Comandos DDL

¿Cúales son los comandos DDL?

El **lenguaje DDL** permite a los programadores llevar a cabo las tareas de definición de las estructuras que almacenarán los datos así como de los procedimientos o funciones que permitan consultarlos.

- **CREATE** Utilizado para crear nuevas tablas, stored procedures e índices.
- **DROP** Empleado para eliminar tablas, stored procedures e índices
- **ALTER** Utilizado para modificar las tablas agregando campos o cambiando la definición de los campos

CREATE DATABASE

MySQL implementa una base de datos como un directorio que contiene todos los archivos que corresponden a las tablas de la base de datos.
Para crear una nueva base de datos en MySQL, use la instrucción CREATE DATABASE con la siguiente *sintaxis:*

```
CREATE DATABASE [IF NOT EXISTS] database_name
[CHARACTER SET charset_name]
[COLLATE collation_name]
```

Primero, se especifica la cláusula CREATE DATABASE y luego database_name. El **nombre de la base de datos debe ser único** dentro de la instancia del servidor MySQL. Si intenta crear una base de datos con un nombre que ya existe, MySQL emite un **error**.
En segundo lugar, **para evitar un error en caso de que cree** accidentalmente **una base de datos que ya existe**, puede especificar la opción IF NOT EXISTS. En este caso, MySQL **no emite un error, sino que termina la declaración CREATE DATABASE.**
En tercer lugar, **se especifica el conjunto de caracteres y la clasificación** para la nueva base de datos en el momento de la creación. **Si se omite** las cláusulas CHARACTER SET y COLLATE, MySQL **usa el conjunto de caracteres y la clasificación predeterminados para la nueva base de datos**.
En MySQL, el esquema es el sinónimo de la base de datos. Crear un nuevo esquema también significa crear una nueva base de datos.

DROP DATABASE

La declaración DROP DATABASE descarta todas las tablas en la base de datos y elimina la base de datos de forma permanente. Por lo tanto, debe tener mucho cuidado al usar esta declaración.

La sintaxis de la declaración DROP DATABASE:

```
DROP DATABASE [IF EXISTS] database_name;
```

En esta declaración, especifica el nombre de la base de datos que desea eliminar.

Si intenta eliminar una base de datos que no existe, MySQL emitirá un error.

Para evitar que se produzca un error si elimina una base de datos que no existe, puede usar la opción IF EXISTS. En este caso, MySQL termina la declaración sin emitir ningún error.

La declaración DROP DATABASE devuelve el número de tablas que se eliminaron.

En MySQL, el esquema es el sinónimo de la base de datos, por lo tanto, puede usarlos indistintamente:

```
DROP SCHEMA [IF EXISTS] database_name;
```

CREATE TABLE

La declaración CREATE TABLE le permite crear una nueva tabla en una base de datos.
La *sintaxis básica* de la declaración CREATE TABLE:

```
CREATE TABLE [IF NOT EXISTS] table_name(
   column_1_definition,
   column_2_definition,
   ...,
   table_constraints
) ENGINE=storage_engine;
```

- El **nombre de la tabla debe ser único dentro de una base de datos**. El IF NOT EXISTS es **opcional**. Le permite verificar si la tabla que crea ya existe en la base de datos. Si este es el caso, MySQL i**gnorará toda la declaración y no creará ninguna tabla nueva**.
- **Especifica una lista de columnas de la tabla** en la sección column_list, las columnas están separadas por comas.
- Puede especificar **opcionalmente** el **motor de almacenamiento para la tabla** en la cláusula ENGINE. Puede usar cualquier motor de almacenamiento como InnoDB y MyISAM. Si no declara explícitamente un motor de almacenamiento, MySQL **usará InnoDB por defecto**.

CREATE TABLE

InnoDB se convirtió en el motor de almacenamiento predeterminado desde MySQL versión 5.5. El motor de almacenamiento InnoDB ofrece muchos beneficios de un sistema de gestión de bases de datos relacionales, como **transacciones ACID, integridad referencial** y **recuperación de fallos**. En las versiones anteriores, MySQL usaba MyISAM como motor de almacenamiento predeterminado.

CREATE TABLE

A continuación se muestra la *sintaxis* para la definición de una columna:

```
column_name data_type(length) [NOT NULL] [DEFAULT value] [AUTO_INCREMENT] column_
constraint;
```

- El column_name especifica el nombre de la columna. Cada columna tiene un tipo de datos específico y un tamaño opcional, por ejemplo,VARCHAR(255)
- El NOT NULL asegura que la columna no contendrá NULL . Además, una columna con NOT NULL, puede tener una restricción adicional como CHECK y UNIQUE .
- El DEFAULT especifica un valor predeterminado para la columna.
- El AUTO_INCREMENT indica que el valor de la columna se incrementa en uno automáticamente siempre que se una nueva fila insertada en la tabla. Cada tabla tiene **un máximo de una columna** AUTO_INCREMENT .

Después de la lista de columnas, puede definir restricciones de tabla como UNIQUE , CHECK , PRIMARY KEY y FOREIGN KEY .

Por ejemplo, si desea **establecer una columna o un grupo de columnas como clave principal**, use la siguiente *sintaxis*:

```
PRIMARY KEY (col1,col2,...)
```

Ejemplos de declaraciones CREATE TABLE

Ejemplo simple de CREATE TABLE

La siguiente instrucción crea una nueva tabla llamada tasks:

```
CREATE TABLE IF NOT EXISTS tasks (
    task_id INT AUTO_INCREMENT PRIMARY KEY,
    title VARCHAR(255) NOT NULL,
    start_date DATE,
    due_date DATE,
    status TINYINT NOT NULL,
    priority TINYINT NOT NULL,
    description TEXT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
)  ENGINE=INNODB;
```

.

Ejemplos de declaraciones CREATE TABLE

La tabla de tareas tiene las siguientes columnas:

- El task_id es una **columna de incremento automático**. Si usa la instrucción INSERT para insertar una nueva fila en la tabla sin especificar un valor para la columna task_id, MySQL generará automáticamente un número entero secuencial para task_id comenzando desde 1. Es la **columna de clave principal de la tabla tasks**. Significa que los valores en la columna task_id identificarán de manera única las filas en la tabla.
- La columna title es una **columna de cadena de caracteres variable cuya longitud máxima es 255**. Esto significa que no puede insertar una cadena cuya longitud es mayor que 255 en esta columna. La restricción **NOT NULL** indica que **la columna no acepta NULL**. En otras palabras, usted tiene que proporcionar un valor no nulo cuando se inserta o actualizar esta columna.
- El start_date y due_date **son columnas DATE**. Debido a que estas columnas no tienen la restricción NOT NULL, **pueden almacenar NULL**. Si no proporciona un valor para la columna start_date cuando inserta una nueva fila, la columna start_date **tomará la fecha actual del servidor de la base de datos**.
- El status y priority son las **columnas TINYINT que no permiten NULL**.
- La columna description es una **columna TEXT que acepta NULL**.
- El created_at es una **columna TIMESTAMP que acepta la hora actual como el valor predeterminado**.

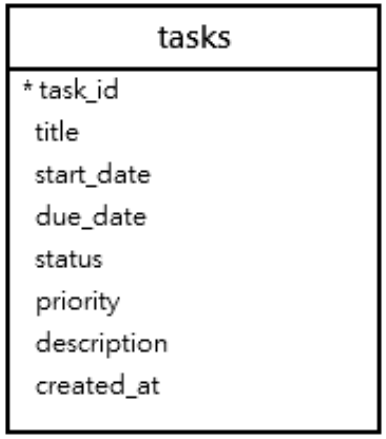
Ejemplos de declaraciones CREATE TABLE

Una vez que ejecute la instrucción CREATE TABLE para crear la tabla tasks, puede ver su estructura utilizando la instrucción DESCRIBE:

```
DESCRIBE tasks;
```

Field	Type	Null	Key	Default	Extra
task_id	int(11)	NO	PRI	NULL	auto_increment
title	varchar(255)	NO		NULL	
start_date	date	YES		NULL	
due_date	date	YES		NULL	
status	tinyint(4)	NO		NULL	
priority	tinyint(4)	NO		NULL	
description	text	YES		NULL	
created_at	timestamp	YES		CURRENT_TIMESTAMP	DEFAULT_GENERATED

El **diagrama** de la base de datos de la tabla tasks:



Ejemplos de declaraciones CREATE TABLE (con clave foránea)

Supongamos que cada tarea tiene una lista de verificación o lista de tareas pendientes. Para almacenar listas de verificación de tareas, puede crear una nueva tabla checklists con el siguiente nombre:

```
CREATE TABLE IF NOT EXISTS checklists (
    todo_id INT AUTO_INCREMENT,
    task_id INT,
    todo VARCHAR(255) NOT NULL,
    is_completed BOOLEAN NOT NULL DEFAULT FALSE,
    PRIMARY KEY (todo_id , task_id),
    FOREIGN KEY (task_id)
        REFERENCES tasks (task_id)
        ON UPDATE RESTRICT ON DELETE CASCADE
);
```

La **tabla checklists** tiene una **clave primaria que consta de dos columnas**. Por lo tanto, usamos una restricción de tabla para definir la clave primaria:

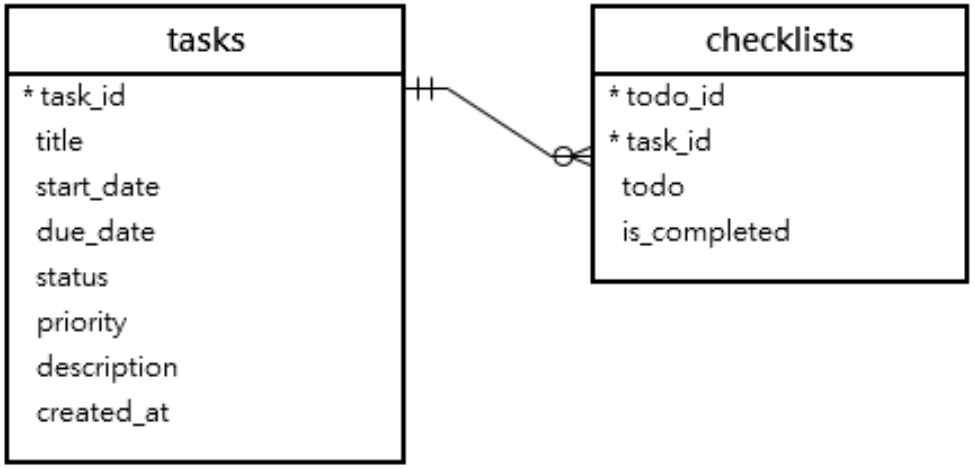
```
PRIMARY KEY (todo_id , task_id)
```

Ejemplos de declaraciones CREATE TABLE (con clave foránea)

Además, task_id es la columna de clave foránea que hace referencia a la columna task_id de la tabla tasks, utilizamos una restricción de clave foránea para establecer esta relación:

```
FOREIGN KEY (task_id)
    REFERENCES tasks (task_id)
    ON UPDATE RESTRICT
    ON DELETE CASCADE
```

Esta imagen ilustra la tabla checklists y su relación con la tabla tasks:



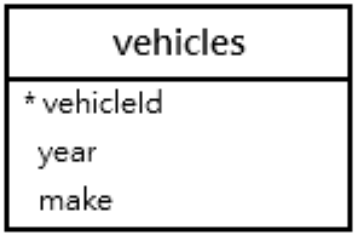
ALTER TABLE

Es una instrucción MySQL para:

- **agregar** una **columna**
- **modificar** una **columna**
- **renombrar** una **columna**
- **borrar** una **columna**
- **renombrar una tabla**.

Creemos una tabla con el nombre **vehicles**

```
CREATE TABLE vehicles (
    vehicleId INT,
    year INT NOT NULL,
    make VARCHAR(100) NOT NULL,
    PRIMARY KEY(vehicleId)
);
```



ALTER TABLE
Añadir una columna

Agregar una columna a una tabla

Para agregar una columna a una tabla, usa la *sintaxis* ALTER TABLE ADD:

```
ALTER TABLE table_name
ADD
    new_column_name column_definition
    [FIRST | AFTER column_name]
```

En esta *sintaxis*:

- table_name: especifica el nombre de la tabla a la que desea agregar una nueva columna o columnas después de las palabras clave ALTER TABLE.
- new_column_name: especifica el nombre de la nueva columna.
- column_definition: especifique el tipo de datos, el tamaño máximo y la restricción de columna de la nueva columna
- FIRST | AFTER column_name especifica la posición de la nueva columna en la tabla. Puede agregar una columna después de una columna existente (AFTER column_name) o como la primera columna (FIRST). Si omite esta cláusula, la columna se agrega al final de la lista de columnas de la tabla.

ALTER TABLE
Añadir una columna (ejemplo)

En el ejemplo usa la instrucción ALTER TABLE ADD para agregar una columna al final de la tabla **vehicles**:

```
ALTER TABLE vehicles
ADD model VARCHAR(100) NOT NULL;
```

Esta declaración muestra la lista de columnas de la tabla vehicles:

```
DESCRIBE vehicles;
```

Field	Type	Null	Key	Default	Extra
vehicleId	int(11)	NO	PRI	NULL	
year	int(11)	NO		NULL	
make	varchar(100)	NO		NULL	
model	varchar(100)	NO		NULL	

Como se muestra claramente en la salida, **la columna model** se ha agregado **al final** de la **tabla vehicles.**

ALTER TABLE
Añadir múltiples columnas

Agregar múltiples columnas a una tabla

Para agregar varias columnas a una tabla, use la siguiente forma de la declaración ALTER TABLE ADD:

```
ALTER TABLE table_name
    ADD new_column_name column_definition
    [FIRST | AFTER column_name],
    ADD new_column_name column_definition
    [FIRST | AFTER column_name],
    ...;
```

Por ejemplo, esta declaración agrega dos **columnas color** y **note** a la **tabla vehicles**:

```
ALTER TABLE vehicles
ADD color VARCHAR(50),
ADD note VARCHAR(255);
```

Esta declaración muestra la nueva estructura de la tabla vehicles:

```
DESCRIBE vehicles;
```

Field	Type	Null	Key	Default	Extra
vehicleId	int(11)	NO	PRI	NULL	
year	int(11)	NO		NULL	
make	varchar(100)	NO		NULL	
model	varchar(100)	NO		NULL	
color	varchar(50)	YES		NULL	
note	varchar(255)	YES		NULL	

ALTER TABLE
Modificar columna

Sintaxis básica para **modificar una columna en una tabla**:

```
ALTER TABLE table_name
MODIFY column_name column_definition
[ FIRST | AFTER column_name];
```

Es una buena práctica ver los atributos de una columna antes de modificarla.
Si se desea **cambiar la columna note a una columna NOT NULL con un máximo de 100 caracteres**.
Primero, muestre la lista de columnas de la vehicles tabla:

```
DESCRIBE vehicles;
```

Field	Type	Null	Key	Default	Extra
vehicleId	int(11)	NO	PRI	NULL	
year	int(11)	NO		NULL	
make	varchar(100)	NO		NULL	
model	varchar(100)	NO		NULL	
color	varchar(50)	YES		NULL	
note	varchar(255)	YES		NULL	

Modificar la **columna note**:

```
ALTER TABLE vehicles
MODIFY note VARCHAR(100) NOT NULL;
```

```
DESCRIBE vehicles;
```

Field	Type	Null	Key	Default	Extra
vehicleId	int(11)	NO	PRI	NULL	
year	int(11)	NO		NULL	
make	varchar(100)	NO		NULL	
model	varchar(100)	NO		NULL	
color	varchar(50)	YES		NULL	
note	varchar(100)	NO		NULL	

ALTER TABLE
Modificar varias columnas

La siguiente declaración le permite **modificar varias columnas**:

```
ALTER TABLE table_name
    MODIFY column_name column_definition
    [ FIRST | AFTER column_name],
    MODIFY column_name column_definition
    [ FIRST | AFTER column_name],
    ...;
```

las columnas actuales de la **tabla vehicles**:

Field	Type	Null	Key	Default	Extra
vehicleId	int(11)	NO	PRI	NULL	
year	int(11)	NO		NULL	
make	varchar(100)	NO		NULL	
model	varchar(100)	NO		NULL	
color	varchar(50)	YES		NULL	
note	varchar(100)	NO		NULL	

Usando la ALTER TABLE MODIFY para modificar varias columnas:

```
ALTER TABLE vehicles
MODIFY year SMALLINT NOT NULL,
MODIFY color VARCHAR(20) NULL AFTER make;
```

En este ejemplo:

- Se modifica el tipo de datos de la **columna year de INT a SMALLINT**
- Se modifica la **columna color estableciendo la longitud máxima en 20, eliminando la restricción NOT NULL** y cambiando su posición para que **aparezca después de la columna make.**

Field	Type	Null	Key	Default	Extra
vehicleId	int(11)	NO	PRI	NULL	
year	smallint(6)	NO		NULL	
make	varchar(100)	NO		NULL	
color	varchar(20)	YES		NULL	
model	varchar(100)	NO		NULL	
note	varchar(100)	NO		NULL	

ALTER TABLE - Cambiar el nombre de una columna en una tabla

Sintaxis:

```
ALTER TABLE table_name
    CHANGE COLUMN original_name new_name column_definition
    [FIRST | AFTER column_name];
```

En esta sintaxis:

- Se especifica el **nombre de la tabla a la que pertenece la columna**.
- ***Se especifica el nombre de la columna y el nuevo nombre seguido de la definición de la columna después de las*** palabras clave CHANGE COLUMN
- Usar la opción FIRST o AFTER column_name para determinar la nueva posición de la columna.

El siguiente ejemplo utiliza la instrucción **ALTER TABLE CHANGE COLUMN** para *cambiar el nombre de la columna note a vehicleCondition*:

```
ALTER TABLE vehicles
CHANGE COLUMN note vehicleCondition VARCHAR(100) NOT NULL;

DESCRIBE vehicles;
```

Field	Type	Null	Key	Default	Extra
vehicleId	int(11)	NO	PRI	NULL	
year	smallint(6)	NO		NULL	
make	varchar(100)	NO		NULL	
color	varchar(20)	YES		NULL	
model	varchar(100)	NO		NULL	
vehicleCondition	varchar(100)	NO		NULL	

ALTER TABLE - Borrar una columna

Para eliminar una columna en una tabla, usa la instrucción ALTER TABLE DROP COLUMN:

```
ALTER TABLE table_name
DROP COLUMN column_name;
```

En esta *sintaxis*:

- Se especifica el **nombre de la tabla** en la que desea **eliminar una columna** después de las palabras clave ALTER TABLE.
- Se especifica el **nombre de la columna** que desea **eliminar** después de las palabras clave DROP COLUMN.

Este ejemplo muestra cómo **eliminar la columna vehicleCondition de la tabla vehicles**:

```
ALTER TABLE vehicles
DROP COLUMN vehicleCondition;
```

ALTER TABLE - Cambiar nombre de tabla

Para cambiar el nombre de una tabla , usa la declaración ALTER TABLE RENAME TO:

```
ALTER TABLE table_name
RENAME TO new_table_name;
```

En esta *sintaxis*:

- Se especifica el nombre de la tabla cuyo nombre desea cambiar después de las palabras clave ALTER TABLE.
- Se especifica el nuevo nombre para la tabla después de las palabras clave RENAME TO.

Este ejemplo **cambia el nombre de la tabla vehicles a cars**:

```
ALTER TABLE vehicles
RENAME TO cars;
```

DROP TABLE

Para eliminar las tablas existentes, usa la DROP TABLE.
Sintaxis

```
DROP [TEMPORARY] TABLE [IF EXISTS] table_name [, table_name] ...
[RESTRICT | CASCADE]
```

- La declaración DROP TABLE **elimina una tabla y sus datos permanentemente de la base de datos**. Se puede **eliminar varias tablas** usando una sola declaración DROP TABLE, **cada tabla está separada por una coma** (,).
- La opción TEMPORARY le permite **eliminar solo tablas temporales**. Asegura que no elimine accidentalmente tablas no temporales.
- La opción IF EXISTS **borra** condicionalmente **una tabla solo si existe**. Si descarta una tabla no existente con la opción IF EXISTS genera una NOTA, que puede recuperarse utilizando la instrucción SHOW WARNINGS.
- La declaración DROP TABLE **no elimina los privilegios de usuario específicos asociados con las tablas**. Por lo tanto, si crea una tabla con el mismo nombre que la descartada, MySQL aplicará los privilegios existentes a la nueva tabla, lo que puede suponer un **riesgo para la seguridad**.
- Las opciones RESTRICT y CASCADE están reservadas para las futuras versiones de MySQL.
- **Para ejecutar** la declaración DROP TABLE, **debe tener privilegios DROP para la tabla que desea eliminar**.

DROP TABLE - Ejemplos

A) Usar DROP TABLE para descartar una tabla.
Cree una tabla insurances:

```
CREATE TABLE insurances (
    id INT AUTO_INCREMENT,
    title VARCHAR(100) NOT NULL,
    effectiveDate DATE NOT NULL,
    duration INT NOT NULL,
    amount DEC(10 , 2 ) NOT NULL,
    PRIMARY KEY(id)
);
```

Eliminar la tabla insurances:

```
DROP TABLE insurances;
```

B) Usar DROP TABLE para eliminar varias tablas:
Cree dos tablas llamadas CarAccessories y CarGadgets

```
CREATE TABLE CarAccessories (
    id INT AUTO_INCREMENT,
    name VARCHAR(100) NOT NULL,
    price DEC(10 , 2 ) NOT NULL,
    PRIMARY KEY(id)
);

CREATE TABLE CarGadgets (
    id INT AUTO_INCREMENT,
    name VARCHAR(100) NOT NULL,
    price DEC(10 , 2 ) NOT NULL,
    PRIMARY KEY(id)
);
```

Eliminar las tablas CarAccessories y CarGadgets:

```
DROP TABLE CarAccessories, CarGadgets;
```

DROP TABLE - Ejemplos

C) Usar DROP TABLE para **descartar una tabla no existente**
Esta declaración intenta descartar una tabla no existente:

```
DROP TABLE aliens;
```

MySQL emitió el siguiente error:

```
Error Code: 1051. Unknown table 'classicmodels.aliens'
```

Sin embargo, si usa la opción IF EXISTS en la declaración DROP TABLE:

```
DROP TABLE IF EXISTS aliens;
```

MySQL emitió una advertencia en su lugar:

```
0 row(s) affected, 1 warning(s): 1051 Unknown table 'classicmodels.aliens'
```

Para mostrar la advertencia, puede usar la declaración SHOW WARNINGS:

```
SHOW WARNINGS;
```

Level	Code	Message
Error	1051	Unknown table 'classicmodels.aliens'

DROP TABLE basado en un patrón

Suponga que tiene muchas tablas cuyos nombres comienzan test en su base de datos y desea eliminarlas todas con una sola instrucción DROP TABLE.
Se puede utilizar el siguiente script que cumple con eliminar múltiples tablas que comienzan con un patrón determinado.

```
-- set table schema and pattern matching for tables
SET @schema = 'classicmodels';
SET @pattern = 'test%';

-- build dynamic sql (DROP TABLE tbl1, tbl2...;)
SELECT CONCAT('DROP TABLE ',GROUP_CONCAT(CONCAT(@schema,'.',table_name)),';')
INTO @droplike
FROM information_schema.tables
WHERE @schema = database()
AND table_name LIKE @pattern;

-- display the dynamic sql statement
SELECT @droplike;

-- execute dynamic sql
PREPARE stmt FROM @droplike;
EXECUTE stmt;
DEALLOCATE PREPARE stmt;
```

1. Se declaran dos variables que acepten el esquema de la base de datos y un patrón que desee que coincidan con las tablas.
2. Construya una declaración DROP TABLE dinámica.
3. La consulta indica a MySQL que vaya a la information_schema tabla, que contiene información sobre todas las tablas de todas las bases de datos, y que concatene todas las tablas de la base de datos @schema(classicmodels) que coincidan con el patrón @pattern (test%) con el prefijo DROP TABLE . La función GROUP_CONCAT crea una lista de tablas separadas por comas.
4. Muestra el SQL dinámico para verificar si funciona correctamente.
5. Ejecuta la instrucción usando una instrucción preparada.

ÍNDICES

Un **índice** es una estructura de datos como B-Tree que **mejora la velocidad de recuperación** de datos en una tabla **a costa de escrituras y almacenamiento adicionales para mantenerla**.

El **optimizador** de consultas puede **usar índices para localizar datos rápidamente sin tener que escanear cada fila** de una tabla para una consulta determinada.

Cuando **crea una tabla con una clave primaria o clave única** , MySQL **crea** automáticamente **un índice especial** llamado PRIMARY. Este índice **se llama** índice agrupado. Una vez que se crea un índice agrupado, **todas las filas de la tabla se almacenarán de acuerdo con las columnas clave utilizadas para crear el índice agrupado**. **Debido a qu**e un índice agrupado **almacena las filas en orden**, **cada tabla tiene solo un índice agrupado**.

El **índice PRIMARY** es especial porque el índice en sí **se almacena junto con los datos en la misma tabla**. El índice agrupado **impone el orden de las filas en la tabla**.

Si no tiene una clave principal para una tabla, MySQL buscará el **primer índice UNIQUE donde se encuentran todas las columnas de clave NOT NULL** y utilizará este índice UNIQUE como índice agrupado.

Otros índices distintos del índice PRIMARY se denominan índices secundarios o índices no agrupados.

CREATE INDEX

Normalmente, **crea índices para una tabla en el momento de la creación**. Por ejemplo, la siguiente instrucción crea una nueva tabla con un índice que consta de dos columnas c2 y c3.

```
CREATE TABLE t(
   c1 INT PRIMARY KEY,
   c2 INT NOT NULL,
   c3 INT NOT NULL,
   c4 VARCHAR(10),
   INDEX (c2,c3)
);
```

Para agregar un índice para una columna o un conjunto de columnas, utilizar la instrucción CREATE INDEX de la siguiente manera:

```
CREATE INDEX index_name ON table_name (column_list)
```

Por ejemplo, para agregar un nuevo índice para la columna c4, use la siguiente instrucción:

```
CREATE INDEX idx_c4 ON t(c4);
```

ÍNDICES - Ejemplo

La siguiente declaración busca **empleados** cuyo **cargo** es **Sales Rep**

```
SELECT
    employeeNumber,
    lastName,
    firstName
FROM
    employees
WHERE
    jobTitle = 'Sales Rep';
```

employeeNumber	lastName	firstName
1165	Jennings	Leslie
1166	Thompson	Leslie
1188	Firrelli	Julie
1216	Patterson	Steve
1286	Tseng	Foon Yue
1323	Vanauf	George
1337	Bondur	Loui
1370	Hernandez	Gerard
1401	Castillo	Pamela
1501	Bott	Larry

Tenemos 17 filas que indican que 17 empleados cuyo título de trabajo es el representante de ventas.

Para ver cómo MySQL realizó internamente esta consulta, agregue la cláusula EXPLAIN **al comienzo de la declaración SELECT de la siguiente manera**:

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	employees	NULL	ALL	NULL	NULL	NULL	NULL	23	10.00	Using where

Como puede ver, MySQL tuvo que escanear toda la tabla que consta de 23 filas para encontrar a los empleados con el título **Sales Rep** del trabajo.

ÍNDICES – Agregando índices al ejemplo

Creemos un índice para la columna **jobTitle** usando la instrucción CREATE INDEX:

```
CREATE INDEX jobTitle ON employees(jobTitle);
```

Y ejecute la declaración anterior nuevamente:

```
EXPLAIN SELECT
    employeeNumber,
    lastName,
    firstName
FROM
    employees
WHERE
    jobTitle = 'Sales Rep';
```

El resultado es:

	id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
▶	1	SIMPLE	employees	NULL	ref	jobTitle	jobTitle	52	const	17	100.00	NULL

MySQL solo tuvo que **ubicar 17 filas del índice jobTitle** como se indica en la columna clave sin escanear toda la tabla.

Para mostrar los índices de una tabla, use la SHOW INDEXES

```
SHOW INDEXES FROM employees;
```

	Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed	Null	Index_type	Comment	Index_commer	Visible
▶	employees	0	PRIMARY	1	employeeNumber	A	23	NULL	NULL		BTREE			YES
	employees	1	reportsTo	1	reportsTo	A	7	NULL	NULL	YES	BTREE			YES
	employees	1	officeCode	1	officeCode	A	7	NULL	NULL		BTREE			YES
	employees	1	jobtitle	1	jobTitle	A	7	NULL	NULL		BTREE			YES

DROP INDEX

Para eliminar un índice existente de una tabla, use la instrucción DROP INDEX:

```
DROP INDEX index_name ON table_name
[algorithm_option | lock_option];
```

En esta sintaxis:

1. Especificar el nombre del índice que desea eliminar después de las palabras clave DROP INDEX.
2. Especificar el nombre de la tabla a la que pertenece el índice.

El algorithm_option le permite especificar un algoritmo específico utilizado para la eliminación del índice. Sintaxis: `ALGORITHM [=] {DEFAULT|INPLACE|COPY}`

Para la eliminación del índice, se admiten los siguientes algoritmos:

- COPY: La tabla se copia en la nueva tabla fila por fila, luego se realiza en la copia de la tabla original. Las declaraciones de manipulación de datos concurrentes como INSERT y UPDATE no están permitidas.
- INPLACE: La tabla se reconstruye en su lugar en lugar de copiarse a la nueva. MySQL emite un bloqueo exclusivo de metadatos en la tabla durante las fases de preparación y ejecución de la operación de eliminación de índice. Este algoritmo permite declaraciones de manipulación de datos concurrentes.
  Tenga en cuenta que la cláusula ALGORITHM es opcional. Si lo omite, MySQL usa INPLACE. En caso de INPLACE que no sea compatible, MySQL usa COPY.
  Usar DEFAULT tiene el mismo efecto que omitir la cláusula ALGORITHM.

DROP INDEX

lock_option establece el nivel de concurrente, leer y escribir en la tabla, mientras se está quitando el índice.

Sintaxis:

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

Se admiten los siguientes **modos de bloqueo**:

- DEFAULT: esto le permite tener el nivel máximo de concurrencia para un algoritmo dado. Primero, **permite lecturas y escrituras concurrentes** si es compatible. **Sino**, permita **lecturas concurrentes** si es compatible. **Sino**, imponga **acceso exclusivo**.
- NONE: es compatible, **puede tener lecturas y escrituras concurrentes**. De lo contrario, MySQL emite un error.
- SHARED: es compatible, **puede tener lecturas concurrentes, pero no escrituras**. MySQL emite un error si las lecturas concurrentes no son compatibles.
- EXCLUSIVE: esto exige **acceso exclusivo**.

DROP INDEX - Ejemplo

Creemos una nueva tabla:

```
CREATE TABLE leads(
    lead_id INT AUTO_INCREMENT,
    first_name VARCHAR(100) NOT NULL,
    last_name VARCHAR(100) NOT NULL,
    email VARCHAR(255) NOT NULL,
    information_source VARCHAR(255),
    INDEX name(first_name,last_name),
    UNIQUE email(email),
    PRIMARY KEY(lead_id)
);
```

La siguiente declaración **elimina el índice name** de la tabla leads:

```
DROP INDEX name ON leads;
```

La siguiente declaración **elimina el índice email** de la tabla leads **con un algoritmo y bloqueo específicos**:

```
DROP INDEX email ON leads
ALGORITHM = INPLACE
LOCK = DEFAULT;
```

DROP PRIMARY KEY

Para **descartar la clave primaria cuyo nombre de índice es** PRIMARY, utilice la siguiente instrucción:

```
DROP INDEX `PRIMARY` ON table_name;
```

Ejemplo

1. La siguiente instrucción **crea una nueva tabla** denominada t con una clave primaria:

```
CREATE TABLE t(
    pk INT PRIMARY KEY,
    c VARCHAR(10)
);
```

2. Para **descartar el índice de clave principal**, utiliza la siguiente instrucción:

```
DROP INDEX `PRIMARY` ON t;
```