# Optimización de consultas

Apuntes de BD para DAM

Blanca Calderón

Curso 2023/2024

# Índice

1. [Optimización de consultas](#_bookmark0) 1
   1. [**Índices**](#_bookmark1) 1
      1. [Tipos de índices](#_bookmark2) 1
      2. [Índices en MySQL](#_bookmark3) 2
      3. [Gestión de índices](#_bookmark4) 4
         1. [**Crear índices**](#_bookmark5) **4**
            1. [CREATE INDEX](#_bookmark6) 4
            2. [ALTER TABLE](#_bookmark7) 6
            3. [CREATE TABLE](#_bookmark8) 8
         2. [**Mostrar los índices**](#_bookmark9) 10
            1. [SHOW INDEX](#_bookmark10) 10
            2. [DESCRIBE](#_bookmark11) 10
         3. [**Eliminar índices**](#_bookmark12) 11
            1. [DROP INDEX](#_bookmark13) 11
            2. [ALTER TABLE](#_bookmark14) 11
         4. [**Actualizar y reordenar índices**](#_bookmark15) 12
            1. [OPTIMIZE TABLE](#_bookmark16) 12
            2. [ANALYZE TABLE](#_bookmark17) 12
      4. [Optimización de consultas e índices](#_bookmark18) 12
         1. [EXPLAIN](#_bookmark19) 12
   2. [Ejemplos de optimización de consultas](#_bookmark20) 13
      1. [Ejemplo 1 (INDEX)](#_bookmark21) 13
      2. [Ejemplo 2 (FULLTEXT INDEX)](#_bookmark22) 16
      3. [Ejemplo 3 (FULLTEXT INDEX)](#_bookmark23) 17
2. [Ejercicios](#_bookmark24) 21
   1. [Base de datos: Jardinería](#_bookmark25) 21
3. [Referencias](#_bookmark28) 24

## 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 puede 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.

### 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.

### Índices en MySQL

La mayoría de los índices que se utilizan en MySQL son almacenados en [árboles B](https://es.wikipedia.org/wiki/%C3%81rbol-B) (*B‑trees*). Los [árboles B](https://es.wikipedia.org/wiki/%C3%81rbol-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 (es un tipo de índice 'especial' que se aplica a campos de texto de gran tamaño, como el [tipo TEXT](https://dev.mysql.com/doc/refman/8.0/en/blob.html) y sus variantes, aunque también puede realizarse sobre char y varchar.)

Ejemplo de un árbol B (*B‑tree*):

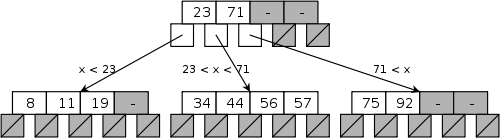


Imagen: Ejemplo de un árbol B. *B‑tree*. Nagae. 2007. Wikipedia.

Los índices que se utilizan sobre datos espaciales se almacenan en [Árboles R](https://es.wikipedia.org/wiki/%C3%81rbol-R) (*R‑trees*).

* SPATIAL (Line o Curve)

Ejemplo de un árbol R (*R‑tree*):

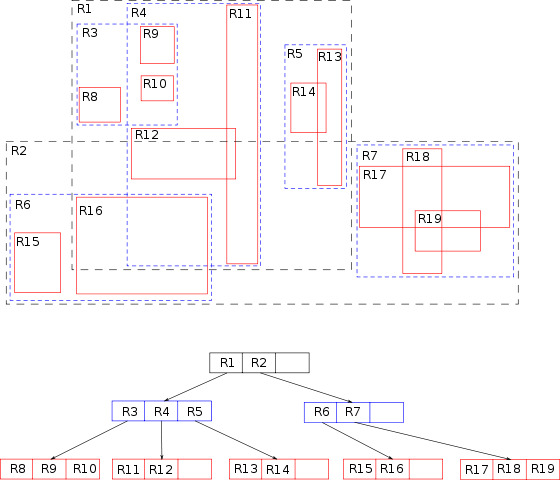


Imagen: Ejemplo de un árbol R. *R‑tree*. Skinkie. 2010. Wikipedia.

Y por último, las tablas almacenadas en memoria utilizan [índices *hash*](https://es.wikipedia.org/wiki/Tabla_hash).

* MEMORY

Ejemplo de índices *hash*:

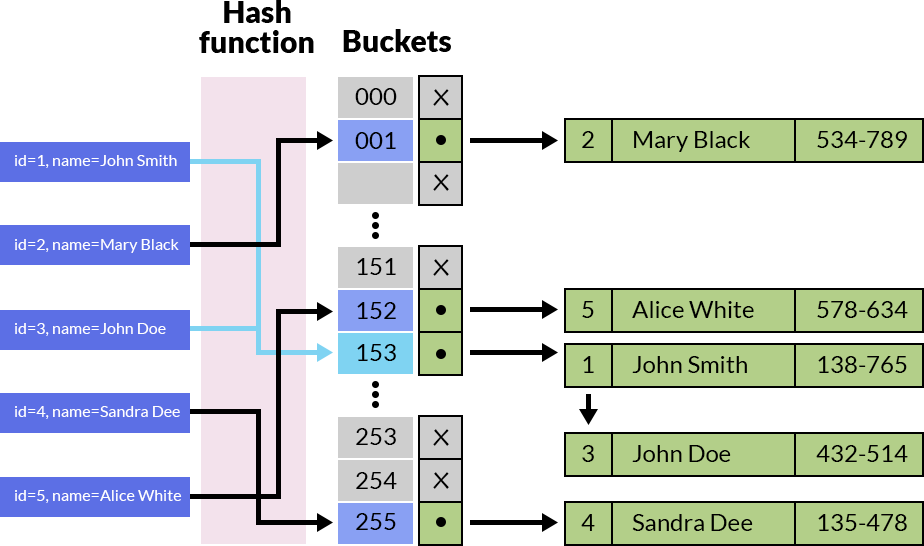


Imagen: Ejemplo de índices *hash*. [Emil Drkušić. 2016](https://www.vertabelo.com/blog/technical-articles/all-about-indexes-part-2-mysql-index-structure-and-performance).

### Gestión de índices

#### Crear índices

* + - * 1. **CREATE INDEX** La sintaxis para crear índices en MySQL es la siguiente:

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

**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](https://dev.mysql.com/doc/refman/5.5/en/create-index.html).

#### Ejemplo 1: Uso de *INDEX*

El siguiente ejemplo crea un índice con el nombre idx\_pais sobre la columna pais de la tabla cliente.

1 **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.

1 **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.

1 **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.

1 **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.

1 **CREATE** FULLTEXT **INDEX** idx\_nombre\_descripcion **ON** producto(nombre, descripcion);

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

1. **SELECT** \*
2. **FROM** producto
3. **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:**

1

2

3

4

5

6

7

8

9

**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. **\***: Indica un comodín para hacer coincidencias parciales. Puede encontrar más información sobre los operadores en la [documentación oficial de MySQL](https://dev.mysql.com/doc/refman/8.0/en/fulltext-boolean.html).

**WITH QUERY EXPANSION**: Esta opción se utiliza para ampliar los resultados de búsqueda mostrando contenidos relacionados.

* + - * 1. **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.

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

**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}

|  |  |  |
| --- | --- | --- |
| 1. index\_option: { 2. KEY\_BLOCK\_SIZE [=] **value** | | |
| 26 | | | index\_type |
| 27 | | | WITH PARSER parser\_name |
| 28 | | | COMMENT 'string' |
| 29 | | | {VISIBLE | INVISIBLE} |
| 30 } |  |  |

Puede encontrar más información sobre la creación de índices en MySQL con la sentencia ALTER TABLE en la [documentación oficial](https://dev.mysql.com/doc/refman/8.0/en/alter-table.html).

#### Ejemplo 1: Uso de INDEX

El siguiente ejemplo crea un índice con el nombre idx\_nombre sobre la columna nombre de la tabla

cliente.

1 **ALTER TABLE** cliente **ADD INDEX** idx\_nombre (nombre);

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

1 **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.

1 **ALTER TABLE** empleado **ADD UNIQUE INDEX** idx\_email (email);

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

1 **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.

1 **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.

1 **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.

1 **ALTER TABLE** cliente **ADD INDEX** idx\_nombre\_cliente (nombre\_cliente(25));

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

1 **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.

1 **ALTER TABLE** producto **ADD** FULLTEXT **INDEX** idx\_nombre\_descripcion (nombre, descripcion);

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

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

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

IN NATURAL LANGUAGE MODE.

1

2

3

4

5

6

7

8

9

10

**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. **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.

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

**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

21 | check\_constraint\_definition

22 }

Puede encontrar la sintaxis completa de la sentencia CREATE TABLE en la [documentación oficial](https://dev.mysql.com/doc/refman/8.0/en/create-table.html).

#### Ejemplo 1:

El siguiente ejemplo crea un índice con el nombre idx\_nombre sobre la columna nombre de la tabla

cliente.

1. **CREATE TABLE** cliente (
2. id **INT** UNSIGNED AUTO\_INCREMENT **PRIMARY KEY**,
3. nombre **VARCHAR**(50) **NOT NULL**,
4. email **VARCHAR**(15) **NOT NULL**,
5. telefono **VARCHAR**(9) **NOT NULL**,
6. **INDEX** idx\_nombre (nombre)

7 );

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

1. **CREATE TABLE** cliente (
2. id **INT** UNSIGNED AUTO\_INCREMENT **PRIMARY KEY**,
3. nombre **VARCHAR**(50) **NOT NULL**,
4. email **VARCHAR**(15) **NOT NULL**,
5. telefono **VARCHAR**(9) **NOT NULL**,
6. **INDEX** (nombre)

7 );

#### 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.

1. **CREATE TABLE** cliente (
2. id **INT** UNSIGNED AUTO\_INCREMENT **PRIMARY KEY**,
3. nombre **VARCHAR**(50) **NOT NULL**,
4. email **VARCHAR**(15) **NOT NULL UNIQUE**,
5. telefono **VARCHAR**(9) **NOT NULL**

6 );

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

1. **CREATE TABLE** cliente (
2. id **INT** UNSIGNED AUTO\_INCREMENT **PRIMARY KEY**,
3. nombre **VARCHAR**(50) **NOT NULL**,
4. email **VARCHAR**(15) **NOT NULL**,
5. telefono **VARCHAR**(9) **NOT NULL**,
6. **UNIQUE** (email)

7 );

#### Mostrar los índices

* + - * 1. **SHOW INDEX** La sintaxis de la sentencia SHOW INDEX para mostrar los índices de una tabla en MySQL es la siguiente:

1. SHOW {**INDEX** | INDEXES | KEYS}
2. {**FROM** | **IN**} tbl\_name
3. [{**FROM** | **IN**} db\_name]
4. [**WHERE** expr]

Puede encontrar más información en la [documentación oficial](https://dev.mysql.com/doc/refman/5.7/en/show-index.html).

#### Ejemplo:

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

1 SHOW **INDEX FROM** cliente;

2

3 +--

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

4 | **Table**| Non\_unique | Key\_name | Seq\_in\_index | Column\_name | **Collation** | Cardinality | Sub\_part | Packed | **Null** | Index\_type | Comment | Index\_comment |

5 +--

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

1. | cliente | 0 | **PRIMARY** | 1 | codigo\_cliente | A | 36 | **NULL** | **NULL** |

| BTREE | | |

1. | cliente | 1 | codigo\_empleado\_rep\_ventas | 1 | codigo\_empleado\_rep\_ventas | A | 11 | **NULL** | **NULL** | YES | BTREE | | |

8 +--

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

* + - * 1. **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.

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 **DESCRIBE** cliente;  2  3 +----------------------------+---------------+------+-----+---------+ +  4 | Field | Type | **Null** | **Key** | **Default** | Extra |  5 +----------------------------+---------------+------+-----+---------+ + | | | | | | | | | | | |
| 6 | | codigo\_cliente | | | **int**(11) | | | **NO** | | | PRI | | | **NULL** | | | | |
| 7 | | nombre\_cliente | | | **varchar**(50) | | | **NO** | | |  | | | **NULL** | | | | |
| 8 | | nombre\_contacto | | | **varchar**(30) | | | YES | | |  | | | **NULL** | | | | |
| 9 | | apellido\_contacto | | | **varchar**(30) | | | YES | | |  | | | **NULL** | | | | |
| 10 | | telefono | | | **varchar**(15) | | | **NO** | | |  | | | **NULL** | | | | |
| 11 | | fax | | | **varchar**(15) | | | **NO** | | |  | | | **NULL** | | | | |

20 +----------------------------+---------------+------+ +

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 12 | | linea\_direccion1 | | | **varchar**(50) | | | **NO** | | | | | **NULL** | | | | |
| 13 | | linea\_direccion2 | | | **varchar**(50) | | | YES | | | | | **NULL** | | | | |
| 14 | | ciudad | | | **varchar**(50) | | | **NO** | | | | | **NULL** | | | | |
| 15 | | region | | | **varchar**(50) | | | YES | | | | | **NULL** | | | | |
| 16 | | pais | | | **varchar**(50) | | | YES | | | | | **NULL** | | | | |
| 17 | | codigo\_postal | | | **varchar**(10) | | | YES | | | | | **NULL** | | | | |
| 18 | | codigo\_empleado\_rep\_ventas | | | **int**(11) | | | YES | | | MUL | | **NULL** | | | | |
| 19 | 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.

#### Eliminar índices

* + - * 1. **DROP INDEX** La sintaxis para eliminar índices con la sentencia DROP INDEX en MySQL es la siguiente:

1

2

3

4

5

6

7

8

**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](https://dev.mysql.com/doc/refman/5.7/en/drop-index.html).

#### Ejemplo:

El siguiente ejemplo elimina un índice con el nombre idx\_nombre de la tabla cliente.

1 **DROP INDEX** idx\_nombre **ON** cliente;

* + - * 1. **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.

1

2

3

4

5

6

7

**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](https://dev.mysql.com/doc/refman/8.0/en/alter-table.html).

#### Ejemplo:

El siguiente ejemplo elimina un índice con el nombre idx\_nombre de la tabla cliente.

1 **ALTER TABLE** cliente **DROP INDEX** idx\_nombre;

#### Actualizar y reordenar índices

* + - * 1. OPTIMIZE TABLE OPTIMIZE TABLE nos permite desfragmentar una tabla, así como actualizar y reordenar los índices. La sintaxis en MySQL es la siguiente:

1. OPTIMIZE [NO\_WRITE\_TO\_BINLOG | **LOCAL**]
2. **TABLE** tbl\_name [, tbl\_name] ...

Puede encontrar más información sobre OPTIMIZE TABLE en la [documentación oficial](https://dev.mysql.com/doc/refman/5.7/en/optimize-table.html).

* + - * 1. ANALYZE TABLE ANALYZE TABLE analiza y almacena la distribución de claves en una tabla. Esta 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.

1. ANALYZE [NO\_WRITE\_TO\_BINLOG | **LOCAL**]
2. **TABLE** tbl\_name [, tbl\_name] ...

Puede encontrar más información sobre ANALYZE TABLE en la [documentación oficial](https://dev.mysql.com/doc/refman/5.7/en/analyze-table.html).

### Optimización de consultas e índices

#### 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.

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

{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

21

22

23

24

25

**SELECT** statement

| **DELETE** statement

| **INSERT** statement

| REPLACE statement

| **UPDATE** statement

}

## Ejemplos de optimización de consultas

### Ejemplo 1 (INDEX)

Suponga que estamos trabajando con la base de datos jardineria y queremos optimizar la siguiente con‑ sulta.

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.

20 +----------------------------+---------------+------+----

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 3 +----------------------------+---------------+------+-----+---------+ +  4 | Field | Type | **Null** | **Key** | **Default** | Extra |  5 +----------------------------+---------------+------+-----+---------+ + | | | | | | | | | | | |
| 6 | | codigo\_cliente | | | **int**(11) | | | **NO** | | | PRI | | | **NULL** | | | | |
| 7 | | nombre\_cliente | | | **varchar**(50) | | | **NO** | | |  | | | **NULL** | | | | |
| 8 | | nombre\_contacto | | | **varchar**(30) | | | YES | | |  | | | **NULL** | | | | |
| 9 | | apellido\_contacto | | | **varchar**(30) | | | YES | | |  | | | **NULL** | | | | |
| 10 | | telefono | | | **varchar**(15) | | | **NO** | | |  | | | **NULL** | | | | |
| 11 | | fax | | | **varchar**(15) | | | **NO** | | |  | | | **NULL** | | | | |
| 12 | | linea\_direccion1 | | | **varchar**(50) | | | **NO** | | |  | | | **NULL** | | | | |
| 13 | | linea\_direccion2 | | | **varchar**(50) | | | YES | | |  | | | **NULL** | | | | |
| 14 | | ciudad | | | **varchar**(50) | | | **NO** | | |  | | | **NULL** | | | | |
| 15 | | region | | | **varchar**(50) | | | YES | | |  | | | **NULL** | | | | |
| 16 | | pais | | | **varchar**(50) | | | YES | | |  | | | **NULL** | | | | |
| 17 | | codigo\_postal | | | **varchar**(10) | | | YES | | |  | | | **NULL** | | | | |
| 18 | | codigo\_empleado\_rep\_ventas | | | **int**(11) | | | YES | | | MUL | | | **NULL** | | | | |
| 19 | limite\_credito | **decimal**(15,2) | YES | | | | | | | | | | **NULL** | |  -+---------+ + | | | |

* SHOW INDEX

1 SHOW **INDEX FROM** cliente;

2

3 +--

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

4 | **Table** | Non\_unique | Key\_name | Seq\_in\_index | Column\_name | **Collation** | Cardinality | Sub\_part | Packed | **Null** | Index\_type | Comment | Index\_comment |

5 +--

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

1. | cliente | 0 | **PRIMARY** | 1 | codigo\_cliente | A | 36 | **NULL** | **NULL** |

| BTREE | | |

1. | cliente | 1 | codigo\_empleado\_rep\_ventas | 1 | codigo\_empleado\_rep\_ventas | A | 11 | **NULL** | **NULL** | YES | BTREE | | |

8 +--

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

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:

1 **CREATE INDEX** idx\_pais **ON** cliente(pais);

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

20 +----------------------------+---------------+------+----

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 3 +----------------------------+---------------+------+-----+---------+ +  4 | Field | Type | **Null** | **Key** | **Default** | Extra |  5 +----------------------------+---------------+------+-----+---------+ + | | | | | | | | | | | |
| 6 | | codigo\_cliente | | | **int**(11) | | | **NO** | | | PRI | | | **NULL** | | | | |
| 7 | | nombre\_cliente | | | **varchar**(50) | | | **NO** | | |  | | | **NULL** | | | | |
| 8 | | nombre\_contacto | | | **varchar**(30) | | | YES | | |  | | | **NULL** | | | | |
| 9 | | apellido\_contacto | | | **varchar**(30) | | | YES | | |  | | | **NULL** | | | | |
| 10 | | telefono | | | **varchar**(15) | | | **NO** | | |  | | | **NULL** | | | | |
| 11 | | fax | | | **varchar**(15) | | | **NO** | | |  | | | **NULL** | | | | |
| 12 | | linea\_direccion1 | | | **varchar**(50) | | | **NO** | | |  | | | **NULL** | | | | |
| 13 | | linea\_direccion2 | | | **varchar**(50) | | | YES | | |  | | | **NULL** | | | | |
| 14 | | ciudad | | | **varchar**(50) | | | **NO** | | |  | | | **NULL** | | | | |
| 15 | | region | | | **varchar**(50) | | | YES | | |  | | | **NULL** | | | | |
| 16 | | pais | | | **varchar**(50) | | | YES | | | MUL | | | **NULL** | | | | |
| 17 | | codigo\_postal | | | **varchar**(10) | | | YES | | |  | | | **NULL** | | | | |
| 18 | | codigo\_empleado\_rep\_ventas | | | **int**(11) | | | YES | | | MUL | | | **NULL** | | | | |
| 19 | limite\_credito | **decimal**(15,2) | YES | | | | | | | | | | **NULL** | |  -+---------+ + | | | |

* + SHOW INDEX

1 SHOW **INDEX FROM** cliente;

2

3 +--

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

4 | **Table** | Non\_unique | Key\_name | Seq\_in\_index | Column\_name | **Collation** | Cardinality | Sub\_part | Packed | **Null** | Index\_type | Comment | Index\_comment |

5 +--

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

1. | cliente | 0 | **PRIMARY** | 1 | codigo\_cliente | A | 36 | **NULL** | **NULL** |

| BTREE | | |

1. | cliente | 1 | codigo\_empleado\_rep\_ventas | 1 | codigo\_empleado\_rep\_ventas | A | 11 | **NULL** | **NULL** | YES | BTREE | | |
2. | cliente | 1 | idx\_paix | 1 | pais

| A | 5 | **NULL** | **NULL** | YES |

BTREE | | |

9 +--

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

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.

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** |

7 +--

key\_len | ref | **rows** | filtered | Extra |

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

8 | 1 | SIMPLE | cliente | **NULL** | ref | idx\_pais | idx\_pais |

203 | const | 2 | 100.00 | **NULL** |

9 +--

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

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.

### 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:

1. **SELECT** \*
2. **FROM** producto
3. **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.

1. EXPLAIN **SELECT** \*
2. **FROM** producto
3. **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.

1 **CREATE** FULLTEXT **INDEX** idx\_nombre\_descripcion **ON** producto(nombre, descripcion);

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

1. **SELECT** \*
2. **FROM** producto
3. **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.

1. EXPLAIN **SELECT** \*
2. **FROM** producto
3. **WHERE MATCH**(nombre, descripcion) AGAINST ('acero');

### 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.

***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 volcá***

***nica. 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>.

1. **SELECT** \*
2. **FROM** lugares
3. **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 ca‑ rácter >. 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:

1. **SELECT** REGEXP\_REPLACE(descripcion, "<[^>]+>", "")
2. **FROM** lugares
3. **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.

1 EXPLAIN **SELECT** REGEXP\_REPLACE(descripcion, "<[^>]+>", "")

2 **FROM** lugares

3 **WHERE** REGEXP\_REPLACE(descripcion, "<[^>]+>", "") **LIKE** '%museo del Louvre%';

4

5 +--

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

6 | id | select\_type | **table** | partitions | type | possible\_keys | **key** | key\_len | ref | **rows** | filtered | Extra |

7 +--

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

8 | 1 | SIMPLE | lugares | **NULL** | **ALL** | **NULL** | **NULL** |

**NULL** | **NULL** | 5 | 100.00 | **Using where** |

9 +--

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

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.

1 **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:

1. **SELECT** \*, **MATCH**(descripcion) AGAINST ('museo del Louvre')
2. **FROM** lugares
3. **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

2

3

4

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

| 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.

1 EXPLAIN **SELECT** \*, **MATCH**(descripcion) AGAINST ('museo del Louvre')

2 **FROM** lugares

3 **WHERE MATCH**(descripcion) AGAINST ('museo del Louvre');

4

5 +--

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

6 | id | select\_type | **table** | partitions | type | possible\_keys | **key**

| key\_len | ref | **rows** | filtered | Extra |

7 +--

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

8 | 1 | SIMPLE | lugares | **NULL** | fulltext | idx\_nombre | idx\_nombre | 0 | const | 1 | 100.00 | **Using where** |

9 +--

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

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:

1. **SELECT** \*
2. **FROM** lugares
3. **WHERE**
4. **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

3

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

| 1 | París

| Viaje a <strong>París</strong>... |

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

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.

# *Ejercicios*

## 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?.

1. **SELECT** \*
2. **FROM** producto
3. **WHERE** codigo\_producto = 'OR-114';
4. **SELECT** \*
5. **FROM** producto
6. **WHERE** nombre = 'Evonimus Pulchellus';
   * 1. 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.
7. **SELECT AVG**(total)
8. **FROM** pago
9. **WHERE YEAR**(fecha\_pago) = 2008;
10. **SELECT AVG**(total)
11. **FROM** pago
12. **WHERE** fecha\_pago >= '2008-01-01' **AND** fecha\_pago <= '2008-12-31';

Nota: [Lectura recomendada sobre la función YEAR y el uso de índices](http://www.mysqltutorial.org/mysql-year/).

* + 1. 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.

1. **SELECT** \*
2. **FROM** cliente **INNER JOIN** pedido
3. **ON** cliente.codigo\_cliente = pedido.codigo\_cliente
4. **WHERE** cliente.nombre\_cliente **LIKE** 'A%';
   * 1. ¿Por qué no es posible optimizar el tiempo de ejecución de las siguientes consultas, incluso haciendo uso de índices?

Unidad 10. Optimización de consultas Curso 2023/2024

1

2

3

4

5

6

7

8

9

**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';

* + 1. Crea un índice de tipo FULLTEXT sobre las columnas nombre y descripcion de la tabla producto.
    2. 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.
    1. Crea un índice de tipo INDEX compuesto por las columnas apellido\_contacto y nombre\_contacto de la tabla cliente.
    2. 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?
    3. 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. Necesi‑ tarás utilizar la función COUNT y DISTINCT.
       - Haciendo uso de la función LEFT ve calculando el número de caracteres que necesitas utilizar como pre‑ fijo 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.

# 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](https://www.adictosaltrabajo.com/2015/09/11/introduccion-a-indices-en-mysql/).Rafael Vindel Amor.
    - [Optimización de consultas en MySQL](https://www.adictosaltrabajo.com/tutoriales/optimizacion-de-consultas-en-mysql/). Eduardo Sánchez Contreras.
    - [Optimization](https://dev.mysql.com/doc/refman/8.0/en/optimization.html). Documentación oficial de MySQL.
    - [Optimization and Indexes](https://dev.mysql.com/doc/refman/8.0/en/optimization-indexes.html). Documentación oficial de MySQL.
    - [Optimizing SQL Statements](https://dev.mysql.com/doc/refman/8.0/en/statement-optimization.html). Documentación oficial de MySQL.
    - [Use the index Luke](https://use-the-index-luke.com/). A guide to database performance for developers.
    - [MySQL Indexes](https://www.mysqltutorial.org/mysql-index/). MySQLTutorial.org
    - [Building the best INDEX for a given SELECT](https://mariadb.com/kb/en/building-the-best-index-for-a-given-select/). Documentación oficial de MariaDB.

***La salida que proporcionaste corresponde a la información obtenida mediante la instrucción EXPLAIN para una consulta en la tabla producto. Aquí está el significado de cada columna:***

id: Identificador de la consulta en el plan de ejecución.

select\_type: Tipo de la consulta.

table: Nombre de la tabla.

partitions: Particiones que se están utilizando.

type: Tipo de unión.

possible\_keys: Índices que podrían ser utilizados para la consulta.

key: Índice que realmente se está utilizando.

key\_len: Longitud del índice utilizado.

ref: Columna utilizada con el índice para la comparación.

rows: Número de filas examinadas.

filtered: Porcentaje de filas filtradas.

Extra: Información adicional sobre la consulta

***En MySQL, el atributo TYPE en la salida del plan de ejecución EXPLAIN se refiere al método de acceso utilizado para recuperar filas de la tabla. Aquí tienes algunos de los valores que puede tomar el atributo TYPE y su significado:***

ALL: Este valor indica que se está realizando un escaneo completo de la tabla. Esto significa que se están examinando todas las filas de la tabla para satisfacer la consulta. Es el método menos eficiente ya que implica examinar todas las filas, especialmente en tablas grandes.

index: Significa que se está utilizando un índice no único para buscar filas que coincidan con los criterios de la consulta. Aunque se está utilizando un índice, puede no ser eficiente si se están recuperando muchas filas o si el índice no es selectivo.

range: Esto indica que se está utilizando un rango de índice para buscar filas en función de un rango de valores. Por ejemplo, cuando se utiliza una cláusula WHERE con operadores como > o <, se puede utilizar un rango de índice para seleccionar las filas.

ref: Este tipo indica que se está utilizando un índice para buscar filas que coincidan con un valor específico o una lista de valores. Es más eficiente que un escaneo completo de la tabla, pero menos eficiente que un rango de índice.

eq\_ref: Similar a ref, pero se utiliza cuando el índice es único y se puede realizar una búsqueda de una sola fila.

const: Indica que se está utilizando una constante para buscar una única fila. Este es el método más eficiente ya que implica acceder directamente a una fila utilizando un índice único o una clave primaria.

system: Esto indica que se está accediendo a una sola fila de la tabla del sistema, como la tabla information\_schema.

***ORDEN DE LOS METODOS DE ACCESO DE LA CONSULTA ORDENADAS DE LA MAS EFICIENTE A LA MENOS.***

const: Acceso directo a una fila utilizando un índice único o una clave primaria. Es el método más eficiente ya que implica un acceso directo a la fila deseada.

eq\_ref: Similar a ref, pero se utiliza cuando el índice es único y se puede realizar una búsqueda de una sola fila. También es muy eficiente, ya que implica un acceso directo a una fila a través de un índice único.

ref: Utiliza un índice para buscar filas que coincidan con un valor específico o una lista de valores. Es más eficiente que un escaneo completo de la tabla, pero menos eficiente que const y eq\_ref.

range: Utiliza un rango de índice para buscar filas en función de un rango de valores. Es eficiente para buscar filas dentro de un rango determinado, pero puede ser menos eficiente que ref dependiendo del tamaño del rango y la selectividad del índice.

index: Utiliza un índice no único para buscar filas que coincidan con los criterios de la consulta. Es más eficiente que un escaneo completo de la tabla, pero puede ser menos eficiente que range y ref dependiendo de la selectividad del índice.

ALL: Realiza un escaneo completo de la tabla, examinando todas las filas para satisfacer la consulta. Es el método menos eficiente, especialmente en tablas grandes, ya que implica examinar todas las filas.

system: Acceso a una sola fila de la tabla del sistema, como la tabla information\_schema. Este método es específico para consultas internas del sistema y generalmente no se encuentra en consultas de usuario estándar.