Solución Detallada: Optimización de Consultas SQL para Empresa de Logística

Problemas Identificados y Soluciones

Consulta Original	**Problema**	**Solución Aplicada**	**Impacto**
SELECT * FROM envios WHERE estado = 'Entregado';	Escaneo completo de tabla (Full Scan) por falta de índice en estado.	Crear índice en columna estado: CREATE INDEX idx_envios_estado ON envios(estado);	Reduce el tiempo de búsqueda de O(n) a O(log n).
SELECT * FROM envios WHERE YEAR(fecha_creacion) = 2023;	Uso de función YEAR() evita uso de índices.	Reemplazar con rango de fechas: WHERE fecha_creacion >= '2023-01-01' AND fecha_creacion < '2024-01-01';	Habilita el uso de índices en fecha_creacion.
SELECT e.id_envio, c.nombre FROM envios e JOIN clientes c ON e.id_cliente = c.id_cliente;	Falta de índice en clientes.id_cliente.	Crear índice en clave foránea: CREATE INDEX idx_clientes_id ON clientes(id_cliente);	Acelera el JOIN al evitar escaneos secuenciales.
SELECT * FROM envios WHERE id_envio = 12345;	SELECT * recupera datos innecesarios.	Especificar columnas requeridas: SELECT id_envio, estado FROM envios WHERE id_envio = 12345;	Reduce transferencia de datos y mejora rendimiento.

Consulta Original	Problema	Solución Aplicada	Impacto
SELECT * FROM envios ORDER BY fecha_creacion DESC;	Ordenamiento costoso sin índice.	Crear índice descendente: CREATE INDEX idx_fecha_creacion_desc ON envios(fecha_creacion DESC);	Evita ordenamiento en memoria (filesort).
SELECT * FROM envios e JOIN clientes c ON e.id_cliente = c.id_cliente WHERE e.estado = 'Entregado';	Combinación de JOIN + filtro sin índices óptimos.	Crear índice compuesto: CREATE INDEX idx_envios_cliente_estado ON envios(id_cliente, estado);	Optimiza filtrado y unión simultáneamente.

Análisis de Plan de Ejecución (Antes vs Después)

Herramienta: EXPLAIN ANALYZE en MySQL.

Ejemplo: Consulta con JOIN

sql
Copy
Download
-- Antes (sin índices)
EXPLAIN ANALYZE
SELECT e.id_envio, c.nombre FROM envios e JOIN clientes c ON e.id_cliente = c.id_cliente;

- **Resultado:**
    - Full Table Scan en ambas tablas.
    - Costo alto (~500 ms para 10K registros).

sql
Copy
Download
-- *Después (con índices)*
EXPLAIN ANALYZE
SELECT e.id_envio, c.nombre FROM envios e JOIN clientes c ON e.id_cliente = c.id_cliente;

- **Resultado:**
    - Index Scan en clientes.id_cliente y envios.id_cliente.
    - Costo reducido (~50 ms).

Técnicas Adicionales (Plus)

1. **Particionamiento de Tablas:**
    - Dividir la tabla envios por rangos de fecha:

sql
Copy
Download
CREATE TABLE envios_2023 PARTITION OF envios
FOR VALUES FROM ('2023-01-01') TO ('2024-01-01');

    - **Beneficio:** Consultas filtradas por fecha solo escanean particiones relevantes.
2. **Monitoreo con** EXPLAIN ANALYZE**:**
    - Identificar operaciones costosas (ej: filesort, temporary tables).
3. **Caché de Consultas:**
    - Habilitar caché en MySQL:

sql
Copy
Download
SET GLOBAL query_cache_size = 1000000;

Conclusiones

- **Índices adecuados** son clave para optimizar consultas con filtros (WHERE) y uniones (JOIN).
- **Evitar funciones en columnas indexadas** (ej: YEAR()).
- **Especificar columnas necesarias** en lugar de SELECT *.
- **Índices compuestos** mejoran consultas con múltiples condiciones.

Próximos pasos:

- Implementar las optimizaciones en MySQL Workbench.
- Medir el rendimiento con EXPLAIN ANALYZE antes/después.
- Explorar particionamiento para datasets grandes (>1M registros).