Unidad 10. Optimización de consultas

Apuntes de BD para DAW, DAM y ASIR

José Juan Sánchez Hernández

Curso 2023/2024

Índice

1 Optimización de consultas **1**

1.1 Índices . . . 1

1.1.1 Tipos de índices . . . 1

1.1.2 Índices en MySQL . . . 2

1.1.3 Gestión de índices . . . 4

1.1.3.1 Crear índices . . . 4

1.1.3.1.1 `CREATE INDEX` . . . 4

1.1.3.1.2 `ALTER TABLE` . . . 6

1.1.3.1.3 `CREATE TABLE` . . . 8

1.1.3.2 Mostrar los índices . . . 10

1.1.3.2.1 `SHOW INDEX` . . . 10

1.1.3.2.2 `DESCRIBE` . . . 10

1.1.3.3 Eliminar índices . . . 11

1.1.3.3.1 `DROP INDEX` . . . 11

1.1.3.3.2 `ALTER TABLE` . . . 11

1.1.3.4 Actualizar y reordenar índices . . . 12

1.1.3.4.1 `OPTIMIZE TABLE` . . . 12

1.1.3.4.2 `ANALYZE TABLE` . . . 12

1.1.4 Optimización de consultas e índices . . . 12

1.1.4.1 `EXPLAIN` . . . 12

1.2 Ejemplos de optimización de consultas . . . 13

1.2.1 Ejemplo 1 (`INDEX`) . . . 13

1.2.2 Ejemplo 2 (`FULLTEXT INDEX`) . . . 16

1.2.3 Ejemplo 3 (`FULLTEXT INDEX`) . . . 17

2 Ejercicios **21**

2.1 Base de datos: Jardinería . . . 21

3 Práctica **23**

3.1 Base de datos: `netflix` . . . 23

4 Referencias **24**

5 Licencia **25**

Índice de figuras

Índice de cuadros

1 Optimización de consultas

1.1 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`.

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.

1.1.1 Tipos de índices

Los sistemas gestores de bases de datos utilizan diferentes tipos de índices, algunos de los más utilizados son los siguientes:

- **Índices de clave primaria.** Identifican de forma única una fila dentro de una tabla y no admiten valores nulos.
- **Índices de clave ajena.** Este índice hace referencia a una columna que es clave primaria en otra tabla.
- **Índices únicos.** Garantiza que los valores de una columna son únicos. Son similares a los índices de clave primaria, pero permiten valores nulos.
- **Índices con valores repetidos.** Permiten optimizar búsquedas sobre columnas que contienen valores repetidos.
- **Índices de múltiples columnas.** Utilizan varias columnas en lugar de una sola.
- **Índices de texto completo.** Se utilizan para optimizar las búsquedas en campos de texto.

1.1.2 Índices en MySQL

La mayoría de los índices que se utilizan en MySQL son almacenados en árboles B (*B-trees*). Los árboles B son unas estructuras de datos que se utilizan para almacenar datos de forma ordenada, y permiten realizar operaciones de búsqueda, inserción y borrado de forma eficiente. Estas estructuras mejoran el rendimiento de las consultas en las bases de datos.

Algunos índices de MySQL que utilizan árboles B son:

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

Ejemplo de un árbol B (*B-tree*):

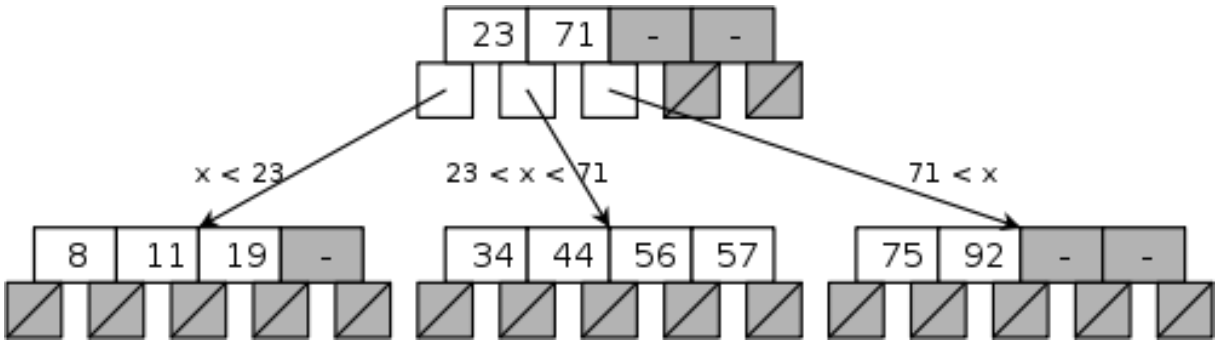

Imagen: Ejemplo de un árbol B. *B-tree*. Nagae. 2007. Wikipedia.

Los índices que se utilizan sobre datos espaciales se almacenan en Árboles R (*R-trees*).

- `SPATIAL`

Ejemplo de un árbol R (*R-tree*):

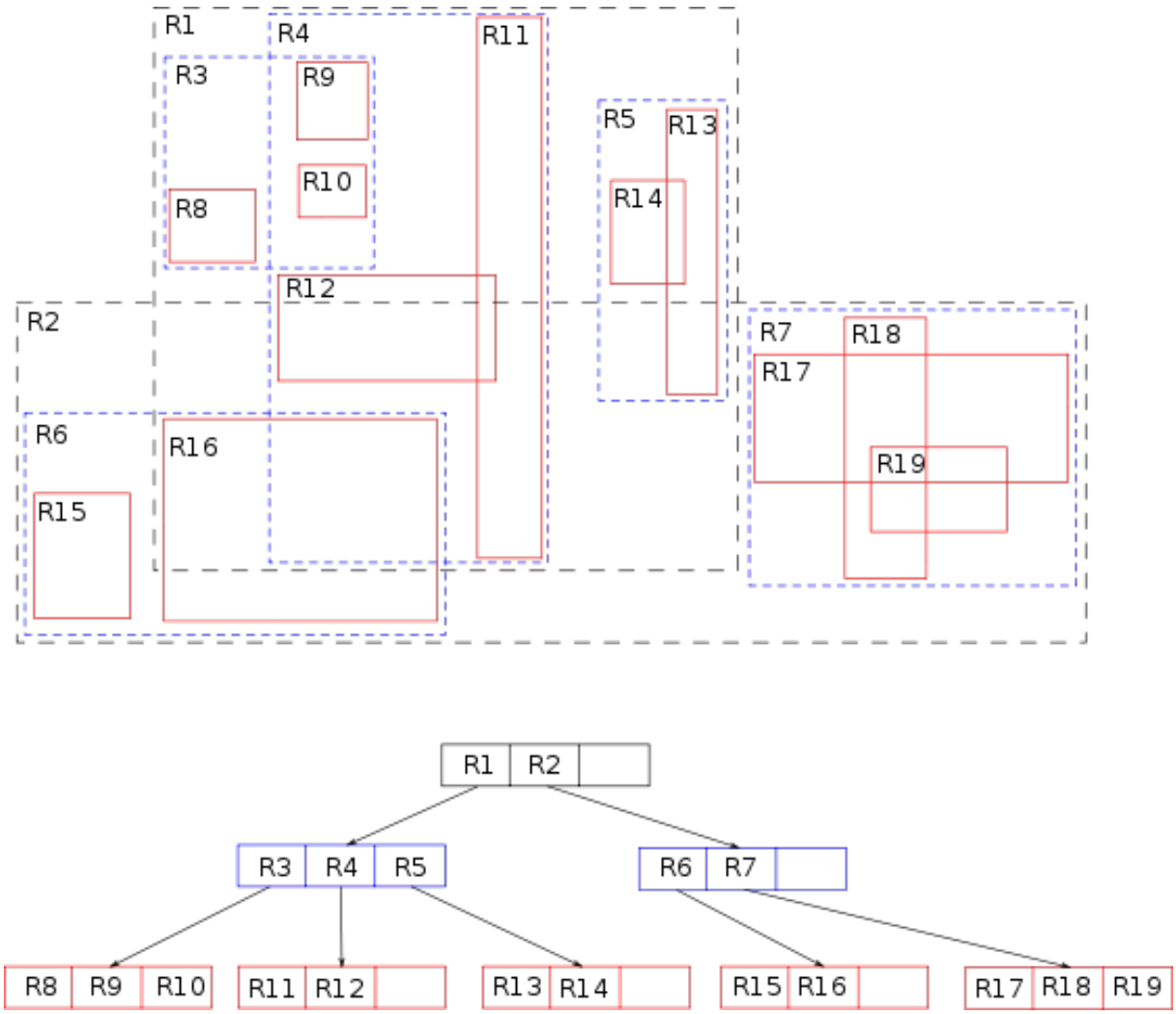

Imagen: Ejemplo de un árbol R. *R-tree*. Skinkie. 2010. Wikipedia.

Y por último, las tablas almacenadas en memoria utilizan índices *hash*.

- `MEMORY`

Ejemplo de índices *hash*:

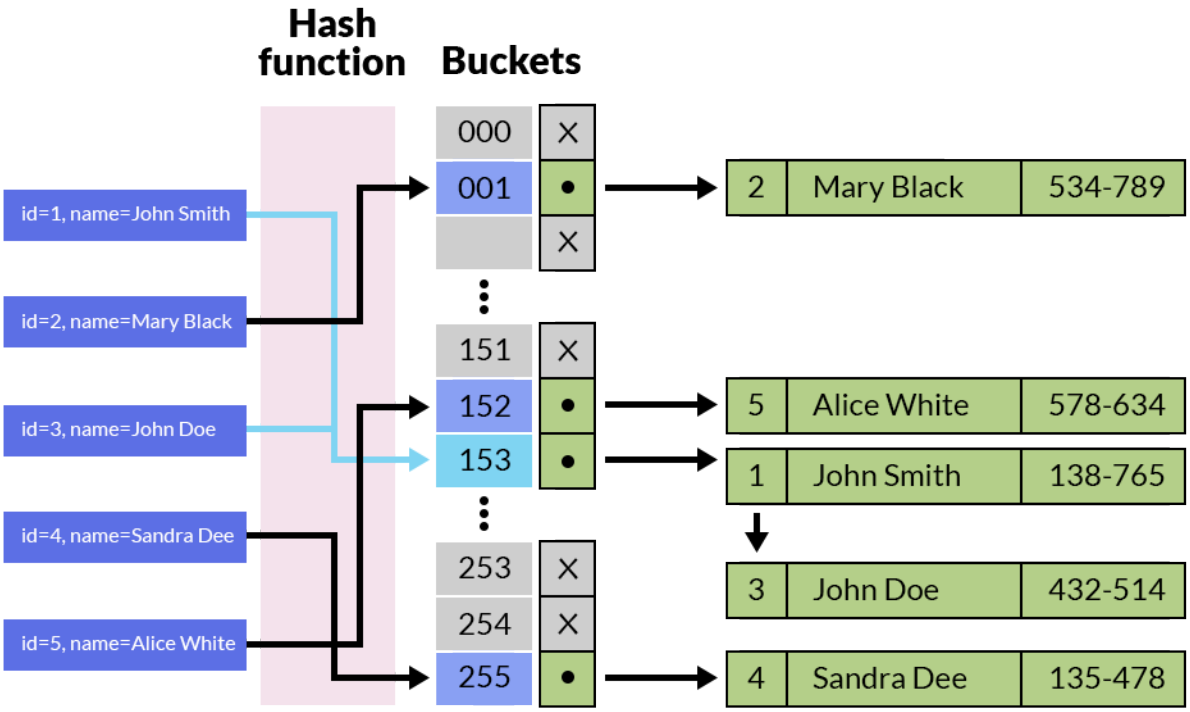

Imagen: Ejemplo de índices *hash*. Emil Drkušić. 2016.

1.1.3 Gestión de índices

1.1.3.1 Crear índices

1.1.3.1.1 `CREATE INDEX` 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}
```

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

Ejemplo 1: Uso de `INDEX`

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(pais);
```

Ejemplo 2: Uso de `UNIQUE INDEX`

El siguiente ejemplo crea un índice de tipo `UNIQUE` con el nombre `idx_email` sobre la columna `email` de la tabla `empleado`.

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

Ejemplo 3: Uso de `INDEX` con varias columnas

El siguiente ejemplo crea un índice de tipo `INDEX` con el nombre `idx_apellido_nombre` compuesto por las columnas `apellido_contacto` y `nombre_contacto` de la tabla `cliente`.

```
CREATE INDEX idx_apellido_nombre ON cliente(apellido_contacto, nombre_contacto)
    ;
```

Este índice será útil en las consultas donde se realicen búsquedas por el apellido y el nombre del cliente, o solamente por el apellido, pero no será útil en aquellas consultas donde sólo se utilice el nombre, ya que tendría que recorrer toda la tabla para encontrarlo.

Ejemplo 4: Uso de `INDEX` con el prefijo de una columna

En este ejemplo vamos a crear un índice sobre un prefijo de la columna `nombre_cliente` de la tabla `cliente`. La columna `nombre_cliente` está definida como un `VARHCAR(50)`, pero en este caso vamos a crear un índice de sólo 25 caracteres.

El uso de índices sobre un prefijo de una columna, es útil para reducir el tamaño que ocuparán los índices y optimizar así su almacenamiento, pero para que las búsquedas sobre los índices sigan siendo eficientes, habrá que buscar un tamaño de índice adecuado que nos permita diferenciarlos con el menor número de bytes posibles.

```
CREATE INDEX idx_nombre_cliente ON cliente(nombre_cliente(25));
```

Ejemplo 5: Uso de `FULLTEXT INDEX`

En este ejemplo vamos a crear un índice de tipo `FULLTEXT` compuesto por las columnas `nombre` y `descripcion` de la tabla `producto`, para poder realizar búsquedas más eficientes sobre esas columnas.

```
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');
```

A continuación, se muestra cuál es la sintaxis para realizar una búsqueda con el operador `MATCH()AGAINST()` sobre un índice de tipo `FULLTEXT INDEX`:

```
MATCH (col1,col2,...) AGAINST (expr [search_modifier])

search_modifier:
  {
       IN NATURAL LANGUAGE MODE
     | IN NATURAL LANGUAGE MODE WITH QUERY EXPANSION
     | IN BOOLEAN MODE
     | WITH QUERY EXPANSION
  }
```

Dependiendo del modificador, podemos utilizar tres tipos de búsqueda sobre los índices de tipo `FULLTEXT INDEX`:

- `IN NATURAL LANGUAGE MODE`: Esta es la opción que se utiliza por defecto cuando no se indica de forma explícita un tipo de búsqueda. Utiliza un algoritmo de búsqueda similar a cómo procesamos y buscamos información.
- `IN BOOLEAN MODE`: Con esta opción podemos utilizar operadores booleanos en la búsqueda. Algunos de los operadores son: + para indicar que la palabra tiene que aparecer en el resultado, o − para indicar que la palabra no tiene que aparecer en el resultado. Puede encontrar más información sobre los operadores en la documentación oficial de MySQL.
- `WITH QUERY EXPANSION`: Esta opción se utiliza para ampliar los resultados de búsqueda mostrando contenidos relacionados.

1.1.3.1.2 ALTER TABLE También es posible crear índices con la sentencia `ALTER TABLE`. A continuación se muestra una versión reducida de la sintaxis de la sintaxis `ALTER TABLE` para añadir índices y restricciones a una tabla en MySQL.

```
ALTER TABLE tbl_name
    [alter_option [, alter_option] ...]

alter_option: {
  | ADD {INDEX | KEY} [index_name]
        [index_type] (key_part,...) [index_option] ...
  | ADD {FULLTEXT | SPATIAL} [INDEX | KEY] [index_name]
        (key_part,...) [index_option] ...
  | ADD [CONSTRAINT [symbol]] PRIMARY KEY
        [index_type] (key_part,...)
        [index_option] ...
  | ADD [CONSTRAINT [symbol]] UNIQUE [INDEX | KEY]
        [index_name] [index_type] (key_part,...)
        [index_option] ...
  | ADD [CONSTRAINT [symbol]] FOREIGN KEY
        [index_name] (col_name,...)
        reference_definition

key_part: {col_name [(length)] | (expr)} [ASC | DESC]

index_type:
    USING {BTREE | HASH}

```

```
index_option: {
    KEY_BLOCK_SIZE [=] value
  | index_type
  | WITH PARSER parser_name
  | COMMENT 'string'
  | {VISIBLE | INVISIBLE}
}
```

Puede encontrar más información sobre la creación de índices en MySQL con la sentencia `ALTER TABLE` en la documentación oficial.

Ejemplo 1: Uso de `INDEX`

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

```
ALTER TABLE cliente ADD INDEX idx_nombre (nombre);
```

Al crear los índices con `ALTER TABLE` podemos omitir el nombre del índice.

```
ALTER TABLE cliente ADD INDEX (nombre);
```

Ejemplo 2: Uso de `UNIQUE INDEX`

El siguiente ejemplo crea un índice de tipo `UNIQUE` con el nombre `idx_email` sobre la columna `email` de la tabla `empleado`.

```
ALTER TABLE empleado ADD UNIQUE INDEX idx_email (email);
```

Al crear los índices con `ALTER TABLE` podemos omitir el nombre del índice.

```
ALTER TABLE empleado ADD UNIQUE INDEX (email);
```

Ejemplo 3: Uso de `INDEX` con varias columnas

El siguiente ejemplo crea un índice de tipo `INDEX` con el nombre `idx_apellido_nombre` compuesto por las columnas `apellido_contacto` y `nombre_contacto` de la tabla `cliente`.

```
ALTER TABLE cliente ADD INDEX idx_apellido_nombre (apellido_contacto,
    nombre_contacto);
```

Al crear los índices con `ALTER TABLE` podemos omitir el nombre del índice.

```
ALTER TABLE cliente ADD INDEX (apellido_contacto, nombre_contacto);
```

Ejemplo 4: Uso de `INDEX` con el prefijo de una columna

En este ejemplo vamos a crear un índice sobre un prefijo de la columna `nombre_cliente` de la tabla `cliente`. La columna `nombre_cliente` está definida como un `VARHCAR(50)`, pero en este caso vamos a crear un índice de sólo 25 caracteres.

```
ALTER TABLE cliente ADD INDEX idx_nombre_cliente (nombre_cliente(25));
```

Al crear los índices con `ALTER TABLE` podemos omitir el nombre del índice.

```
ALTER TABLE cliente ADD INDEX (nombre_cliente(25));
```

Ejemplo 5: Uso de `FULLTEXT INDEX`

En este ejemplo vamos a crear un índice `FULLTEXT` sobre las columnas `nombre` y `descripcion` de la tabla `producto`, para permitir realizar búsquedas más eficientes sobre esas columnas.

```
ALTER TABLE producto ADD FULLTEXT INDEX idx_nombre_descripcion (nombre,
    descripcion);
```

Al crear los índices con `ALTER TABLE` podemos omitir el nombre del índice.

```
ALTER TABLE producto ADD FULLTEXT INDEX (nombre, descripcion);
```

Existen tres tipos de búsquedas con índices de tipo `FULLTEXT INDEX`:

- `IN NATURAL LANGUAGE MODE`.

```
MATCH (col1,col2,...) AGAINST (expr [search_modifier])


search_modifier:
  {
       IN NATURAL LANGUAGE MODE
     | IN NATURAL LANGUAGE MODE WITH QUERY EXPANSION
     | IN BOOLEAN MODE
     | WITH QUERY EXPANSION
  }
```

1.1.3.1.3 CREATE TABLE También es posible crear índices al crear la tabla con la sentencia `CREATE TABLE`. A continuación se muestra una versión reducida de la sintaxis `CREATE TABLE` para añadir índices y restricciones a una tabla en MySQL.

```
CREATE TABLE [IF NOT EXISTS] tbl_name
    (create_definition,...)
    [table_options]
    [partition_options]

create_definition: {
    col_name column_definition
  | {INDEX | KEY} [index_name] [index_type] (key_part,...)
      [index_option] ...
  | {FULLTEXT | SPATIAL} [INDEX | KEY] [index_name] (key_part,...)
      [index_option] ...
  | [CONSTRAINT [symbol]] PRIMARY KEY
      [index_type] (key_part,...)
      [index_option] ...
  | [CONSTRAINT [symbol]] UNIQUE [INDEX | KEY]
      [index_name] [index_type] (key_part,...)
      [index_option] ...
  | [CONSTRAINT [symbol]] FOREIGN KEY
      [index_name] (col_name,...)
      reference_definition
```

```
  | check_constraint_definition
}
```

Puede encontrar la sintaxis completa de la sentencia `CREATE TABLE` en la documentación oficial.

Ejemplo 1:

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

```
CREATE TABLE cliente (
  id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
  nombre VARCHAR(50) NOT NULL,
  email VARCHAR(15) NOT NULL,
  telefono VARCHAR(9) NOT NULL,
  INDEX idx_nombre (nombre)
);
```

En este caso, el nombre de índice no es obligatorio y se puede omitir.

```
CREATE TABLE cliente (
  id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
  nombre VARCHAR(50) NOT NULL,
  email VARCHAR(15) NOT NULL,
  telefono VARCHAR(9) NOT NULL,
  INDEX (nombre)
);
```

Ejemplo 2:

El siguiente ejemplo crea un índice de tipo `UNIQUE` sobre la columna `email` de la tabla `cliente`. En este caso, sería suficiente con añadir la palabra reservada `UNIQUE` después de la definición de los atributos de la columna `email`.

```
CREATE TABLE cliente (
  id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
  nombre VARCHAR(50) NOT NULL,
  email VARCHAR(15) NOT NULL UNIQUE,
  telefono VARCHAR(9) NOT NULL
);
```

También se podría crear debajo de la definición de todas las columnas.

```
CREATE TABLE cliente (
  id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
  nombre VARCHAR(50) NOT NULL,
  email VARCHAR(15) NOT NULL,
  telefono VARCHAR(9) NOT NULL,
    UNIQUE (email)
);
```

1.1.3.2 Mostrar los índices

1.1.3.2.1 SHOW INDEX La sintaxis de la sentencia `SHOW INDEX` 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]
```

Puede encontrar más información en la documentación oficial.

Ejemplo:

Este ejemplo muestra los índices que existen en la tabla `cliente` de la base de datos `jardinería`.

```
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      |         |               |
+--
    -------+------------+---------------------------+--------------+-------------------------
```

1.1.3.2.2 DESCRIBE También es posible obtener información sobre los índices que existen en una tabla con la sentencia `DESCRIBE`.

Ejemplo:

Este ejemplo muestra información de la tabla `cliente` de la base de datos `jardinería`.

```
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_empleado_rep_ventas | int(11)        | YES  | MUL | NULL    |       |
| limite_credito             | decimal(15,2)  | YES  |     | NULL    |       |
+----------------------------+----------------+------+-----+---------+-------+
```

En la columna `Key` podemos observar que en las columnas `codigo_cliente` y `codigo_empleado_rep_ventas` son dos índices.

1.1.3.3 Eliminar índices

1.1.3.3.1 DROP INDEX La sintaxis para eliminar índices con la sentencia `DROP INDEX` 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}
```

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

Ejemplo:

El siguiente ejemplo elimina un índice con el nombre `idx_nombre` de la tabla `cliente`.

```
DROP INDEX idx_nombre ON cliente;
```

1.1.3.3.2 ALTER TABLE También es posible eliminar índices con la sentencia `ALTER TABLE`. A continuación se muestra una versión reducida de la sintaxis `ALTER TABLE` para eliminar índices en una tabla en MySQL.

```
ALTER TABLE tbl_name
    [alter_option [, alter_option] ...]

alter_option: {
  | DROP {INDEX | KEY} index_name
  | DROP PRIMARY KEY
  | DROP FOREIGN KEY fk_symbol
```

Puede encontrar más información sobre la creación de índices en MySQL con la sentencia `ALTER TABLE` en la documentación oficial.

Ejemplo:

El siguiente ejemplo elimina un índice con el nombre `idx_nombre` de la tabla `cliente`.

```
ALTER TABLE cliente DROP INDEX idx_nombre;
```

1.1.3.4 Actualizar y reordenar índices

1.1.3.4.1 OPTIMIZE TABLE `OPTIMIZE TABLE` nos permite desfragmentar una tabla, así como actualizar y reordenar los índices. La sintaxis en MySQL es la siguiente:

```
OPTIMIZE [NO_WRITE_TO_BINLOG | LOCAL]
    TABLE tbl_name [, tbl_name] ...
```

Puede encontrar más información sobre `OPTIMIZE TABLE` en la documentación oficial.

1.1.3.4.2 ANALYZE TABLE `ANALYZE TABLE` analiza y almacena la distribución de claves en una tabla. Ésta distribución se usa para determinar el orden que el servidor seguirá para combinar tablas en un JOIN, así como para decidir qué índices se usarán en una consulta. Es útil después de insertar una gran cantidad de datos y cuando creamos un nuevo índice.

```
ANALYZE [NO_WRITE_TO_BINLOG | LOCAL]
    TABLE tbl_name [, tbl_name] ...
```

Puede encontrar más información sobre `ANALYZE TABLE` en la documentación oficial.

1.1.4 Optimización de consultas e índices

1.1.4.1 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: {
```

```
20      SELECT statement
21    | DELETE statement
22    | INSERT statement
23    | REPLACE statement
24    | UPDATE statement
25  }
```

1.2 Ejemplos de optimización de consultas

1.2.1 Ejemplo 1 (INDEX)

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

```
1  SELECT nombre_contacto, telefono
2  FROM cliente
3  WHERE pais = 'France';
```

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

```
1  EXPLAIN SELECT nombre_contacto, telefono
2  FROM cliente
3  WHERE pais = 'France';
4
5  +--
       --+-------------+---------+------------+------+---------------+------+---------+------+--
6  | id | select_type | table   | partitions | type | possible_keys | key  |
       key_len | ref  | rows | filtered | Extra       |
7  +--
       --+-------------+---------+------------+------+---------------+------+---------+------+--
8  |  1 | SIMPLE      | cliente | NULL       | ALL  | NULL          | NULL | NULL
          | NULL |   36 |    10.00 | Using where |
9  +--
       --+-------------+---------+------------+------+---------------+------+---------+------+--
```

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`

```
1  DESCRIBE cliente;
2
```

```
+---------------------------+--------------+------+-----+---------+-------+
| 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_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      |         |               |
+--
    -------+------------+---------------------------+--------------+-------------------------
    
```

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_paix                  |            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.

1.2.2 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');
```

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');
```

1.2.3 Ejemplo 3 (FULLTEXT INDEX)

En este ejemplo vamos a trabajar con una base de datos llamada `viajes` que contiene la tabla `lugares` que almacena en una columna la descripción con texto enriquecido con etiquetas HTML.

El script SQL de creación de la base de datos es el siguiente.

```sql
DROP DATABASE IF EXISTS viajes;
CREATE DATABASE viajes CHARACTER SET utf8mb4;
USE viajes;

CREATE TABLE lugares (
  id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
  nombre VARCHAR(100) NOT NULL,
  descripcion TEXT NOT NULL
);

INSERT INTO lugares VALUES (1, 'París', 'Viaje a <strong>París</strong>, fascinado por la <strong>Torre Eiffel</strong> iluminada de noche y el museo del <strong>Louvre</strong> con la <strong>Mona Lisa</strong>.');

INSERT INTO lugares VALUES (2, 'Santorini', 'Pintoresco pueblo de <strong>Santorini</strong> con casas blancas y tejados azules, playas de arena volcánica. Cuenta con museos fascinantes como el del Louvre que muestran la rica historia de la isla y su cultura.');

INSERT INTO lugares VALUES (3, 'Gran Cañon', 'Impresionante <strong>Gran Cañón</strong> con paredes rocosas y espectaculares puestas de sol.');

INSERT INTO lugares VALUES (4, 'Machu Pichu', 'Ruinas antiguas de <strong>Machu Picchu</strong>, caminar por calles empedradas y admirar templos y terrazas.');

INSERT INTO lugares VALUES (5, 'Tokio', 'Contraste de tradición y modernidad en <strong>Tokio</strong>, con templos históricos y brillantes letreros de neón.');
```

El problema que queremos resolver es que queremos realizar una búsqueda de una frase exacta sobre la columna `descripcion`, pero tenemos el inconveniente de que esta columna contiene etiquetas HTML, lo que dificulta la búsqueda de una frase exacta.

Por ejemplo, suponga que queremos buscar todas las filas que contengan la frase `museo del Louvre`. Si utilizamos la siguiente consulta no obtendremos ningún resultado, porque en la tabla `lugares` la fila que contiene esa frase tiene la palabra Louvre está encerrada entre etiquetas: `museo del <strong>Louvre</strong>`.

```sql
SELECT *
FROM lugares
WHERE descripcion LIKE '%museo del Louvre%';
```

Paso 1

La primera solución que vamos a realizar consiste en utilizar la función `REGEXP_REPLACE` para eliminar las etiquetas HTML que aparecen en el texto de la descripción.

La expresión regular que nos permite eliminar las etiquetas HTML es: `"<[^>]+>"`. Vamos a analizar cada uno de los elementos que forman la expresión:

- `"<"`: Busca el caracter < dentro del texto.
- `[^>]+`: Entre los corchetes indicamos que vamos a seleccionar todos los caracteres que no sean el carácter >. El símbolo + indica que se deben buscar uno o más caracteres que cumplan la regla definida dentro de los corchetes.
- `">"`: Busca el caracter > dentro del texto.

La consulta SQL quedaría así:

```sql
SELECT REGEXP_REPLACE(descripcion, "<[^>]+>", "")
FROM lugares
WHERE REGEXP_REPLACE(descripcion, "<[^>]+>", "") LIKE '%museo del Louvre%';
```

Esta consulta **no es eficiente porque tiene que recorrer todas las filas** de la tabla para hacer la búsqueda.

Podemos utilizar el operador `EXPLAIN` para obtener información sobre cómo se está realizando la consulta.

```sql
EXPLAIN SELECT REGEXP_REPLACE(descripcion, "<[^>]+>", "")
FROM lugares
WHERE REGEXP_REPLACE(descripcion, "<[^>]+>", "") LIKE '%museo del Louvre%';

+--
    --+-------------+---------+------------+------+---------------+---------+---------+------
    

| id | select_type | table   | partitions | type | possible_keys | key     |
    key_len | ref   | rows | filtered | Extra       |
+--
    --+-------------+---------+------------+------+---------------+---------+---------+------
    

|  1 | SIMPLE      | lugares | NULL       | ALL  | NULL          | NULL    |
    NULL    | NULL  |    5 |   100.00 | Using where  |
+--
    --+-------------+---------+------------+------+---------------+---------+---------+------
    
```

En la columna `type` podemos observar que es necesario realizar un escaneo completo de toda la tabla y en la columna `rows` vemos que se han recorrido las 5 filas que tiene la tabla.

Paso 2

Para evitar tener que recorrer toda la tabla durante la búsqueda vamos a crear índice de tipo `FULLTEXT` sobre la columna `descripcion` que es la que contiene el texto enriquecido con etiquetas.

```sql
CREATE FULLTEXT INDEX idx_nombre ON lugares(descripcion);
```

Hacemos una búsqueda sobre el índice que acabamos de crear, pero tenemos el inconveniente de que no podemos utilizar la función `REGEXP_REPLACE` dentro de las cláusulas `MATCH` y `AGAINST`.

Por lo tanto, no vamos a poder utilizar una búsqueda de frase completa porque con las cláusulas `MATCH` y `AGAINST` no podemos eliminar las etiquetas HTML que aparecen en el texto de la descripción.

La consulta SQL quedaría así:

```
SELECT *, MATCH(descripcion) AGAINST ('museo del Louvre')
FROM lugares
WHERE MATCH(descripcion) AGAINST ('museo del Louvre');
```

Esta consulta es más eficiente que la anterior porque está haciendo uso de índices, pero **el resultado no es correcto del todo** porque devuelve filas con contenido relacionado con las palabras de búsqueda.

```
+---+-----------+-----------------------------------+---------------------+
| 1 | París     | Viaje a <strong>París</strong>... | 0.805271565914154   |
| 2 | Santorini | Pintoresco pueblo de <strong>...  | 0.31671249866485596 |
+---+-----------+-----------------------------------+---------------------+
```

Podemos utilizar el operador `EXPLAIN` para obtener información sobre cómo se está realizando la consulta.

```
EXPLAIN SELECT *, MATCH(descripcion) AGAINST ('museo del Louvre')
FROM lugares
WHERE MATCH(descripcion) AGAINST ('museo del Louvre');

+--
    --+-------------+---------+------------+----------+---------------+------------+---------

| id | select_type | table   | partitions | type     | possible_keys | key
           | key_len | ref   | rows | filtered | Extra         |
+--
    --+-------------+---------+------------+----------+---------------+------------+---------

| 1  | SIMPLE      | lugares | NULL       | fulltext | idx_nombre    |
   idx_nombre | 0       | const | 1    | 100.00   | Using where   |
+--
    --+-------------+---------+------------+----------+---------------+------------+---------
```

En la columna `type` podemos observar que no es necesario realizar un escaneo completo de toda la tabla porque está utilizando un índice de tipo `FULLTEXT`, y en la columna `rows` vemos que sólo se ha escaneado 1 fila de la tabla.

Paso 3

Podemos mejorar la consulta anterior para hacer uso del índice de tipo `FULLTEXT` y filtrar únicamente las filas que coinciden con la búsqueda exacta haciendo uso de la función `REGEXP_REPLACE`. En este caso vamos a utilizar dos condiciones en la cláusula `WHERE`:

- La primera para hacer uso del índice con las cláusulas `MATCH` y `AGAINST`, y filtrar únicamente las filas que pueden tener el resultado que estamos buscando.
- Y la segunda será una expresión regular con la función `REGEXP_REPLACE` para eliminar las etiquetas HTML y hacer una comparación exacta con la cadena que estamos buscando.

La consulta optimizada quedaría así:

```
SELECT *
FROM lugares
WHERE
  MATCH(descripcion) AGAINST ('museo del Louvre') AND
```

```
5    REGEXP_REPLACE(descripcion, "<[^>]+>", "") LIKE '%museo del Louvre%';
```

Esta consulta devuelve el resultado que estamos buscando.

```
1  +---+-----------+-----------------------------------+
2  | 1 | París     | Viaje a <strong>París</strong>... |
3  +---+-----------+-----------------------------------+
```

Podemos utilizar el operador `EXPLAIN` para obtener información sobre cómo se está realizando la consulta.

```
1  EXPLAIN SELECT *
2  FROM lugares
3  WHERE
4    MATCH(descripcion) AGAINST ('museo del Louvre') AND
5    REGEXP_REPLACE(descripcion, "<[^>]+>", "") LIKE '%museo del Louvre%';
6
7  +--
       --+-------------+---------+------------+----------+---------------+------------+---------
8  | id | select_type | table   | partitions | type     | possible_keys | key
             | key_len | ref   | rows | filtered | Extra        |
9  +--
       --+-------------+---------+------------+----------+---------------+------------+---------
10 | 1  | SIMPLE      | lugares | NULL       | fulltext | idx_nombre    |
       idx_nombre | 0       | const | 1    | 100.00   | Using where  |
11 +--
       --+-------------+---------+------------+----------+---------------+------------+---------
```

En la columna `type` podemos observar que no es necesario realizar un escaneo completo de toda la tabla porque está utilizando un índice de tipo `FULLTEXT`, y en la columna `rows` vemos que sólo se ha escaneado 1 fila de la tabla.

2 Ejercicios

2.1 Base de datos: Jardinería

1. Consulte cuáles son los índices que hay en la tabla `producto` utilizando las instrucciones SQL que nos permiten obtener esta información de la tabla.
2. Haga uso de `EXPLAIN` para obtener información sobre cómo se están realizando las consultas y diga cuál de las dos consultas realizará menos comparaciones para encontrar el producto que estamos buscando. ¿Cuántas comparaciones se realizan en cada caso? ¿Por qué?.

```sql
SELECT *
FROM producto
WHERE codigo_producto = 'OR-114';
```

```sql
SELECT *
FROM producto
WHERE nombre = 'Evonimus Pulchellus';
```

3. Suponga que estamos trabajando con la base de datos `jardineria` y queremos saber optimizar las siguientes consultas. ¿Cuál de las dos sería más eficiente?. Se recomienda hacer uso de `EXPLAIN` para obtener información sobre cómo se están realizando las consultas.

```sql
SELECT AVG(total)
FROM pago
WHERE YEAR(fecha_pago) = 2008;
```

```sql
SELECT AVG(total)
FROM pago
WHERE fecha_pago >= '2008-01-01' AND fecha_pago <= '2008-12-31';
```

Nota: Lectura recomendada sobre la función YEAR y el uso de índices.

4. Optimiza la siguiente consulta creando índices cuando sea necesario. Se recomienda hacer uso de `EXPLAIN` para obtener información sobre cómo se están realizando las consultas.

```sql
SELECT *
FROM cliente INNER JOIN pedido
ON cliente.codigo_cliente = pedido.codigo_cliente
WHERE cliente.nombre_cliente LIKE 'A%';
```

5. ¿Por qué no es posible optimizar el tiempo de ejecución de las siguientes consultas, incluso haciendo uso de índices?

```sql
SELECT *
FROM cliente INNER JOIN pedido
ON cliente.codigo_cliente = pedido.codigo_cliente
WHERE cliente.nombre_cliente LIKE '%A%';

SELECT *
FROM cliente INNER JOIN pedido
ON cliente.codigo_cliente = pedido.codigo_cliente
WHERE cliente.nombre_cliente LIKE '%A';
```

6. Crea un índice de tipo `FULLTEXT` sobre las columnas `nombre` y `descripcion` de la tabla `producto`.

7. Una vez creado el índice del ejercicio anterior realiza las siguientes consultas haciendo uso de la función `MATCH`, para buscar todos los productos que:

- Contienen la palabra `planta` en el nombre o en la descripción. Realice una consulta para cada uno de los modos de búsqueda *full-text* que existen en MySQL (`IN NATURAL LANGUAGE MODE`, `IN BOOLEAN MODE` y `WITH QUERY EXPANSION`) y compare los resultados que ha obtenido en cada caso.
- Contienen la palabra `planta` seguida de cualquier carácter o conjunto de caracteres, en el nombre o en la descripción.
- **Empiezan** con la palabra `planta` en el nombre o en la descripción.
- Contienen la palabra `tronco` **o** la palabra `árbol` en el nombre o en la descripción.
- Contienen la palabra `tronco` **y** la palabra `árbol` en el nombre o en la descripción.
- Contienen la palabra `tronco` pero no contienen la palabra `árbol` en el nombre o en la descripción.
- Contiene la frase `proviene de las costas` en el nombre o en la descripción.

8. Crea un índice de tipo `INDEX` compuesto por las columnas `apellido_contacto` y `nombre_contacto` de la tabla `cliente`.

9. Una vez creado el índice del ejercicio anterior realice las siguientes consultas haciendo uso de `EXPLAIN`:

- Busca el cliente `Javier Villar`. ¿Cuántas filas se han examinado hasta encontrar el resultado?
- Busca el cliente anterior utilizando solamente el apellido `Villar`. ¿Cuántas filas se han examinado hasta encontrar el resultado?
- Busca el cliente anterior utilizando solamente el nombre `Javier`. ¿Cuántas filas se han examinado hasta encontrar el resultado? ¿Qué ha ocurrido en este caso?

10. Calcula cuál podría ser un buen valor para crear un índice sobre un prefijo de la columna `nombre_cliente` de la tabla `cliente`. Tenga en cuenta que un buen valor será aquel que nos permita utilizar el menor número de caracteres para diferenciar todos los valores que existen en la columna sobre la que estamos creando el índice.

- En primer lugar calculamos cuántos valores distintos existen en la columna `nombre_cliente`. Necesitarás utilizar la función `COUNT` y `DISTINCT`.
- Haciendo uso de la función `LEFT` ve calculando el número de caracteres que necesitas utilizar como prefijo para diferenciar todos los valores de la columna. Necesitarás la función `COUNT`, `DISTINCT` y `LEFT`.
- Una vez que hayas encontrado el valor adecuado para el prefijo, crea el índice sobre la columna `nombre_cliente` de la tabla `cliente`.
- Ejecuta algunas consultas de prueba sobre el índice que acabas de crear.

3 Práctica

Práctica 14. Optimización de consultas en MySQL

3.1 Base de datos: `netflix`

- `netflix-schema-without-index.sql`.
- `netflix-data.sql`.

4 Referencias

- *Capítulo 7, Optimización de consultas,* del libro **Gestión de Bases de Datos**. 2ª Edición. Ra-Ma. Luis Hueso Ibáñez.
- Introducción a índices en MySQL.Rafael Vindel Amor.
- Optimización de consultas en MySQL. Eduardo Sánchez Contreras.
- Optimization. Documentación oficial de MySQL.
- Optimization and Indexes. Documentación oficial de MySQL.
- Optimizing SQL Statements. Documentación oficial de MySQL.
- Use the index Luke. A guide to database performance for developers.
- MySQL Indexes. MySQLTutorial.org
- Building the best INDEX for a given SELECT. Documentación oficial de MariaDB.

5 Licencia