Bases de Datos: Sistemas de Gestión y DDL

Bienvenidos a esta presentación sobre el Lenguaje de Definición de Datos (DDL) en sistemas de gestión de bases de datos. Exploraremos las principales sentencias, restricciones y buenas prácticas para el diseño eficiente de bases de datos.

Sentencias Principales del DDL

CREATE TABLE

Instrucción principal para crear una tabla con nombre único y definir sus columnas.

ALTER TABLE

Permite modificar la estructura de una tabla existente.

SHOW CREATE TABLE

Muestra la sentencia completa de creación de una tabla con todas sus restricciones.

DROP TABLE

Elimina una tabla de la base de datos.

Estas sentencias son fundamentales para la creación y gestión de la estructura de la base de datos.

Estructura de CREATE TABLE

```
CREATE TABLE nombre_tabla (
  columna1 tipo_dato [opciones],
  columna2 tipo_dato [opciones],
  ...
  [CONSTRAINT nombre_constraint tipo_constraint (columnas)]
);
```

Donde:

- nombre_tabla: Nombre único asignado a la tabla
- columna1, columna2: Nombres únicos de las columnas
- tipo_dato: INT, VARCHAR, DATE, DECIMAL, etc.
- opciones: NOT NULL, DEFAULT, PRIMARY KEY, etc.
- CONSTRAINT: Define restricciones a nivel de tabla

Tipos de Datos Fundamentales

1

Datos Numéricos Enteros

- TINYINT
- SMALLINT
- INT
- BIGINT

2

Datos Numéricos Reales

- DECIMAL(p,d)
- Indica tamaño de parte entera y decimal

3

Cadenas

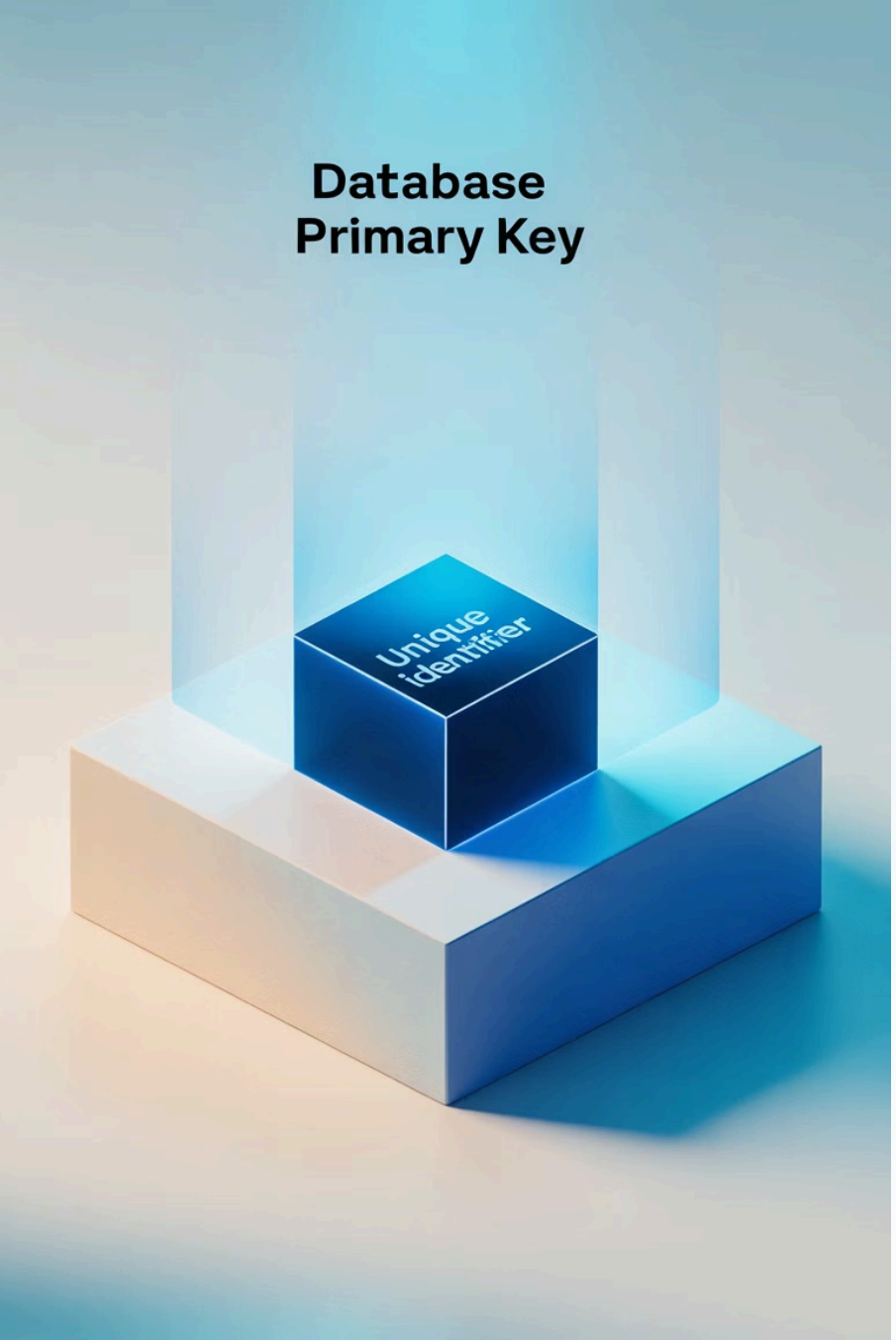
- VARCHAR
- CHAR
- TEXT

4

Fechas

- DATE
- DATETIME
- TIMESTAMP

La elección adecuada del tipo de dato es crucial para la eficiencia y el rendimiento de la base de datos.



Claves Primarias (PRIMARY KEY)

Una clave primaria permite identificar de manera única cada fila en una tabla. Características importantes:

- Una tabla solo puede tener una clave primaria
- Las columnas que forman la PK no pueden contener valores NULL
- Puede estar formada por una o varias columnas

Sintaxis a nivel de columna:

```
columna tipo_dato PRIMARY KEY
```

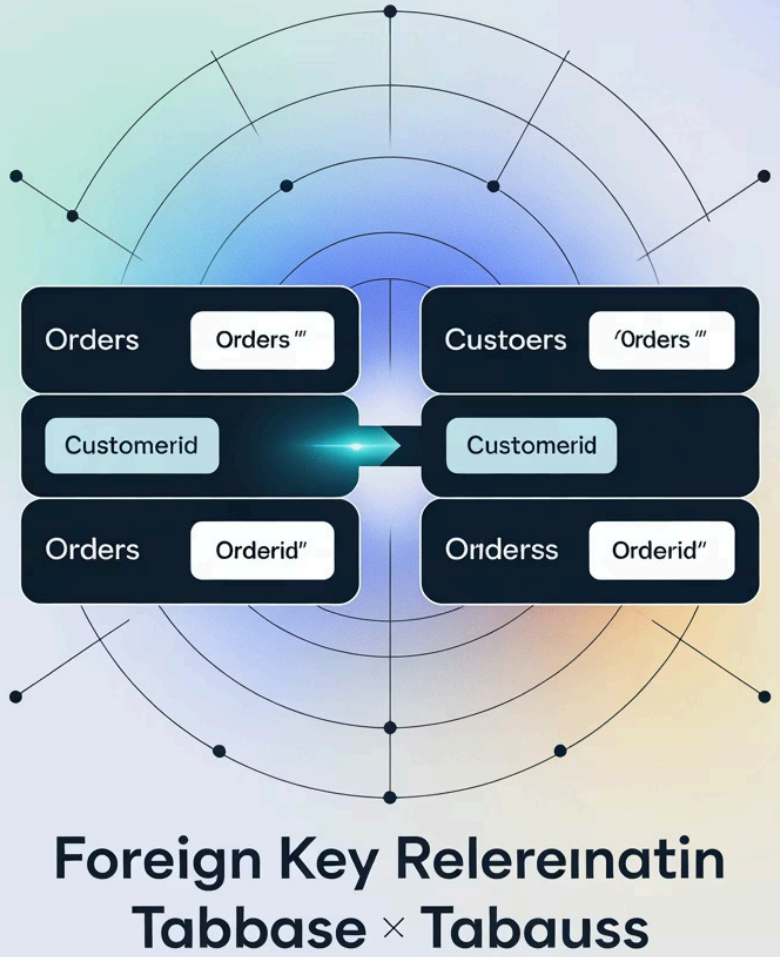
Sintaxis a nivel de tabla:

```
CONSTRAINT pk_nombre_tabla
PRIMARY KEY (columna1,
columna2, ...)
```

Claves Foráneas (FOREIGN KEY)

```
CONSTRAINT fk_nombre_restriccion
FOREIGN KEY (columna_local)
REFERENCES tabla_referenciada (columna_referenciada)
ON DELETE opcion
ON UPDATE opcion
```

Las claves foráneas establecen relaciones entre tablas, garantizando la integridad referencial. Permiten definir comportamientos específicos cuando se eliminan o actualizan registros relacionados.



Opciones ON DELETE: RESTRICT

ON DELETE RESTRICT

Impide la eliminación de la fila en la tabla madre si existen filas relacionadas en la tabla hija.

Cuándo usar: Cuando es crucial mantener la integridad referencial y evitar eliminaciones accidentales que podrían dejar datos huérfanos.

Es el comportamiento por defecto en muchos SGBDs si no se especifica ON DELETE.

```
-- Ejemplo:
CREATE TABLE clientes (
 cliente_id INT PRIMARY KEY,
 nombre VARCHAR(255)
);

CREATE TABLE pedidos (
 pedido_id INT PRIMARY KEY,
 cliente_id INT,
 FOREIGN KEY (cliente_id)
 REFERENCES clientes (cliente_id)
 ON DELETE RESTRICT
);
```

Ejemplo de ON DELETE RESTRICT

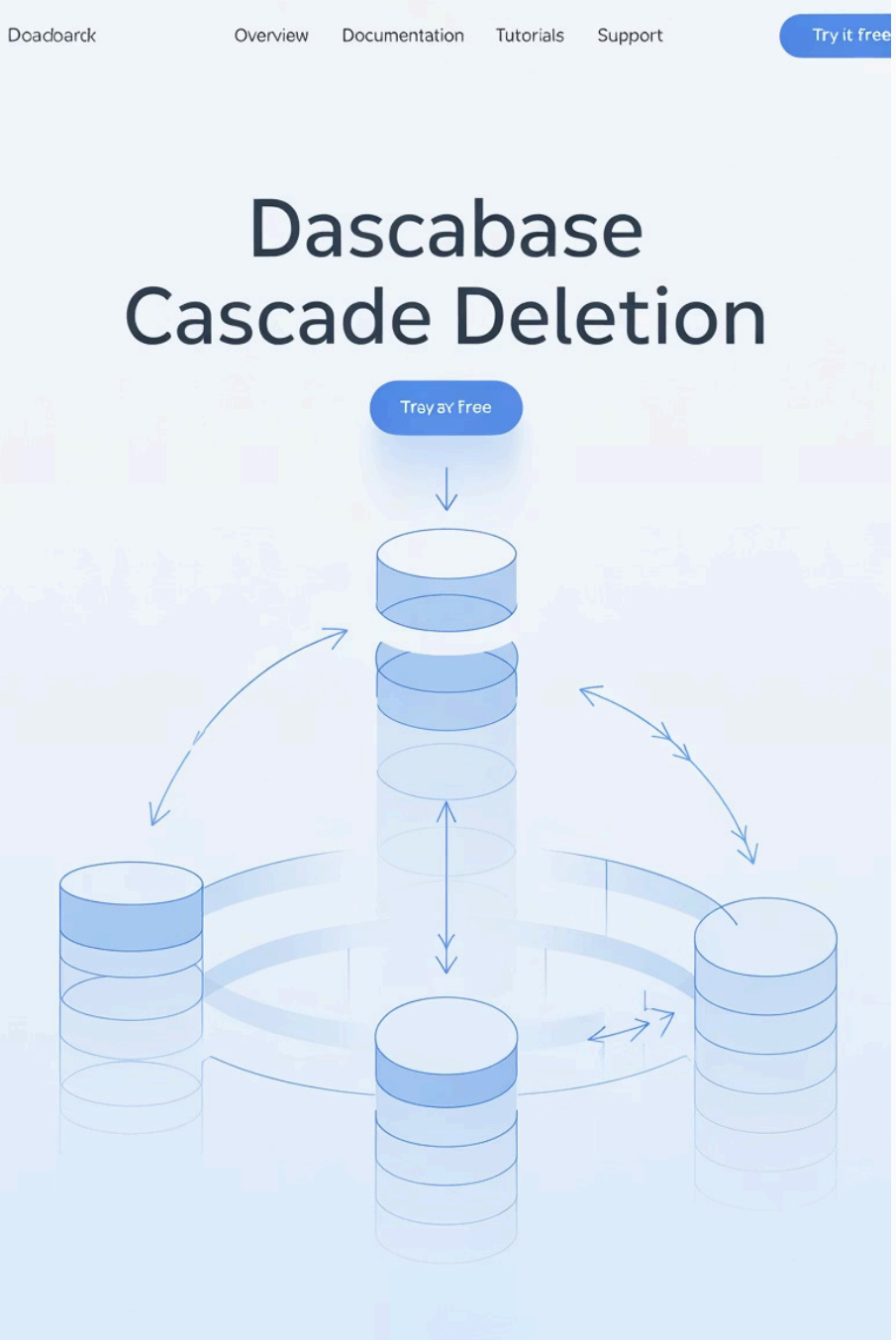
```
-- Insertamos algunos datos
INSERT INTO clientes (cliente_id, nombre)
VALUES (1, 'Juan Pérez');

INSERT INTO pedidos (pedido_id, cliente_id)
VALUES (101, 1);

-- Intentamos eliminar el cliente con ID 1
DELETE FROM clientes WHERE cliente_id = 1;
-- ¡Esto fallará!
-- Error: No se puede eliminar el cliente
-- porque tiene pedidos asociados.
```

En este caso, la instrucción DELETE fallará porque la restricción ON DELETE RESTRICT impide eliminar el cliente mientras existan pedidos asociados a él.

Primero deberías eliminar los pedidos del cliente, y luego podrías eliminar al cliente.



Opciones ON DELETE: CASCADE

ON DELETE CASCADE

Elimina automáticamente las filas relacionadas en la tabla hija cuando se elimina la fila en la tabla madre.

Cuándo usar: Cuando la eliminación de una fila en la tabla madre implica lógicamente la eliminación de las filas relacionadas en la tabla hija.

¡Precaución! Puede producir eliminación de datos no deseada, debemos estar muy seguros en estos casos.

```
CREATE TABLE clientes (
 cliente_id INT PRIMARY KEY,
 nombre VARCHAR(255)
);

CREATE TABLE pedidos (
 pedido_id INT PRIMARY KEY,
 cliente_id INT,
 FOREIGN KEY (cliente_id)
 REFERENCES clientes (cliente_id)
 ON DELETE CASCADE
);
```

ON UPDATE CASCADE

Si se modifica la clave primaria en la tabla referenciada, la misma modificación se propaga a la tabla que hace referencia a esa clave foránea.

Por ejemplo, si se cambia el ID de un cliente en la tabla de clientes, se actualizarán todos los pedidos relacionados con ese cliente en la tabla de pedidos.

1

Actualización en tabla madre

Se modifica el ID de un cliente de 1 a 100

2

Propagación automática

El sistema detecta la relación y aplica ON UPDATE CASCADE

3

Actualización en tabla hija

Todos los pedidos con cliente_id=1 ahora tienen cliente_id=100

Cuándo usar RESTRICT vs CASCADE

Usar RESTRICT cuando:

- La integridad referencial es crítica
- Se quiere evitar la eliminación accidental de datos relacionados
- La eliminación en la tabla madre no implica lógicamente la eliminación en la tabla hija
- Se necesita control estricto sobre las eliminaciones

Ejemplo: En una aplicación bancaria, no permitir la eliminación de un cliente si tiene cuentas bancarias activas.

Usar CASCADE cuando:

- Existe certeza de que la eliminación en la tabla madre implica lógicamente la eliminación en la tabla hija
- Se busca simplificar el proceso de eliminación
- Se han analizado completamente las implicaciones de eliminar datos en cascada

Ejemplo: En un sistema de gestión de tareas, al eliminar un proyecto, eliminar automáticamente todas las tareas asociadas.

Buenas Prácticas para Restricciones

1

Planificación

Analizar cuidadosamente las relaciones entre las tablas y cómo deben comportarse las eliminaciones antes de definir las restricciones FOREIGN KEY.

2

Documentación

Documentar las restricciones FOREIGN KEY y las opciones ON DELETE para que otros desarrolladores entiendan el comportamiento de la base de datos.

3

Pruebas

Realizar testeos exhaustivos para asegurar que las restricciones FOREIGN KEY y las opciones ON DELETE funcionan como se espera.

Estas prácticas ayudan a mantener la integridad de los datos y evitar problemas futuros.

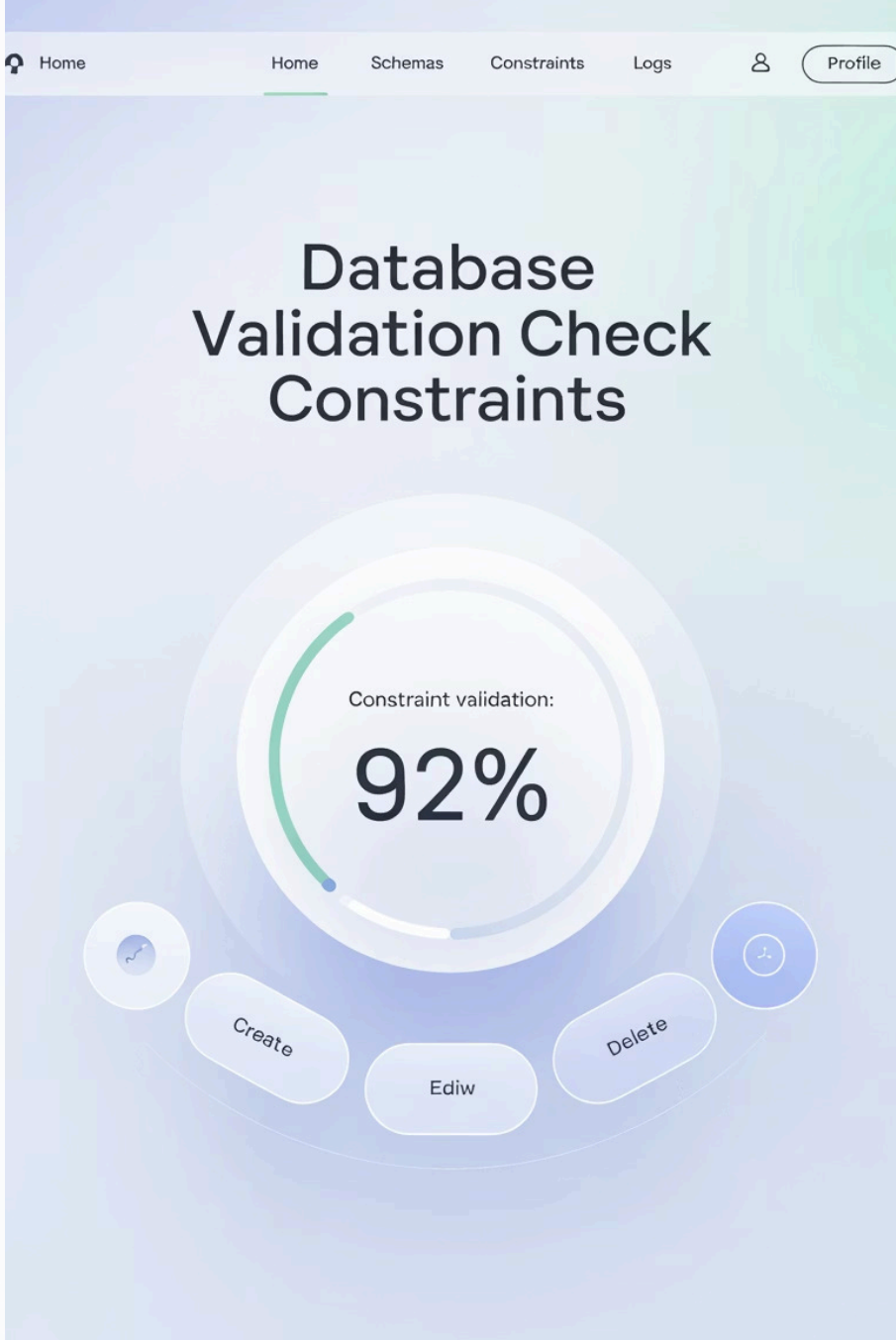
Restricción CHECK

La condición CHECK define una restricción de integridad que se puede aplicar a columnas o tablas para asegurar que los datos insertados cumplan ciertos criterios de validación.

Si se intenta insertar o actualizar un valor que no cumpla con la condición CHECK, se producirá un error y la operación será rechazada.

En MySQL, las restricciones CHECK existen desde la versión 8.0.16, pero con limitaciones importantes.

```
CREATE TABLE productos (
 codigo_producto VARCHAR(9)
PRIMARY KEY,
 CONSTRAINT ck_codigo_formato
CHECK (
 LENGTH(codigo_producto) = 9 AND
 SUBSTRING(codigo_producto, 2, 1)
= '-' AND
 SUBSTRING(codigo_producto, 8, 1)
= '-' AND
 SUBSTRING(codigo_producto, 1, 1)
 BETWEEN 'A' AND 'Z' AND
 SUBSTRING(codigo_producto, 9, 1)
 BETWEEN 'A' AND 'Z'
 )
);
```



Probando la Restricción CHECK

```
-- Insertamos valores para probar la restricción CHECK
INSERT INTO productos (codigo_producto) VALUES ('A-12345-Z');
-- Correcto: Cumple con el formato A-12345-Z

INSERT INTO productos (codigo_producto) VALUES ('B-54321-B');
-- Error: Primera y última letra son iguales

INSERT INTO productos (codigo_producto) VALUES ('1-12345-Z');
-- Error: Primer carácter no es una letra

INSERT INTO productos (codigo_producto) VALUES ('A12345Z');
-- Error: No tiene el formato con guiones
```

Solo el primer valor es correcto y cumple con todas las restricciones definidas. El resto generará mensajes de error específicos.

Para visualizar la estructura completa de la tabla con sus restricciones y chequeos, usar: SHOW CREATE TABLE productos;

Índices: Optimizando el Rendimiento

Los índices son estructuras especiales que se utilizan para localizar rápidamente las filas de una tabla. Sin índices, el SGBD tendría que escanear toda la tabla para encontrar los datos buscados.

Son como el catálogo de una biblioteca que nos permite encontrar rápidamente un libro sin revisar toda la colección.

Sentencias principales:

- CREATE INDEX
- ALTER TABLE ... ADD INDEX
- DROP INDEX
- SHOW INDEX FROM tabla

Creación de Índices

```
CREATE [UNIQUE | FULLTEXT | SPATIAL] INDEX nombre_indice
ON nombre_tabla (columna1 [ASC | DESC], columna2 [ASC | DESC], ...);
```

UNIQUE

Garantiza que todos los valores en la columna (o combinación de columnas) indexadas sean únicos. Útil para evitar datos repetidos y para búsquedas rápidas.

FULLTEXT

Se utiliza para indexar columnas que contienen texto largo (TEXT o VARCHAR) y realizar búsquedas de texto completas (MATCH...AGAINST). Para búsquedas tipo "Google" dentro de tus datos.

SPATIAL

Para columnas que almacenan datos geográficos (coordenadas, formas, etc.) y realizar consultas espaciales. Útil para aplicaciones con mapas o ubicaciones.

Visualización de Índices

Para ver la definición de los índices de una tabla y poder optimizar nuestras consultas:

```
SHOW INDEX FROM nombre_tabla;
```

Si no conocemos los índices disponibles, difícilmente podamos crear consultas óptimas.

Para analizar cómo MySQL está ejecutando tus consultas y detectar problemas de rendimiento:

```
EXPLAIN SELECT * FROM productos
WHERE precio > 1000
ORDER BY fecha_lanzamiento DESC;
```

La salida de EXPLAIN tendrá pistas sobre dónde enfocar los esfuerzos de indexación.

Buenas Prácticas para Índices

1 Analizar las consultas más lentas

Usar EXPLAIN para ver cómo MySQL está ejecutando tus consultas. Si ves "Using filesort", "Using temporary", o un type de ALL (full table scan), es una señal de que necesitas un índice.

2 Indexar columnas estratégicas

Priorizar las columnas en tus cláusulas WHERE, JOIN, ORDER BY y GROUP BY, son las principales candidatas a ser indexadas.

3 Considerar índices compuestos

Si tus consultas a menudo filtran u ordenan por varias columnas juntas, un índice compuesto puede ser muy beneficioso. Recuerda el "orden de las columnas" en el índice compuesto.

4 No indexar en exceso

Demasiados índices pueden ser contraproducentes. Ralentizan las operaciones de escritura y consumen mucho espacio. Sé selectivo.

Más Buenas Prácticas para Índices

Evitar indexar columnas con pocos valores únicos

Si una columna solo tiene dos valores (por ejemplo, activo SI/NO), un índice en esa columna probablemente no te dará mucho beneficio, ya que se tendría que escanear la mitad de la tabla.

Usar tipos de datos correctos

Utilizar el tipo de dato más pequeño y apropiado para tus columnas también contribuye a un mejor rendimiento del índice y de la tabla en general. Por ejemplo, INT en lugar de BIGINT si tus números no serán tan grandes.

Mantenimiento de la base de datos

A medida que los datos cambian, los índices pueden fragmentarse. Considera OPTIMIZE TABLE para reorganizar el espacio de disco y mejorar el rendimiento en tablas con muchas eliminaciones o actualizaciones.

Conclusiones

> "Entender y aplicar los principios de indexación es una de las habilidades más valiosas que puedes desarrollar como programador para optimizar el rendimiento de las aplicaciones."

Diseño Estructurado

Utiliza CREATE TABLE con restricciones adecuadas para garantizar la integridad de los datos desde el principio.

Integridad Referencial

Implementa FOREIGN KEY con las opciones ON DELETE/UPDATE apropiadas según la lógica de negocio.

Rendimiento Optimizado

Crea índices estratégicamente para mejorar la velocidad de las consultas más frecuentes.

Empieza pequeño, experimenta con EXPLAIN, y poco a poco irás desarrollando la habilidad para saber cuándo y dónde crear índices.