AWAKELAB

Desarrollador de aplicaciones Full Stack

Python Trainee

Objetivo de la jornada

4.1 Reconoce la sintaxis básica para la creación de expresiones DDL que resuelven un requerimiento de definición de datos
4.2 Construye sentencias de creación de una tabla utilizando DDL y definiendo campos, tipos de dato, nulidad, llaves primarias y foráneas de acuerdo a un modelo de datos existente para satisfacer un requerimiento
4.3 Construye sentencias utilizando DDL para la modificación de los atributos de una tabla de acuerdo a los requerimientos planteados

3.5.- Sentencias para la definición de tablas

3.5.1.- El lenguaje de definición de datos DDL

El lenguaje de definición de datos (en inglés Data Definition Language, o DDL), es el que se encarga de la modificación de la estructura de los objetos de la base de datos. Incluye órdenes para modificar, borrar o definir las tablas en las que se almacenan las bases de datos. Existen cuatro operaciones básicas: CREATE, ALTER, DROP y TRUNCATE.

Los elementos, llamados objetos, de la base de datos (tablas, vistas, columnas, índices, etc.) se almacenan en el diccionario de datos. Por otro lado, muchos Sistemas Gestores de Bases de Datos aportan elementos para organizar estos objetos (como catálogos y esquemas).

Los objetos son manipulados y creados por los usuarios. En principio solo los administradores y los usuarios propietarios pueden acceder a cada objeto, salvo que se modifiquen los privilegios del objeto para permitir el acceso a otros usuarios.

Se debe tener en cuenta que ninguna instrucción DDL puede ser anulada por una instrucción ROLLBACK por lo que hay que tener mucha precaución a la hora de utilizarlas. Es decir, las instrucciones DDL generan acciones que no se pueden deshacer. Salvo que se disponga de alguna copia de seguridad o de otros elementos de recuperación.

Según los estándares actuales, una base de datos es un conjunto de objetos pensados para gestionar datos. Estos objetos están contenidos en esquemas, los esquemas suelen estar asociados al perfil de un usuario en particular.
En SQL estándar, existe el concepto de catálogo, que sirve para almacenar esquemas, y estos sirven para almacenar objetos. Así el nombre completo de un objeto vendría dado por:

```
catálogo.esquema.objeto
```

Es decir, los objetos pertenecen a esquemas, y estos últimos a catálogos.

En casi todos los sistemas de bases de datos hay un catálogo por defecto, de modo que si no se indica un catálogo alguno al crear objetos, estos se almacenan allí. Del mismo modo, hay esquemas por defecto.

MySQL no posee catálogos; sin embargo sí existen esquemas. Los esquemas de MySQL están relacionados con los usuarios: cada usuario puede tener diferentes accesos a cada esquema existente en el servidor.

3.5.2.- Creación de una tabla

Los nombres de las tablas deben cumplir las siguientes reglas (se comentan las reglas de MySQL, en otros SGBD podrían cambiar):

- Deben comenzar con una letra.
- No deben tener más de 30 caracteres.
- Solo se permiten utilizar letras del alfabeto (inglés), números o el signo de subrayado (también los signos $ y #, pero esos se utilizan de manera especial, por lo que no son recomendados).
- No puede haber dos tablas con el mismo nombre dentro del mismo esquema (pueden coincidir los nombres si están en distintos esquemas).
- No puede coincidir con el nombre de una palabra reservada SQL (por ejemplo no se puede llamar SELECT a una tabla).
- En el caso de que el nombre tenga espacios en blanco o caracteres nacionales (permitido solo en algunas bases de datos), entonces se suele entrecomillar con comillas dobles. En el estándar SQL se pueden utilizar comillas dobles al poner el nombre de la tabla a fin de hacerla sensible a las mayúsculas (se diferenciará entre “FACTURAS” y “Facturas”).

Los conceptos anteriores son también las propiedades que debe cumplir cualquier nombre de objeto en una base de datos (nombres de vistas, columnas, restricciones, etc.).

3.5.3.- Definición de campos

El comando CREATE es la orden SQL que permite crear una tabla. Por defecto será almacenada en el espacio y esquema del usuario que crea la tabla.

La sintaxis del comando es la siguiente:

```
CREATE TABLE [esquema.] nombreDeTabla ( nombreDeLaColumna1 tipoDeDatos [DEFAULT valor] [restricciones] [, ...] );
```

Ejemplo 1: Creación de la tabla "escuela", con solo un campo de tipo texto

```
CREATE TABLE escuela (nombre VARCHAR(50));
```

Solo se podrá crear la tabla si el usuario posee los permisos necesarios para ello. Si la tabla pertenece a otro esquema (suponiendo que el usuario tenga permiso para grabar tablas en ese otro esquema), se antepone al nombre de la tabla.

Ejemplo 2: Creación de la tabla "escuela" en un esquema específico

```
CREATE TABLE otroEsquema.escuela (nombre VARCHAR(50));
```

Se puede indicar un valor por defecto para el atributo mediante la cláusula DEFAULT.

Ejemplo 3: Creación de la tabla proveedores con un campo por defecto

```
CREATE TABLE escuela (nombre VARCHAR(50), localidad VARCHAR(30) DEFAULT 'Copiapó' );
```

De este modo si se añade una escuela y no se indica la localidad, se tomará Copiapó como localidad de dicha escuela.

Se puede utilizar DEFAULT y usar funciones del sistema.

Ejemplo 4: Creación de la tabla escuela con un campo de tipo DATE con la fecha actual como valor por defecto

```
CREATE TABLE escuela (
   nombre VARCHAR(50),
   localidad VARCHAR(30) DEFAULT 'Copiapó',
   fecha_creacion DATETIME DEFAULT CURRENT_TIMESTAMP()
);
```

3.5.4.- Tipos de dato

Después de la fase de diseño de una base de datos, y una vez se ha realizado el paso a tablas del mismo, es necesario crear las tablas correspondientes dentro de la base de datos. Para cada campo de cada una de las tablas, es necesario determinar el tipo de datos que contiene, para de esa forma ajustar el diseño de la base de datos, y conseguir un almacenamiento óptimo con la menor utilización de espacio.

Los tipos de datos que puede haber en un campo, se pueden agrupar en tres grandes grupos:

- Tipos numéricos
- Tipos de Fecha
- Tipos de Cadena

Tipos numéricos

Existen tipos de datos numéricos, que se pueden dividir en dos grandes grupos, los que están en coma flotante (con decimales) y los que no.

- **TinyInt:** Es un número entero con o sin signo. Con signo el rango de valores válidos va desde -128 a 127. Sin signo, el rango de valores es de 0 a 255.
- **Bit ó Bool:** Un número entero que puede ser 0 ó 1.
- **SmallInt:** Número entero con o sin signo. Con signo el rango de valores va desde -32768 a 32767. Sin signo, el rango de valores es de 0 a 65535.
- **MediumInt:** Número entero con o sin signo. Con signo el rango de valores va desde -8.388.608 a 8.388.607. Sin signo el rango va desde 0 a16777215.
- **Integer, Int:** Número entero con o sin signo. Con signo el rango de valores va desde -2147483648 a 2147483647. Sin signo el rango va desde 0 a 429.4967.295.
- **BigInt:** Número entero con o sin signo. Con signo el rango de valores va desde -9.223.372.036.854.775.808 a 9.223.372.036.854.775.807. Sin signo el rango va desde 0 a 18.446.744.073.709.551.615.
- **Float:** Número pequeño en coma flotante de precisión simple. Los valores válidos van desde -3.402823466E+38 a -1.175494351E-38, 0 y desde 1.175494351E-38 a 3.402823466E+38.
- **xReal, Double:** Número en coma flotante de precisión doble. Los valores permitidos van desde -1.7976931348623157E+308 a -2.2250738585072014E-308, 0 y desde 2.2250738585072014E-308 a 1.7976931348623157E+308.
- **Decimal, Dec, Numeric:** Número en coma flotante desempaquetado. El número se almacena como una cadena.

Tipo de campo	Tamaño de almacenamiento
TINYINT	1 byte
SMALLINT	2 bytes
MEDIUMINT	3 bytes
INT	4 bytes
INTEGER	4 bytes
BIGINT	8 bytes
FLOAT(X)	4 u 8 bytes
FLOAT	4 bytes
DOUBLE	8 bytes

DOUBLE PRECISION	8 bytes
REAL	8 bytes
DECIMAL(M,D	M+2 bytes sí D > 0, M+1 bytes sí D = 0
NUMERIC(M,D)	M+2 bytes if D > 0, M+1 bytes if D = 0

Tipos fecha

A la hora de almacenar fechas, hay que tener en cuenta que Mysql no comprueba de una manera estricta si una fecha es válida o no. Simplemente comprueba que el mes está comprendido entre 0 y 12 y que el día está comprendido entre 0 y 31.

- **Date:** Tipo fecha, almacena una fecha. El rango de valores va desde el 1 de enero de 1001 al 31 de diciembre de 9999. El formato de almacenamiento es de año-mes-dia.
- **DateTime:** Combinación de fecha y hora. El rango de valores va desde el 1 de enero de 1001 a las 0 horas, 0 minutos y 0 segundos al 31 de diciembre de 9999 a las 23 horas, 59 minutos y 59 segundos. El formato de almacenamiento es de año-mes-dia horas:minutos:segundos.
- **TimeStamp:** Combinación de fecha y hora. El rango va desde el 1 de enero de 1970 al año 2037. El formato de almacenamiento depende del tamaño del campo:
    - 14 bytes: AñoMesDiaHoraMinutoSegundo aaaammddhhmmss
    - 12 bytes: AñoMesDiaHoraMinutoSegundo aammddhhmmss
    - 8 bytes: AñoMesDia aaaammdd
    - 6 bytes: AñoMesDia aammdd
    - 4 bytes: AñoMes aamm
    - 2 bytes: Año aa
- **Time:** Almacena una hora. El rango de horas va desde -838 horas, 59 minutos y 59 segundos a 838, 59 minutos y 59 segundos. El formato de almacenamiento es de 'HH:MM:SS'.
- **Year:** Almacena un año. El rango de valores permitidos va desde el año 1901 al año 2155. El campo puede tener tamaño dos o tamaño 4 dependiendo de si queremos almacenar el año con dos o cuatro dígitos.

Tipo de campo	Tamaño de almacenamiento
DATE	3 bytes
DATETIME	8 bytes
TIMESTAMP	4 bytes
TIME	3 bytes
YEAR	1 byte

Tipos de cadena

- **Char(n):** Almacena una cadena de longitud fija. La cadena podrá contener desde 0 a 255 caracteres.
- **VarChar(n):** Almacena una cadena de longitud variable. La cadena podrá contener desde 0 a 255 caracteres. Dentro de los tipos de cadena se pueden distinguir otros dos subtipos, los tipo Test y los tipo BLOB (Binary large Object). La diferencia entre un tipo y otro es el tratamiento que reciben a la hora de realizar ordenamientos y comparaciones. Mientras que el tipo test se ordena sin tener en cuenta las Mayúsculas y las minúsculas, el tipo BLOB se ordena teniéndolas en cuenta. Los tipos BLOB se utilizan para almacenar datos binarios como pueden ser ficheros.
- **TinyText y TinyBlob:** Columna con una longitud máxima de 255 caracteres.
- **Blob y Text:** Un texto con un máximo de 65535 caracteres.
- **MediumBlob y MediumText:** Un texto con un máximo de 16.777.215 caracteres.
- **LongBlob y LongText:** Un texto con un máximo de caracteres 4.294.967.295. Hay que tener en cuenta que debido a los protocolos de comunicación los paquetes pueden tener un máximo de 16 Mb.
- **Enum:** Campo que puede tener un único valor de una lista que se especifica. El tipo Enum acepta hasta 65535 valores distintos.
- **Set:** Un campo que puede contener ninguno, uno o varios valores de una lista. La lista puede tener un máximo de 64 valores.

Tipo de campo	Tamaño de almacenamiento
CHAR(n)	n bytes
VARCHAR(n)	n +1 bytes
TINYBLOB, TINYTEXT	Longitud+1 bytes
BLOB, TEXT	Longitud +2 bytes
MEDIUMBLOB, MEDIUMTEXT	Longitud +3 bytes
LONGBLOB, LONGTEXT	Longitud +4 bytes
ENUM('value1','value2',...)	1 ó dos bytes dependiendo del número de valores.
SET('value1','value2',...)	1, 2, 3, 4 ó 8 bytes, dependiendo del número de valores.

Diferencia de almacenamiento entre los tipos Char y VarChar:

Valor	CHAR(4)	Almacenamiento	VARCHAR(4)	Almacenamiento
''	''	4 bytes	''	1 byte
'ab'	'ab'	4 bytes	'ab'	3 bytes
'abcd'	'abcd'	4 bytes	'abcd'	5 bytes
'abcdefgh'	'abcd'	4 bytes	'abcd'	5 bytes

3.5.5.- La restricción de nulidad

La restricción NOT NULL permite prohibir los nulos en una determinada tabla. Eso obliga a que la columna tenga que tener obligatoriamente un valor para que sea almacenado el registro.

Se puede colocar durante la creación (o modificación) del campo, añadiendo la palabra NOT NULL tras el tipo.

Ejemplo 5: Creación de una tabla con un campo que no permite valores nulos en su ingreso.

```
CREATE TABLE escuela (
   nombre VARCHAR(50) NOT NULL,
   localidad VARCHAR(30) DEFAULT 'Copiapó',
   fecha_creacion DATETIME DEFAULT CURRENT_TIMESTAMP()
);
```

La restricción NOT NULL es la única que solo se puede poner seguida al nombre de la columna a la que se aplica. La razón es que NOT NULL sólo se puede aplicar a una columna a la vez.

3.5.6.- Definición de la llave primaria

La clave primaria de una tabla la forman las columnas que indican cada registro de la misma. La clave primaria hace que los campos que la forman no puedan quedar vacíos ni que se puedan repetir valores.

Además pasan a formar parte del índice principal de la tabla, que se usa para acceder más rápidamente a estos datos, que sean NOT NULL (sin posibilidad de quedar vacíos) y que los valores de los campos sean de tipo UNIQUE (sin posibilidad de repetición).

Si la clave está formada por un solo campo basta seguir una sintaxis similar a la que se indica a continuación.

Ejemplo 6: Creación de una tabla con una llave primaria compuesta por un solo campo

```
CREATE TABLE escuela (
   idescuela INT PRIMARY KEY,
   nombre VARCHAR(50) NOT NULL,
   localidad VARCHAR(30) DEFAULT 'Copiapó',
   fecha_creacion DATETIME DEFAULT CURRENT_TIMESTAMP()
);
```

Ejemplo 7: Creación de una tabla con una llave primaria compuesta por más de un campo

```
CREATE TABLE escuela (
   idescuela INT NOT NULL,
   nombre VARCHAR(50) NOT NULL,
   localidad VARCHAR(30) DEFAULT 'Copiapó',
   fecha_creacion DATETIME DEFAULT CURRENT_TIMESTAMP(),
   CONSTRAINT escuela_pk PRIMARY KEY (idescuela, localidad)
);
```

3.5.7.- Definición de llaves foráneas

Una clave secundaria o foránea se usa para indicar que uno o más campos de una tabla están relacionados con la clave principal (o incluso con una clave candidata) de otra tabla y, por lo tanto, no podrán contener valores que no estén relacionados en la otra tabla.

Ejemplo 8: Creación de una tabla con una llave foránea

```
CREATE TABLE profesor (
   profesor_id INT,
   nombre VARCHAR(25),
   apellido VARCHAR(50),
   idescuela INT,
   fecha_de_contratacion DATE,
   sueldo INT,
   CONSTRAINT profesores_pk PRIMARY KEY (profesor_id),
```

```
   CONSTRAINT profesores_escuela_FK FOREIGN KEY (idescuela)
REFERENCES escuela(idescuela)
);
```

Significa esta instrucción (en cuanto a claves foráneas) que el campo idescuela se relaciona con la columna idescuela de la tabla escuela.

Si el campo al que se hace referencia es la clave principal, se puede obviar el nombre del campo.

Ejemplo 9: Creación de una tabla con una llave foránea, apuntando a claves primarias

```
CREATE TABLE profesor_asignatura (
   profesor_id INT NOT NULL,
   asignatura_id INT NOT NULL,
   PRIMARY KEY (profesor_id, asignatura_id),
   CONSTRAINT profesorasignatura_profesor_fk FOREIGN KEY
(profesor_id) REFERENCES profesor(profesor_id),
   CONSTRAINT profesorasignatura_asignatura_fk FOREIGN KEY
(asignatura_id) REFERENCES asignatura(asignatura_id)
);
```

En este caso se entiende que los campos hacen referencia a las claves principales de las tablas. Si la relación está formada por más de una columna, el orden de los campos debe de ser el mismo; sin embargo, en este caso, es mejor indicar explícitamente el nombre.

De hecho, cuando una relación la forman más de una columna, se debe indicar (como siempre ocurre en las restricciones de más de una columna) tras la lista de columnas de la tabla. Aunque cualquier restricción (sea de una sola columna o no), se puede indicar también al final.

Si la definición de clave secundaria se pone al final, hace falta colocar el texto FOREIGN KEY para indicar en qué campos se coloca la restricción de clave foránea. En el ejemplo anterior es absolutamente necesario (al no indicar explícitamente la lista de columnas en el apartado REFERENCES) que la clave

principal de la tabla sea a la que hace referencia la clave que forman las columnas tipo y modelo, y que además deben estar en ese orden.

Las restricciones de tipo FOREIGN KEY provocan una restricción de integridad referencial, en la que no se pueden indicar datos en las claves secundarias que no existan en las claves principales relacionadas.

Lo malo es que la integridad referencial provoca varios problemas, debidos a sus efectos secundarios.

Por ejemplo, se considerará que se relacionan los profesores con una tabla de escuelas con el id de la escuela respectiva. El id es la clave de la tabla escuelas; ahora bien, no se puede borrar una escuela de la tabla que tenga profesores asociados. Tampoco se puede modificar el id de la escuela por la misma razón.

Ante esto, se dispone de la posibilidad de aplicar políticas especiales. Estas políticas son palabras claves que se colocan tras la cláusula REFERENCES al añadir una restricción de tipo FOREIGN KEY.

Así las políticas que dictan qué hacer cuando se borran datos principales relacionados con claves secundarias son:

- ON DELETE SET NULL: Coloca nulos en todas las claves secundarias relacionadas.
- ON DELETE CASCADE: Borra todas las filas relacionadas con aquella que se ha eliminado.
- ON DELETE SET DEFAULT. Coloca en las filas relacionadas el valor por defecto de esa columna en la columna relacionada.
- ON DELETE RESTRICT. No hace nada.

Las mismas se pueden aplicar en el caso de modificar claves principales. Así tendremos ON UPDATE RESTRICT, ON UPDATE CASCADE, ON UPDATE SET NULL y ON UPDATE SET DEFAULT.

Ejemplo 10: Ejemplo de establecimiento de borrado en cascada y de restricción de acción

```
CREATE TABLE profesor_asignatura (
  profesor_id INT NOT NULL,
  asignatura_id INT NOT NULL,
```

```
    PRIMARY KEY (profesor_id, asignatura_id),
    CONSTRAINT profesorasignatura_profesor_fk FOREIGN KEY
(profesor_id) REFERENCES profesor(profesor_id) ON DELETE RESTRICT,
    CONSTRAINT profesorasignatura_asignatura_fk FOREIGN KEY
(asignatura_id) REFERENCES asignatura(asignatura_id) ON DELETE
CASCADE
);
```

3.5.8.- Creando un modelo de datos con integridad referencial

La integridad referencial es una limitación que se aplica a una base de datos relacional, en la que los datos y las relaciones entre ellos están organizados en tablas de filas y columnas, para que no se introduzcan datos inconsistentes.

La mayoría de los sistemas gestores de bases de datos relacionales definen reglas de integridad referencial que los programadores aplican cuando crean las relaciones entre dos tablas.

Regla de integridad referencial

Básicamente, la integridad referencial dice que una base de datos no puede tener valores de claves externas sin pareja. Tal como se indicó antes, una clave externa o foránea es una columna en una tabla de base de datos que tiene valores que están en la columna de clave primaria, un identificador único que identifica una fila en una tabla, en otra tabla.

Por ejemplo, considera la tabla llamada "escuela", en la que hay una columna llamada "idescuela" como clave primaria. Se relaciona con otra tabla llamada "profesor", donde "profesor_id" es una clave externa. Un profesor no puede pertenecer a una determinada escuela si el "identificador de escuela" correspondiente no existe ya en la tabla "escuela". Si el programa que añade los profesores fuerza la integridad referencial, cualquier intento por insertar un registro a una escuela desconocida no podrá realizarse.

Ventajas

Además de asegurar que estas referencias entre los datos están intactas y son válidas, definir la integridad referencial de una base de datos tiene muchas ventajas.

La integridad referencial usa el código existente en un motor de base de datos en lugar de pedir a los programadores que escriban código de programa personalizado desde cero.

Como resultado, el desarrollo de programas es más rápido, menos propenso a errores y consistente entre varios programas de aplicación que acceden a la base de datos.

Consecuencias

Desafortunadamente, los lenguajes de programación suelen no tener los mecanismos para aplicar la integridad referencial e, incluso cuando un sistema de gestión de base de datos soporta esos mecanismos, los programadores fracasan al usarlos.

La consecuencia de ignorar la integridad referencial es que el código de programación que tiene defectos, o errores, funciona mal y es difícil extenderlo.

Aplicación

Los programadores pueden aplicar la integridad referencial, y evitar los registros "huérfanos" en una base de datos, habilitándola en una relación entre dos tablas.
En el caso de MySQL, aplicar la integridad referencial hace que cualquier operación que la contravenga sea rechazada. Este tipo de operaciones incluyen las actualizaciones de la base de datos que cambian el objeto de una referencia, o las acciones que eliminan el objeto de una referencia. Además, como se mencionaba en clases anteriores, MySQL también tiene un conjunto de opciones, conocidas como opciones en "cascada".

Las opciones en cascada permiten que actualizaciones referenciales y acciones de eliminación sean propagados por la base de datos, de forma que todas las filas relacionadas cambien de la misma forma.

3.5.9.- Modificar un campo de una tabla

MySQL permite cambiar el tipo de datos y propiedades de una determinada columna. La sintaxis para generar este comportamiento sería la siguiente:

```
ALTER TABLE nombreTabla MODIFY (columna tipo [propiedades] [columnaSiguiente tipo [propiedades] ...]
```

Los cambios que se permiten en las columnas de una base de datos MySQL son:

- Incrementar precisión o largo de los tipos de datos
- Solo se puede reducir la anchura de una columna si ésta posee nulos en todos los registros, o bien si todos los valores existentes tienen un tamaño menor o igual a la nueva anchura.
- Se puede pasar de CHAR a VARCHAR y viceversa (si no se modifica la anchura)
- Se puede pasar de DATE a TIMESTAMP y viceversa
- Cualquier otro cambio solo es posible si la tabla está vacía
- Si, a través de este comando, se modifica el valor de la propiedad DEFAULT de una tabla; dicho cambio sólo tendrá efecto en la inserción de nuevas filas.

Ejemplo 11: Modificar el campo de una tabla, usando el comando MODIFY

```
ALTER TABLE escuela MODIFY COLUMN localidad VARCHAR(50) DEFAULT 'Talca';
```

A fin de cambiar el nombre de una columna, se puede utilizar un comando similar al siguiente:

```
ALTER TABLE nombreTabla RENAME COLUMN nombreAntiguo TO nombreNuevo
```

Ejemplo 12: Modificar el nombre del campo de una tabla

```
ALTER TABLE escuela RENAME COLUMN localidad TO nombrelocalidad;
```

A cada columna se le puede asignar un valor por defecto durante su creación mediante la propiedad DEFAULT. Se puede poner esta propiedad durante la creación o modificación de la tabla, añadiendo la palabra DEFAULT tras el tipo de datos del campo y colocando detrás el valor que se desea por defecto.

La palabra DEFAULT se puede añadir durante la creación o la modificación de la tabla (comando ALTER TABLE).

El valor indicado con DEFAULT se aplica cuando se añaden filas a una tabla dejando el valor de la columna vacío en lugar de NULL, a la columna se le asignará el valor por defecto indicado.

3.5.10.- Modificar una condición de nulidad

Al crear una tabla se establece si un determinado campo impedirá el registro de valores nulos. Esto se realiza indicando la frase NOT NULL al final de cada campo al que se desee asignar esta propiedad.

En algunas situaciones se desea modificar esta característica, lo cual se puede lograr a través del mismo comando ALTER indicado en los puntos anteriores. Para tal efecto, se debe añadir a dicho comando la instrucción MODIFY, y agregar o quitar la opción NOT NULL, dependiendo del efecto que se desee lograr.

Ejemplo 13: Ejemplo de modificación de la condición de nulidad de un campo

```
ALTER TABLE escuela MODIFY COLUMN nombrelocalidad VARCHAR(50) NOT NULL;
```

Para que el ejemplo anterior sea efectivo, la columna “nombrelocalidad” no debe tener valores nulos. De ser así MySQL retornará un error, y no se

continuará la operación. Inevitablemente se deberán actualizar todos los campos de la fila que no cumplan dicha condición.

3.5.11.- Eliminación de una tabla

La orden DROP TABLE seguida del nombre de una tabla, permite eliminar la tabla en cuestión.

Al borrar una tabla:

- Desaparecen todos los datos.
- Cualquier vista y sinónimo referente a la tabla seguirá existiendo, pero ya no funcionará (conviene eliminarlos).
- Las transacciones pendientes son aceptadas (COMMIT), en aquellas bases de datos que tengan la posibilidad de utilizar transacciones.
- Lógicamente, solo se pueden eliminar las tablas sobre las que se tiene permiso de borrado.

Normalmente, el borrado de una tabla es irreversible, y no hay ninguna petición de confirmación, por lo que conviene ser muy cuidadoso con esta operación.

Ejemplo 14: Eliminación de una tabla

```
DROP TABLE profesor_asignatura;
```

3.5.12.- Truncado de una tabla

El comando TRUNCATE TABLE se utiliza para eliminar todas las filas de una tabla.

Eliminar filas con la instrucción TRUNCATE TABLE puede ser más eficaz que eliminar y volver a crear una tabla. Eliminar y volver a crear una tabla invalida los objetos dependientes de la tabla, requiere que se vuelvan a otorgar privilegios de objeto en la tabla, e implica que se vuelvan a crear los índices, las restricciones de integridad y los activadores en la tabla, y al mismo tiempo volver a especificar sus parámetros de almacenamiento. Truncar no tiene ninguno de estos efectos.

Eliminar filas con la declaración TRUNCATE TABLE puede ser más rápido que eliminar todas las filas con la declaración DELETE, especialmente si la tabla tiene numerosos desencadenadores, índices y otras dependencias.

Ejemplo 15: Eliminación de datos de una tabla “truncando” datos

```
TRUNCATE TABLE profesor_asignatura;
```

Anexo: Referencias

[1] ¿Qué son DDL, DML y DCL en MySQL?
Referencia: https://estudyando.com/que-son-ddl-dml-y-dcl-en-mysql/

[2] Create table
Referencia: https://www.w3schools.com/sql/sql_create_table.asp

[3] Drop table
Referencia: https://www.w3schools.com/sql/sql_drop_table.asp

[4] Alter table
Referencia: https://www.w3schools.com/sql/sql_alter.asp