Optimización de consultas

1 Índices

Si quisiéramos buscar un valor específico en la columna de una tabla y la columna sobre la que queremos buscar no tuviese un índice, tendríamos que recorrer toda la tabla comparando fila a fila hasta encontrar el valor que coincide con el valor buscado. Para tablas con pocas filas puede que esto no sea un problema, pero imagina las operaciones de comparación que tendría que realizar sobre una tabla con millones de filas.

La mejor forma de optimizar el rendimiento de una consulta es creando índices sobre las columnas que se utilizan en la cláusula `WHERE`. Los índices se comportan como punteros sobre las filas de la tabla y nos permiten determinar rápidamente cuáles son las filas que cumplen la condición de la cláusula `WHERE`.

También es posible optimizar consultas que usan las sentencias `ORDER BY` o `GROUP BY`. Ordenar o agrupar los resultados de una consulta es un proceso costoso que se debe intentar evitar pero para hacerlo también podemos hacer uso de los índices.

Nota: MySQL siempre tendrá en cuenta la cláusula `WHERE` por encima de la de `ORDER BY o GROUP BY`.

Por lo tanto siempre será mejor tener un índice que nos devuelva un número pequeño de filas a otro que nos sirva solo para 'ordenar' o ‘agrupar’.

Recordar que Mysql sólo va a hacer uso de **un índice**, por lo que si se aplica uno en la cláusula `WHERE`, no va a utilizar otro para aplicarlo a la parte `ORDER BY` o `GROUP BY`, únicamente se aplicaría un índice a distintas cláusulas en el caso de que tengamos un índice sobre una columna que se vaya a aplicar en la parte `WHERE` y además la misma columna se utiliza en la parte `ORDER BY` (y/o `GROUP BY`).

Además hay que tener en cuenta que una vez creada una tabla ya tenemos por defecto un índice, el `PRIMARY KEY`, que se crea automáticamente. Y si existiera en dicha tabla un campo que fuera `FOREIGN KEY`, también se crearía de manera automática un índice normal sobre ese campo.

Todos los tipos de datos de MySQL pueden ser indexados, pero tenga en cuenta que no es conveniente crear un índice para cada una de las columnas de una tabla, ya que el exceso de índices innecesarios pueden provocar un incremento del espacio de almacenamiento y un aumento del tiempo para MySQL a la hora de decidir qué índices necesita utilizar. Los índices además añaden una sobrecarga a las

operaciones de inserción, actualización y borrado, porque cada índice tiene que ser actualizado después de realizar cada una de estas operaciones.

Debe tratar de buscar un equilibrio entre el número de índices y el tiempo de respuesta de su consulta, de modo que pueda reducir el tiempo de respuesta de su consulta utilizando el menor número de índices posible.

1.1 Índices en MySQL

La mayoría de los índices que se utilizan en MySQL son almacenados en Árboles B (*B-trees*).

- `PRIMARY KEY`
- `UNIQUE`
- `INDEX`
- `FULLTEXT`

Ejemplo de un árbol B (*B-tree*):

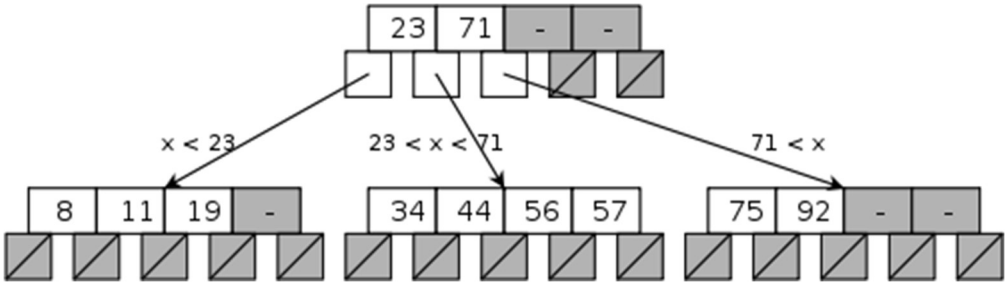

Imagen: Ejemplo de un árbol B. *B-tree*. Nagae. 2007. Wikipedia.

1.2 Gestión de índices

1.2.1 Crear índices

La sintaxis para crear índices en MySQL es la siguiente:

```
CREATE [ONLINE|OFFLINE] [UNIQUE|FULLTEXT|SPATIAL] INDEX index_name
    [index_type]
    ON tbl_name (index_col_name,...)
    [index_option] ...

index_col_name:
    col_name [(length)] [ASC | DESC]

index_option:
    KEY_BLOCK_SIZE [=] value
  | index_type
  | WITH PARSER parser_name
```

```
  | COMMENT 'string'

index_type:

    USING {BTREE | HASH}
```

Ejemplo:

El siguiente ejemplo crea un índice con el nombre `idx_pais` sobre la columna `pais` de la tabla `cliente`.

```
CREATE INDEX idx_pais ON cliente(país);
```

Puede encontrar más información sobre la creación de índices en MySQL en la documentación oficial.

1.2.2 Mostrar los índices

La sintaxis para mostrar los índices de una tabla en MySQL es la siguiente:

```
SHOW {INDEX | INDEXES | KEYS}
    {FROM | IN} tbl_name
    [{FROM | IN} db_name]

    [WHERE expr]
```

Ejemplo:

```
SHOW INDEX FROM cliente;
```

Puede encontrar más información en la documentación oficial.

1.2.3 Eliminar índices

La sintaxis para eliminar índices en MySQL es la siguiente:

```
DROP INDEX index_name ON tbl_name
    [algorithm_option | lock_option] ...

algorithm_option:
    ALGORITHM [=] {DEFAULT|INPLACE|COPY}

lock_option:

    LOCK [=] {DEFAULT|NONE|SHARED|EXCLUSIVE}
```

Ejemplo:

```
DROP INDEX idx_pais ON cliente;
```

Puede encontrar más información sobre cómo eliminar índices en MySQL en la documentación oficial.

1.3 Optimización de consultas e índices

1.3.1 `EXPLAIN`

`EXPLAIN` nos permite obtener información sobre cómo se llevarán a cabo las consultas. Nos permite detectar cuando un índice se usa o no, si se usa correctamente o ver si las consultas se ejecutan de forma óptima.

```
{EXPLAIN | DESCRIBE | DESC}
    tbl_name [col_name | wild]

{EXPLAIN | DESCRIBE | DESC}
    [explain_type]
    {explainable_stmt | FOR CONNECTION connection_id}

explain_type: {
    EXTENDED
  | PARTITIONS
  | FORMAT = format_name
}

format_name: {
    TRADITIONAL
  | JSON
}

explainable_stmt: {
    SELECT statement
  | DELETE statement
  | INSERT statement
  | REPLACE statement
  | UPDATE statement

}
```

2 Ejemplos de optimización de consultas

2.1 Ejemplo 1 (`INDEX`)

Suponga que estamos trabajando con la base de datos `jardineria` y queremos optimizar la siguiente consulta.

```sql
SELECT nombre_contacto, telefono
FROM cliente
WHERE pais = 'France';
```

Lo primero que tenemos que hacer es hacer uso de `EXPLAIN` para obtener información sobre cómo se está realizando la consulta.

```sql
EXPLAIN SELECT nombre_contacto, telefono
FROM cliente
WHERE pais = 'France';
```

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	cliente	NULL	ALL	NULL	NULL	NULL	NULL	36	10.00	Using where

Tenemos que fijarnos en los valores que nos aparecen en las columnas `type` y `rows`. En este caso tenemos `type = ALL`, que quiere decir que es necesario realizar un escaneo completo de todas las filas de la tabla. Y `rows = 36`, quiere decir que en este caso ha tenido que examinar 36 filas. Que es el número total de filas que tiene la tabla.

Para obtener información sobre la tabla y sobre los índices que existen en ella podemos usar `DESCRIBE` o `SHOW INDEX`.

- `DESCRIBE cliente;`

Field	Type	Null	Key	Default	Extra
codigo_cliente	int(11)	NO	PRI	NULL	
nombre_cliente	varchar(50)	NO		NULL	
nombre_contacto	varchar(30)	YES		NULL	
apellido_contacto	varchar(30)	YES		NULL	
telefono	varchar(15)	NO		NULL	
fax	varchar(15)	NO		NULL	
linea_direccion1	varchar(50)	NO		NULL	
linea_direccion2	varchar(50)	YES		NULL	
ciudad	varchar(50)	NO		NULL	
region	varchar(50)	YES		NULL	
pais	varchar(50)	YES		NULL	
codigo_postal	varchar(10)	YES		NULL	
codigo_emplea...	int(11)	YES	MUL	NULL	
limite_credito	decimal(1...	YES		NULL	

- `SHOW INDEX`

```sql
SHOW INDEX FROM cliente;
```

Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed	Null	Index_type	Comment	Index_comment
cliente	0	PRIMARY	1	codigo_cliente	A	36	NULL	NULL		BTREE		
cliente	1	codigo_empleado_rep_ventas	1	codigo_empleado_rep_ventas	A	11	NULL	NULL	YES	BTREE		

Según los resultados obtenidos con `DESCRIBE` y `SHOW INDEX` podemos observar que no existe ningún índice sobre la columna `pais`.

Para crear un índice sobre la columna `pais` hacemos uso de `CREATE INDEX`:

```
CREATE INDEX idx_pais ON cliente(pais);
```

Volvemos a ejecutar `DESCRIBE` o `SHOW INDEX` para comprobar que hemos creado el índice de forma correcta:

- DESCRIBE

```
DESCRIBE cliente;
```

```
+---------------------------+--------------+------+-----+---------+------+
| Field                     | Type         | Null | Key | Default | Extra|
+---------------------------+--------------+------+-----+---------+------+
| codigo_cliente            | int(11)      | NO   | PRI | NULL    |
|
| nombre_cliente            | varchar(50)  | NO   |     | NULL    |
|
| nombre_contacto           | varchar(30)  | YES  |     | NULL    |
|
| apellido_contacto         | varchar(30)  | YES  |     | NULL    |
|
| telefono                  | varchar(15)  | NO   |     | NULL    |
|
| fax                       | varchar(15)  | NO   |     | NULL    |
|
| linea_direccion1          | varchar(50)  | NO   |     | NULL    |
|
| linea_direccion2          | varchar(50)  | YES  |     | NULL    |
|
| ciudad                    | varchar(50)  | NO   |     | NULL    |
|
| region                    | varchar(50)  | YES  |     | NULL    |
|
| pais                      | varchar(50)  | YES  | MUL | NULL    |
|
| codigo_postal             | varchar(10)  | YES  |     | NULL    |
|
| codigo_empleado_rep_ventas | int(11)     | YES  | MUL | NULL    |
|
| limite_credito            | decimal(15,2) | YES  |     | NULL    |
|

+---------------------------+--------------+------+-----+---------+------+
```

- SHOW INDEX

```
SHOW INDEX FROM cliente;
```

Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed	Null	Index_type	Comment	Index_comment
cliente	0	PRIMARY	1	codigo_cliente	A	36	NULL	NULL		BTREE		
cliente	1	codigo_empleado_rep_ventas	1	codigo_empleado_rep_ventas	A	11	NULL	NULL	YES	BTREE		
cliente	1	idx_pais	1	pais	A	5	NULL	NULL	YES	BTREE		

Una vez que hemos comprobado que el índice se ha creado de forma correcta podemos volver a ejecutar la consulta con `EXPLAIN` para comprobar si hemos conseguido optimizarla.

```
EXPLAIN SELECT nombre_contacto, telefono
FROM cliente
WHERE pais = 'France';
```

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	cliente	NULL	ref	idx_pais	idx_pais	203	const	2	100.00	NULL

De nuevo tenemos que fijarnos en los valores que nos aparecen en las columnas `type` y `rows`. En este caso ambos valores han cambiado, ahora `type` es igual a `ref`, y por lo tanto ya no es necesario realizar un escaneo completo de todas las filas de la tabla. Y el valor de `rows` es igual a `2`, que quiere decir que en este caso ha tenido que examinar solamente 2 filas.

2.2 Ejemplo 2 (`FULLTEXT INDEX`)

Suponga que estamos trabajando con la base de datos `jardineria` y queremos buscar todos los productos que contienen la palabra `acero` en el nombre o en la descripción del producto. Una posible solución podrías ser esta:

```
SELECT *
FROM producto
WHERE nombre LIKE '%acero%' OR descripcion LIKE '%acero%';
```

Si la analizamos con `EXPLAIN` veremos que no es muy eficiente porque esta consulta realiza un escaneo completo de toda la tabla.

```
EXPLAIN SELECT *
FROM producto
WHERE nombre LIKE '%acero%' OR descripcion LIKE '%acero%';
```

En estos casos es muy útil hacer uso de los índices de tipo `FULLTEXT INDEX`.

En primer lugar vamos a modificar la tabla `producto` para crear el índice `FULLTEXT` con las dos columnas sobre las que queremos realizar la búsqueda.

```
CREATE FULLTEXT INDEX idx_nombre_descripcion ON producto(nombre, descripcion);
```

Una vez creado el índice ejecutamos la consulta haciendo uso de `MATCH` y `AGAINST`.

```
SELECT *
FROM producto
WHERE MATCH(nombre, descripcion) AGAINST ('acero');
```

`MATCH` y `AGAINST` solo se puede utilizar cuando hay creado un índice FULLTEXT, no en otro caso.

Si analizamos la consulta con `EXPLAIN` veremos que ya no es necesario escanear toda la tabla para encontrar el resultado que buscamos.

```
EXPLAIN SELECT *
FROM producto
WHERE MATCH(nombre, descripcion) AGAINST ('acero');
```