Manual Detallado de MySQL y SQL: De Cero a Experto

Autor: Manus AI

1. Introducción a las Bases de Datos y SQL

1.1. ¿Qué es una Base de Datos?

Una **Base de Datos (BD)** es un conjunto organizado de datos relacionados que se almacena y se accede electrónicamente. Permite almacenar, gestionar y recuperar información de manera eficiente.

- **DBMS (Database Management System):** Es el software que interactúa con el usuario, las aplicaciones y la base de datos en sí para capturar y analizar los datos. Ejemplos: MySQL, PostgreSQL, Oracle.
- **RDBMS (Relational Database Management System):** Un tipo de DBMS que almacena los datos en forma de tablas, donde los datos se relacionan entre sí. MySQL es un RDBMS.

1.2. ¿Qué es SQL?

SQL (Structured Query Language) es el lenguaje estándar para gestionar y manipular bases de datos relacionales. Permite a los usuarios interactuar con los datos mediante comandos sencillos.

El lenguaje SQL se divide en varios subconjuntos:

Subconjunto	Abreviatura	Propósito Principal	Ejemplos de Comandos
Lenguaje de Definición de Datos	**DDL**	Definir la estructura de la base de datos (tablas, esquemas).	`CREATE`, `ALTER`, `DROP`
Lenguaje de Manipulación de Datos	**DML**	Manipular los datos dentro de las tablas (consultar, insertar, modificar).	`SELECT`, `INSERT`, `UPDATE`, `DELETE`
Lenguaje de Control de Datos	**DCL**	Controlar el acceso a los datos (permisos y seguridad).	`GRANT`, `REVOKE`
Lenguaje de Control de Transacciones	**TCL**	Gestionar las transacciones y asegurar la integridad de los datos.	`COMMIT`, `ROLLBACK`, `SAVEPOINT`

1.3. Instalación y Configuración de MySQL

Para empezar, necesitará instalar el servidor MySQL y el cliente de línea de comandos. El proceso varía según el sistema operativo, pero la instalación suele incluir el servidor, el cliente y herramientas de desarrollo.

2. Administración de MySQL desde la Terminal

Esta sección detalla los comandos esenciales para interactuar con su servidor MySQL directamente desde la línea de comandos (terminal o consola), lo cual es fundamental para la administración y el desarrollo.

2.1. Acceso a la Línea de Comandos de MySQL

El cliente de MySQL se utiliza para conectarse al servidor.

Sintaxis básica de conexión:

```
mysql -u [usuario] -p
```

Ejemplo práctico:

```
# Conexión como el usuario 'root'
mysql -u root -p
# El sistema le pedirá la contraseña.
```

Una vez dentro, el prompt cambiará a `mysql>`.

2.2. Gestión de Bases de Datos

Tarea	Comando SQL	Ejemplo Práctico
Ver bases de datos existentes	`SHOW DATABASES;`	`mysql> SHOW DATABASES;`
Crear una nueva base de datos	`CREATE DATABASE [nombre_bd];`	`mysql> CREATE DATABASE biblioteca;`
Seleccionar una base de datos	`USE [nombre_bd];`	`mysql> USE biblioteca;`
Eliminar una base de datos	`DROP DATABASE [nombre_bd];`	`mysql> DROP DATABASE biblioteca;`

2.3. Gestión de Usuarios y Permisos (DCL)

La seguridad es clave. Es esencial crear usuarios específicos con los permisos mínimos necesarios.

Crear un nuevo usuario:

```
-- Sintaxis para crear un usuario y asignarle una contraseña
CREATE USER '[nombre_usuario]'@'[host]' IDENTIFIED BY '[contraseña]';
-- host puede ser 'localhost' (solo acceso local) o '%' (acceso desde
cualquier host)
```

Ejemplo práctico de creación de usuario:

```
mysql> CREATE USER 'dev_user'@'localhost' IDENTIFIED BY 'MiPasswordSegura';
```

Asignar permisos (`GRANT`):

El comando `GRANT` se usa para otorgar privilegios a un usuario sobre una base de datos específica.

```
-- Sintaxis para otorgar todos los privilegios sobre una BD
GRANT ALL PRIVILEGES ON [nombre_bd].* TO '[nombre_usuario]'@'[host]';
-- Para que los cambios surtan efecto, es necesario recargar los privilegios
FLUSH PRIVILEGES;
```

Ejemplo práctico de asignación de permisos:

```
mysql> GRANT SELECT, INSERT, UPDATE, DELETE ON biblioteca.* TO
'dev_user'@'localhost';
mysql> FLUSH PRIVILEGES;
```

Eliminar permisos (`REVOKE`) y usuarios:

```
mysql> REVOKE ALL PRIVILEGES ON biblioteca.* FROM 'dev_user'@'localhost';
mysql> DROP USER 'dev_user'@'localhost';
```

2.4. Salir de la Terminal de MySQL

Simplemente use el comando `EXIT` o `QUIT` .

```
mysql> EXIT;
```

3. SQL Básico: Lenguaje de Definición de Datos (DDL)

El DDL se utiliza para definir la estructura de la base de datos, es decir, las tablas y sus columnas.

3.1. Creación de Tablas (`CREATE TABLE`)

Las tablas son la estructura fundamental donde se almacenan los datos.

Sintaxis básica:

```
CREATE TABLE [nombre_tabla] (
    [nombre_columna1] [tipo_dato] [restricciones],
    [nombre_columna2] [tipo_dato] [restricciones],
    ...
    [definicion_claves]
);
```

Ejemplo práctico:

```
USE biblioteca;

CREATE TABLE libros (
    libro_id INT PRIMARY KEY AUTO_INCREMENT,
    titulo VARCHAR(255) NOT NULL,
    autor VARCHAR(100) NOT NULL,
    anio_publicacion YEAR,
    isbn VARCHAR(20) UNIQUE
);
```

3.2. Tipos de Datos en MySQL

Elegir el tipo de dato correcto es crucial para la eficiencia y el almacenamiento.

Categoría	Tipo de Dato	Descripción
Numéricos	`INT`, `TINYINT`, `DECIMAL(p,s)`, `FLOAT`	Números enteros, pequeños enteros, números con punto decimal fijo o flotante.
Cadenas	`VARCHAR(L)`, `CHAR(L)`, `TEXT`	Cadenas de longitud variable (L = longitud máxima), longitud fija, o texto largo.
Fecha y Hora	`DATE`, `TIME`, `DATETIME`, `TIMESTAMP`	Fecha (YYYY-MM-DD), Hora (HH:MM:SS), Fecha y Hora, Marca de tiempo.
Booleanos	`BOOLEAN`	Sin tipo booleano nativo; se usa `TINYINT(1)` (0 para falso, 1 para verdadero).

3.3. Restricciones (Constraints)

Aseguran la integridad de los datos.

- `PRIMARY KEY`: Identificador único para cada fila. No puede ser `NULL`.
- `NOT NULL`: Asegura que una columna debe contener un valor.
- `UNIQUE`: Asegura que todos los valores en una columna son diferentes.
- `FOREIGN KEY`: Enlaza dos tablas, asegurando la integridad referencial.
- `DEFAULT`: Asigna un valor por defecto si no se especifica uno.

3.4. Modificación y Eliminación de Estructuras (`ALTER` y `DROP`)

Modificar la estructura de una tabla (`ALTER TABLE`):

```sql
-- Añadir una nueva columna
ALTER TABLE libros ADD COLUMN editorial VARCHAR(100);

-- Modificar el tipo de dato de una columna
ALTER TABLE libros MODIFY COLUMN anio_publicacion INT;

-- Eliminar una columna
ALTER TABLE libros DROP COLUMN isbn;
```

Eliminar una tabla o base de datos (`DROP`):

```
DROP TABLE libros;
DROP DATABASE biblioteca;
```

4. SQL Intermedio: Lenguaje de Manipulación de Datos (DML)

El DML se utiliza para gestionar los datos *dentro* de las tablas.

4.1. Inserción de Datos (`INSERT INTO`)

```
-- Sintaxis básica
INSERT INTO [nombre_tabla] ([columna1], [columna2], ...)
VALUES ([valor1], [valor2], ...);

-- Ejemplo práctico
INSERT INTO libros (titulo, autor, anio_publicacion)
VALUES ('Cien años de soledad', 'Gabriel García Márquez', 1967);
```

4.2. Consulta de Datos (`SELECT`)

El comando `SELECT` es el más usado y permite recuperar datos.

Sintaxis básica:

```
SELECT [columnas] FROM [tabla] [cláusulas_opcionales];
```

Ejemplos prácticos:

Consulta	Comando SQL	Descripción
Seleccionar todas las columnas	`SELECT * FROM libros;`	Muestra todos los datos de la tabla `libros`.
Seleccionar columnas específicas	`SELECT titulo, autor FROM libros;`	Muestra solo el título y el autor.
Filtrar con `WHERE`	`SELECT * FROM libros WHERE anio_publicacion > 2000;`	Libros publicados después del año 2000.
Ordenar con `ORDER BY`	`SELECT * FROM libros ORDER BY autor ASC;`	Ordena por autor de forma ascendente.
Limitar resultados	`SELECT * FROM libros LIMIT 10;`	Muestra solo los primeros 10 resultados.

4.3. Actualización de Datos (`UPDATE`)

Se utiliza para modificar datos existentes en una o más filas. **¡La cláusula `WHERE` es crítica!** Sin ella, se actualizarán *todas* las filas.

```
-- Sintaxis
UPDATE [nombre_tabla]
SET [columna1] = [nuevo_valor1], [columna2] = [nuevo_valor2]
WHERE [condición];

-- Ejemplo práctico
UPDATE libros
SET anio_publicacion = 1982
WHERE titulo = 'Cien años de soledad';
```

4.4. Eliminación de Datos (`DELETE FROM`)

Se utiliza para eliminar filas de una tabla. **¡La cláusula `WHERE` es crítica!** Sin ella, se eliminarán *todas* las filas.

```sql
-- Sintaxis
DELETE FROM [nombre_tabla] WHERE [condición];

-- Ejemplo práctico
DELETE FROM libros WHERE libro_id = 5;
```

5. SQL Avanzado: Consultas Complejas y Funciones

5.1. Uniones de Tablas (`JOIN`s)

Los `JOIN`s permiten combinar filas de dos o más tablas basándose en una columna relacionada entre ellas (generalmente una clave foránea).

Para los ejemplos, asumimos una tabla `autores` (`autor_id`, `nombre`) y la tabla `libros` (`libro_id`, `titulo`, `autor_id_fk`).

Tipo de JOIN	Descripción	Ejemplo Práctico
INNER JOIN	Devuelve solo las filas que tienen valores coincidentes en ambas tablas.	`SELECT l.titulo, a.nombre FROM libros l INNER JOIN autores a ON l.autor_id_fk = a.autor_id;`
LEFT JOIN	Devuelve todas las filas de la tabla izquierda, y las filas coincidentes de la tabla derecha. Si no hay coincidencia, devuelve `NULL` para las columnas de la derecha.	`SELECT * FROM autores a LEFT JOIN libros l ON a.autor_id = l.autor_id_fk;`
RIGHT JOIN	Devuelve todas las filas de la tabla derecha, y las filas coincidentes de la tabla izquierda. (Menos usado, ya que un LEFT JOIN con las tablas invertidas logra lo mismo).	
FULL JOIN	Devuelve filas cuando hay una coincidencia en una de las tablas. (En MySQL se simula con `UNION` de `LEFT JOIN` y `RIGHT JOIN`).	

5.2. Agregación y Agrupación

Las funciones de agregación realizan un cálculo sobre un conjunto de filas y devuelven un único valor.

Función de Agregación	Propósito
`COUNT()`	Cuenta el número de filas.
`SUM()`	Calcula la suma total de una columna numérica.
`AVG()`	Calcula el valor promedio de una columna numérica.
`MAX()`	Encuentra el valor máximo de una columna.
`MIN()`	Encuentra el valor mínimo de una columna.

Agrupación (`GROUP BY`):

Se utiliza para agrupar filas que tienen los mismos valores en columnas especificadas en un conjunto de filas resumidas.

Ejemplo práctico:

```
-- Contar cuántos libros tiene cada autor
SELECT autor_id_fk, COUNT(libro_id) AS total_libros
FROM libros
GROUP BY autor_id_fk;
```

Filtrado de Grupos (`HAVING`):

La cláusula `HAVING` se utiliza para filtrar los resultados de la agrupación. Es el equivalente de `WHERE` para los grupos.

```
-- Mostrar solo los autores que tienen más de 5 libros
SELECT autor_id_fk, COUNT(libro_id) AS total_libros
FROM libros
GROUP BY autor_id_fk
HAVING total_libros > 5;
```

5.3. Subconsultas (Subqueries)

Una subconsulta es una consulta anidada dentro de otra consulta SQL.

Ejemplo práctico:

```sql
-- Encontrar el título del libro más antiguo
SELECT titulo
FROM libros
WHERE anio_publicacion = (
    SELECT MIN(anio_publicacion) FROM libros
);
```

6. MySQL Específico: Administración y Programación

6.1. Vistas (`CREATE VIEW`)

Una **Vista** es una tabla virtual basada en el conjunto de resultados de una consulta SQL. No almacena datos físicamente, sino que simplifica consultas complejas y mejora la seguridad.

```sql
CREATE VIEW vista_libros_modernos AS
SELECT titulo, autor, anio_publicacion
FROM libros
WHERE anio_publicacion > 2010;

-- Usar la vista es como usar una tabla normal
SELECT * FROM vista_libros_modernos;
```

6.2. Índices

Los índices son estructuras que mejoran la velocidad de las operaciones de recuperación de datos. Son cruciales para la optimización del rendimiento.

```sql
-- Crear un índice simple
CREATE INDEX idx_autor ON libros (autor);

-- Mostrar índices
SHOW INDEX FROM libros;
```

6.3. Procedimientos Almacenados (Stored Procedures)

Un **Procedimiento Almacenado** es un conjunto de sentencias SQL que se almacenan en la base de datos y se pueden ejecutar con una sola llamada. Mejoran el rendimiento y la modularidad.

```sql
DELIMITER //

CREATE PROCEDURE ObtenerLibrosPorAutor (IN autor_nombre VARCHAR(100))
BEGIN
    SELECT titulo, anio_publicacion
    FROM libros
    WHERE autor = autor_nombre;
END //

DELIMITER ;

-- Ejecutar el procedimiento
CALL ObtenerLibrosPorAutor('Gabriel García Márquez');
```

6.4. Triggers

Un **Trigger** es un procedimiento almacenado que se ejecuta automáticamente (se "dispara") cuando ocurre un evento específico (INSERT, UPDATE o DELETE) en una tabla.

```
DELIMITER //

CREATE TRIGGER antes_insertar_libro
BEFORE INSERT ON libros
FOR EACH ROW
BEGIN
    -- Asegura que el título se guarde en mayúsculas
    SET NEW.titulo = UPPER(NEW.titulo);
END //

DELIMITER ;
```

6.5. Manejo de Transacciones (TCL)

Una **Transacción** es una secuencia de una o más operaciones SQL que se ejecutan como una única unidad lógica de trabajo. Siguen las propiedades **ACID** (Atomicidad, Consistencia, Aislamiento, Durabilidad).

```
-- Iniciar el modo de transacción (si no está activado por defecto)
START TRANSACTION;

-- Operación 1: Descontar stock
UPDATE productos SET stock = stock - 1 WHERE producto_id = 101;

-- Operación 2: Registrar la venta
INSERT INTO ventas (producto_id, cantidad) VALUES (101, 1);

-- Si todo salió bien, confirmar los cambios
COMMIT;

-- Si algo falló, deshacer todos los cambios desde START TRANSACTION
-- ROLLBACK;
```

7. Ejemplos Prácticos y Casos de Estudio

Caso de Estudio: Sistema de Gestión de Pedidos

Objetivo: Diseñar un esquema de base de datos para una tienda online sencilla y realizar consultas complejas.

Esquema de Tablas:

1. **Clientes** (`cliente_id` PK, `nombre`, `email`)
2. **Productos** (`producto_id` PK, `nombre_producto`, `precio`, `stock`)
3. **Pedidos** (`pedido_id` PK, `cliente_id_fk`, `fecha_pedido`)
4. **Detalle_Pedido** (`detalle_id` PK, `pedido_id_fk`, `producto_id_fk`, `cantidad`, `precio_unitario`)

Ejercicios de Consulta Resueltos:

Ejercicio 1: Obtener el valor total de cada pedido.

```
SELECT
    p.pedido_id,
    c.nombre AS nombre_cliente,
    SUM(dp.cantidad * dp.precio_unitario) AS total_pedido
FROM
    Pedidos p
INNER JOIN
    Clientes c ON p.cliente_id_fk = c.cliente_id
INNER JOIN
    Detalle_Pedido dp ON p.pedido_id = dp.pedido_id_fk
GROUP BY
    p.pedido_id, c.nombre
ORDER BY
    total_pedido DESC;
```

Ejercicio 2: Encontrar los clientes que nunca han realizado un pedido.

```sql
SELECT
    c.nombre, c.email
FROM
    Clientes c
LEFT JOIN
    Pedidos p ON c.cliente_id = p.cliente_id_fk
WHERE
    p.pedido_id IS NULL;
```

Ejercicio 3: Mostrar los 5 productos más vendidos (por cantidad).

```sql
SELECT
    pr.nombre_producto,
    SUM(dp.cantidad) AS total_cantidad_vendida
FROM
    Detalle_Pedido dp
INNER JOIN
    Productos pr ON dp.producto_id_fk = pr.producto_id
GROUP BY
    pr.nombre_producto
ORDER BY
    total_cantidad_vendida DESC
LIMIT 5;
```

8. Herramientas Recomendadas

Aunque la terminal es poderosa, las herramientas con interfaz gráfica (GUI) facilitan la administración y el desarrollo.

Herramienta	Tipo	Descripción
MySQL Workbench	Cliente Oficial GUI	Herramienta oficial de Oracle. Ofrece diseño de bases de datos visual, desarrollo de SQL, y administración del servidor.
DBeaver	Cliente Universal GUI	Cliente de base de datos universal que soporta MySQL, PostgreSQL, Oracle, y muchos más. Excelente para trabajar con múltiples bases de datos.
DataGrip	IDE (JetBrains)	Un IDE de base de datos profesional y potente para desarrolladores. Ofrece autocompletado avanzado y refactorización. (De pago)
phpMyAdmin	Herramienta Web	Una interfaz web popular para la administración de MySQL. Ideal para entornos de alojamiento compartido.
HeidiSQL	Cliente GUI (Windows)	Un cliente ligero y rápido, muy popular entre los usuarios de Windows.
Visual Studio Code	Editor de Código	Con extensiones como *SQLTools*, puede conectarse y ejecutar consultas directamente desde su editor de código.

FIN DEL MANUAL