

Fundamentos de SQL: Lenguaje para Bases de Datos Relacionales

Una introducción completa al **Structured Query Language**.

¿Qué es SQL?

Definición y Propósito

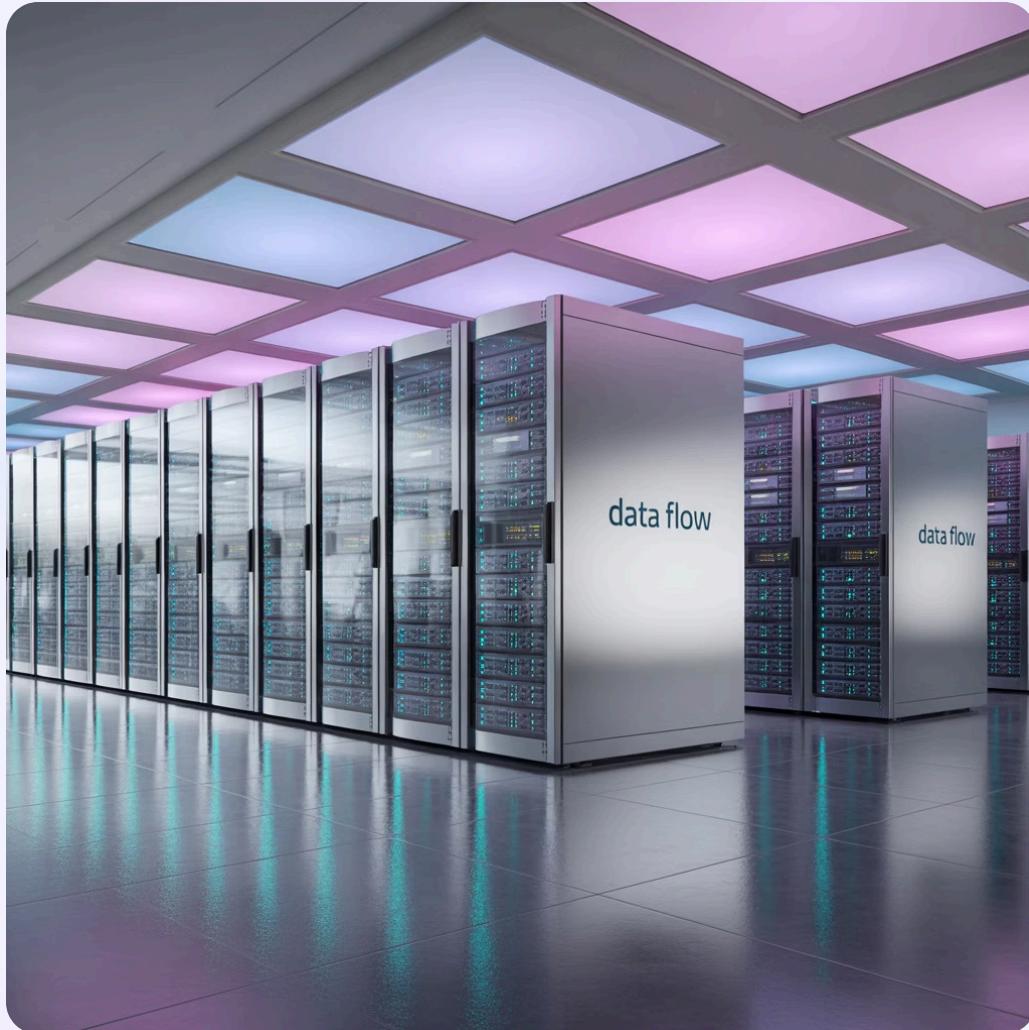
SQL (Structured Query Language) es el lenguaje estándar para gestionar bases de datos relacionales.

Desarrollado en los años 70 en IBM, se ha convertido en el lenguaje universal para la manipulación de datos estructurados.

- Permite consultar datos de forma eficiente
- Facilita la modificación de información almacenada
- Posibilita la administración completa de la base de datos

The screenshot shows the CodeFlow software interface. At the top, there is a navigation bar with the CodeFlow logo, Dashboard, Schemas, Queries, Documentation, and a Sign Up button. Below the navigation bar, the main title "Codeflow" is displayed in large, bold letters, followed by the tagline "Visualize, Optimize, Deploy". The interface features a dark-themed dashboard with several floating windows and cards. One window on the left shows a file icon with the number "100". Another window on the right displays a table with data rows and percentages. A central window shows a list of items with small icons next to them. A card at the bottom left provides technical details: "VBT" and "29 0.02K". A card at the bottom right shows a small diagram with three components connected by lines. The overall theme is data visualization and management.

Relevancia de SQL en el Mundo Actual



¿Por qué aprender SQL?

SQL sigue siendo la tecnología fundamental en el ecosistema de datos a pesar del auge de soluciones NoSQL.

- Presente en prácticamente todas las empresas
- Habilidad altamente demandada en el mercado laboral
- Base para otros lenguajes y tecnologías de datos
- Esencial para roles como analista de datos, desarrollador y DBA

Arquitectura de una Base de Datos Relacional



Tablas

Estructuras que almacenan datos en filas y columnas. Cada tabla representa una entidad (usuarios, productos, etc.).



Claves

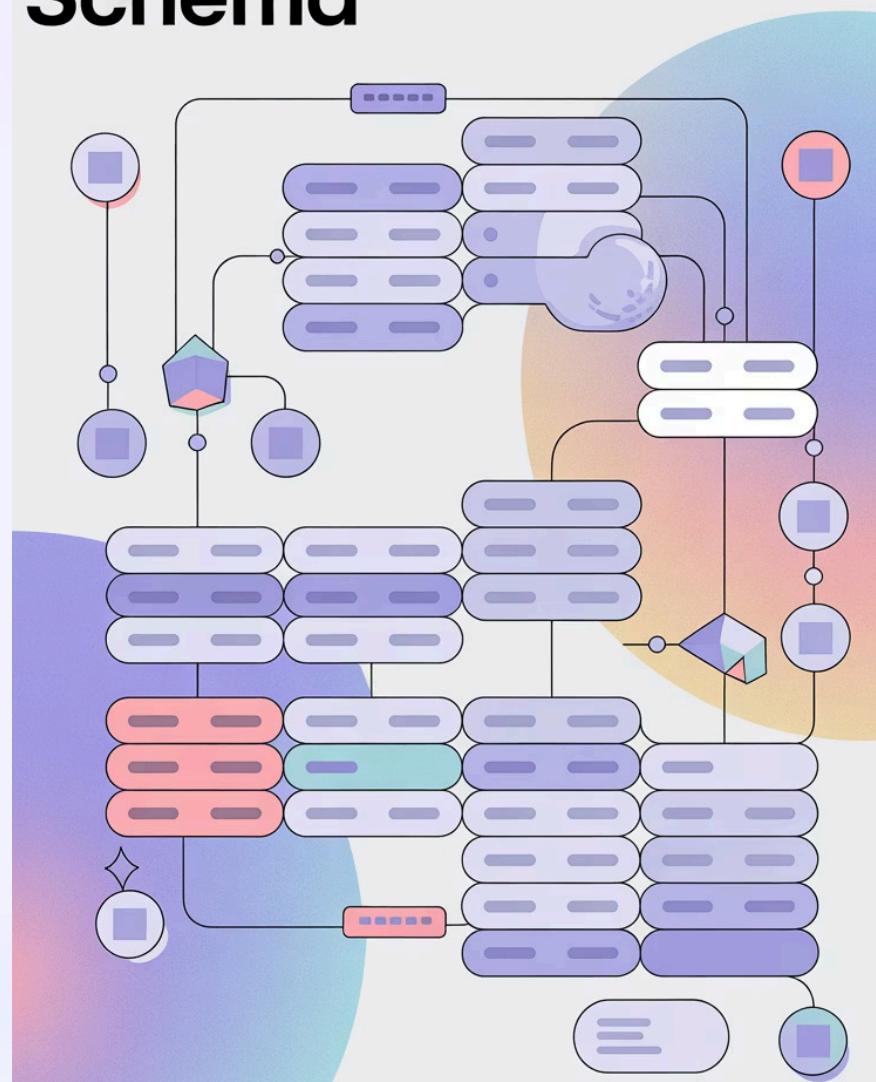
Identificadores únicos (claves primarias) y referencias a otras tablas (claves foráneas) que establecen relaciones.



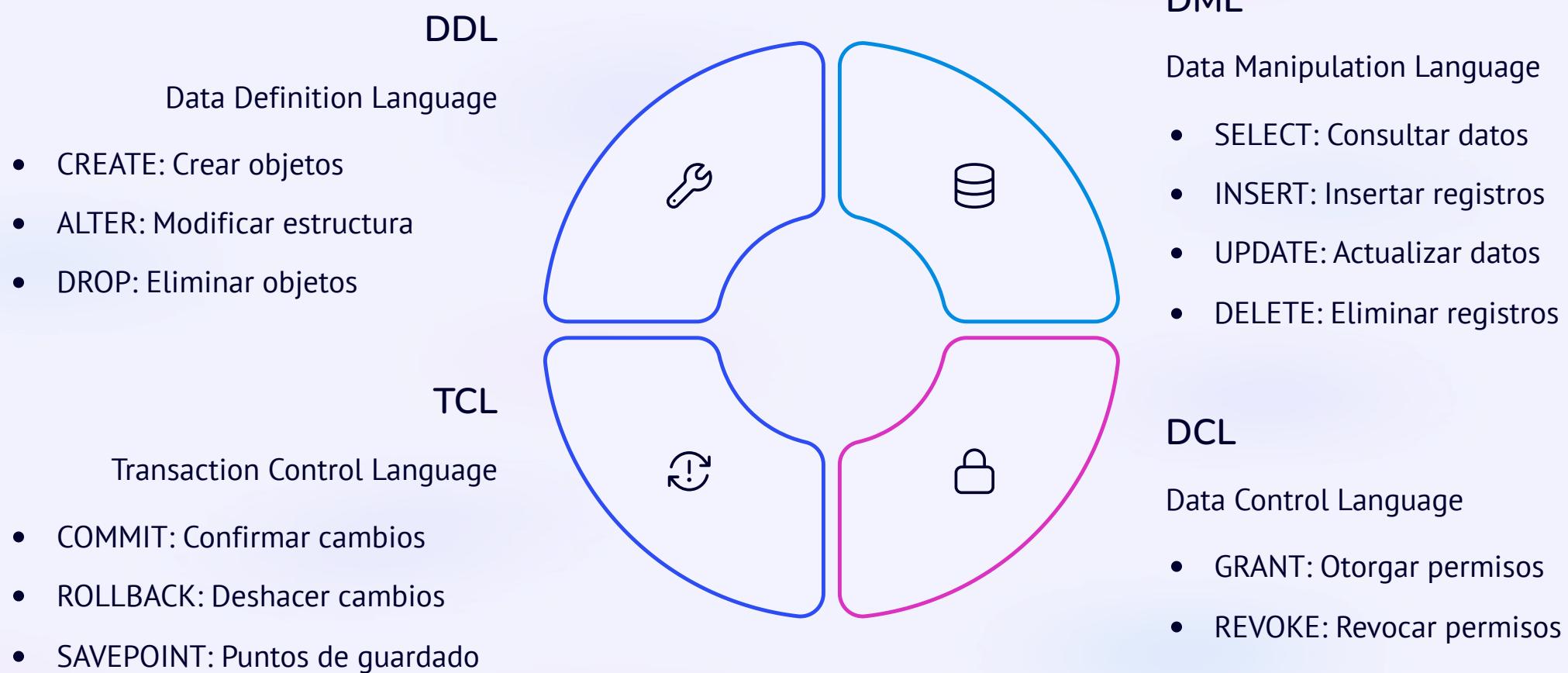
Relaciones

Conexiones lógicas entre tablas (uno a uno, uno a muchos, muchos a muchos) que modelan la realidad.

Database Schema



Tipos de Sentencias SQL



SELECT - La Consulta Fundamental

Sintaxis Básica

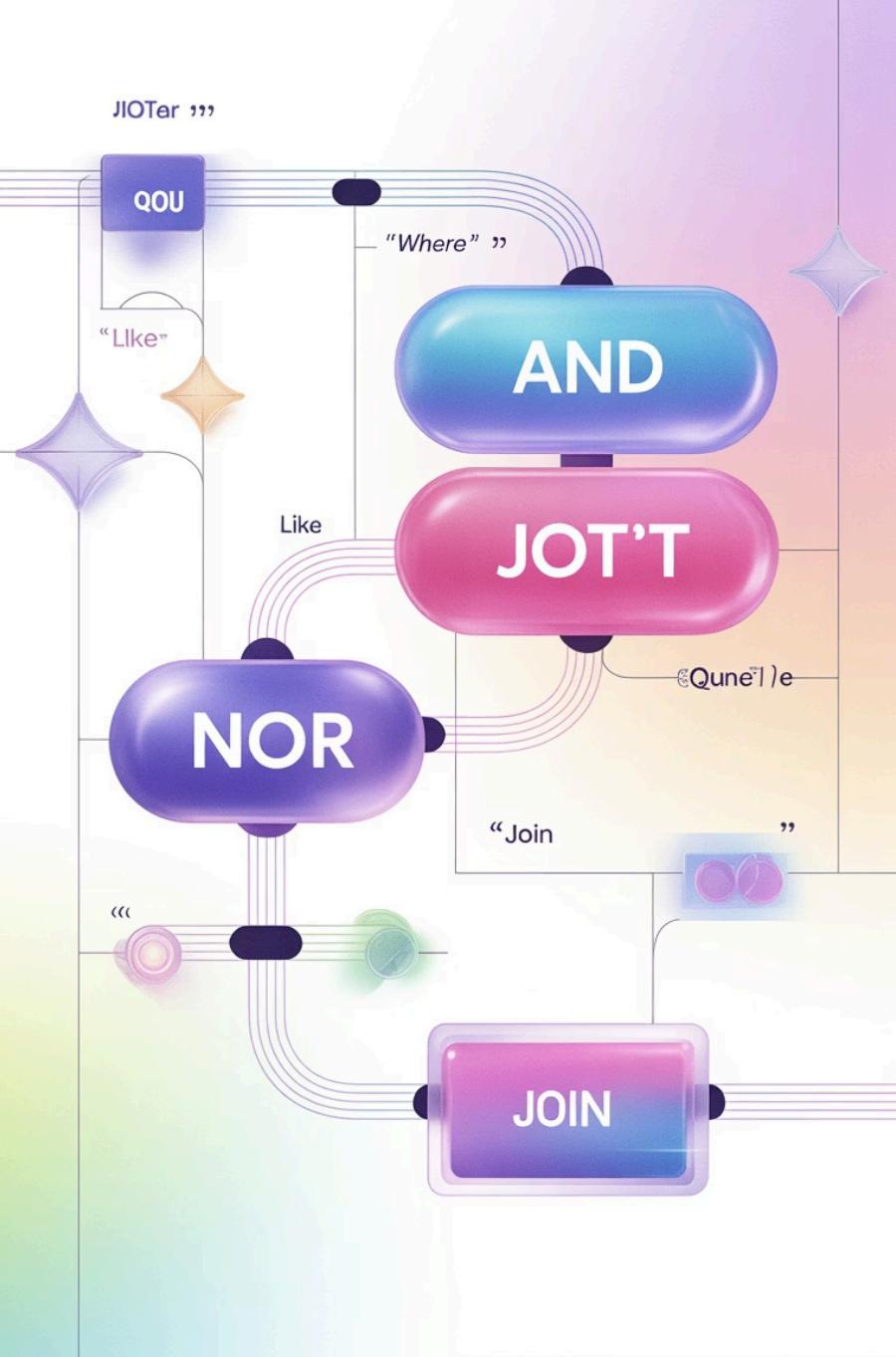
```
SELECT columna1, columna2, ...
FROM tabla
WHERE condición;
```

La sentencia SELECT es la más utilizada en SQL y permite recuperar datos de una o más tablas según criterios específicos.



Ejemplo Práctico

```
SELECT nombre, edad, email
FROM alumnos
WHERE carrera = 'Informática'
AND edad > 20;
```



Operadores y Filtros en Consultas SELECT

Operadores de Comparación

Permiten establecer condiciones precisas para filtrar datos.

- Igualdad: = (igual), <> o != (diferente)
- Comparación: > (mayor), < (menor), >= (mayor o igual), <= (menor o igual)
- Pertenencia: IN (dentro de lista), NOT IN (fuera de lista)

Operadores Lógicos

Combinan múltiples condiciones para crear filtros complejos.

- AND: ambas condiciones deben cumplirse
- OR: al menos una condición debe cumplirse
- NOT: niega una condición

Patrones de Texto

Búsqueda flexible de texto con comodines.

- LIKE '%texto%': contiene "texto" en cualquier posición
- LIKE 'texto%': comienza con "texto"
- LIKE '%texto': termina con "texto"

INSERT - Agregando Nuevos Datos



Sintaxis y Ejemplos

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

La sentencia INSERT permite agregar nuevos registros a una tabla existente.

Ejemplo Práctico

```
INSERT INTO alumnos  
(nombre, apellido, edad, carrera)  
VALUES  
('Lucía', 'García', 22, 'Informática');
```

También es posible insertar múltiples registros en una sola operación:

```
INSERT INTO alumnos (nombre, edad)  
VALUES  
('Carlos', 19),  
('María', 21);
```

UPDATE - Modificando Datos Existentes

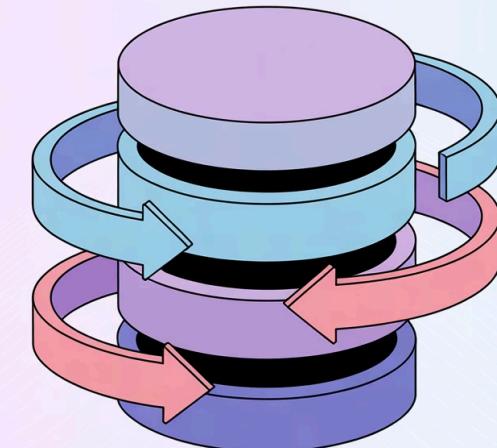
Sintaxis y Uso

```
UPDATE tabla  
SET columna1 = valor1, columna2 = valor2, ...  
WHERE condición;
```

La sentencia UPDATE modifica registros existentes que cumplan con determinada condición.

¡Atención! Si se omite la cláusula WHERE, el cambio afectará a *todos* los registros de la tabla.

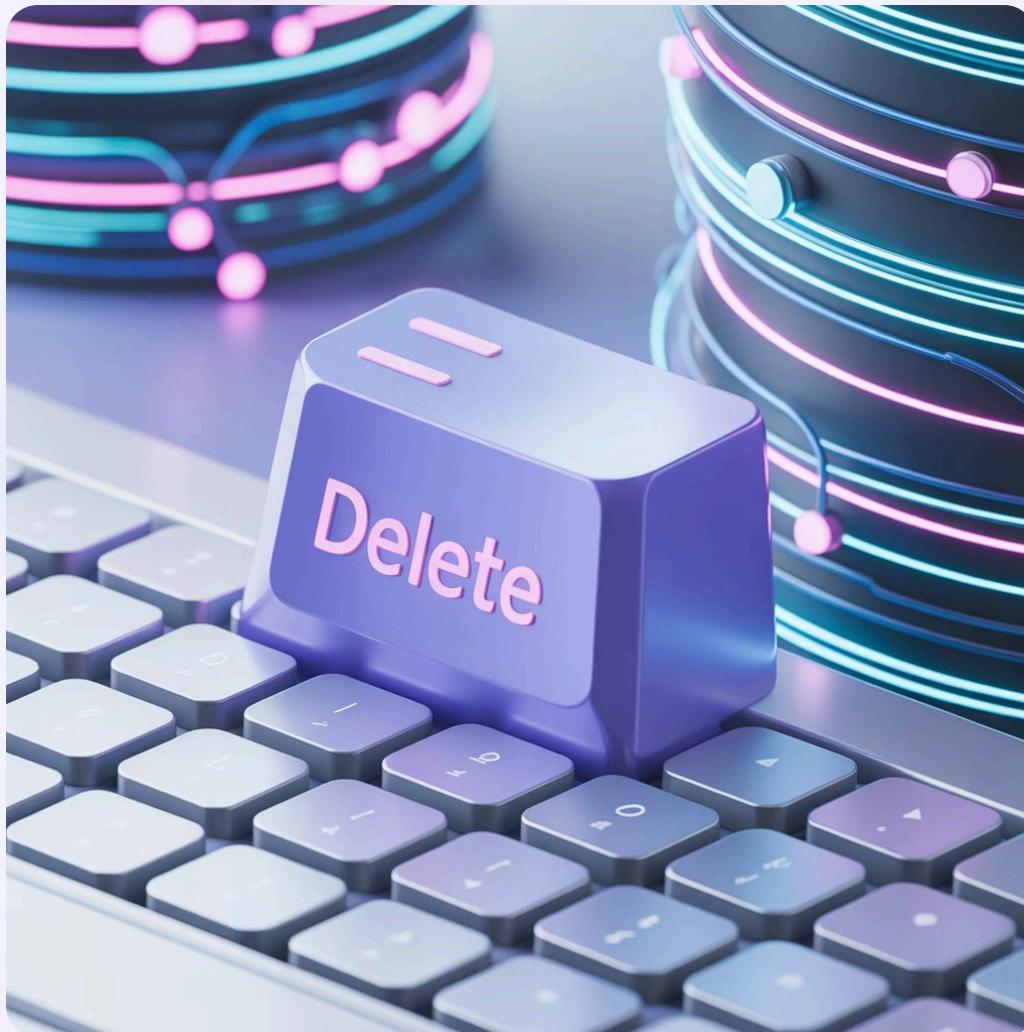
Database Update



Ejemplo Práctico

```
UPDATE alumnos  
SET  
edad = 23,  
email = 'lucia.garcia@universidad.es'  
WHERE nombre = 'Lucía' AND apellido = 'García';
```

DELETE - Eliminando Registros



Sintaxis y Consideraciones

```
DELETE FROM tabla  
WHERE condición;
```

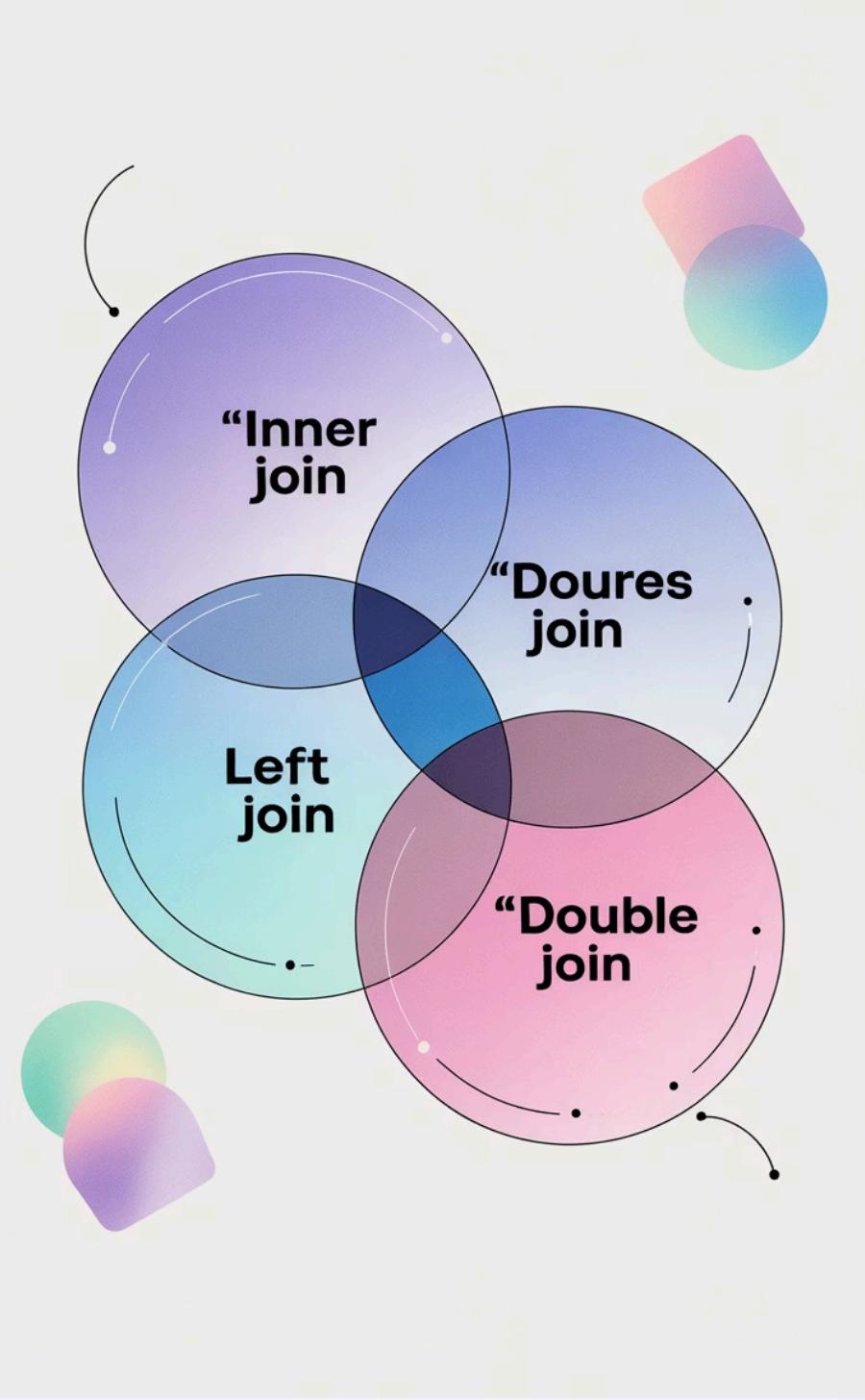
La sentencia DELETE elimina registros de una tabla que cumplan con la condición especificada.

Advertencia: Si se omite WHERE, se eliminarán *todos* los registros. Esta operación generalmente no se puede deshacer sin una copia de seguridad.

Ejemplo Práctico

```
DELETE FROM alumnos  
WHERE edad < 18;
```

Recomendación: antes de ejecutar DELETE, verifique los datos con SELECT.



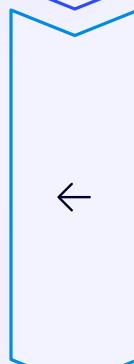
JOINS - Uniendo Tablas



INNER JOIN

Devuelve registros cuando hay coincidencias en ambas tablas.

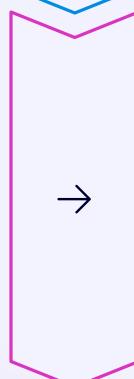
```
SELECT a.nombre, c.nombre_carrera  
FROM alumnos a  
INNER JOIN carreras c  
ON a.id_carrera = c.id;
```



LEFT JOIN

Devuelve todos los registros de la tabla izquierda y los que coinciden de la derecha.

```
SELECT a.nombre, c.nombre_carrera  
FROM alumnos a  
LEFT JOIN carreras c  
ON a.id_carrera = c.id;
```



RIGHT JOIN

Devuelve todos los registros de la tabla derecha y los que coinciden de la izquierda.

```
SELECT a.nombre, c.nombre_carrera  
FROM alumnos a  
RIGHT JOIN carreras c  
ON a.id_carrera = c.id;
```

Modelo de Datos de Ejemplo

Para nuestros ejemplos, utilizaremos un modelo simplificado de gestión académica:

Tabla: alumnos

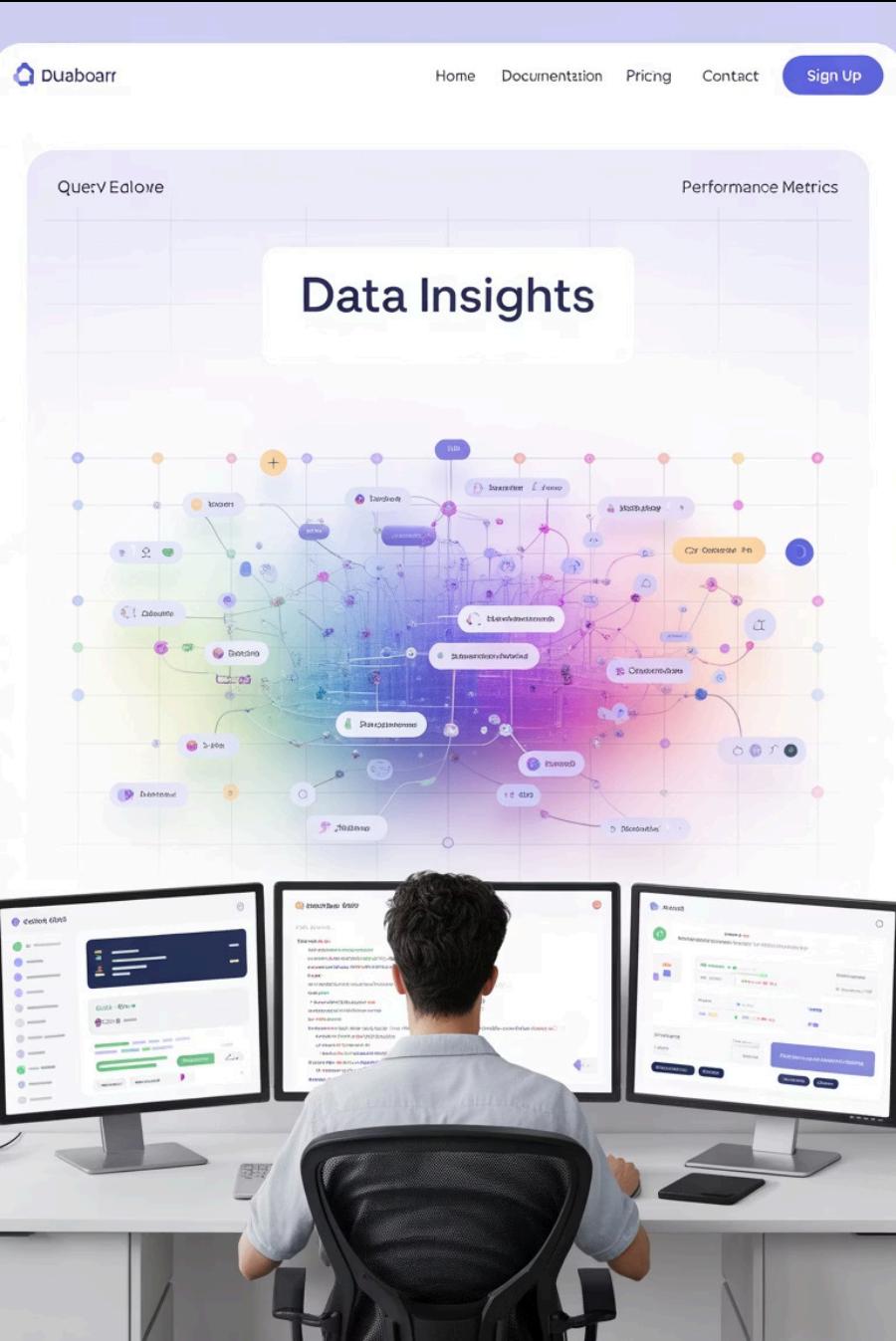
- id (PK): Identificador único
- nombre: Nombre del alumno
- apellido: Apellido del alumno
- edad: Edad del alumno
- id_carrera (FK): Referencia a carrera

Tabla: carreras

- id (PK): Identificador único
- nombre_carrera: Nombre de la carrera
- duración: Duración en años
- departamento: Departamento responsable

Tabla: asignaturas

- id (PK): Identificador único
- nombre: Nombre de la asignatura
- créditos: Número de créditos
- id_carrera (FK): Carrera asociada



Implementación de JOINS en Consultas Complejas

Ejemplo de Consulta con Múltiples JOINS

```
SELECT
    a.nombre,
    a.apellido,
    c.nombre_carrera,
    asig.nombre AS asignatura,
    m.calificación
FROM alumnos a
INNER JOIN carreras c ON a.id_carrera = c.id
INNER JOIN matriculas m ON a.id = m.id_alumno
INNER JOIN asignaturas asig ON m.id_asignatura = asig.id
WHERE c.nombre_carrera = 'Informática'
ORDER BY a.apellido, a.nombre, asig.nombre;
```

Esta consulta recupera todos los alumnos de Informática junto con las asignaturas en las que están matriculados y sus calificaciones, ordenando los resultados alfabéticamente.

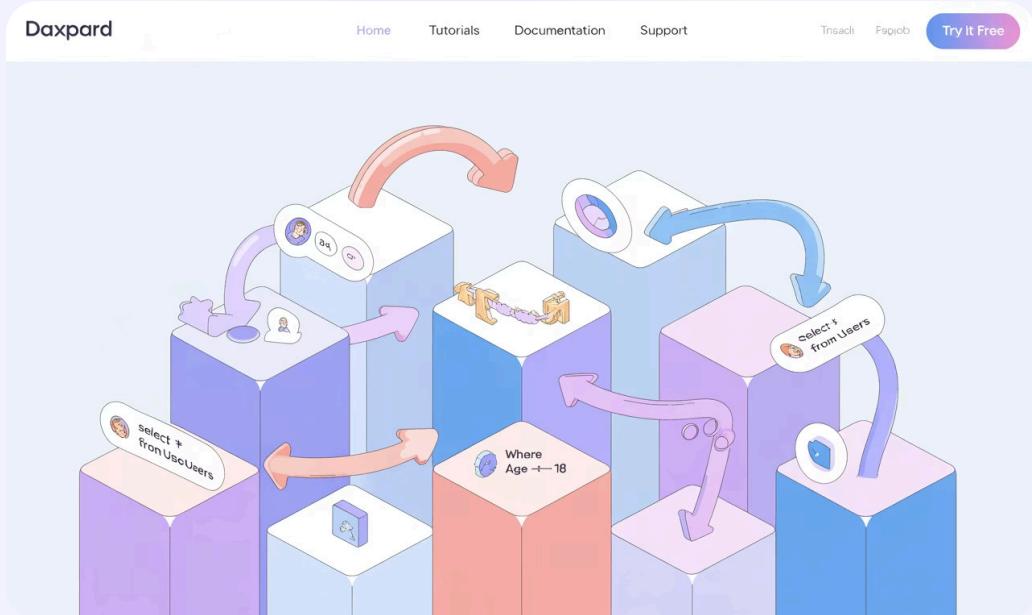
Subconsultas - Consultas Anidadas

¿Qué son las subconsultas?

Las subconsultas son consultas SQL anidadas dentro de otra consulta principal. Permiten realizar operaciones complejas utilizando el resultado de una consulta como parte de otra.

Pueden utilizarse en diversas cláusulas como:

- En la cláusula WHERE
- En la cláusula FROM (subconsulta como tabla)
- En la cláusula SELECT



Ejemplo Práctico

```
-- Alumnos con edad superior a la media
SELECT nombre, edad
FROM alumnos
WHERE edad > (
    SELECT AVG(edad)
    FROM alumnos
);
```

Funciones de Agregación en SQL

¿Qué son las funciones de agregación?

Las funciones de agregación realizan cálculos sobre conjuntos de valores y devuelven un único resultado.

Funciones para contar registros



COUNT()

Cuenta el número de filas o valores no nulos.

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

Funciones para cálculos numéricos



SUM()

Calcula la suma de los valores de una columna numérica.

```
SELECT SUM(créditos)  
FROM asignaturas;
```



AVG()

Calcula el promedio de los valores de una columna numérica.

```
SELECT AVG(edad)  
FROM alumnos;
```

Funciones para valores extremos



MAX()

Encuentra el valor máximo de una columna.

```
SELECT MAX(calificación)  
FROM matriculas;
```

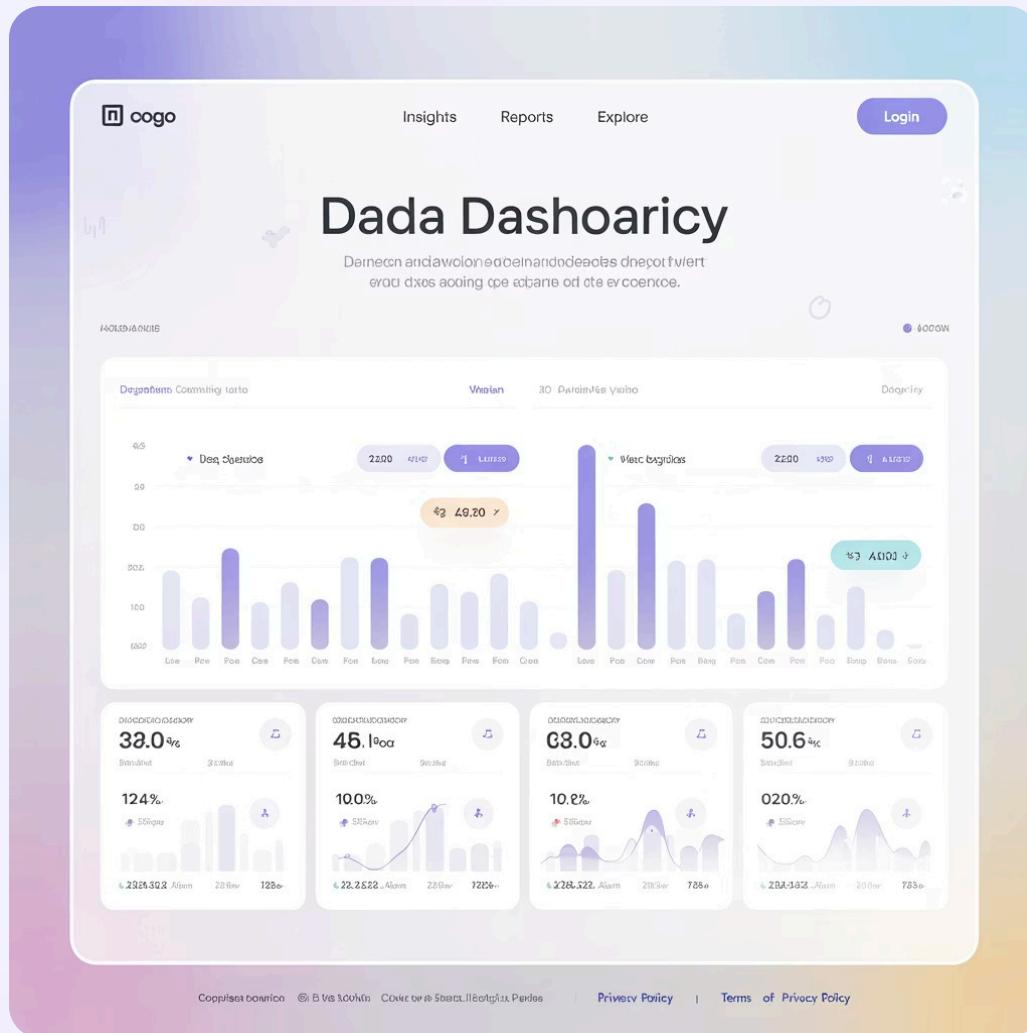


MIN()

Encuentra el valor mínimo de una columna.

```
SELECT MIN(calificación)  
FROM matriculas;
```

GROUP BY - Agrupación de Resultados



Agrupando Datos

La cláusula GROUP BY permite agrupar filas con valores idénticos en la columna especificada, aplicando funciones de agregación a cada grupo.

```
SELECT  
carrera,  
COUNT(*) as total_alumnos  
FROM alumnos  
GROUP BY carrera;
```

Filtrado con HAVING

La cláusula HAVING filtra los resultados agrupados, mientras que WHERE filtra antes de la agrupación.

```
SELECT  
carrera,  
COUNT(*) as total_alumnos  
FROM alumnos  
GROUP BY carrera  
HAVING COUNT(*) > 10;
```

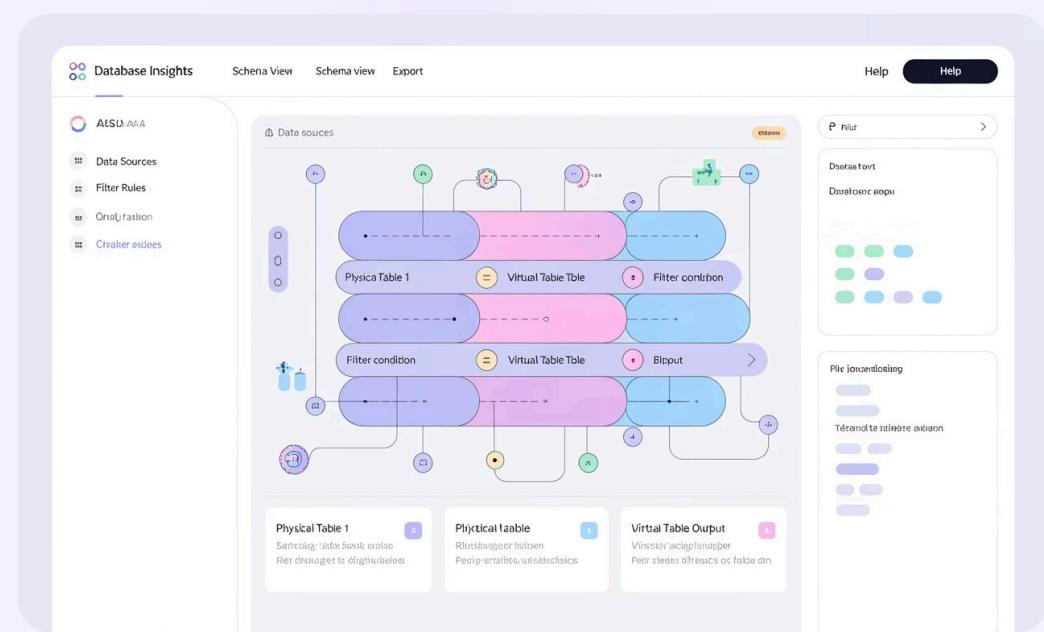
Vistas (Views) - Consultas Predefinidas

¿Qué son las vistas?

Las vistas son consultas SQL almacenadas que actúan como tablas virtuales. No contienen datos físicamente, sino que muestran datos de otras tablas de forma dinámica.

Ventajas

- Simplifican consultas complejas
- Mejoran la seguridad ocultando tablas base
- Proporcionan abstracción de datos
- Facilitan la compatibilidad con versiones anteriores



Sintaxis y Ejemplo

```
CREATE VIEW nombre_vista AS  
SELECT ...;
```

```
-- Vista de alumnos activos  
CREATE VIEW vista_alumnos_activos AS  
SELECT nombre, apellido, carrera  
FROM alumnos  
WHERE estado = 'activo';
```

```
-- Consultar la vista  
SELECT * FROM vista_alumnos_activos;
```

Índices - Optimizando el Rendimiento



¿Qué son los índices?

Los índices son estructuras de datos especiales que mejoran la velocidad de recuperación de datos, similar a un índice en un libro.

Tipos de Índices

- Índices de clave primaria (automáticos)
- Índices únicos
- Índices compuestos (múltiples columnas)
- Índices de texto completo

Creación de Índices

```
CREATE INDEX idx_apellido  
ON alumnos(apellido);
```

Los índices aceleran las consultas pero pueden ralentizar las operaciones de escritura (INSERT, UPDATE, DELETE).



Buenas Prácticas en SQL



Optimización de Consultas

- Selecciona solo las columnas necesarias (evita SELECT *)
- Utiliza índices adecuadamente
- Limita los resultados con WHERE cuando sea posible
- Usa JOINs en lugar de subconsultas cuando sea adecuado



Seguridad y Prevención

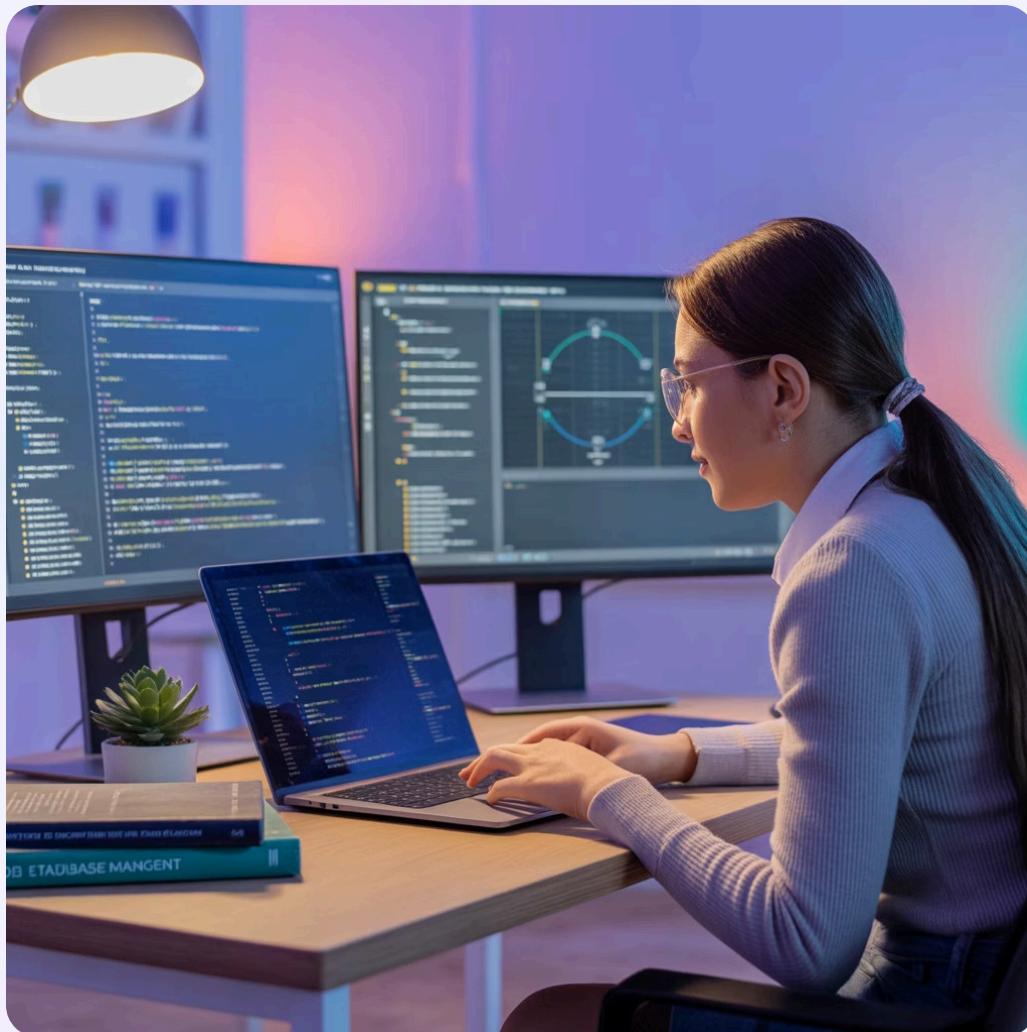
- Usa consultas parametrizadas para evitar inyección SQL
- Implementa control de acceso con GRANT/REVOKE
- Realiza copias de seguridad regularmente
- Prueba las sentencias DELETE/UPDATE con SELECT primero



Legibilidad y Mantenimiento

- Usa indentación consistente
- Incluye comentarios para explicar consultas complejas
- Utiliza nombres descriptivos para tablas y columnas
- Mantén un estilo coherente (mayúsculas para palabras clave)

Recursos para Seguir Aprendiendo



Próximos Pasos

Para profundizar en SQL, recomendamos:
Practicar con bases de datos de ejemplo.