# MYSQL

**Características principales de MySQL:**

1. **Open Source**: MySQL es de código abierto, lo que significa que es gratuito para usar, modificar y distribuir. Sin embargo, también tiene versiones comerciales con soporte adicional proporcionadas por Oracle Corporation.
2. **Escalabilidad y flexibilidad**: MySQL es muy escalable y puede manejar tanto bases de datos pequeñas para proyectos individuales como bases de datos grandes para aplicaciones empresariales.
3. **Compatibilidad multiplataforma**: MySQL puede ejecutarse en diferentes sistemas operativos, como Windows, Linux, MacOS, y más.
4. **Alta Performance**: Es conocido por su alta velocidad de ejecución y rendimiento, especialmente en operaciones de lectura.
5. **Seguridad**: Ofrece múltiples niveles de seguridad, incluyendo autenticación de usuario, control de acceso basado en privilegios, y conexiones seguras (SSL).
6. **Replica y alta disponibilidad**: Soporta replicación de datos, lo que permite tener copias de la base de datos en diferentes servidores, mejorando la disponibilidad y redundancia de los datos.
7. **Comunidades y soporte**: MySQL tiene una gran comunidad de usuarios y desarrolladores que contribuyen con mejoras, plugins, y soluciones a problemas.
8. **Integración con otras tecnologías**: Es muy compatible e integrable con otros lenguajes de programación y tecnologías como PHP, Python, Java, .NET, y más.

**Usos comunes de MySQL:**

* **Desarrollo Web**: Se usa comúnmente como backend para almacenar datos de aplicaciones web.
* **Aplicaciones de comercio electrónico**: Para gestionar inventarios, transacciones y datos de clientes.
* **Gestión de contenidos**: Utilizado en plataformas CMS como WordPress, Joomla y Drupal.

**TIPOS DE DATOS**

**1. Tipos de Datos Numéricos**

**a. Números Enteros**

* **TINYINT**: Un número entero muy pequeño. Rango: -128 a 127 o 0 a 255 (con UNSIGNED).
* **SMALLINT**: Un número entero pequeño. Rango: -32,768 a 32,767 o 0 a 65,535 (con UNSIGNED).
* **MEDIUMINT**: Un número entero de tamaño medio. Rango: -8,388,608 a 8,388,607 o 0 a 16,777,215 (con UNSIGNED).
* **INT** o **INTEGER**: Un número entero estándar. Rango: -2,147,483,648 a 2,147,483,647 o 0 a 4,294,967,295 (con UNSIGNED).
* **BIGINT**: Un número entero grande. Rango: -9,223,372,036,854,775,808 a 9,223,372,036,854,775,807 o 0 a 18,446,744,073,709,551,615 (con UNSIGNED).

En MySQL, el término **UNSIGNED** se utiliza con los tipos de datos numéricos enteros para indicar que el campo no almacenará números negativos. Por defecto, los tipos de datos numéricos enteros en MySQL pueden almacenar tanto números positivos como negativos (es decir, son *signed*, o con signo). Al declarar un campo como **UNSIGNED**, estás indicando que el rango del campo solo abarcará números positivos.

**¿Qué significa UNSIGNED?**

Cuando un tipo de dato numérico entero es declarado como **UNSIGNED**, se elimina el rango de los números negativos, permitiendo que el rango completo sea utilizado solo para números positivos y aumentando así la capacidad de almacenar números más grandes.

**Ejemplo sin UNSIGNED (con signo):**

* **TINYINT**: Rango de **-128 a 127**
* **INT**: Rango de **-2,147,483,648 a 2,147,483,647**

**Ejemplo con UNSIGNED (sin signo):**

* **TINYINT UNSIGNED**: Rango de **0 a 255**
* **INT UNSIGNED**: Rango de **0 a 4,294,967,295**

Como puedes ver, al especificar UNSIGNED, se duplica el rango positivo del tipo de dato porque se utiliza el mismo número de bits, pero se omite la representación de números negativos.

**Ejemplo en una Tabla**

Supongamos que estamos creando una tabla para almacenar información de usuarios y queremos almacenar la edad de los usuarios. La edad es un número que nunca será negativo, por lo que podemos usar TINYINT UNSIGNED para ahorrar espacio y permitir un rango más amplio en el lado positivo.

**SQL para crear la tabla:**

sql

Copiar código

CREATE TABLE usuarios (

id INT UNSIGNED AUTO\_INCREMENT PRIMARY KEY,

nombre VARCHAR(100),

edad TINYINT UNSIGNED

);

En este ejemplo:

* **id INT UNSIGNED**: El identificador del usuario (id) es un número entero sin signo. Como los identificadores nunca serán negativos, usamos UNSIGNED para permitir valores más grandes, permitiendo hasta 4,294,967,295 en lugar de solo 2,147,483,647.
* **edad TINYINT UNSIGNED**: La edad del usuario es almacenada como un TINYINT UNSIGNED. Dado que la edad no puede ser negativa, usamos UNSIGNED para duplicar el rango positivo disponible, permitiendo valores de 0 a 255.

**Ventajas de Usar UNSIGNED**

1. **Mayor Rango de Valores Positivos**: Al usar UNSIGNED, puedes almacenar valores más grandes en el mismo espacio de almacenamiento. Esto es útil cuando sabes que los valores siempre serán positivos.
2. **Eficiencia de Almacenamiento**: Reducir el espacio de almacenamiento es importante para la optimización del rendimiento de la base de datos, especialmente cuando se trabaja con grandes cantidades de datos.

En resumen, UNSIGNED es una opción útil cuando estás seguro de que tus datos numéricos no necesitan ser negativos y deseas maximizar el rango positivo disponible dentro del mismo tamaño de almacenamiento.

**b. Números Decimales y de Punto Flotante**

* **FLOAT(M,D)**: Un número de coma flotante de precisión simple. M es la cantidad máxima de dígitos y D es la cantidad de dígitos a la derecha del punto decimal. La precisión y el rango dependen de la implementación y el hardware, pero generalmente cubre números en el rango de aproximadamente -3.402823466E+38 a 3.402823466E+38.
* **DOUBLE(M,D)** o **DOUBLE PRECISION**: Un número de coma flotante de doble precisión. Similar a FLOAT pero con mayor precisión. Rango aproximado de -1.7976931348623157E+308 a 1.7976931348623157E+308.
* **DECIMAL(M,D)** o **NUMERIC(M,D)**: Un número decimal exacto. M es el número máximo de dígitos (hasta 65), y D es el número de dígitos a la derecha del punto decimal. Utilizado para almacenar números que requieren una representación exacta, como datos monetarios.

**2. Tipos de Datos de Cadena**

* **CHAR(M)**: Una cadena de longitud fija. M puede ser de 0 a 255. Si la cadena es más corta que M, se rellena con espacios a la derecha. Utilizado para datos de longitud fija, como códigos de país.
* **VARCHAR(M)**: Una cadena de longitud variable. M especifica la longitud máxima de la cadena (hasta 65,535 caracteres). Utilizado para almacenar datos de texto cuya longitud puede variar.
* **TINYTEXT**: Una cadena de texto muy corta. Longitud máxima de 255 caracteres. Utilizado para almacenar pequeñas cantidades de texto.
* **TEXT**: Una cadena de texto. Longitud máxima de 65,535 caracteres. Adecuado para almacenar texto más largo.
* **MEDIUMTEXT**: Una cadena de texto de longitud media. Longitud máxima de 16,777,215 caracteres. Utilizado para grandes cantidades de texto.
* **LONGTEXT**: Una cadena de texto muy larga. Longitud máxima de 4,294,967,295 caracteres. Utilizado para datos de texto muy grandes.
* **BINARY(M)**: Similar a CHAR pero para datos binarios. M representa la longitud fija de los datos almacenados en bytes.
* **VARBINARY(M)**: Similar a VARCHAR pero para datos binarios. M representa la longitud máxima de los datos almacenados en bytes.

**3. Tipos de Datos de Fecha y Hora**

* **DATE**: Almacena una fecha en formato 'YYYY-MM-DD'. Rango: '1000-01-01' a '9999-12-31'.
* **DATETIME**: Almacena una fecha y hora en formato 'YYYY-MM-DD HH:MM

'. Rango: '1000-01-01 00:00:00' a '9999-12-31 23:59:59'. No depende del huso horario.

* **TIMESTAMP**: Similar a DATETIME, pero depende del huso horario. Se utiliza comúnmente para registrar el momento de creación o actualización de filas. Rango: '1970-01-01 00:00:01' UTC a '2038-01-19 03:14:07' UTC.
* **TIME**: Almacena una hora en formato 'HH:MM

'. Rango: '-838:59:59' a '838:59:59'. Utilizado para almacenar intervalos de tiempo.

* **YEAR**: Almacena un año en formato de dos o cuatro dígitos. Rango: 1901 a 2155 (cuatro dígitos) o 70 a 69 (dos dígitos, que representan 1970 a 2069).

**4. Tipos de Datos de Valor Booleano**

* **BOOLEAN** o **BOOL**: Este tipo de dato es un alias para TINYINT(1) en MySQL. Generalmente, el valor 0 representa FALSE y cualquier otro valor representa TRUE.

**5. Tipos de Datos Espaciales (Geoespaciales)**

* **GEOMETRY**: Almacena datos espaciales de geometría de cualquier tipo.
* **POINT**: Representa un punto en un espacio de coordenadas 2D.
* **LINESTRING**: Representa una línea compuesta por dos o más puntos.
* **POLYGON**: Representa un polígono en 2D definido por un anillo cerrado de puntos.
* **MULTIPOINT**, **MULTILINESTRING**, **MULTIPOLYGON**, **GEOMETRYCOLLECTION**: Representan colecciones de objetos espaciales de su respectivo tipo.

**6. Tipos de Datos JSON**

* **JSON**: Utilizado para almacenar datos en formato JSON (JavaScript Object Notation). Este tipo de dato permite almacenar objetos JSON completos y es útil para aplicaciones que manejan datos semi-estructurados.

**7. Tipos de Datos de Valores Binarios Grandes (BLOB)**

* **TINYBLOB**: Almacena un pequeño valor binario de hasta 255 bytes.
* **BLOB**: Almacena un valor binario mediano de hasta 65,535 bytes.
* **MEDIUMBLOB**: Almacena un valor binario más grande de hasta 16,777,215 bytes.
* **LONGBLOB**: Almacena un valor binario muy grande de hasta 4,294,967,295 bytes.

**Resumen**

Cada tipo de dato en MySQL tiene su propósito y es importante elegir el tipo de dato adecuado para la eficiencia del almacenamiento y el rendimiento de la base de datos. Por ejemplo, los tipos INT y VARCHAR son muy utilizados en aplicaciones comunes, mientras que los tipos TEXT y BLOB son adecuados para almacenar grandes cantidades de datos. Los tipos de datos de fecha y hora son cruciales para aplicaciones que requieren un seguimiento preciso del tiempo, y los tipos de datos JSON y espaciales se utilizan en aplicaciones modernas que manejan datos semi-estructurados y geoespaciales.

Ejemplos de datos:

### 1. Tipos de Datos Numéricos

#### a. ****Números Enteros****

* **TINYINT**: Usado para almacenar un valor pequeño, como un indicador de estado.

sql

Copiar código

CREATE TABLE productos (

id INT UNSIGNED AUTO\_INCREMENT PRIMARY KEY,

nombre VARCHAR(100),

stock TINYINT UNSIGNED -- Almacena valores de 0 a 255

);

* **SMALLINT**: Útil para almacenar números más grandes que TINYINT pero que aún no necesitan tanto espacio como INT.

sql

Copiar código

CREATE TABLE empleados (

id INT UNSIGNED AUTO\_INCREMENT PRIMARY KEY,

nombre VARCHAR(100),

edad SMALLINT UNSIGNED -- Almacena edades de 0 a 65,535

);

* **MEDIUMINT**: Ideal para números intermedios, como los registros de puntuaciones en un sistema de juego.

sql

Copiar código

CREATE TABLE puntuaciones (

id INT UNSIGNED AUTO\_INCREMENT PRIMARY KEY,

jugador VARCHAR(100),

puntos MEDIUMINT UNSIGNED -- Almacena puntos de 0 a 16,777,215

);

* **INT**: El tipo más común para números enteros, usado para identificadores y valores que pueden ser bastante grandes.

sql

Copiar código

CREATE TABLE usuarios (

id INT UNSIGNED AUTO\_INCREMENT PRIMARY KEY,

nombre VARCHAR(100),

saldo INT -- Almacena un saldo positivo o negativo

);

* **BIGINT**: Útil para números extremadamente grandes, como un identificador único global o un registro de operaciones financieras.

sql

Copiar código

CREATE TABLE transacciones (

id BIGINT UNSIGNED AUTO\_INCREMENT PRIMARY KEY,

usuario\_id INT UNSIGNED,

monto BIGINT -- Almacena montos grandes, como en operaciones de criptomonedas

);

#### b. ****Números Decimales y de Punto Flotante****

* **FLOAT**: Usado para almacenar números con decimales, como calificaciones o porcentajes.

sql

Copiar código

CREATE TABLE calificaciones (

id INT UNSIGNED AUTO\_INCREMENT PRIMARY KEY,

estudiante VARCHAR(100),

promedio FLOAT(5,2) -- Almacena un promedio como 95.75

);

* **DOUBLE**: Utilizado para almacenar números de punto flotante de alta precisión, como mediciones científicas.

sql

Copiar código

CREATE TABLE mediciones (

id INT UNSIGNED AUTO\_INCREMENT PRIMARY KEY,

valor DOUBLE -- Almacena un número de alta precisión

);

* **DECIMAL**: Ideal para almacenar datos financieros donde la precisión es crucial.

sql

Copiar código

CREATE TABLE cuentas (

id INT UNSIGNED AUTO\_INCREMENT PRIMARY KEY,

balance DECIMAL(10, 2) -- Almacena un balance monetario con precisión de centavos

);

### 2. Tipos de Datos de Cadena

* **CHAR**: Usado para almacenar datos de longitud fija, como códigos de país.

sql

Copiar código

CREATE TABLE paises (

id INT UNSIGNED AUTO\_INCREMENT PRIMARY KEY,

codigo CHAR(2), -- Almacena un código de país de 2 caracteres, como 'US'

nombre VARCHAR(100)

);

* **VARCHAR**: Utilizado para almacenar cadenas de texto de longitud variable, como nombres o direcciones.

sql

Copiar código

CREATE TABLE clientes (

id INT UNSIGNED AUTO\_INCREMENT PRIMARY KEY,

nombre VARCHAR(100), -- Almacena nombres de hasta 100 caracteres

direccion VARCHAR(255) -- Almacena direcciones de hasta 255 caracteres

);

* **TINYTEXT**: Para almacenar pequeñas cantidades de texto.

sql

Copiar código

CREATE TABLE comentarios (

id INT UNSIGNED AUTO\_INCREMENT PRIMARY KEY,

comentario TINYTEXT -- Almacena un pequeño comentario de hasta 255 caracteres

);

* **TEXT**: Usado para almacenar texto más largo, como descripciones de productos.

sql

Copiar código

CREATE TABLE articulos (

id INT UNSIGNED AUTO\_INCREMENT PRIMARY KEY,

titulo VARCHAR(100),

contenido TEXT -- Almacena un artículo de texto completo

);

* **MEDIUMTEXT**: Adecuado para almacenar grandes cantidades de texto, como un cuerpo de blog extenso.

sql

Copiar código

CREATE TABLE posts (

id INT UNSIGNED AUTO\_INCREMENT PRIMARY KEY,

titulo VARCHAR(100),

contenido MEDIUMTEXT -- Almacena contenido de blog extenso

);

* **LONGTEXT**: Para almacenar grandes bloques de texto, como documentos legales completos.

sql

Copiar código

CREATE TABLE documentos\_legales (

id INT UNSIGNED AUTO\_INCREMENT PRIMARY KEY,

titulo VARCHAR(100),

documento LONGTEXT -- Almacena un documento legal completo

);

* **BINARY**: Similar a CHAR pero para datos binarios de longitud fija.

sql

Copiar código

CREATE TABLE archivos (

id INT UNSIGNED AUTO\_INCREMENT PRIMARY KEY,

archivo BINARY(16) -- Almacena datos binarios de longitud fija, como un hash MD5

);

* **VARBINARY**: Similar a VARCHAR pero para datos binarios de longitud variable.

sql

Copiar código

CREATE TABLE imagenes (

id INT UNSIGNED AUTO\_INCREMENT PRIMARY KEY,

datos VARBINARY(255) -- Almacena datos de imagen binarios

);

### 3. Tipos de Datos de Fecha y Hora

* **DATE**: Para almacenar solo fechas, como fechas de nacimiento.

sql

Copiar código

CREATE TABLE empleados (

id INT UNSIGNED AUTO\_INCREMENT PRIMARY KEY,

nombre VARCHAR(100),

fecha\_nacimiento DATE -- Almacena fechas de nacimiento

);

* **DATETIME**: Para almacenar fechas y horas completas, como registros de eventos.

sql

Copiar código

CREATE TABLE eventos (

id INT UNSIGNED AUTO\_INCREMENT PRIMARY KEY,

titulo VARCHAR(100),

fecha\_evento DATETIME -- Almacena la fecha y hora de un evento

);

* **TIMESTAMP**: Similar a DATETIME, pero con una diferencia en cómo se manejan los husos horarios.

sql

Copiar código

CREATE TABLE logs (

id INT UNSIGNED AUTO\_INCREMENT PRIMARY KEY,

mensaje VARCHAR(255),

creado\_en TIMESTAMP DEFAULT CURRENT\_TIMESTAMP -- Almacena la fecha y hora de creación automáticamente

);

* **TIME**: Para almacenar solo la hora, como horas de apertura.

sql

Copiar código

CREATE TABLE horarios (

id INT UNSIGNED AUTO\_INCREMENT PRIMARY KEY,

apertura TIME, -- Almacena la hora de apertura, como '09:00:00'

cierre TIME -- Almacena la hora de cierre, como '17:00:00'

);

* **YEAR**: Para almacenar solo el año, como el año de fabricación de un vehículo.

sql

Copiar código

CREATE TABLE vehiculos (

id INT UNSIGNED AUTO\_INCREMENT PRIMARY KEY,

marca VARCHAR(100),

modelo VARCHAR(100),

ano YEAR -- Almacena el año de fabricación, como '2022'

);

### 4. Tipos de Datos de Valor Booleano

* **BOOLEAN**: Para almacenar valores de verdadero/falso, como una bandera de activo.

sql

Copiar código

CREATE TABLE usuarios (

id INT UNSIGNED AUTO\_INCREMENT PRIMARY KEY,

nombre VARCHAR(100),

activo BOOLEAN -- Almacena un valor booleano, 1 para verdadero, 0 para falso

);

### 5. Tipos de Datos Espaciales (Geoespaciales)

* **POINT**: Para almacenar una ubicación geográfica.

sql

Copiar código

CREATE TABLE ubicaciones (

id INT UNSIGNED AUTO\_INCREMENT PRIMARY KEY,

nombre VARCHAR(100),

coordenadas POINT -- Almacena coordenadas geográficas

);

### 6. Tipos de Datos JSON

* **JSON**: Para almacenar datos en formato JSON, ideal para datos semi-estructurados.

sql

Copiar código

CREATE TABLE configuraciones (

id INT UNSIGNED AUTO\_INCREMENT PRIMARY KEY,

usuario\_id INT UNSIGNED,

preferencias JSON -- Almacena las preferencias del usuario en formato JSON

);

### 7. Tipos de Datos de Valores Binarios Grandes (BLOB)

* **TINYBLOB**: Para almacenar pequeñas cantidades de datos binarios.

sql

Copiar código

CREATE TABLE archivos\_pequenos (

id INT UNSIGNED AUTO\_INCREMENT PRIMARY KEY,

datos TINYBLOB -- Almacena datos binarios pequeños

);

* **BLOB**: Para almacenar datos binarios de tamaño mediano, como documentos.

sql

Copiar código

CREATE TABLE documentos (

id INT UNSIGNED AUTO\_INCREMENT PRIMARY KEY,

titulo VARCHAR(100),

archivo BLOB -- Almacena un archivo binario de tamaño mediano

);

* **MEDIUMBLOB**: Para almacenar grandes archivos binarios, como videos.

sql

Copiar código

CREATE TABLE videos (

id INT UNSIGNED AUTO\_INCREMENT PRIMARY KEY,

titulo VARCHAR(100),

archivo MEDIUMBLOB -- Almacena un archivo de video grande

);

* **LONGBLOB**: Para almacenar archivos binarios muy grandes, como películas en alta definición.

sql

Copiar código

CREATE TABLE peliculas (

id INT UNSIGNED AUTO\_INCREMENT PRIMARY KEY,

titulo VARCHAR(100),

archivo LONGBLOB -- Almacena un archivo binario muy grande, como una película en HD

);

Estos ejemplos ilustran cómo se pueden usar los diferentes tipos de datos en MySQL dependiendo del tipo de información que deseas almacenar. El tipo de datos adecuado para cada columna dependerá de las necesidades específicas de la aplicación y del tipo de datos que se almacenarán.