26

MySQL Data types

A MySQL table can store many different data types. The most used types include:

- **Numeric**. Integer and decimal values of varying precision.
- **Date and time**. Time-based data using various formats.
- **String**. Textual data with different lengths.
- **Spatial**. Geometric-based information for geographic data.
- **JSON**. Stores JSON documents.



Numeric Data Types

Numeric data types are used for storing numbers, whether whole numbers or decimal values. SQL supports a range of number formats, suitable for tasks from counting items to handling precise financial calculations.

Integer Types

Data Type	Description	Example
TINYINT	Very small integers (usually 0 to 255)	100
SMALLINT	Small integers	32767
MEDIUMINT	Medium-range integers *(MySQL specific)*	8388607
INT or INTEGER	Standard integer	2147483647
BIGINT	Large integers	9223372036854775807

```sql
CREATE TABLE users (
  id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
  username VARCHAR(100) NOT NULL,
  age SMALLINT UNSIGNED,                -- Age cannot be negative
  is_active TINYINT(1) DEFAULT 1,       -- 1 = active, 0 = inactive
  credits BIGINT UNSIGNED DEFAULT 0     -- Large value counter
);
```

```sql
CREATE TABLE blog_stats (
  post_id INT UNSIGNED PRIMARY KEY,
  views MEDIUMINT UNSIGNED DEFAULT 0,   -- Up to ~16M views
  likes INT UNSIGNED DEFAULT 0,         -- Up to ~4.2B likes
  comments SMALLINT UNSIGNED DEFAULT 0  -- Up to ~65K comments
);
```

Numeric Data Types

Numeric data types are used for storing numbers, whether whole numbers or decimal values. SQL supports a range of number formats, suitable for tasks from counting items to handling precise financial calculations.

Decimal & Floating-Point Types

Data Type	Description	Example
DECIMAL(p, s)	Fixed-point number with '***p***' digits, '***s***' after decimal. High precision.	DECIMAL(5,2) → 123.45
NUMERIC(p, s)	Synonym for DECIMAL. Used interchangeably.	NUMERIC(8,3) → 12345.678
FLOAT(p)	Approximate floating-point. Can lose precision.	FLOAT(6) → 3.14159
REAL	Single-precision floating-point	2.71828
DOUBLE PRECISION / DOUBLE	Double-precision floating-point	1.2345678901

```
CREATE TABLE orders (
  order_id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
  total_amount DECIMAL(12, 2) NOT NULL,    -- Up to 10 digits before decimal, 2 decimal places
  tax_amount DECIMAL(8, 2) NOT NULL,
  discount DECIMAL(5, 2) DEFAULT 0.00
);
```

```
CREATE TABLE sensor_readings (
  reading_id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
  temperature DOUBLE NULL,      -- e.g. 23.567891234
  humidity FLOAT NULL,          -- e.g. 45.7
  pressure DOUBLE NULL          -- e.g. 1013.25
);
```

Character/String Data Types

The **character/string** data types are used to store alphanumeric values such as names, emails, or any other textual content. SQL provides various text data types to accommodate both fixed and variable-length strings, as well as longer text like descriptions or articles.

Data Type	Description	Example
CHAR(n)	Fixed-length string. Always uses '***n***' characters (padded with spaces).	'ABC ' (CHAR(6))
VARCHAR(n)	Variable-length string. Uses space only as needed (up to n chars).	'OpenAI'
TEXT	Long text. Can store large strings (length varies by DBMS).	Articles, blogs
TINYTEXT	Max 255 characters (MySQL).	'Short bio'
MEDIUMTEXT	Medium-sized text (~16MB, MySQL).	Blog posts
LONGTEXT	Very long text (~4GB, MySQL).	Books, documents
NCHAR(n)	Fixed-length Unicode string. Useful for multilingual text.	Unicode text
NVARCHAR(n)	Variable-length Unicode string	Multilingual text

```sql
CREATE TABLE users (
  user_id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
  country_code CHAR(2) NOT NULL,            -- Always 2 characters (ES, US, FR...)
  username VARCHAR(50) NOT NULL,            -- Size varies per user
  email VARCHAR(100) UNIQUE NOT NULL
);
```

```sql
CREATE TABLE profiles (
  profile_id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
  display_name VARCHAR(80) NOT NULL,
  short_bio TINYTEXT,                -- Up to 255 chars
  about_me TEXT                      -- Longer description, up to ~64KB
);
```

Difference Between CHAR And VARCHAR Data

CHAR Data Type:

- **CHAR** is a fixed-length data type. When you define a column as CHAR(n), it reserves n bytes of storage, padding shorter strings with spaces to meet the defined length.
- **Use Case:** Ideal for storing data where entries are of consistent length, such as fixed-length codes or identifiers.

VARCHAR Data Type:

- **VARCHAR** is a variable-length data type. Defining a column as VARCHAR(n) allows storage of strings up to n characters, using only as much space as needed for each entry, plus additional bytes for length information.
- **Use Case:** Suitable for storing data with variable lengths, such as names or addresses.

Example:

```
CREATE TABLE Departments (
    DeptCode CHAR(4),
    DeptName VARCHAR(50)
);

INSERT INTO Departments (DeptCode, DeptName)
VALUES ('HR', 'Human Resources');

SELECT DeptCode, LENGTH(DeptCode) AS CodeLength FROM Departments;
```

Output:

DeptCode	CodeLength
'HR '	4

Date and Time Data Types

Date and time data types are essential for storing information like birthdays, timestamps, or event logs. These help in managing and querying data chronologically in formats like date-only, time-only, or both

Data Type	Description	Example
DATE	Stores date only	'2025-04-08'
TIME	Stores time only	'14:30:00'
DATETIME	Stores date and time	'2025-04-08 14:30:00'
TIMESTAMP	Like DATETIME, but it auto-updates based on the current system time.	'2025-04-08 14:30:00'
YEAR	Stores year (usually 2 or 4 digits)	'2025'

```
CREATE TABLE employees (
  employee_id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
  full_name VARCHAR(150) NOT NULL,
  birth_date DATE NOT NULL,            -- Format: YYYY-MM-DD
  hire_date  DATE NOT NULL
);
```

```
CREATE TABLE store_hours (
  id INT AUTO_INCREMENT PRIMARY KEY,
  day_of_week CHAR(3) NOT NULL,        -- Mon, Tue, Wed...
  open_time TIME,                      -- e.g. '09:30:00'
  close_time TIME                      -- e.g. '18:00:00'
);
```

Boolean Data Type

The **Boolean data** type is used for storing truth values—typically representing yes/no or true/false conditions. It's often used in decision-making columns like “is_active” or “is_verified”. Stores true/false values.

Data Type	Description	Example
BOOLEAN	Stores TRUE or FALSE. Often implemented as 1 (true) or 0 (false).	TRUE / FALSE

```
CREATE TABLE accounts (
  id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
  username VARCHAR(100) NOT NULL,
  is_active BOOLEAN DEFAULT TRUE,      -- Stored as TINYINT(1)
  is_admin  BOOLEAN DEFAULT FALSE
);
```

```
INSERT INTO accounts (username, is_active, is_admin)
VALUES ('JohnDoe', TRUE, FALSE);
```

```
CREATE TABLE system_settings (
  setting_id INT AUTO_INCREMENT PRIMARY KEY,
  feature_name VARCHAR(150) NOT NULL,
  enabled BOOLEAN DEFAULT TRUE
);
```

```
SELECT feature_name
FROM system_settings
WHERE enabled = TRUE;
```

Binary Data Types

The **Binary data** types are designed to store binary data such as images, multimedia files, or encrypted content. They're commonly used when working with non-textual data in applications like file storage or document databases. Store binary data, such as files, images, etc.

Data Type	Description	Use Case
BINARY(n)	Fixed-length binary data	Hashes, file signatures
VARBINARY(n)	Variable-length binary data	Encrypted data
BLOB	Binary Large Object (up to several GBs)	Images, audio, video
TINYBLOB	Small BLOB (max 255 bytes)	Small images or icons
MEDIUMBLOB	Medium BLOB (~16 MB)	Medium files
LONGBLOB	Large BLOB (~4 GB)	High-res images/videos

```
CREATE TABLE file_hashes (
  file_id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
  checksum BINARY(32) NOT NULL        -- e.g. SHA-256 hash
);
```

```
CREATE TABLE secure_tokens (
  token_id INT AUTO_INCREMENT PRIMARY KEY,
  access_token VARBINARY(255) NOT NULL,   -- length varies per token
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

JSON Data Type

JSON documents are stored in MySQL in an internal format that provides for easy access to document components. The JSON binary format is organized in such a manner that the server may quickly search for values within the JSON document by key or array index.

The maximum size of the JSON document is approximately similar to the LONGBLOB or LONGTEXT.

```
CREATE TABLE products (
  id INT PRIMARY KEY AUTO_INCREMENT,
  name VARCHAR(100),
  details JSON                                -- stored in binary JSON format
);
```

```
INSERT INTO products (name, details)
VALUES ('Laptop', JSON_OBJECT('brand','Dell','RAM',16,'SSD',512));
```

```
SELECT details->'$.brand' AS brand,
       details->'$.RAM'   AS ram_size
FROM products;
```

Spatial Data Types

Use one of the many different **spatial data** types that MySQL supports when storing geometric or geographic data. They are utilized to represent information about geometric shapes and physical location.

Spatial Data Types	Description
`GEOMETRY`	A spatial value of any type
`POINT`	A point (a pair of X-Y coordinates)
`LINESTRING`	A curve (one or more `POINT` values)
`POLYGON`	A polygon
`GEOMETRYCOLLECTION`	A collection of `GEOMETRY` values
`MULTILINESTRING`	A collection of `LINESTRING` values
`MULTIPOINT`	A collection of `POINT` values
`MULTIPOLYGON`	A collection of `POLYGON` values

```
CREATE TABLE locations (
  id INT AUTO_INCREMENT PRIMARY KEY,
  name VARCHAR(100),
  position POINT NOT NULL                -- e.g. (40.4168 -3.7038) Madrid
);
```

```
INSERT INTO locations (name, position)
VALUES ('Madrid', ST_PointFromText('POINT(-3.7038 40.4168)'));
```