ACTIVIDAD 3 – ÍNDICES

Cristóbal Suárez Abad

ADMINISTRACIÓN DE SISTEMAS GESTORES DE BASES DE DATOS - 2º ASIR

Contenido

Crea las tablas y/o índices necesarios para cada uno de los siguientes casos.

CASO 1

Situación: *Una tienda online tiene una tabla* `pedidos` *con 10 millones de filas. Las consultas que buscan pedidos de un cliente específico tardan varios minutos en completarse.*

Asumiendo que la columna donde están las "id" de los usuarios sea "id_cliente".

CREATE INDEX idx_pedidos_cliente
ON pedidos (id_cliente);

CASO 2

Situación: *Una plataforma inmobiliaria permite filtrar por* `ciudad` *y* `precio` *simultáneamente. Las consultas suelen ser:* `SELECT * FROM viviendas WHERE ciudad = 'Madrid' AND precio < 300000 ORDER BY precio;`.

CREATE INDEX idx_ciudad_precio ON viviendas (ciudad, precio);

CASO 3

Situación: *Una nueva red social necesita asegurar que ningún usuario registre el mismo correo electrónico. Además, quieren que los usuarios puedan buscar palabras clave dentro de sus "biografías" (campos de texto largo).*

Para garantizar que no registren el mismo correo usamos "Índice Único (UNIQUE)":

CREATE UNIQUE INDEX idx_email_unico ON usuarios (email);

Y para búsquedas dentro de texto usamos "Índice de Texto Completo (FULLTEXT)":

CREATE FULLTEXT INDEX idx_bio_user ON usuarios (biografia);

Las búsquedas se deben hacer ahora con "Match" y "Against"[1][2].

SELECT * FROM usuarios WHERE MATCH (biografia) AGAINST ('economía');

[1] https://codersfree.com/courses-status/aprende-php-y-mysql-desde-cero/indice-fulltext-en-mysql
[2] https://www.datacamp.com/doc/mysql/mysql-match-against

*Aquí tienes una ampliación de la actividad con **4 casos complejos** adicionales. Estos están diseñados para que el alumno no solo decida "poner un índice", sino que tenga que analizar el orden de las columnas, el tipo de motor y el impacto en el almacenamiento.*

CASO 4

***Situación:** Una plataforma de gestión de empleados tiene una tabla `nomina` con 5 millones de registros. Las consultas más frecuentes son:*

1. `SELECT * FROM nomina WHERE departamento_id = 5 AND año = 2023;`
2. `SELECT * FROM nomina WHERE departamento_id = 5;`
3. `SELECT * FROM nomina WHERE año = 2023;` *(Esta es la que más tarda).*

El administrador propone crear un único índice compuesto: `CREATE INDEX idx_dep_año ON nomina (departamento_id, año);`.

- ***El Reto:** ¿Por qué este índice no acelera la consulta número 3? ¿Cómo debería reestructurarse la estrategia de indexación para cubrir las tres consultas de forma eficiente sin penalizar demasiado las inserciones?*

Porque los índices compuestos siguen la regla de "**Regla del Prefijo de Izquierda**", por lo cual se filtra desde la columna de la izquierda a la derecha; pero no sirve de nada si solo estás usando el de la derecha.

En este caso sirve para las dos primeras. Para la tercera, se debería crear una propia:

CREATE INDEX idx_año ON nomina (año);

CASO 5

Situación: *Un periódico digital tiene una tabla* `articulos` *con un campo* `contenido` *de tipo* `LONGTEXT`*. Actualmente, usan la siguiente consulta para buscar palabras clave:*

```
SELECT titulo FROM articulos WHERE contenido LIKE '%economía%';
```

A pesar de tener un índice normal en la columna `contenido (contenido_index)`*, la consulta tarda más de 10 segundos y el servidor se bloquea cuando hay muchos usuarios buscando. Borra y crea los índices que consideres necesarios.*

Están usando un "**Índice Normal (B-Tree)**" para hacer búsquedas en un campo "LONGTEXT", el cual no está diseñado para eso. Debe usar un "**Índice de Texto Completo (FULLTEXT)**".

Borrar el índice anterior: DROP INDEX contenido_index ON articulos;

Crear uno nuevo: CREATE FULLTEXT INDEX idx_contenido_ft ON articulos (contenido);

Ahora se recomienda cambiar la consulta[3]:

```
SELECT titulo FROM articulos WHERE MATCH (contenido) AGAINST
('economía');
```

[3] https://codersfree.com/courses-status/aprende-php-y-mysql-desde-cero/indice-fulltext-en-mysql

CASO 6

Situación: *Un alumno recibe una base de datos heredada de un desarrollador anterior. Al ejecutar* `SHOW INDEX FROM productos;`*, observa lo siguiente:*

1. `PRIMARY KEY` *en* `id`
2. `INDEX idx_categoria` *en* `categoria_id`
3. `INDEX idx_cat_prov` *en* `(categoria_id, proveedor_id)`
4. `UNIQUE INDEX idx_slug` *en* `slug_producto`

- *¿Cambiarías algo? Hazlo*

Si, el segundo, “INDEX idx_categoria en categoria_id”, es innecesario porque el 3 (que es compuesto), ya cubre esa opción con “**Regla del Prefijo de Izquierda”.**

Así, que lo borraría:

DROP INDEX idx_categoria ON productos;

CASO 7

Situación: *Una multinacional quiere generar un reporte:* `SELECT * FROM ventas WHERE pais = 'España' AND estado_pedido = 'Completado';`*. La columna* `pais` *tiene 200 valores distintos (alta cardinalidad relativa), mientras que* `estado_pedido` *solo tiene 3 valores: 'Pendiente', 'Cancelado', 'Completado' (baja cardinalidad).*

El alumno dispone de dos opciones:

- ***Opción A:*** *Crear un índice en* `estado_pedido`*.*
- ***Opción B:*** *Crear un índice en* `pais`*.*
- ***Opción C:*** *Crear un índice compuesto* `(pais, estado_pedido)`*.*
- *Razona cada una de las opciones*

Opción A: Una mala opción porque apenas mejoraría el rendimiento. "*Evitar índices en columnas con baja cardinalidad (pocos valores distintos como género o booleanos). No aportan ventajas significativas*."

Opción B: Mejor que la anterior. Al contrario que la "A", esta tiene una alta cardinalidad y mejoraría el rendimiento.

Opción C: La mejor opción, porque es capaz de filtrar por ambos criterios. Además, siguiendo la Regla del Prefijo Más a la Izquierda, filtrará primero por "país", quitándose de encima muchos registros innecesarios.

CASO 8

Situación: *Una aplicación de marketing ejecuta miles de veces por segundo la consulta:* `SELECT email FROM suscriptores WHERE estado = 'activo';` *Actualmente hay un índice en la columna* `estado`*. El servidor tiene mucha carga de I/O (lectura de disco) porque, tras encontrar el registro en el índice, MySQL debe ir a la tabla principal a buscar el email.*

Eso es porque está usando un "Índice Normal" en vez de uno compuesto. Debería crear uno así:

CREATE INDEX idx_estado_email ON suscriptores (estado, email);

CASO 9

Situación: *Tenemos una tabla* `usuarios` *donde el* `id` *es* `INT` *y una tabla* `logs_actividad` *donde el campo* `usuario_id` *fue creado por error como* `VARCHAR(20)`*. Al hacer un* `JOIN` *entre ambas, la consulta es extremadamente lenta, a pesar de que ambos campos están indexados.*

- ***¿Qué propones para mejorar esto?***

Convertir "usuario_id" a INT[4].

ALTER TABLE logs_actividad MODIFY usuario_id INT NOT NULL;

Indicamos el "NOT NULL", porque al ser una "PRIMARY KEY", si lo modificamos se puede "perder" esa característica, por eso es bueno definirla de nuevo; por si acaso.

[4] https://forum.espocrm.com/forum/general/58093-how-to-change-the-column-type-id-from-varchar-to-int

CASO 10

Situación: *Un panel de control financiero muestra las ventas de un mes específico ordenadas por fecha:* `SELECT * FROM facturas WHERE empresa_id = 45 AND fecha BETWEEN '2023-01-01' AND '2023-01-31' ORDER BY fecha DESC;`

Crear un índice compuesto que recoja ambos campos[5]. Ayuda a mejorar el rendimiento: "*Mejora significativa en operaciones WHERE, JOIN, ORDER BY y GROUP BY, las más utilizadas en consultas complejas*."

CREATE INDEX idx_empresa_fecha ON facturas (empresa_id, fecha DESC);

O sin "DESC".

CASO 11

Situación: *Una base de datos de ciberseguridad almacena millones de URLs en una columna* `url_detectada` *(VARCHAR 2048). Quieren indexar esta columna para búsquedas rápidas, pero el índice resultante ocupa más que la propia tabla y ralentiza todo el sistema.*

Usamos "Prefijos de Columna", los cuales permiten indexar solo los primeros "N" caracteres, permitiendo tener un índice más pequeño.

Sería algo así (si la tabla se llamase "tabla_url").

CREATE INDEX idx_url_corto ON tabla_url (url_detectada(100));

El único problema es que no distingue URLs que solo difieren al final.

[5] https://use-the-index-luke.com/sql/sorting-grouping/indexed-order-by

CASO 12

Situación: *El equipo de desarrollo quiere borrar un índice que parece no usarse (`idx_antiguo`), pero tienen miedo de que al borrarlo la base de datos de producción colapse si alguna consulta crítica sí lo usaba.*

- ***El Reto:*** *Proponer un método para "desactivar" el índice sin borrarlo físicamente, permitiendo que el optimizador de MySQL lo ignore pero que se pueda recuperar instantáneamente si algo sale mal.*
- ***Concepto clave:*** *Invisible Indexes (disponible desde MySQL 8.0) y monitorización del rendimiento.*

"Invisible indexes make it possible to test the effect of removing an index on query performance, without making a destructive change that must be undone should the index turn out to be required. Dropping and re-adding an index can be expensive for a large table, whereas making it invisible and visible are fast, in-place operations."[6]

Gestión de Índices Invisibles:

- Podemos usar primero un "**ALTER TABLE**" y convertirlo en invisible para probar que no pasa nada:

ALTER TABLE nombre_tabla ALTER INDEX idx_antiguo INVISIBLE;

- Más adelante podemos borrarla si no surgen problemas:

DROP INDEX idx_antiguo ON nombre_tabla;

- Si hay problemas, se recupera en un segundo con:

ALTER TABLE nombre_tabla ALTER INDEX idx_antiguo VISIBLE;

[6] https://dev.mysql.com/doc/refman/9.1/en/invisible-indexes.html