🗄️ Consultas SQL Básicas

INSERT, SELECT, UPDATE y DELETE

Python Full Stack - Clase 22

¿Qué aprenderemos hoy?

- 🔍 Recuperar datos con SELECT y sus variantes
- ➕ Insertar nuevos registros con INSERT
- ✏️ Actualizar información existente con UPDATE
- 🗑️ Eliminar registros con DELETE
- 🎯 Usar condiciones WHERE para filtrar datos
- 📊 Ordenar y limitar resultados
- 💡 Entender la importancia de WHERE en UPDATE y DELETE

Utilizaremos la base de datos de ejemplo: `x` . x.sql

🔍 SELECT: Recuperar Datos

¿Qué es SELECT?

SELECT es el comando SQL que nos permite **leer** información de nuestras tablas.

Piensa en SELECT como hacer una **pregunta** a la base de datos:

- "¿Qué usuarios tengo?"
- "¿Cuáles son los tweets más recientes?"
- "¿Quién tiene más de 18 años?"

Sintaxis Básica de SELECT

```sql
SELECT *
FROM nombre_tabla;
```

Desglosando:

- `SELECT *` → Selecciona **todas las columnas**
- `FROM nombre_tabla` → De la tabla especificada
- `;` → Termina la consulta (punto y coma)

Ejemplo: Ver Todos los Usuarios

```sql
SELECT *
FROM usuarios;
```

¿Qué hace esto?

- Muestra **todas las columnas** de la tabla `usuarios`
- Muestra **todos los registros** (filas)

Resultado: Verás id, nombre, apellido, email, etc. de todos los usuarios.

Seleccionar Columnas Específicas

En lugar de `*`, podemos elegir qué columnas queremos ver:

```
SELECT nombre, apellido
FROM usuarios;
```

¿Qué hace esto?

- Solo muestra las columnas `nombre` y `apellido`
- Ignora las demás columnas (id, email, etc.)

Ventaja: Más rápido y claro cuando solo necesitas cierta información.

SELECT con Condiciones WHERE

WHERE nos permite **filtrar** los resultados:

```
SELECT *
FROM usuarios
WHERE id = 5;
```

¿Qué hace esto?

- Solo muestra el usuario cuyo `id` es igual a 5
- Si no existe, no muestra nada

Analogía: Como buscar en un directorio telefónico solo las personas llamadas "Juan".

Operadores de Comparación

Podemos usar diferentes operadores en WHERE:

```
-- Igual a
WHERE id = 5
-- Mayor que
WHERE id > 3
-- Mayor o igual que
WHERE id >= 3
-- Menor que
WHERE id < 10
-- Menor o igual que
WHERE id <= 4
-- Diferente de
WHERE id != 3
```

Múltiples Condiciones: OR y AND

Podemos combinar condiciones:

```sql
-- Usuarios con id 4 O id 5
SELECT nombre
FROM usuarios
WHERE id = 4 OR id = 5;

-- Usuarios con id mayor a 3 Y menor a 10
SELECT *
FROM usuarios
WHERE id > 3 AND id < 10;
```

OR = Al menos una condición debe cumplirse

AND = Todas las condiciones deben cumplirse

Búsqueda con LIKE

LIKE nos permite buscar patrones en texto:

```sql
-- Nombres que terminan en 'e'
SELECT *
FROM usuarios
WHERE nombre LIKE '%e';
```

El símbolo `%` es un comodín:

- `'%e'` → Termina en 'e' (ej: "María", "José")
- `'r%'` → Comienza con 'r' (ej: "Roberto", "Rosa")
- `'%navidad%'` → Contiene 'navidad' en cualquier parte

Más Ejemplos con LIKE

```sql
-- Nombres que comienzan con 'A'
SELECT *
FROM usuarios
WHERE nombre LIKE 'A%';

-- Nombres que NO comienzan con 'A'
SELECT *
FROM usuarios
WHERE nombre NOT LIKE 'A%';

-- Tweets que contienen la palabra 'navidad'
SELECT *
FROM tweets
WHERE tweet LIKE '%navidad%';
```

Ordenar Resultados: ORDER BY

ORDER BY nos permite ordenar los resultados:

```sql
-- Orden ascendente (A-Z, 1-9)
SELECT nombre
FROM usuarios
ORDER BY nombre ASC;

-- Orden descendente (Z-A, 9-1)
SELECT *
FROM tweets
ORDER BY created_at DESC;
```

ASC = Ascendente (por defecto, puede omitirse)

DESC = Descendente

Combinando WHERE y ORDER BY

Podemos filtrar Y ordenar al mismo tiempo:

```sql
SELECT *
FROM tweets
WHERE tweet LIKE '%navidad%'
ORDER BY created_at DESC;
```

¿Qué hace esto?

1. Busca tweets que contengan 'navidad'
2. Los ordena del más reciente al más antiguo

Orden de ejecución: Primero WHERE, luego ORDER BY

Limitar Resultados: LIMIT

LIMIT nos permite mostrar solo una cantidad específica de registros:

```sql
-- Solo los primeros 3 tweets
SELECT *
FROM tweets
LIMIT 3;
```

Uso común: Paginación (mostrar 10 resultados por página)

LIMIT con OFFSET

OFFSET nos permite "saltar" registros:

```
-- Tweets del 3 al 5 (salta los primeros 2)
SELECT *
FROM tweets
LIMIT 3
OFFSET 2;
```

Sintaxis alternativa:

```
SELECT *
FROM tweets
LIMIT 2, 3;  -- OFFSET primero, luego LIMIT
```

Útil para: Paginación (página 2, página 3, etc.)

💪 Ejercicio: Práctica con SELECT

Abre MySQL Workbench y ejecuta estas consultas:

1. Selecciona todos los usuarios de la tabla `usuarios`
2. Selecciona solo los nombres de los usuarios
3. Encuentra el usuario con id = 1
4. Busca usuarios cuyo nombre termine en 'a'
5. Ordena los tweets por fecha de creación (más recientes primero)
6. Muestra solo los primeros 5 tweets

Pistas:

- Usa `SELECT *` para ver todas las columnas
- Recuerda usar `WHERE` para filtrar
- `ORDER BY` va después de `WHERE`

⏱️ **Tiempo:** 10 minutos

+ INSERT: Agregar Datos

¿Qué es INSERT?

INSERT es el comando SQL que nos permite **agregar nuevos registros** a nuestras tablas.

Piensa en INSERT como **llenar una nueva fila** en una hoja de cálculo:

- Agregar un nuevo usuario
- Crear un nuevo tweet
- Registrar un nuevo favorito

Sintaxis Básica de INSERT

```
INSERT INTO nombre_tabla (columna1, columna2, columna3)
VALUES ('valor1', 'valor2', 'valor3');
```

Desglosando:

- `INSERT INTO nombre_tabla` → Especifica la tabla
- `(columna1, columna2, ...)` → Las columnas que vamos a llenar
- `VALUES (...)` → Los valores correspondientes

⚠️ **Importante:** El orden de los valores debe coincidir con el orden de las columnas.

Ejemplo: Insertar un Tweet

```sql
INSERT INTO tweets (tweet, usuario_id, created_at, updated_at)
VALUES ('Nuevo tweet desde query', 1, '2024-04-24', '2024-04-24');
```

¿Qué hace esto?

- Crea un nuevo registro en la tabla `tweets`
- Asigna valores a las columnas especificadas
- El `id` se genera automáticamente (si tiene AUTO_INCREMENT)

INSERT a través de MySQL Workbench

Método visual (más fácil para principiantes):

1. Despliega la tabla donde quieres agregar datos
2. Haz clic en la última fila (que dice `NULL`)
3. Haz doble clic en cada columna e ingresa los valores
4. Haz clic en **Apply**
5. Confirma en la ventana que aparece

Ventaja: Visual y fácil de entender

INSERT como Comando SQL

Método por código (más profesional):

```
INSERT INTO usuarios (nombre, apellido, email)
VALUES ('María', 'González', 'maria@email.com');
```

Ventajas:

- Más rápido cuando conoces la estructura
- Puedes copiar y pegar
- Es el método que usarás en aplicaciones

⚠️ Reglas Importantes de INSERT

1. **No incluyas el ID** si tiene AUTO_INCREMENT (se genera automáticamente)
2. **Las fechas** deben estar en formato 'YYYY-MM-DD'
3. **Los textos** van entre comillas simples `'texto'`
4. **Los números** NO llevan comillas
5. **El orden** de valores debe coincidir con el orden de columnas

Ejemplo Completo: Insertar Usuario

```
INSERT INTO usuarios (nombre, apellido, nombre_usuario, email, fecha_nacimiento)
VALUES ('Carlos', 'Rodríguez', 'carlos_r', 'carlos@email.com', '1990-05-15');
```

Verificando que se insertó:

```
SELECT *
FROM usuarios
WHERE nombre_usuario = 'carlos_r';
```

💪 Ejercicio: Práctica con INSERT

¡Hora de practicar!

Crea nuevos registros en tu base de datos:

1. Inserta un nuevo usuario con tus datos (nombre, apellido, email)
2. Inserta 3 tweets diferentes asociados a diferentes usuarios
3. Verifica que los registros se crearon correctamente usando SELECT

Pistas:

- Usa `INSERT INTO` seguido del nombre de la tabla
- Especifica las columnas entre paréntesis
- Los valores van en `VALUES`
- Verifica con `SELECT * FROM tabla WHERE ...`

⏱️ **Tiempo:** 10 minutos

✏️ UPDATE: Modificar Datos

¿Qué es UPDATE?

UPDATE es el comando SQL que nos permite **modificar información existente** en nuestras tablas.

Piensa en UPDATE como **editar una celda** en una hoja de cálculo:

- Cambiar el nombre de un usuario
- Actualizar un tweet
- Modificar una fecha

Sintaxis Básica de UPDATE

```
UPDATE nombre_tabla
SET columna1 = 'nuevo_valor', columna2 = 'otro_valor'
WHERE condicion;
```

Desglosando:

- `UPDATE nombre_tabla` → Especifica la tabla a modificar
- `SET columna = valor` → Qué columna cambiar y a qué valor
- `WHERE condicion` → **MUY IMPORTANTE:** Qué registros modificar

⚠️ ADVERTENCIA: WHERE es OBLIGATORIO

Sin WHERE, UPDATE modifica TODOS los registros:

```
-- ❌ PELIGROSO: Cambia TODOS los tweets
UPDATE tweets
SET tweet = 'Texto modificado';
```

Con WHERE, solo modifica los que cumplen la condición:

```
-- ✅ CORRECTO: Solo cambia el tweet con id = 14
UPDATE tweets
SET tweet = 'Otro tweet modificado'
WHERE id = 14;
```

Regla de oro: SIEMPRE usa WHERE en UPDATE (a menos que realmente quieras cambiar todo).

Ejemplo: Actualizar un Tweet

```
UPDATE tweets
SET tweet = 'Otro tweet modificado'
WHERE id = 14;
```

¿Qué hace esto?

- Busca el tweet con `id = 14`
- Cambia su columna `tweet` al nuevo texto
- No modifica ningún otro registro

Actualizar Múltiples Columnas

Puedes actualizar varias columnas a la vez:

```
UPDATE usuarios
SET nombre = 'María', apellido = 'López', email = 'maria.nueva@email.com'
WHERE id = 5;
```

¿Qué hace esto?

- Actualiza nombre, apellido y email del usuario con id = 5
- Todo en una sola consulta

UPDATE a través de MySQL Workbench

Método visual:

1. Despliega la tabla
2. Localiza el registro que quieres modificar
3. Haz doble clic en la celda que quieres cambiar
4. Escribe el nuevo valor
5. Haz clic en **Apply**
6. Confirma en la ventana

Ventaja: Visual y seguro (puedes ver qué cambias)

UPDATE con Condiciones Complejas

Puedes usar condiciones más complejas:

```sql
-- Actualizar todos los tweets de un usuario específico
UPDATE tweets
SET tweet = 'Tweet actualizado'
WHERE usuario_id = 3;

-- Actualizar usuarios mayores de 18 años
UPDATE usuarios
SET activo = 1
WHERE fecha_nacimiento < '2006-01-01';
```

💪 Ejercicio: Práctica con UPDATE

¡Hora de practicar!

Modifica registros existentes:

1. Actualiza el nombre de un usuario específico (usa WHERE con id)
2. Modifica el texto de un tweet
3. Actualiza la fecha de actualización (`updated_at`) de varios tweets de un mismo usuario

Pistas:

- Empieza con `UPDATE nombre_tabla`
- Usa `SET columna = valor`
- **SIEMPRE** incluye `WHERE` para especificar qué modificar
- Verifica los cambios con SELECT

⏱️ **Tiempo:** 10 minutos

🗑️ DELETE: Eliminar Datos

¿Qué es DELETE?

DELETE es el comando SQL que nos permite **eliminar registros** de nuestras tablas.

Piensa en DELETE como **borrar una fila completa** en una hoja de cálculo:

- Eliminar un usuario
- Borrar un tweet
- Remover un favorito

Sintaxis Básica de DELETE

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

Desglosando:

- `DELETE FROM nombre_tabla` → Especifica la tabla
- `WHERE condicion` → **MUY IMPORTANTE:** Qué registros eliminar

⚠️ ADVERTENCIA: WHERE es OBLIGATORIO

Sin WHERE, DELETE elimina TODOS los registros:

```
-- ❌ PELIGROSO: Borra TODOS los tweets
DELETE FROM tweets;
```

Con WHERE, solo elimina los que cumplen la condición:

```
-- ✅ CORRECTO: Solo elimina el tweet con id = 14
DELETE FROM tweets
WHERE id = 14;
```

Regla de oro: SIEMPRE usa WHERE en DELETE (a menos que realmente quieras borrar todo).

Ejemplo: Eliminar un Tweet

```sql
DELETE FROM tweets
WHERE id = 14;
```

¿Qué hace esto?

- Busca el tweet con `id = 14`
- Lo elimina permanentemente
- No elimina ningún otro registro

⚠️ **Cuidado:** Una vez eliminado, no se puede recuperar fácilmente.

DELETE a través de MySQL Workbench

Método visual:

1. Despliega la tabla
2. Localiza el registro que quieres eliminar
3. Haz clic derecho sobre la fila
4. Selecciona **Delete Row(s)**
5. Haz clic en **Apply**
6. Confirma en la ventana

Ventaja: Visual y puedes ver exactamente qué eliminas

DELETE con Condiciones Complejas

Puedes eliminar múltiples registros que cumplan una condición:

```sql
-- Eliminar todos los tweets de un usuario
DELETE FROM tweets
WHERE usuario_id = 3;

-- Eliminar usuarios inactivos
DELETE FROM usuarios
WHERE activo = 0;
```

⚠️ Siempre verifica primero con SELECT:

```sql
-- Primero verifica qué se va a eliminar
SELECT * FROM tweets WHERE usuario_id = 3;

-- Luego elimina
DELETE FROM tweets WHERE usuario_id = 3;
```

Error: SQL SAFE UPDATES

Si recibes un error sobre "SQL SAFE UPDATES", ejecuta:

```
SET SQL_SAFE_UPDATES = 0;
```

¿Qué hace esto?

- Desactiva la protección de MySQL Workbench
- Te permite ejecutar UPDATE y DELETE sin WHERE
- ⚠️ **Úsalo con cuidado:** Asegúrate de saber lo que haces

💪 Ejercicio: Práctica con DELETE

¡Hora de practicar!

Elimina registros de forma segura:

1. Primero, usa SELECT para ver qué registros vas a eliminar
2. Elimina un tweet específico (usa WHERE con id)
3. Elimina todos los tweets de un usuario específico
4. Verifica que se eliminaron correctamente

Pistas:

- Empieza con `DELETE FROM nombre_tabla`
- **SIEMPRE** incluye `WHERE` para especificar qué eliminar
- Verifica primero con SELECT antes de eliminar
- Si hay error de SQL_SAFE_UPDATES, ejecuta `SET SQL_SAFE_UPDATES = 0;`

⏱️ **Tiempo:** 10 minutos

Resumen de Conceptos Clave

Las 4 Operaciones Básicas (CRUD)

Operación	Comando	Propósito	¿Requiere WHERE?
Create	`INSERT`	Agregar nuevos registros	No
Read	`SELECT`	Leer/consultar datos	Opcional (para filtrar)
Update	`UPDATE`	Modificar datos existentes	**SÍ (obligatorio)**
Delete	`DELETE`	Eliminar registros	**SÍ (obligatorio)**

SELECT: Sintaxis Completa

```
SELECT columnas
FROM tabla
WHERE condicion
ORDER BY columna ASC/DESC
LIMIT cantidad
OFFSET inicio;
```

Ejemplo completo:

```
SELECT nombre, apellido
FROM usuarios
WHERE fecha_nacimiento > '2000-01-01'
ORDER BY nombre ASC
LIMIT 10;
```

INSERT: Sintaxis

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

Ejemplo:

```
INSERT INTO usuarios (nombre, email)
VALUES ('Juan', 'juan@email.com');
```

UPDATE: Sintaxis

```
UPDATE tabla
SET columna1 = 'valor1', columna2 = 'valor2'
WHERE condicion;  -- ⚠️ OBLIGATORIO
```

Ejemplo:

```
UPDATE usuarios
SET nombre = 'Pedro'
WHERE id = 5;
```

DELETE: Sintaxis

```
DELETE FROM tabla
WHERE condicion;  -- ⚠️ OBLIGATORIO
```

Ejemplo:

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

Lo Más Importante

✅ **SELECT** → Para leer datos (WHERE es opcional para filtrar)
✅ **INSERT** → Para agregar nuevos registros
✅ **UPDATE** → Para modificar (SIEMPRE usa WHERE)
✅ **DELETE** → Para eliminar (SIEMPRE usa WHERE)
✅ **WHERE** → Es tu mejor amigo para evitar errores catastróficos

Buenas Prácticas

1. **Siempre usa WHERE** en UPDATE y DELETE
2. **Verifica primero** con SELECT antes de modificar o eliminar
3. **Prueba en datos de prueba** antes de trabajar con datos reales
4. **Haz respaldos** de tu base de datos regularmente
5. **Usa transacciones** cuando trabajes con datos importantes

Errores Comunes a Evitar

❌ Olvidar WHERE en UPDATE/DELETE

```sql
-- ❌ MAL: Borra todo
DELETE FROM usuarios;
```

❌ Orden incorrecto en INSERT

```sql
-- ❌ MAL: Valores en orden incorrecto
INSERT INTO usuarios (nombre, email) VALUES ('email@email.com', 'Juan');
```

❌ Comillas incorrectas

```sql
-- ❌ MAL: Números con comillas
INSERT INTO usuarios (id) VALUES ('5');
```

Recursos para Seguir Aprendiendo

- **SQLZoo**: http://sqlzoo.net
  - SELECT básico: SELECT basics
  - SELECT nombres: SELECT names
  - SELECT desde World: SELECT from WORLD
- **Documentación MySQL**: https://dev.mysql.com/doc/

Próximos Pasos

En la siguiente clase aprenderemos:

- 🔗 **JOINs**: Combinar datos de múltiples tablas
- 📈 **Funciones agregadas**: COUNT, SUM, AVG, etc.
- 🎯 **Consultas más complejas**: Subconsultas y agrupaciones

Práctica para Casa

Ejercicio sugerido:

1. Crea una base de datos de una librería con tablas: `libros`, `autores`, `ventas`
2. Inserta al menos 10 libros con sus autores
3. Practica SELECT con diferentes filtros y ordenamientos
4. Actualiza precios de algunos libros
5. Elimina libros que ya no están disponibles

Objetivo: Familiarizarte con las 4 operaciones básicas.

Consejos Finales

- 💡 **Practica mucho**: SQL se aprende haciendo
- 💡 **Experimenta**: Prueba diferentes consultas
- 💡 **Lee los errores**: MySQL te dice qué está mal
- 💡 **Usa SELECT primero**: Siempre verifica antes de modificar
- 💡 **Ten paciencia**: Dominar SQL toma tiempo

🎉 ¡Felicidades!

Ya sabes INSERT, SELECT, UPDATE y DELETE

¡Estás listo para manipular datos en bases de datos!

? Preguntas

¿Alguna duda sobre consultas SQL básicas?

🙌 ¡Excelente Trabajo!

¡Nos vemos en la siguiente clase con JOINs y funciones agregadas!

No olvides completar todos los ejercicios 📝