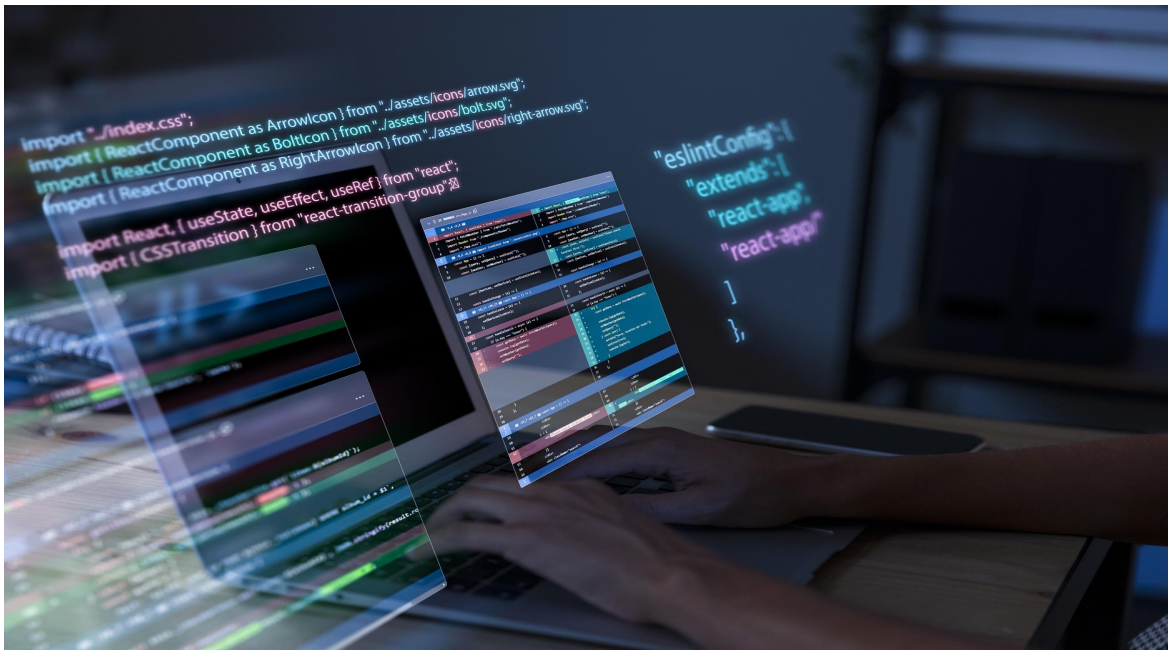
PROMETEO

2º ASIR /ASGB

UNIDAD 6

Rendimiento y optimización



Autor: Luz María Álvarez Moreno

Fecha: 20/11/2025

Sesión 8 – Índices, Estadísticas, Particionamiento y Optimizador de Consultas

1. ¿Qué es un índice?

Un **índice** es una estructura de datos adicional creada para acelerar la búsqueda de registros en una tabla.
MySQL suele usar **B-Tree**, una estructura optimizada para encontrar datos en muy pocos pasos.

Analogía sencilla

Una tabla sin índice es como un **libro sin índice**:

- Para encontrar algo, debes leer página por página.
  Una tabla con índice es como un **libro con índice temático**:
- Te dice exactamente dónde buscar.

Ejemplo

Tabla alumnos con 10.000 registros.

SELECT * FROM alumnos WHERE dni = '12345678A';

- Sin índice : MySQL recorrerá **toda la tabla** (full table scan).
- Con índice en *dni* : MySQL encuentra el registro en **milisegundos**.

Crear un índice

CREATE INDEX idx_dni ON alumnos(dni);

Importante

- **Los índices aceleran las lecturas**

- **Pero hacen más lentas las escrituras** (INSERT, UPDATE, DELETE), porque el índice también debe actualizarse.
- No conviene indexar todo.

Cuándo crear un índice

- Columnas usadas en WHERE
- Columnas usadas en JOIN
- Columnas usadas en ORDER BY / GROUP BY
- Campos de búsqueda únicos (dni, email, username)

2. Tipos de índices

Índice simple

```
CREATE INDEX idx_nombre ON empleados(nombre);
```

Índice compuesto (más de una columna)

```
CREATE INDEX idx_apellidos_nombre ON empleados(apellido, nombre);
```

El orden importa: el índice sirve para consultas que comienzan por la **primera** columna.

Índice único

Evita duplicados.

```
CREATE UNIQUE INDEX idx_email ON usuarios(email);
```

Índice FULLTEXT

Para búsquedas de texto largo (blogs, descripciones).

CREATE FULLTEXT INDEX idx_texto ON articulos(contenido);

Índice PRIMARY KEY

- Es único
- No admite NULL
- MySQL lo usa como índice principal interno

3. Estadísticas en MySQL

Las **estadísticas** son datos que MySQL guarda sobre:

- cuántas filas tiene la tabla
- distribución de valores
- cardinalidad de columnas
- rangos de datos

El optimizador de consultas **usa estas estadísticas para decidir** cómo ejecutar una consulta.

Analogía

Es como un GPS que elige la ruta en función del tráfico.
Si la información está desactualizada, el GPS te manda por el peor camino.

Ver estadísticas de una tabla

SHOW INDEX FROM alumnos;

ANALYZE TABLE alumnos;

Forzar actualización de estadísticas

ANALYZE TABLE alumnos;

4. Particionamiento en MySQL

El **particionamiento** divide una tabla muy grande en partes más pequeñas (particiones), pero MySQL la sigue viendo como **una sola tabla**.

Analogía

Imagina un archivo de miles de papeles.
Si lo divides en carpetas por años (2019, 2020, 2021...), buscar un papel concreto es mucho más rápido.

¿Por qué particionar?

- **Tablas enormes** (millones o cientos de millones de filas)
- Mejor rendimiento en búsquedas
- Consultas por rangos más rápidas
- Limpieza de datos más fácil (DROP PARTITION en vez de DELETE masivo)

Tipos de particionamiento

- **RANGE**
    - **Divide por rangos.**

```
CREATE TABLE ventas (
  id INT,
  fecha DATE
)
PARTITION BY RANGE (YEAR(fecha)) (
  PARTITION p2019 VALUES LESS THAN (2020),
  PARTITION p2020 VALUES LESS THAN (2021),
  PARTITION p2021 VALUES LESS THAN (2022)
);
```

- **LIST**
    - **Por valores concretos.**

```
PARTITION BY LIST (pais) (
  PARTITION p_es VALUES IN ('España'),
  PARTITION p_fr VALUES IN ('Francia')
);
```

- **HASH**
    - Distribución uniforme usando un cálculo interno.

PARTITION BY HASH(id) PARTITIONS 4;

5. El optimizador de consultas

El **optimizador** es la parte de MySQL que decide **cómo** ejecutar una consulta:

- ¿Usar un índice o no?
- ¿Qué orden seguir para hacer JOIN?
- ¿Cargar toda la tabla o solo una parte?
- ¿Qué plan es más rápido?

Analogía

Cuando pides una pizza, puede llegar por distintos caminos.
El optimizador elige **la ruta más rápida** basándose en estadísticas e índices.

6. EXPLAIN: ver el plan de ejecución

Usar EXPLAIN

EXPLAIN SELECT * FROM alumnos WHERE dni='12345678A';

Información útil:

- type: tipo de búsqueda (ALL, index, ref, eq_ref...)
- possible_keys: índices que podría usar
- key: índice que realmente usó

- rows: cuántas filas estima que necesita leer
- Extra: información adicional ("Using index", "Using where", etc.)

Interpretación

- ALL = peor caso, recorre toda la tabla
- ref o eq_ref = bueno
- const = excelente
- Using index = acceso muy rápido

7. Ejemplo completo

Tabla grande:

ventas (id, fecha, cliente_id, total)

Mejoras:

1. **Crear índice compuesto:**

CREATE INDEX idx_fecha_cliente ON ventas(fecha, cliente_id);

2. **Particionar por año:**

PARTITION BY RANGE (YEAR(fecha));

3. **Actualizar estadísticas:**

ANALYZE TABLE ventas;

4. **Analizar una consulta:**

EXPLAIN SELECT * FROM ventas

WHERE fecha BETWEEN '2024-01-01' AND '2024-12-31'

AND cliente_id = 55;

Tema	Idea clave
Índices	Aceleran búsquedas, pero ralentizan escrituras.
Estadísticas	Le dicen a MySQL “dónde buscar”.
Particionamiento	Divide tablas enormes para hacerlas manejables.
Optimizador	Decide la mejor ruta para ejecutar consultas.
EXPLAIN	Te muestra el plan real que usa MySQL.