Universidad de los Andes | Educación Continua
Vicerrectoría Académica

Introducción al Backend y Arquitectura Cliente-Servidor

Curso: Desarrollo Backend con Django

SQL Básico: SELECT, INSERT, UPDATE y DELETE

¿Qué aprenderemos hoy?

1

Importar y exportar datos

Importación y exportación con MySQL Wokbench

2

Comandos principales

SELECT, INSERT, UPDATE y DELETE en detalle

3

Relaciones entre tablas

Tipos de JOINS con ejemplos prácticos

4

Caso práctico

Sistema de gestión de biblioteca con MySQL Workbench

Nuestro escenario práctico: Sistema de Biblioteca

Tabla Libros

id, titulo, autor, anio_publicacion

Tabla Usuarios

id, nombre, correo, telefono

Tabla Préstamos

id, id_usuario, id_libro, fecha_prestamo, fecha_devolucion

Trabajaremos con estas tres tablas relacionadas para gestionar un sistema de préstamos de libros, aplicando los comandos SQL que aprenderemos.

Importar y Exportar Datos en MySQL Workbench

Dominar la importación y exportación de datos es crucial para gestionar eficientemente grandes volúmenes de información, garantizar su respaldo y facilitar la migración entre diferentes entornos de base de datos.

Importar datos desde archivos CSV

Importar Datos desde CSV

01

Iniciar Asistente

En "Navigator", haz clic derecho en la tabla de destino y selecciona `Table Data Import Wizard`.

02

Seleccionar Archivo

Haz clic en `Browse` para elegir tu archivo CSV. Confirma la tabla de destino y pulsa `Next`.

03

Configurar Opciones

Ajusta la asignación de columnas, tipo de datos, codificación (encoding) y delimitador si es necesario. Pulsa `Next`.

04

Revisar y Ejecutar

Revisa el resumen y haz clic en `Next`, luego `Execute` para iniciar la importación. Verifica los resultados.

Exportar datos a archivos CSV

01

Iniciar Asistente

Haz clic derecho en la tabla que deseas exportar y selecciona `Table Data Export Wizard`.

02

Elegir Destino

Selecciona la ruta donde se guardará el archivo y asegúrate de que el formato sea CSV.

03

Configurar Columnas

Elige las columnas que deseas incluir en la exportación y ajusta las opciones de delimitador si es necesario.

04

Revisar y Exportar

Revisa el resumen de la configuración y haz clic en "Export" o "Next" para iniciar el proceso.

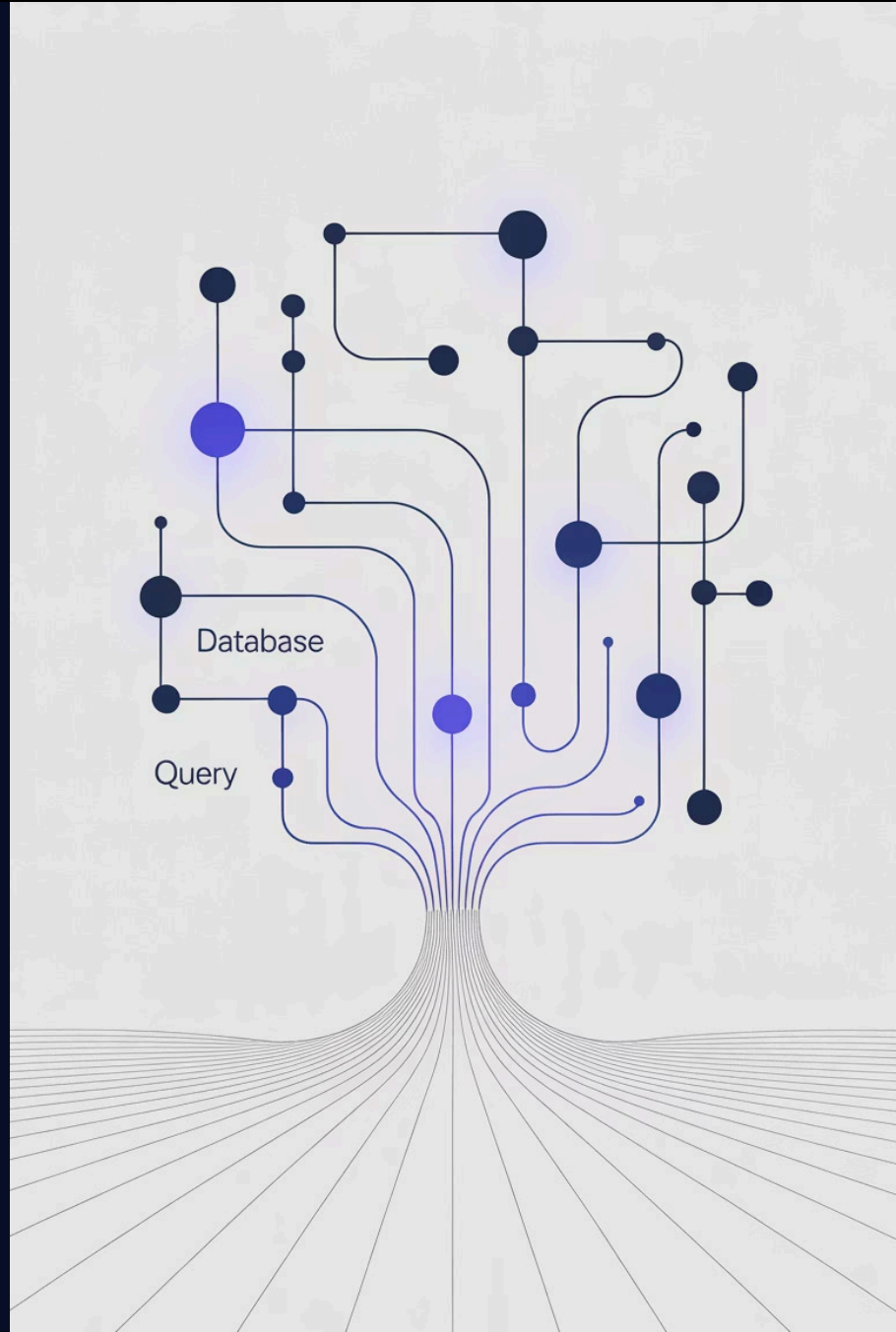
¡Listo! Ahora a practicar SQL

Con la base de datos configurada y con datos, ya están listos para comenzar con los comandos SQL.

Comando SELECT: La base de las consultas

El comando **SELECT** es, sin duda, el más fundamental y utilizado en SQL. Es la herramienta principal para recuperar datos de una o varias tablas en una base de datos.

Desde la consulta más simple para ver todos los registros, hasta las más complejas que combinan, filtran y agregan información, todo comienza con **SELECT**. Entenderlo a fondo es crucial para interactuar eficientemente con cualquier base de datos relacional.



Comando SELECT: La base de las consultas

El comando SELECT es la piedra angular de SQL y nos permite extraer información de las bases de datos para su análisis.

Con él podemos:

- Recuperar datos específicos de una o varias tablas
- Filtrar resultados según condiciones
- Ordenar la información obtenida
- Realizar cálculos y obtener estadísticas

Sintaxis básica de SELECT

```
SELECT columna1, columna2, ...
FROM nombre_tabla
WHERE condicion
ORDER BY columna [ASC|DESC];
```

Donde:

- **SELECT:** Define qué columnas queremos ver
- **FROM:** Especifica de qué tabla obtendremos los datos
- **WHERE:** Establece condiciones para filtrar los resultados
- **ORDER BY:** Ordena los resultados según una o varias columnas

Ejemplo básico: Consultar todos los libros

```
SELECT *
FROM libros;
```

El asterisco (*) indica que queremos todas las columnas de la tabla Libros.

Esta consulta nos devolverá todos los registros con todos los campos de la tabla Libros.

SELECT con columnas específicas

```
SELECT titulo, autor
FROM libros;
```

Con esta consulta obtenemos únicamente el título y el autor de cada libro, ignorando el resto de columnas como id y año de publicación.

Es una buena práctica seleccionar solo las columnas que necesitamos en lugar de usar SELECT *, especialmente cuando trabajamos con tablas grandes.

Filtrando con WHERE

```
SELECT titulo, autor, anio_publicacion
FROM libros
WHERE anio_publicacion > 2000;
```

Esta consulta nos muestra solo los libros publicados después del año 2000.

Operadores comunes en WHERE:

- `=` (igual a)
- `>` (mayor que)
- `<` (menor que)
- `>=` (mayor o igual que)
- `<=` (menor o igual que)
- `!=` o `<>` (diferente de)

Filtros con operadores lógicos

```
SELECT *
FROM libros
WHERE anio_publicacion > 2010 AND autor = 'Gabriel García Márquez';
```

Podemos combinar múltiples condiciones usando operadores lógicos:

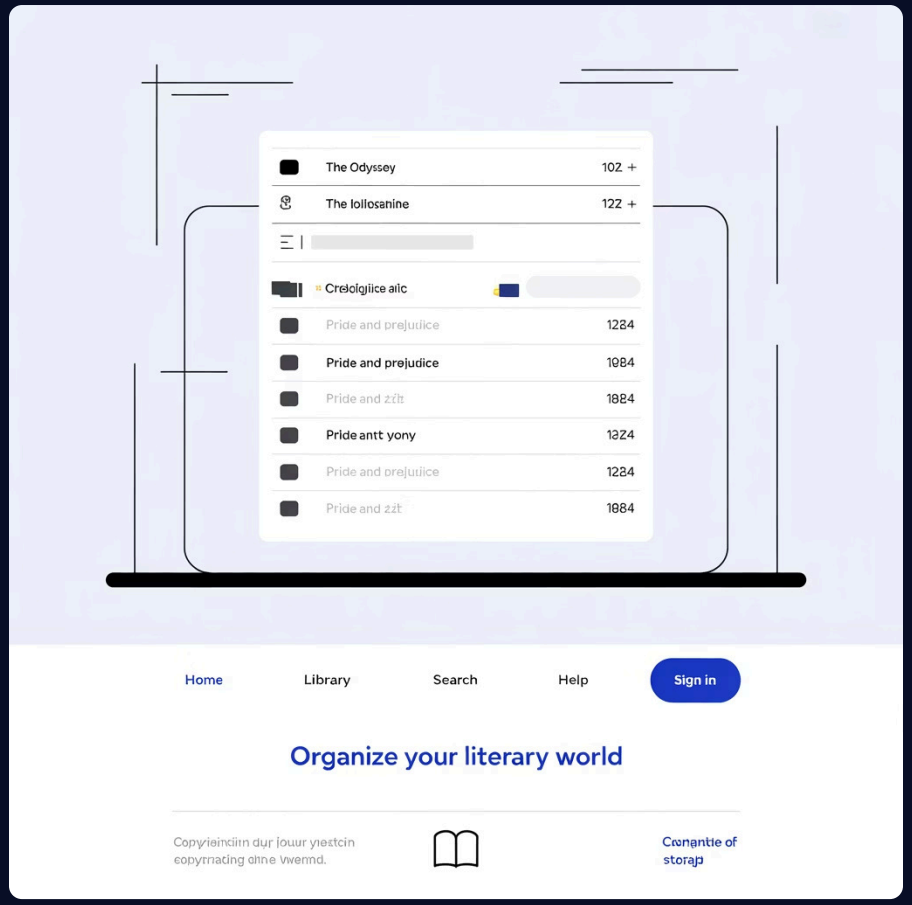
- **AND:** Ambas condiciones deben cumplirse
- **OR:** Al menos una condición debe cumplirse
- **NOT:** Niega una condición

Ordenando resultados con ORDER BY

```
SELECT titulo, autor, anio_publicacion
FROM libros
ORDER BY anio_publicacion DESC;
```

Esta consulta muestra los libros ordenados del más reciente al más antiguo.

- **ASC:** Orden ascendente (predeterminado)
- **DESC:** Orden descendente



Ordenamiento por múltiples columnas

```
SELECT titulo, autor, anio_publicacion
FROM libros
ORDER BY autor ASC, anio_publicacion DESC;
```

Esta consulta primero ordena alfabéticamente por autor (A-Z) y luego, para cada autor, muestra sus libros del más reciente al más antiguo.

El ordenamiento múltiple es útil para crear listados jerárquicos y organizados por varios criterios.

Funciones de agregación

COUNT()

Cuenta el número de registros

```
SELECT COUNT(*) FROM Libros;
```

SUM()

Suma los valores de una columna

```
SELECT SUM(anio_publicacion) FROM libros;
```

AVG()

Calcula el promedio

```
SELECT AVG(anio_publicacion) FROM libros;
```

MIN() y MAX()

Encuentra el valor mínimo y máximo

```
SELECT MIN(anio_publicacion),
MAX(anio_publicacion) FROM libros;
```

Agrupando resultados con GROUP BY

```
SELECT autor, COUNT(*) as total_libros
FROM libros
GROUP BY autor
ORDER BY total_libros DESC;
```

Esta consulta nos muestra cuántos libros tiene cada autor, ordenados del autor con más libros al que tiene menos.

GROUP BY permite:

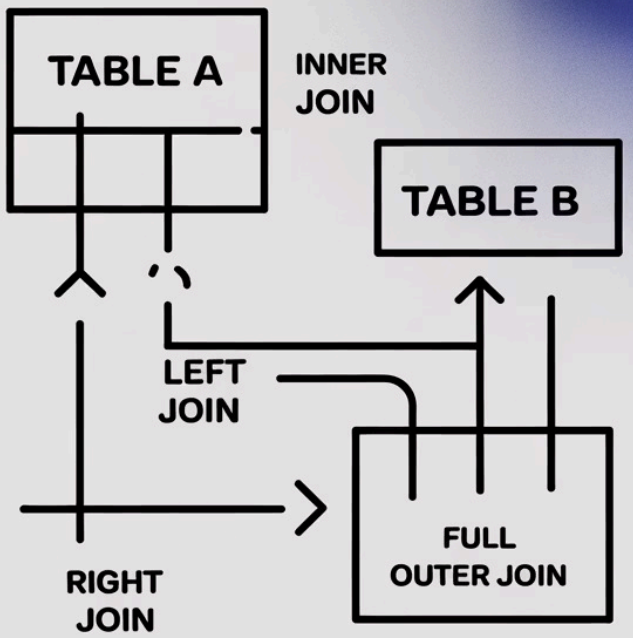
- Agrupar registros según valores de una o varias columnas
- Aplicar funciones de agregación a cada grupo
- Crear resúmenes y estadísticas por categorías

Filtrado de grupos con HAVING

```
SELECT autor, COUNT(*) as total_libros
FROM libros
GROUP BY autor
HAVING total_libros > 5
ORDER BY total_libros DESC;
```

HAVING funciona como WHERE pero se aplica después de agrupar los resultados.

En este ejemplo, solo mostramos los autores que tienen más de 5 libros en nuestra base de datos.



Relaciones entre tablas: JOINS

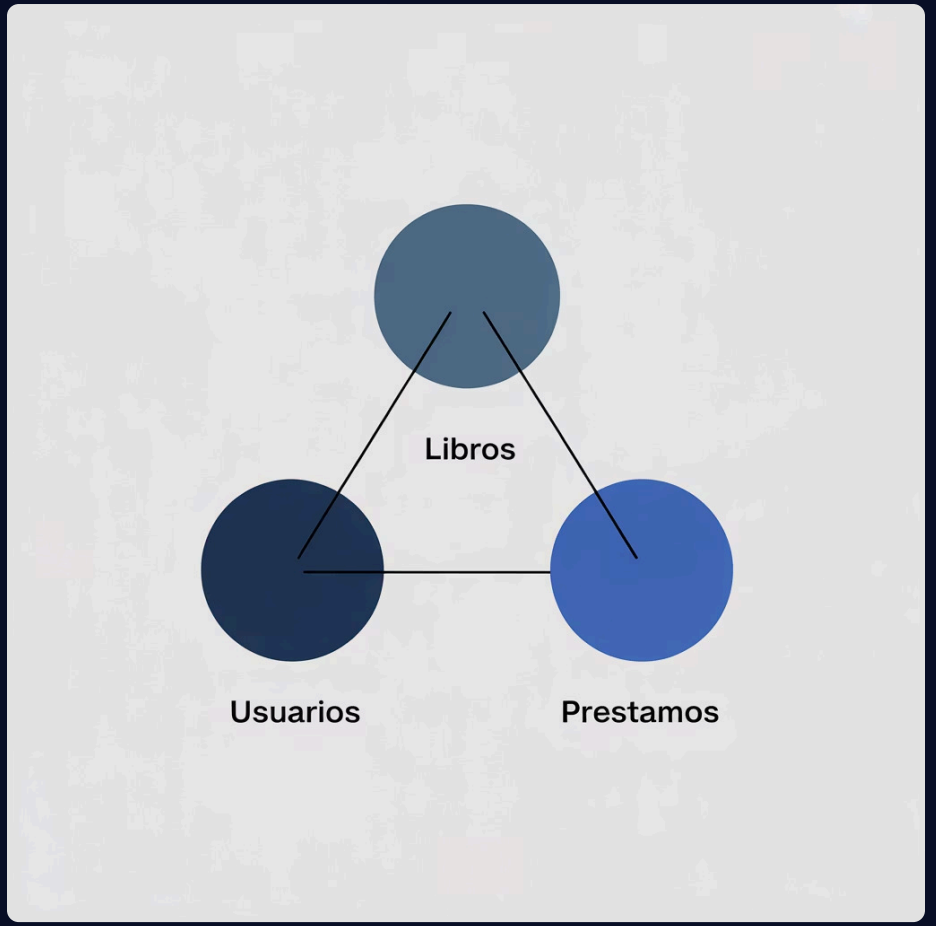
Los JOINS nos permiten combinar datos de múltiples tablas en una sola consulta, aprovechando las relaciones establecidas entre ellas.

¿Por qué necesitamos JOINS?

En una base de datos relacional, la información está distribuida en múltiples tablas para:

- Evitar la redundancia de datos
- Mantener la integridad referencial
- Facilitar el mantenimiento

Los JOINS nos permiten reconstruir estas relaciones al consultar los datos.



Tipos de JOINS en SQL

INNER JOIN

Solo registros que coinciden en ambas tablas

LEFT JOIN

Todos los registros de la tabla izquierda y las coincidencias de la derecha

RIGHT JOIN

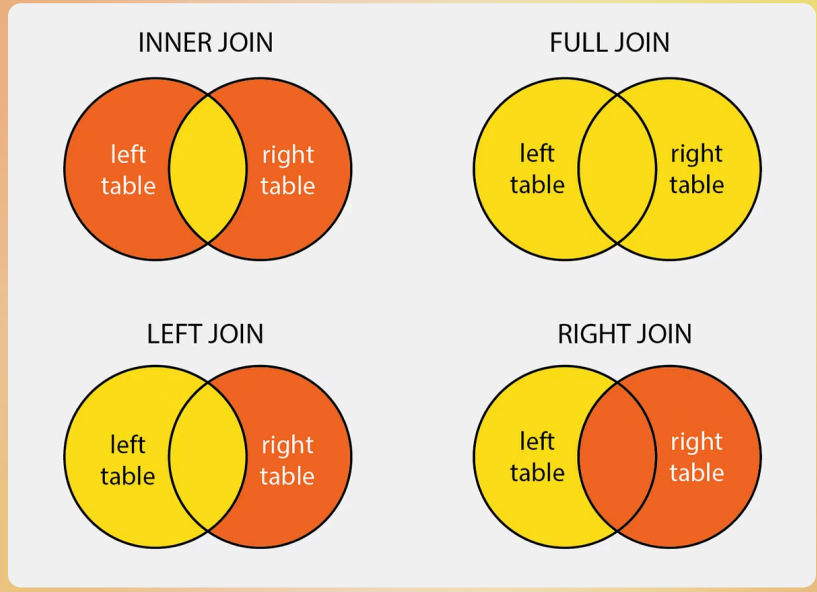
Todos los registros de la tabla derecha y las coincidencias de la izquierda

FULL JOIN

Todos los registros de ambas tablas

Visualización de los diferentes JOINS

Los diagramas de Venn son una forma efectiva de visualizar qué registros se incluyen en cada tipo de JOIN.



INNER JOIN: Solo coincidencias

```
SELECT p.id, u.nombre, l.titulo, p.fecha_prestamo
FROM Prestamos p
INNER JOIN usuarios u ON p.id_usuario = u.id
INNER JOIN libros l ON p.id_libro = l.id;
```

Esta consulta nos muestra todos los préstamos con la información del usuario y del libro, pero solo si existe tanto el usuario como el libro en sus respectivas tablas.

Es el tipo de JOIN más común y el predeterminado si solo escribimos "JOIN".

LEFT JOIN: Prioriza la tabla izquierda

```
SELECT l.titulo, p.fecha_prestamo
FROM libros l
LEFT JOIN prestamos p
  ON l.id = p.id_libro;
```

Esta consulta muestra todos los libros, incluso aquellos que nunca han sido prestados (en ese caso, los campos de Prestamos aparecerán como NULL).

RIGHT JOIN: Prioriza la tabla derecha

```
SELECT u.nombre, p.fecha_prestamo, l.titulo
FROM prestamos p
LEFT JOIN usuarios u ON p.id_usuario = u.id
RIGHT JOIN libros l ON p.id_libro = l.id;
```

Esta consulta muestra todos los libros (tabla derecha), hayan sido prestados o no. Si un libro no ha sido prestado, los campos de Prestamos aparecerán como NULL.

En la práctica, un RIGHT JOIN puede reescribirse como un LEFT JOIN cambiando el orden de las tablas.

FULL JOIN: Todos los registros de ambas tablas

Las consultas con FULL JOIN muestran todos los datos de todas las tablas que se le indiquen en el JOIN.

```
SELECT u.nombre, l.titulo, p.fecha_prestamo
FROM usuarios u
FULL JOIN prestamos p ON u.id = p.id_usuario
FULL JOIN libros l ON p.id_libro = l.id;
```

Esta consulta mostraría todos los usuarios, todos los libros y todos los préstamos, independientemente de si hay coincidencias o no.

Nota: MySQL no soporta directamente FULL JOIN, pero puede simularse con la combinación de LEFT JOIN y UNION.

INSERT: Agregando nuevos registros

El comando INSERT nos permite añadir nuevos registros a nuestras tablas, ampliando nuestra base de datos con nueva información.

Sintaxis básica de INSERT

```
INSERT INTO nombre_tabla (columna1, columna2, ...)
VALUES (valor1, valor2, ...);
```

Donde:

- **INSERT INTO:** Indica la tabla donde se agregarán los datos
- **columnas:** Lista de columnas que recibirán valores (opcional si se proporcionan valores para todas las columnas)
- **VALUES:** Valores a insertar, en el mismo orden que las columnas especificadas

Ejemplo: Agregar un nuevo libro

```
INSERT INTO libros
(titulo, autor, anio_publicacion)
VALUES
('Crónica de una muerte anunciada',
 'Gabriel García Márquez',
 1981);
```

Esta sentencia agrega un nuevo libro a nuestra tabla Libros.

Observaciones importantes:

- Los valores de texto van entre comillas o comillas simples
- Los valores numéricos van sin comillas
- Si no especificamos el id, se asignará automáticamente (si es autoincremental)

Insertar múltiples registros a la vez

```
INSERT INTO usuarios (nombre, correo, telefono)
VALUES
('Ana Gómez', 'ana@ejemplo.com', '3101234567'),
('Carlos Pérez', 'carlos@ejemplo.com', '3207654321'),
('María López', 'maria@ejemplo.com', '3153456789');
```

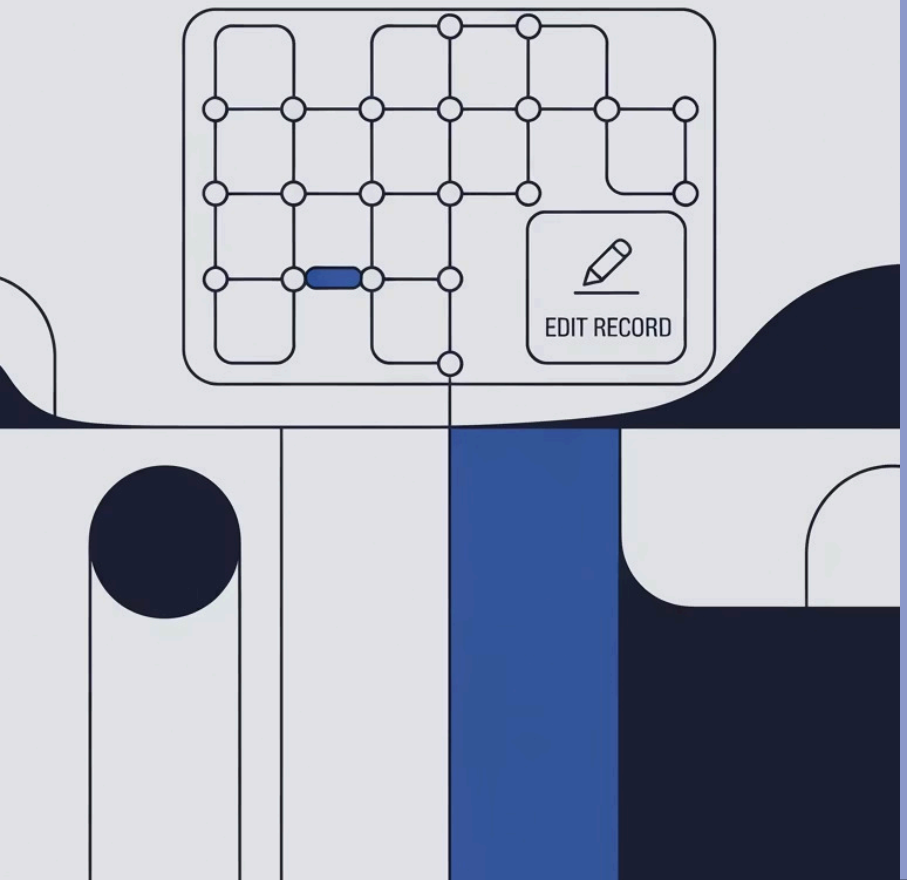
Podemos insertar varios registros en una sola sentencia, separándolos con comas. Esto es más eficiente que hacer múltiples INSERT individuales.

INSERT con subconsulta

```
INSERT INTO usuarios_premium (id, nombre, correo)
SELECT id, nombre, correo
FROM Usuarios
WHERE fecha_registro < '2022-01-01';
```

También podemos insertar datos que provienen de una consulta SELECT. Esto es útil para:

- Copiar datos entre tablas
- Crear tablas de respaldo
- Migrar información basada en ciertos criterios



UPDATE: Modificando registros existentes

El comando UPDATE nos permite modificar datos ya existentes en nuestras tablas, actualizando la información según nuestras necesidades.

Sintaxis básica de UPDATE

```
UPDATE nombre_tabla
SET columna1 = valor1, columna2 = valor2, ...
WHERE condicion;
```

Donde:

- **UPDATE:** Indica la tabla cuyos registros serán modificados
- **SET:** Especifica qué columnas cambiar y sus nuevos valores
- **WHERE:** Define qué registros serán afectados (¡crucial para evitar actualizar toda la tabla!)

Ejemplo: Actualizar un usuario

```
UPDATE usuarios
SET telefono = '3209876543',
    correo = 'ana.nueva@ejemplo.com'
WHERE id = 1;
```

Esta sentencia actualiza el teléfono y el correo del usuario con id = 1.

⚠ ¡Precaución con UPDATE!

Siempre incluye una cláusula WHERE específica, o modificarás todos los registros de la tabla.

Actualización basada en otras tablas

```
UPDATE libros l
JOIN prestamos p ON l.id = p.id_libro
SET l.disponible = 0
WHERE p.fecha_devolucion IS NULL;
```

Podemos actualizar registros basándonos en condiciones que involucran otras tablas.

Este ejemplo marca como no disponibles (disponible = 0) todos los libros que actualmente están prestados.

Actualización con expresiones

```
UPDATE libros
SET precio = precio * 1.1
WHERE anio_publicacion > 2020;
```

Podemos utilizar el valor actual de una columna en la actualización.

Este ejemplo aumenta en un 10% el precio de todos los libros publicados después de 2020.

DELETE: Eliminando registros

El comando DELETE nos permite eliminar registros de nuestras tablas, removiendo información que ya no necesitamos.

Sintaxis básica de DELETE

```
DELETE FROM nombre_tabla
WHERE condicion;
```

Donde:

- **DELETE FROM:** Indica la tabla de la que se eliminarán registros
- **WHERE:** Define qué registros serán eliminados (¡crucial para evitar eliminar toda la tabla!)

Ejemplo: Eliminar un préstamo

```
DELETE FROM prestamos
WHERE id = 5;
```

Esta sentencia elimina el préstamo con id = 5.

```
DELETE FROM prestamos
WHERE fecha_devolucion < '2022-01-01';
```

Esta elimina todos los préstamos devueltos antes de 2022.

¡PELIGRO!

Una vez eliminados los datos con DELETE, no se pueden recuperar fácilmente sin un respaldo.

Siempre verifica dos veces antes de ejecutar DELETE, especialmente en producción.

Eliminar con JOIN

```
DELETE p
FROM prestamos p
JOIN usuarios u ON p.id_usuario = u.id
WHERE u.activo = 0;
```

Podemos eliminar registros basándonos en condiciones que involucran otras tablas.

Este ejemplo elimina todos los préstamos de usuarios que ya no están activos en el sistema.

Reflexión: Riesgos de manipulación sin filtros

Riesgos de ejecutar comandos sin WHERE:

- Pérdida masiva de datos (DELETE sin WHERE elimina toda la tabla)
- Modificación no intencionada de registros (UPDATE sin WHERE actualiza todos los registros)
- Imposibilidad de recuperación sin backups
- Impacto en la integridad referencial

Buenas prácticas:

- Usar transacciones (BEGIN, COMMIT, ROLLBACK)
- Realizar copias de seguridad antes de operaciones masivas
- Probar primero con SELECT para verificar los registros afectados
- Utilizar LIMIT en DELETE para limitar el impacto

Resumen: Comandos SQL básicos

1

SELECT

Recupera datos de una o varias tablas.

```
SELECT columnas FROM tabla WHERE condicion;
```

2

INSERT

Agrega nuevos registros a una tabla.

```
INSERT INTO tabla (cols) VALUES (vals);
```

3

UPDATE

Modifica registros existentes.

```
UPDATE tabla SET col=val WHERE condicion;
```

4

DELETE

Elimina registros de una tabla.

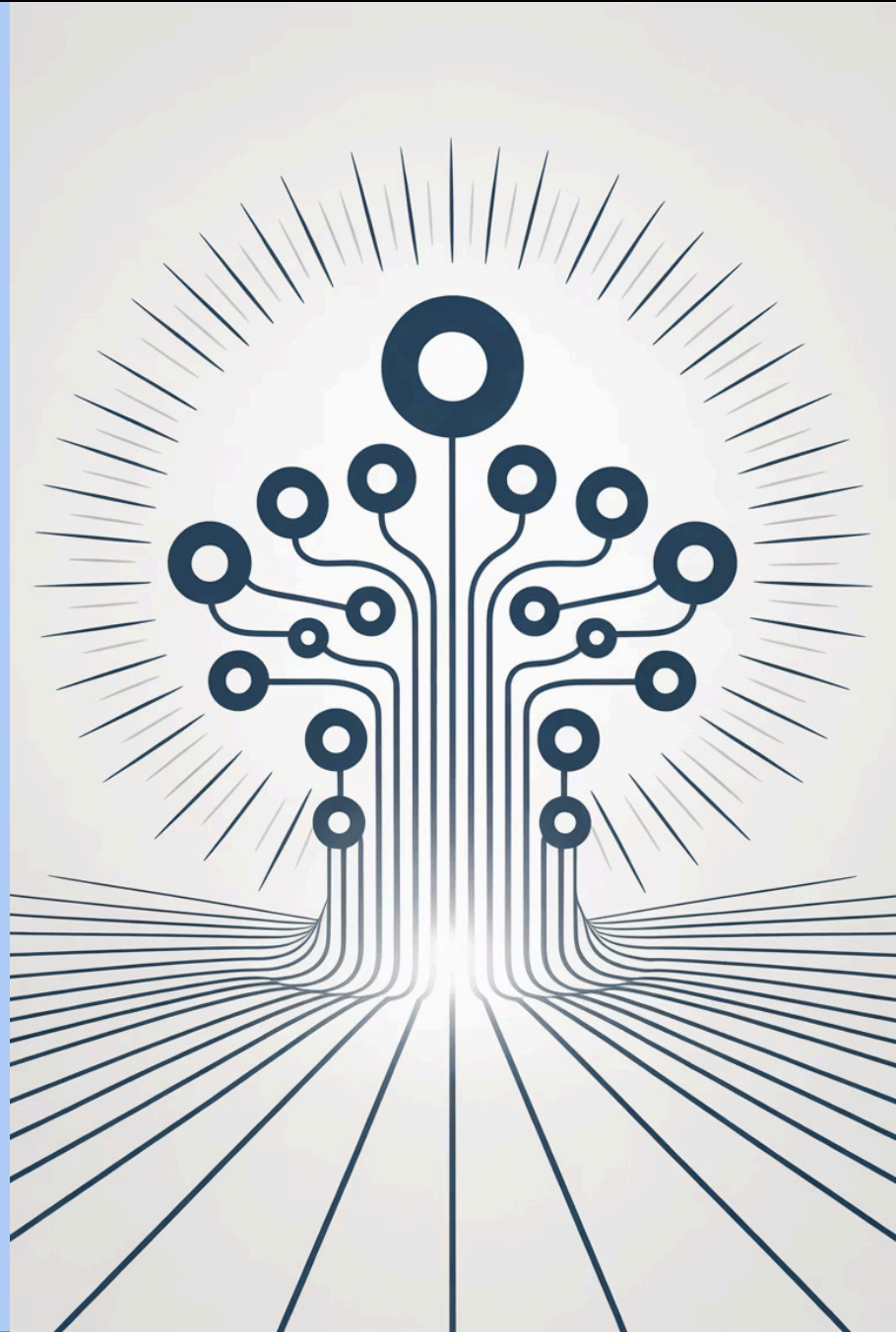
```
DELETE FROM tabla WHERE condicion;
```

Pregunta de reflexión

¿Por qué la cláusula `WHERE` es absolutamente crítica y, a la vez, potencialmente devastadora si se omite al utilizar `UPDATE` o `DELETE`?

Piensa en las consecuencias directas e indirectas de ejecutar estas operaciones sin un filtro adecuado. ¿Qué escenarios podrías enfrentar en un entorno de producción real?

¡Gracias por tu atención!



Universidad de los Andes | Educación Continua
Vicerrectoría Académica

#EDCOUNIANDES

https://educacioncontinua.uniandes.edu.co/

Contacto: educacion.continua@uniandes.edu.co

Universidad de los Andes | Vigilada Mineducación. Reconocimiento como Universidad: Decreto 1297 del 30 de mayo de 1964. Reconocimiento personería jurídica: Resolución 28 del 23 de febrero de 1949 MinJusticia.