ies ntra. sra de los remedios de ubrique

Tema 5 – Lenguaje SQL: DDL

1º DAW – Bases de Datos

J. Carlos Moreno

Curso 2022/2023

Contenido

5 Lenguaje SQL

5.1 Introducción

El SQL (StructuredQueryLanguage)es el lenguaje estándar ANSI/ISO (American NationalStandardsInstitute/InternationalOrganizationforStandardization) de definición, manipulación y control de bases de datos relacionales.

Es un lenguaje declarativo: sólo hay que indicar qué se quiere hacer. En cambio, en los lenguajes procedimentales es necesario especificar cómo hay que hacer cualquier acción sobre la base de datos.

El SQL es un lenguaje muy parecido al lenguaje natural; concretamente, se parece al inglés, y es muy expresivo. Por estas razones, y como lenguaje estándar, el SQL es un lenguaje con el que se puede acceder a todos los principales sistemas gestores de bases de datos comerciales y libres como Oracle, Access, MySQL, Postgre, etc.

5.1.1 Historia

Empezamos con una breve explicación de la forma en que el SQL ha llegado a ser el lenguaje estándar de las bases de datos relacionales:

1) Al principio de los años setenta, los laboratorios de investigación Santa Teresade IBM empezaron a trabajar en el proyecto System R. El objetivo de esteproyecto era implementar un prototipo de SGBD relacional; por lo tanto, tambiénnecesitaban investigar en el campo de los lenguajes de bases de datos relacionales.
2) A mediados de los años setenta, el proyecto de IBM dio como resultadoun primer lenguaje denominado SEQUEL (Structured English QueryLanguage), quepor razones legales se denominó más adelante SQL (StructuredQueryLanguage).
3) Al final de la década de los setenta y al principio de la de los ochenta, una vezfinalizado el proyecto System R, IBM y otras empresas empezaron a utilizar elSQL en sus SGBD relacionales, con lo que este lenguaje adquirió una gran popularidad.
4) En 1982, ANSI (American NationalStandardsInstitute) encargó a uno de sus comités(X3H2) la definición de un lenguaje de bases de datos relacionales. Este comité,después de evaluar diferentes lenguajes, y ante la aceptación comercial delSQL, eligió un lenguaje estándar que estaba basado en éste prácticamente en sutotalidad. El SQL se convirtió oficialmente en el lenguaje estándar de ANSI en elaño 1986, y de ISO (International StandardsOrganization) en 1987. También hasido adoptado como lenguaje estándar por FIPS (Federal InformationProcessingStandard), Unix X/Open y SAA (SystemsApplicationArchitecture) de IBM.
5) En el año 1989, el estándar fue objeto de una revisión y una ampliaciónque dieron lugar al lenguaje que se conoce con el nombre de SQL1 o SQL89.
6) En el año 1992 el estándar volvió a ser revisado y ampliado considerablementepara cubrir carencias de la versión anterior. Esta nueva versión del SQL, que seconoce con el nombre de SQL2 o SQL92.
7) En 1999 se publica SQL-99 o SQL3. Con características adicionales para soportar la gestión de datos orientada a objetos.

8) En 2003 se publica el actual estándar SQL:2003. Añade un nuevo apartado, el apartado 14: SQL/XML

5.1.2 Características

A partir del estándar cada Sistema Gestor de Base de Datos ha desarrollado su propio SQL que puede variar de un sistema a otro, pero con cambios que no suponen ninguna complicación para alguien que conozca un SQL concreto.

Como su nombre indica, el SQL nos permite realizar consultas a la base de datos. Pero el nombre se queda corto ya que SQL además realiza funciones de definición, control y gestión de la base de datos.

Las sentencias SQL se clasifican según su finalidad dando origen a tres 'lenguajes' o mejor dicho sublenguajes:

- DDL (Data DescriptionLanguage), lenguaje de definición de datos, incluye órdenes para definir, modificar o borrar las tablas en las que se almacenan los datos y de las relaciones entre estas. (Es el que más varia de un sistema a otro)
- DCL (Data Control Language), lenguaje de control de datos, contiene elementos útiles para trabajar en un entorno multiusuario, en el que es importante la protección de los datos, la seguridad de las tablas y el establecimiento de restricciones en el acceso, así como elementos para coordinar la compartición de datos por parte de usuarios concurrentes, asegurando que no interfieren unos con otros.
- DML (Data ManipulationLanguage), lenguaje de manipulación de datos, nos permite recuperar los datos almacenados en la base de datos y también incluye órdenes para permitir al usuario actualizar la base de datos añadiendo nuevos datos, suprimiendo datos antiguos o modificando datos previamente almacenados.

5.1.3 Funciones Básicas

Con el SQL se puede definir, manipular y controlar una base de datos relacional. A continuación veremos, aunque sólo en un nivel introductorio, cómo se pueden realizar estas acciones:

1) **Crear Tabla**. Sería necesario crear una tabla que contuviese los datos de editoriales de libros

```
DROP TABLE IF EXISTS `Editoriales`;
CREATE TABLE IF NOT EXISTS `Editoriales` (
 IdEditorial int(4) unsigned AUTO_INCREMENT PRIMARY KEY,
 Nombre     varchar(30),
 Direccion  varchar(50),
 Poblacion  varchar(25),
 IdProvincia  int(2) unsigned,
 CodPostal  char(5) ,
 Nif        char(9) Unique,
 Telefono  char(9),
 Fax        char(9),
 Email      varchar(40) unique,
 Web        varchar(40) unique,
 NombreContacto varchar(40),
 FOREIGN KEY (IdProvincia) REFERENCES `provincias` (IdProvincia)
 ON DELETE CASCADE ON UPDATE CASCADE
);
```

Ilustración 5-1. Crear Tabla Editoriales

2) **Insertar datos** en la tabla editoriales creada anteriormente

```
INSERT INTO `editoriales` VALUES
(1,'Ediciones Paraninfo S.A.','Avda. Filipinas, 50, Bajo, Dcha. puerta A',
'Madrid',28,'28003','A81461477','902995240',null, null, 'http://www.paraninfo.es', 'Patricia García'),
(2,'MCGRAWHILL','C/ Basauri, 17, 1ª Planta','Aravaca',28,
'28023','B28914323', '911803000',null, null, 'http://www.mcgraw-hill.es/', 'Raquel Huertas'),
(3,'RA-MA, S.A. Editorial y Publicaciones','Cl. Jarama, 3A Polígono Industrial IGARSA',
'Paracuellos de Jarama',28, '28860','M16584280', '916628139', '916628131','editorial@ra-ma.com',
'http://www.ra-ma.es/', 'Rocio García'),
(4,'Editorial Planeta, S.A.U.','Avda Diagonal 662-664',
'Barcelona',8,'08034','A08186249','934928978',null,'viajeros@geoplaneta.es','www.editorial.planeta.es',
'Roberto Rodríguez' ),
(5,'ALFAGUARA','Calle Torrelaguna, 60','Madrid',28,'28043','A0818624X','917449060','917449224',
'alfaguara@santillana.es','http://www.alfaguara.com/es/', 'Isidoro Moreno'),
(6, 'Anaya', 'Calle San Francisco, 30 A', 'Madrid', 28, '28014', 'A0012514C',
'917458458', '963547852', 'info@anaya.es', 'www.anaya.com', 'Rosario Vázquez'),
(7, 'Macmillan', 'Avd. de Chile, 40 A', 'Madrid', 28, '28016', 'A1212365C', '917458020',
'963232547', 'info@macmillan.es', 'www.macmillan.com', 'Isidoro Ríos');
```

Ilustración 5-2. Insertar Editoriales

3) **Consultar Tabla**. Consulta sobre la tabla Editoriales de forma que muestre sólo aquellas que son de Madrid, además sólo interesan las columnas IdEditorial, Nombre, Teléfono y Fax.

```
1  select
2      IdEditorial, Nombre, Telefono, Fax
3  from
4      Editoriales
5  where
6      Poblacion = 'Madrid';
```

Ilustración 5-3. Consultar Tabla Editoriales

4) **Controlar Usuarios**. Crear el usuario Marta para que sólo pueda acceder a consultar los campos IdEditorial, Nombre, Dirección, Población y Email de la Tabla Editoriales.

```
grant select(IdEditorial, Nombre, Direccion, Poblacion, Email)
on librosventas.Editoriales
to 'Marta'@'localhost' identified by 'marta_06';
```

Ilustración 5-4. Creación con privilegios del usuario Marta

5.1.4 Software Necesario

El software necesario para poder realizar las prácticas con lenguaje SQL es:

5.1.4.1 XAMPP

Distribución de Apache gratuita que integra en su última versión: MariaDB, PHP y Perl. Usaremos la versión v5.6.15 (PHP 5.6.15)

Ilustración 5-5. Xampp

MariaDB

MariaDB es un remplazo de MySQL con más funcionalidades y mejor rendimiento. MariaDB es un fork de MySQL que nace bajo la licencia GPL. Esto se debe a que Oracle compró MySQL y cambió el tipo de licencia por un privativo, aunque mantuvieron MySQLCommunityEdition bajo licencia GPL. La compatibilidad de MariaDB con MySQL es prácticamente total y por si fuese poco tenemos mejoras de rendimiento y funcionalidad. MariaDB está diseñado para reemplazar a MySQL directamente ya que mantiene las mismas órdenes, APIs y bibliotecas.

5.1.4.2 MySQLWorkbench

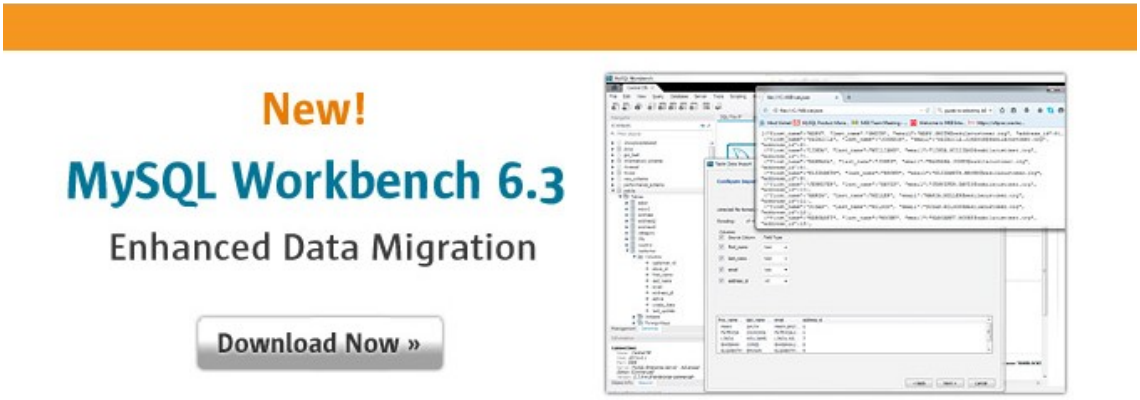

Ilustración 5-6. MySQLWorkbench 6.3

MySQLWorkbench es una herramienta visual de diseño de bases de datos que integra desarrollo de software, Administración de bases de datos, diseño de bases de datos, creación y mantenimiento para el sistema de base de datos MySQL. Es el sucesor de DBDesigner 4 de fabFORCE.net, y reemplaza el anterior conjunto de software, MySQL GUI Tools Bundle.

También es una herramienta de Oracle y usaremos la versión 6.3

5.2 Lenguaje DDL.

El grupo de instrucciones DDL del Lenguaje SQL abarca lo que llamamos las sentencias de definición.

Para poder trabajar con bases de datos relacionales, lo primero que tenemos que hacer es definirlas. Veremos las órdenes del estándar SQL92 para crear y borrar una base de datos relacional y para insertar, borrar y modificar las diferentes tablas que la componen.

En este apartado también veremos cómo se definen los dominios, las aserciones (restricciones) y las vistas.

La sencillez y la homogeneidad del SQL92 hacen que:

1) Para crear bases de datos, tablas, dominios, aserciones y vistas se utilice la sentencia **CREATE**.
2) Para modificar tablas y dominios se utilice la sentencia **ALTER**.
3) Para borrar bases de datos, tablas, dominios, aserciones y vistas se utilice lasentencia **DROP**.

¿Qué es una Vista?

Una vista en el modelo relacional no es sino una tabla virtual derivada de las tablas reales de nuestra base de datos, un esquema externo puede ser un conjunto de vistas.

La adecuación de estas sentencias a cada caso nos dará diferencias que iremos perfilando al hacer la descripción individual de cada una.

Base de Datos BDUOC

Para ilustrar la aplicación de las sentencias de SQL que veremos, utilizaremos una base de datos de ejemplo muy sencilla de una pequeña empresa con sede en Barcelona, Girona, Lleida y Tarragona, que se encarga de desarrollar proyectos informáticos. La información que nos interesará almacenar de esta empresa, que denominaremos BDUOC, será la siguiente:

1) Sobre los empleados que trabajan en la empresa, querremos saber su código de empleado, el nombre y apellido, el sueldo, el nombre y la ciudad de su departamento y el número de proyecto al que están asignados.
2) Sobre los diferentes departamentos en los que está estructurada la empresa, nos interesa conocer su nombre, la ciudad donde se encuentran y el teléfono. Será necesario tener en cuenta que un departamento con el mismo nombre puede estar en ciudades diferentes, y que en una misma ciudad puede haber departamentos con nombres diferentes.
3) Sobre los proyectos informáticos que se desarrollan, querremos saber su código, el nombre, el precio, la fecha de inicio, la fecha prevista de finalización, la fecha real de finalización y el código de cliente para quien se desarrolla.
4) Sobre los clientes para quien trabaja la empresa, querremos saber el código de cliente, el nombre, el NIF, la dirección, la ciudad y el teléfono.

5.2.1 Definición Base de Datos Relacional

5.2.1.1 Crear Base de Datos

SQL dispone de una sentencia para la creación de una base de datos relacional, dicha sentencia está basada en la siguiente sintaxis

```
CREATE {DATABASE | SCHEMA} [IF NOT EXISTS] db_name
    [create_specification] ...

create_specification:
    [DEFAULT] CHARACTER SET [=] charset_name
  | [DEFAULT] COLLATE [=] collation_name
```

La nomenclatura usada en la sintaxis es la siguiente:

- Las palabras en mayúsculas son palabras reservadas del lenguaje
- La notación **[…]** indica que es opcional su utilización
- La notación **{A| … |B}**quiere decir que tenemos que elegir una entre todas las opciones que hay entre las llaves

Observaciones CREATE DATABASE:

- La sentencia de creación de esquemas hace que varias tablas se puedan agrupar bajo un mismo nombre y que tengan un propietario (usuario). Aunque todos los parámetros de la sentencia CREATE SCHEMA son opcionales, como mínimo se debe dar el nombre del esquema o base de datos.

- La creación de una Base de Datos es una tarea administrativa, por lo que se necesita el premiso CREATE para su ejecución.
- **CREATE DATABASE y CREATE SCHEMA** son comandos sinónimos
- **[IF NOT EXISTS]** sólo crea la base de datos si no existe previamente.
- **Db_name**. Es el nombre de la base de datos. Nombre descriptivo sin espacios, caracteres raros y acentos.

Cláusulas CHARATER SET Y COLLATE

Mediante estas dos cláusulas se indica el juego de caracteres que voy a usar en la base de datos, es decir, el tipo de codificación que vamos a usar para almacenar los valores en nuestra BBDD. Si nos fijamos en las sintaxis, estas dos cláusulas son optativas, así que, si no se indican usa los valores establecidos por defecto en MySQL que normalmente suele ser el **utf8_general_ci**

Si fueras chino necesitarías codificar los datos con un tipo de cotejamiento que admitiera los símbolos chinos, al igual que si fueras musulmán o en definitiva si quisieras escribir con signos distintos a los occidentales.

La comunidad de habla española necesita aceptar la ñ o los acentos en nuestros valores, por lo que es muy importante elegir el tipo de cotejamiento adecuado.

En la siguiente tabla se muestran los tipos de cotejamientos más usados junto con el espacio ocupado por cada carácter

Idioma	CHARACTER SET	COLLATE	ESPACIO por carácter
Multilingüe	LATIN1	Latin1_general_ci	1 byte
Español	LATIN1	Latin1_spanish_ci	1 byte
Multilingüe	UTF8	Utf8_general_ci	Hasta 4 byte
Español	UTF8	Utf8_spanish_ci	Hasta 4 byte
Multilingüe	UTF8MB4	Utf8mb4_general_ci	Hasta 8 byte
Español	UTF8MB4	Utf8mb4_spanish_ci	Hasta 8 byte

Tabla 1. Tabla de cotejamientos más usuales

Ejemplos CREATE DATABASE

```sql
-- Muestra los juegos de caracteres disponibles mysql
SHOW CHARACTER SET;

-- Muestra los cotejamientos disponibles de mysql
SHOW COLLATION;

-- Crear la Base de Datos EJEMPLO con el cotejamiento establecido por defecto en MySQL
CREATE DATABASE EJEMPLO;

-- Crear la Base de Datos EJEMPLO sólo si no existe
CREATE DATABASE IF NOT EXISTS EJEMPLO;

-- Crear la Base de Datos BANCO Multilingüe con UTF8
CREATE DATABASE IF NOT EXISTS BANCO
CHARACTER SET UTF8 COLLATE UTF8_GENERAL_CI;

-- Crear la Base de Datos BANCO para Español con UTF8
CREATE DATABASE IF NOT EXISTS BANCO
CHARACTER SET UTF8 COLLATE UTF8_SPANISH_CI;

-- Crear la Base de Datos BANCO para Español con UTF8MB4
CREATE DATABASE IF NOT EXISTS BANCO
CHARACTER SET UTF8MB4 COLLATE UTF8MB4_SPANISH_CI;

-- Crear la Base de Datos BANCO para Español con LATIN1
CREATE DATABASE IF NOT EXISTS BANCO
CHARACTER SET LATIN1 COLLATE LATIN1_SPANISH_CI;

-- Crear la Base de Datos BANCO Multilingüe con LATIN1
CREATE DATABASE IF NOT EXISTS BANCO
CHARACTER SET LATIN1 COLLATE LATIN1_GENEARL_CI;
```

5.2.1.2 Modificación Base de Datos

Este comando normalmente se usa para modificar el juego de caracteres usados por una base de datos.

Para modificar una base de datos usamos un comando sql basado en la sentencia ALTER, a partir de la siguiente sintaxis.

```
ALTER {DATABASE | SCHEMA} [db_name]
    alter_specification ...

alter_specification:
    [DEFAULT] CHARACTER SET [=] charset_name
  | [DEFAULT] COLLATE [=] collation_name
```

Ejemplo:

```sql
-- Modifica la Base de Datos BANCO a Multilingüe con LATIN1
ALTER DATABASE BANCO
CHARACTER SET LATIN1 COLLATE LATIN1_GENEARL_CI;
```

5.2.1.3 Eliminar Base de Datos

Para eliminar una Base de Datos usamos un comando sql basado en la sentencia DROP a partir de la siguiente sintaxis

```
DROP {DATABASE | SCHEMA} [IF EXISTS] db_name
```

Hay que ser muy cuidadoso con el uso de este de DROP DATABASE ya que una vez que ejecutada elimina por completo la base de datos, y además, a no ser que tengamos una copia de seguridad, no podremos volver a recuperar los datos o deshacer la operación.

Ejemplos

```
1  -- Eliminar la Bae de Datos EJEMPLO
2  DROP DATABASE EJEMPLO;
3
4  -- Eliminar la Base de Datos BANCO
5  DROP DATABASE IF EXISTS BANCO;
6
7  -- Eliminar la Bae de Datos BIBLIOTECA
8  DROP SCHEMA IF EXISTS BIBLIOTECA;
```

5.2.2 Definición de Tablas

Una vez creada Base de Datos es preciso crear las tablas que la conforman.

Todas las tablas junto con las especificaciones de sus campos y restricciones (PRIMARY KEY, UNIQUE, FOREIGN KEY, NOT NULL, …) vienen especificadas en el Modelo Relacional, obtenido a partir de Diagrama Entidad Relación previo.

El proceso que hay que seguir para crear una tabla es el siguiente:

1. Lo primero que tenemos que hacer es decidir qué nombre queremos poner a la tabla, aunque dicho nombre ya está representado en el Modelo Relacional.
2. Después, iremos dando el nombre de cada de las columnas, campos o atributos de las tablas.Al igual que los nombres de las tablas serán nombres descriptivos, y que ya están representados también en el Modelo Relacional.
3. A cada una de las columnas le asignaremos un tipo de datos predefinido o bien un dominio definido por el usuario. También podremos dar definiciones por defecto y restricciones de columna.
4. Una vez definidas las columnas, sólo nos quedará dar las restricciones de tabla

5.2.2.1 Crear Tabla: CREATE TABLE

La sintaxis básica para la creación de tablas es la siguiente:

```
CREATE TABLE [IF NOT EXISTS] nombre_tabla(
      definición_columna
      [, definición_columna...]
      [, restricciones_tabla]
);
```

Donde definición_columna es

```
nombre_columna {tipo_datos|dominio} [restric_col]
```

CREATE TABLE crea una tabla con el nombre dado. Por defecto, la tabla se crea en la base de datos actual, además ocurrirá un error si la tabla existe, si no hay base de datos actual o si la base de datos no existe. En caso de que la tabla ya existiera usaremos **IF NOT EXISTS** para evitar errores, de esta forma sólo se ejecutará el comando si la tabla no existiera.

Ejemplos:

Creación de la Tabla Clientes

```
DROP TABLE IF EXISTS clientes;
CREATE TABLE IF NOT EXISTS clientes (
      Id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
      nombre VARCHAR(40),
      direccion VARCHAR(50),
      poblacion VARCHAR(50),
      cod_postal CHAR(5),
      provincia_id INT UNSIGNED,
      nif CHAR(9) UNIQUE,
      telefono CHAR(9),
      email VARCHAR(45) UNIQUE,
      fecha DATE,
      FOREIGN KEY (provincia_id) REFERENCES provincias (id)
      ON DELETE RESTRICT ON UPDATE CASCADE
);
```

Creación de la Tabla Libros

```
DROP TABLE IF EXISTS libros;
CREATE TABLE IF NOT EXISTS libros (
      id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
      isbn CHAR(13) UNIQUE,
      ean CHAR(13) UNIQUE,
      titulo VARCHAR(80) NOT NULL,
      precio_coste DECIMAL(10,2) DEFAULT '0.00',
      precio_venta DECIMAL(10,2) DEFAULT '0.00',
      stock_actual INT,
      stock_minimo INT,
      stock_maximo INT,
      fecha_edicion DATE,
      autor_id INT NOT NULL,
      editorial_id INT UNSIGNED,
      tema_id INT UNSIGNED,
      FOREIGN KEY (editorial_id) REFERENCES editoriales (id)
      ON DELETE SET NULL ON UPDATE CASCADE,
      FOREIGN KEY (autor_id) REFERENCES autores (id)
      ON DELETE SET NULL ON UPDATE CASCADE,
      FOREIGN KEY (tema_id) REFERENCES editoriales (id)
      ON DELETE SET NULL ON UPDATE CASCADE
);
```

5.2.2.2 Tipos de Datos

Al diseñar nuestras tablas tenemos que especificar el tipo de datos y tamaño que podrá almacenar cada campo. Una correcta elección debe procurar que la tabla no se quede corta en su capacidad, que destine un tamaño apropiado a la longitud de los datos, máxima velocidad de ejecución, en definitiva conseguir un almacenamiento óptimo con la menor utilización de espacio posible.

Básicamente mysql admite tres tipos de datos:

- Numéricos
- Cadena o string
- Fecha

5.2.2.2.1 Tipos Numéricos

En este tipo de campos solo pueden almacenarse números, positivos o negativos, enteros o decimales, en notación hexadecimal, científica o decimal.

Los tipos numéricos tipo ***INT o INTEGER*** admiten los atributos ***SIGNED*** y ***UNSIGNED*** indicando en el primer caso que pueden tener valor negativo, y solo positivo en el segundo.

Los tipos numéricos pueden además usar el atributo ***ZEROFILL*** en cuyo caso los números se completaran hasta la máxima anchura disponible con ceros.

```
# Declaro en una tabla una columna

  cantidad INT(5) ZEROFILL

# si a cantidad le asigno el valor 25, la base de datos lo almacenará
# como 00025
```

Los tipos numéricos son:

TinyInt

Cada valor que se le asigne a una columna con este tipo de dato ocupa 1 byte en el espacio de almacenamiento físico de la base de datos.

Es un número entero **corto** con o sin signo.

- Con signo el rango de valores -128 a 127
- Sin signo es de 0 – 255

```
# Declaro en una tabla una columna

  edad TINYINT UNSIGNED

# La cantidad máxima que puedo introducir en la columna edad es de 255
```

Bit ó Boolean

Número entero que puede ser 0 ó 1.

Este tipo de dato numérico se usa para asignar valores de tipo lógico:

- Valor 0 también se representa por la constante *false*.
- Valor 1 por la constante *true*

```
# Declaro en una tabla una columna
```

```
  carnet_conducir BOOLEAN

# La columna carnet_conducir almacenará un 0 o false si no tiene carnet y
# un 1 o true si posee carnet de conducir.
```

SmallInt

Cada valor que se le asigne a una columna con este tipo de dato ocupa 2 byte en el espacio de almacenamiento físico de la base de datos.

Número entero **pequeño** con o sin signo cuyo rango de valores sería:

- Con signo -32768 a 32767
- Sin signo 0 al 65535

```
# Declaro en una tabla una columna

  poblacion SMALLINT UNSIGNED NOT NULL

# La columna población podrá almacenar un máximo de 65535 habitantes.
# Además se le añade la restricción NOT NULL por lo que obligatoriamente
# debemos introducir el valor
```

MediumInt

Cada valor que se le asigne a una columna con este tipo de dato ocupará 3 bytes en el espacio de almacenamiento físico de la base de datos.

Número entero **medio** con o sin signo cuyo rango de valores sería:

- Con signo -8.388.608 a 8.388.607
- Sin signo 0 a16777215

```
# Declaro en una tabla una columna

  habitantes MEDIUMINT

# La columna habitantes podrá almacenar un máximo de 8.388.607
```

Integer o Int

Cada valor que se le asigne a una columna con este tipo de dato ocupará **4 bytes** en el espacio de almacenamiento físico de la base de datos.

Número entero con o sin signo cuyo rango de valores sería:

- Con signo -2 147 483 648 a 2 147 483 647
- Sin signo 0 a 4294967295

```
# Declaro en una tabla una columna

   Num_solicitud INT UNSIGNED UNIQUE
```

```
# La columna num_solicitud podrá almacenar hasta un máximo de 4.294.967.295
# millones de solicitudes
# Además la solicitud ha de ser única
```

BigInt

Cada valor que se le asigne a una columna con este tipo de dato ocupará **8 bytes** en el espacio de almacenamiento físico de la base de datos.

Número entero máxima amplitud con o sin signo cuyo rango de valores sería:

- Con signo desde -9.223.372.036.854.775.808 a 9.223.372.036.854.775.807
- Sin signo 0 a 18.446.744.073.709.551.615

```
# Declaro en una tabla una columna

   visitas BIGINT UNSIGNED

# La columna visitas podrá almacenar hasta 18.446.744.073.709.551.615
# millones de visitas
```

Float

Número pequeño en coma flotante de precisión simple. Se usa para asignar valores con precisión decimal.

El rango de valores si se usa con o sin signo:

- Con signo desde -3.402823466E+38 a -1.175494351E-38
- Sin signo el rango va desde 0 a 1.175494351E-38 a 3.402823466E+38.

```
# Declaro en una tabla una columna

   Suma_total FLOAT(10,5) UNSIGNED

# La columna suma_total podrá almacenar como máximo el valor 99.999,99999
```

xReal, double

Número pequeño en coma flotante de precisión doble. Se usa para asignar valores con precisión decimal. Cada valor ocupará 8 bytes.

El rango de valores permitidos son:

- Con signo desde 1.7976931348623157E+308 a -2.2250738585072014E-308
- Sin signo 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. Este tipo de dato lo usaremos mucho en nuestras prácticas para almacenar valores monetarios como precio, saldo, sueldo, importe, …

```
# Declaro en una tabla una columna

   precio DECIMAL(10,2)

# El precio máximo que podría almacenar sería 99999999.99 €
```

Tabla 2. Tamaño almacenamiento datos numéricos

TIPO DE DATO	TAMAÑO
TINYINT	1 byte
SMALLINT	2 bytes
MEDIUMINT	3 bytes
INT	4 bytes
INTEGER	4 bytes
BIGINT	8 bytes
FLOAT(X)	4 ú 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

5.2.2.2.2 TipoCarácter

Admiten valores alfanuméricos, los tipos son los siguientes:

- **CHAR.** Este tipo se utiliza para almacenar cadenas de longitud fija. Su longitud abarca desde 1 a 255 caracteres.
- **VARCHAR.** Al igual que el anterior se utiliza para almacenar cadenas, en el mismo rango de 1-255 caracteres, pero en este caso, de longitud variable. Un campo CHAR

ocupará siempre el máximo de longitud que le hallamos asignado, aunque el tamaño del dato sea menor (añadiendo espacios adicionales que sean precisos). Mientras que VARCHAR solo almacena la longitud del dato, permitiendo que el tamaño de la base de datos sea menor. Eso si, el acceso a los datos CHAR es mas rápido que VARCHAR.

- **TINYTEXT**, **TINYBLOB** para un máximo de 255 caracteres. La diferencia entre la familia de datatypestext y blob es que la primera es para cadenas de texto plano (sin formato) y case-insensitive (sin distinguir mayúsculas o minúsculas) mientras que blob se usa para objetos binarios: cualquier tipo de datos o información, desde un archivo de texto con todo su formato (se diferencia en esto del tipo Text) hasta imágenes, archivos de sonido o video
- **TEXT** y **BLOB** se usa para cadenas con un rango de 255 – 65535 caracteres. La diferencia entre ambos es que TEXT permite comparar dentro de su contenido sin distinguir mayusculas y minusculas, y BLOB si distingue.
- **MEDIUMTEXT**, **MEDIUMBLOB** textos de hasta 16777215 caracteres.
- **LONGTEXT**, **LONGBLOB**, hasta máximo de 4.294.967.295 caracteres
- **SET** un campo que puede contener ninguno, uno ó varios valores de una lista. La lista puede tener un máximo de 64 valores.
- **ENUM** es igual que SET, pero solo se puede almacenar uno de los valores de la lista

Tabla 3. Tamaño almacenamiento datos carácter

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

Tabla 4. Diferencias de almacenamiento entre CHAR Y VARCHAR

Valor	CHAR(4)	Tamaño	VARCHAR(4)	Tamaño
''	''	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

5.2.2.2.3 TiposFecha

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 esta comprendido entre 0 y 12 y que el día esta comprendido entre 0 y 31.

- **Date**: tipo fecha, almacena una fecha. El rango de valores va desde el 1 de enero del 1001 al 31 de diciembre de 9999. El formato por defecto es YYYY MM DD desde 0000 00 00 a 9999 12 31
- **DateTime**: Combinación de fecha y hora. El rango de valores va desde el 1 de enero del 1001 a las 0 horas, 0 minutos y 0 segundos al 31 de diciembre del 9999 a las 23 horas, 59 minutos y 59 segundos. El formato de almacenamiento es de año-mes-diahoras: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
- **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.

Tabla 5. Tamaño almacenamiento tipos fecha

Tipo de Campo	Tamaño de Almacenamiento
DATE	3 bytes
DATETIME	8 bytes
TIMESTAMP	4 bytes
TIME	3 bytes
YEAR	1 byte

Veamos el siguiente ejemplo:

```
USE test;
DROP TABLE IF EXISTS ejemplo;
CREATE TABLE IF NOT EXISTS ejemplo(
    id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
    fecha_nac TIMESTAMP(6) DEFAULT CURRENT_TIMESTAMP, -- millonésimas de segundos
    llegada TIME(3) default current_timestamp, -- milésimas de segundo
    anno YEAR default current_timestamp,
    fecha_llegada DATE default current_timestamp,
    fecha_hora_nacimiento DATETIME default current_timestamp
);

INSERT INTO ejemplo VALUES (null, default, default, default, default, default);
```

5.2.2.3 Restricciones

En este apartado el usuario podrá definir una serie de restricciones con objeto de mantener la validez de los datos.

Las restricciones definen reglas relativas a los valores permitidos en las columnas y constituyen el mecanismo estándar para exigir la integridad.

Estas restricciones las podemos implementar al momento de crear nuestras tablas (CREATE TABLE) o a la hora de modificarlas mediante (ALTER TABLE).

Dependiendo de cuando y como se definan podemos hablar de:

- Restricciones de Columna
- Restricciones de Tabla
- Definición de Restricciones (CONSTRAINT)

5.2.2.3.1 Restricciones de Columna

Cuando las restricciones se especifican junto con la definición de cada uno de los campos se llaman Restricciones de Columnas. Los tipos son las siguientes:

Restricciones de Columna	
Restricción	Descripción
NOT NULL	La columna no puede tener valores nulos.
UNIQUE	La columna no puede tener valores repetidos. Aplicable a las claves secundarias
PRIMARY KEY	La columna no puede tener valores repetidos ni nulos. Es la clave primaria.
RFERENCES tabla (columna)	La columna es la clave foránea de la columna de la tabla especificada.
DEFAULT 'Valor'	Establece un valor por defecto en dicha columna
CHECK (condiciones)	La columna debe cumplir las condiciones especificadas.

5.2.2.3.2 Restricciones de Tabla

Una vez hemos dado un nombre, hemos definido una tabla y hemos impuestociertas restricciones para cada una de las columnas, podemos aplicar restriccionessobre toda la tabla.

Los tipos de restricciones quese pueden en dicho caso son los siguientes:

Restricciones de Tabla	
Restricción	Descripción
UNIQUE (columna [, columna. . .])	El conjunto de las columnas especificadas no pueden contener valores repetidos. Son claves alternativas.
PRIMARY KEY(columna [, columna. . .])	El conjunto de las columnas especificadas no pueden contener valores nulos ni vacios. Es una clave primaria
FOREIGN KEY(columna [, columna. . .]) REFERENCES tabla[(columna2 [, columna2. . .])]ON DELETE tipo ON CASCADE tipo	El conjunto de las columnas especificadas esuna clave foránea que referencia la claveprimaria formada por el conjunto de lascolumnas2 de la tabla dada. Si las columnasy las columnas2 se denominan exactamente igual, entonces no sería necesario ponercolumnas2.
CHECK (condiciones)	La tabla debe cumplir las condiciones especificadas.

5.2.2.3.3 Definición de restricciones CONSTRAINT

Las restricciones son objetos propios de la base de datos y por lo tanto requieren de un nombre único compuesto del nombre del esquema o de la base de dato al que pertenece y el nombre que lo identifica, un ejemplo sería **nombreEsquema.nombreRestriccion.**

Sólo las restricciones de tipo tabla se pueden declarar asignándoles un nombre para ello debemos de usar la siguiente sintaxis:

```
[ CONSTRAINT name ] { [
    NULL | NOT NULL ] | UNIQUE | PRIMARY KEY | CHECK constraint }
[, ...]NombreRestriccionRestriccionTipoTabla
```

No es obligatorio declarar las restricciones detipo tabla mediante CONSTRAINT pero sí es conveniente para así administrar y gestionar más fácilmente el conjunto de restricciones de la base de datos.

Los tipos de restricciones que se pueden asignar mediante CONSTRAINT son NOT NULL, UNIQUE, PRIMARY KEY y CHECK.

5.2.2.3.4 Tipos Restricciones

Los diferentes tipos de restricciones que existen son los siguientes:

- NOT NULL
- UNIQUE
- PRIMARY KEY
- FOREIGN KEY
- DEFAULT
- CHECK

5.2.2.3.4.1 Restricción NOT NULL

La columna no acepta valores NULL es un campo obligatorio.

La restricción NOT NULL se aplica a un campo para que contenga siempre un valor. Esto significa que no se puede insertar un nuevo registro, o actualizar un registro de la tabla sin introducir un valor en dicho campo.

Normalmente este tipo de restricción suele ser de tipo columna.

Ejemplo de creación de la tabla libros

```
CREATE TABLE libros (

      id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
      titulo varchar(80) NOT NULL,
      autor  varchar(80) NOT NULL,

);
```

5.2.2.3.4.2 Restricción UNIQUE

Puede utilizar restricciones UNIQUE para garantizar que no se escriben valores duplicados en columnas específicas que no forman parte de una clave principal. Tanto la restricción UNIQUE como la restricción PRIMARY KEY exigen la unicidad; sin embargo, debe utilizar la restricción UNIQUE y no PRIMARY KEY si desea exigir la unicidad de una columna o una combinación de columnas que no forman la clave principal.

En una tabla se pueden definir varias restricciones UNIQUE, pero sólo una restricción PRIMARY KEY.Además, a diferencia de las restricciones PRIMARY KEY, las restricciones UNIQUE admiten valores NULL. Sin embargo, de la misma forma que cualquier valor incluido en una restricción UNIQUE, sólo se admite un valor NULL por columna.

Es posible hacer referencia a una restricción UNIQUE con una restricción FOREIGN KEY.

La restricción UNIQUE normalmente se especifica en la misma definición de la columna, pero cuando la clave alternativa está formada por más de una columna es necesario definir la en forma de restricción de tabla, es decir, en la parte inferior de la tabla bien mediante CONSTRATIN o no.

Vemos los siguientes ejemplos:

```
-- Definición UNIQUE en la Columna
CREATE TABLE libros (
      id int unsigned AUTO_INCREMENT Primary Key,
      isbn char(13) unique,
      ean char(13) unique,
);

-- Definición restricción UNIQUE en tabla
CREATE TABLE libros (
      id int unsigned AUTO_INCREMENT Primary Key,
      isbn char(13),
      ean char(13),
      UNIQUE(ISBN),
      UNIQUE(EAN)
);

-- Declaración CONSTRAINT
CREATE TABLE libros (
      id int unsigned AUTO_INCREMENT Primary Key,
      isbn char(13),
      ean char(13),
      CONSTRAINT UN_LibISBN unique(isbn),
      CONSTRAINT UN_LibEAN unique(ean)
);
```

5.2.2.3.4.3 Restricción PRIMARY KEY

Como vimos en el Modelo Relacional, una tabla suele tener una columna o una combinación de columnas cuyos valores identifican de forma única cada fila de la tabla. Estas columnas se

denominan claves principales de la tabla y exigen la integridad de entidad de la tabla. Puede crear una clave principal mediante la definición de una restricción PRIMARY KEY cuando cree (CREATE TABLE) o modifique una tabla (ALTER TABLE).

Una tabla sólo puede tener una restricción PRIMARY KEY y ninguna columna a la que se aplique una restricción PRIMARY KEY puede aceptar valores NULL. Debido a que las restricciones PRIMARY KEY garantizan datos únicos, con frecuencia se definen en una columna de identidad creada específicamente para tal propósito (IdArticulo, IdProveedor, …).

Cuando especifica una restricción PRIMARY KEY en una tabla, Motor de base de datos exige la unicidad de los datos mediante la creación de un índice único para las columnas de clave principal. Este índice también permite un acceso rápido a los datos cuando se utiliza la clave principal en las consultas. De esta forma, las claves principales que se eligen deben seguir las reglas para crear índices únicos.

Si se define una restricción PRIMARY KEY para más de una columna, puede haber valores duplicados dentro de la misma columna, pero cada combinación de valores de todas las columnas de la definición de la restricción PRIMARY KEY debe ser única.

Veamos los siguientes ejemplos:

```
USE TEST;

-- Restricción de Columna
CREATE TABLE Ejemplo3 (
      id INT UNSIGNED AUTO_INCREMENT ZEROFILL PRIMARY KEY,
      Nombre VARCHAR(20),
      Apellidos VARCHAR(40)
);

-- Restricción de tipo Tabla
CREATE TABLE Ejemplo2 (
      id INT UNSIGNED AUTO_INCREMENT ZEROFILL,
      Nombre VARCHAR(20),
      Apellidos VARCHAR(40),
      PRIMARY KEY(id)
);

-- Restricción de Tabla con CONSTRAINT
CREATE TABLE Ejemplo (
      id INT UNSIGNED AUTO_INCREMENT ZEROFILL,
      Nombre VARCHAR(20),
      Apellidos VARCHAR(40),
      CONSTRAINT PK_CodEjemplo PRIMARY KEY(id)
);


-- Clave Primaria compuesta por varios atributos
CREATE TABLE Ejemplo4(
      Profesor_id INT UNSIGNED,
      Dia TINYINT UNSIGNED,
      Tramo TINYINT UNSIGNED,
      Asignatura VARCHAR(25),
      Horas TINYINT UNSIGNED,
      PRIMARY KEY (Profesor_id,Dia,Tramo)
)

-- Clave Primaria compuesta por varios atributos con CONSTRAINT
CREATE TABLE Ejemplo5(
      Profesor_id INT UNSIGNED,
      Dia TINYINT UNSIGNED,
      Tramo TINYINT UNSIGNED,
      Asignatura VARCHAR(25),
      Horas TINYINT UNSIGNED,
      CONSTRAINT PK_Ejemplo5 PRIMARY KEY (profesor_id,Dia,Tramo)
);
```

5.2.2.3.4.4 Restricción FOREIGN KEY

Como vimos en el Modelo Relacional una clave externa (FK) es una columna o combinación de columnas que se utiliza para establecer y exigir un vínculo entre los datos de dos tablas. Puede crear una clave externa mediante la definición de una restricción FOREIGN KEY cuando cree (CREATE TABLE) o modifique una tabla (ALTER TABLE).

En una referencia de clave externa, se crea un vínculo entre dos tablas cuando las columnas de una de ellas hacen referencia a las columnas de la otra que contienen el valor de clave principal. Esta columna se convierte en una clave externa para la segunda tabla.

Una restricción FOREIGN KEY puede hacer referencia a columnas de tablas de la misma base de datos o a columnas de una misma tabla.

Customers

ID	Last Name	First Name
007	Bond	James
024	Bauer	Jack
123	Smith	Jane

Orders

ID	Cust_ID	Item	Qty
1	007	Fancy Gadget	4
2	024	Hand Gun	1
3	024	Bullet-proof vest	1

No es necesario que una restricción FOREIGN KEY esté vinculada únicamente a una restricción PRIMARY KEY de otra tabla; también puede definirse para que haga referencia a las columnas de una restricción UNIQUE de otra tabla. Una restricción FOREIGN KEY puede contener valores NULL, pero si alguna columna de una restricción FOREIGN KEY compuesta contiene valores NULL, se omitirá la comprobación de los valores que componen la restricción FOREIGN KEY. Para asegurarse de que todos los valores de la restricción FOREIGN KEY compuesta se comprueben, especifique NOT NULL en todas las columnas que participan.

Una restricción FOREIGN KEY puede hacer referencia a columnas de tablas de la misma base de datos o a columnas de una misma tabla, se trata de las referencias o relaciones reflexivas.

En una tabla se pueden establecer las restricciones FOREIGN KEY que sean necesarias no habiendo un límite establecido para ello, aunque hay que tener en cuenta el costo de gestión que supone la creación de este tipo de restricciones, que por otro lado fundamentales para establecer la INTEGRIDAD REFERENCIAL de los datos.

Sólo el motor de almacenamiento INNODB soporta las restricciones FOREIGN KEY, normalmente es el configurado por defecto en la instalación XAMPP de MySQL. Existen otros motores de almacenamiento MyISAMespecíficos para bases de datos con una sola tabla o diseñados exclusivamente para hacer consultas.

La sintaxis completa de FOREIGN KEY sería

```
Restricción de Columna:
      NomColumnTipoDato REFERENCES TablaReferencia(SuClavePrincipal)
      [ON DELETE {RESTRICT | CASCADE | SET NULL | NO ACTION}]
      [ON UPDATE {RESTRICT | CASCADE | SET NULL | NO ACTION}]

Restricción de Tabla
      [CONSTRAINT NombRestricción] FOREIGN KEY (ClaveAjena) REFERENCES
      TablaReferencia (SuClavePrincipal)
      [ON DELETE {RESTRICT | CASCADE | SET NULL | NO ACTION}]
      [ON UPDATE {RESTRICT | CASCADE | SET NULL | NO ACTION}]
```

Hay que indicar con respecto a la sintaxis que NO ACTION y RESTRICT en MySQL son equivalentes, y que no incluye el tipo SET DEFAULT.

Dadas las siguientes tablas relacionadas

id	nombre	apellidos	telefono	ciudad	dni	email
1	Javier	Gonzalez Ramirez	610867829	Puerto Serrano	21425881F	karRTRa.hand@example.net
2	Warren	Marc Edwars	547854125	algodonales	65847584G	warren@gmail.com
3	Wava Waters I	Reichel	373080734	South Hayleyshire	41797091p	kara.hand@example.net
4	Kaitlin Ebert	Leffler	729184085	O'Haraview	32592665u	earnest.smith@example.net
9	Prof. Tyshawn Lueilwitz Jr.	Feeney	655166674	Port Henriette	79543255d	rbauch@example.com
10	Angelita Rath	Heathcote	650468819	Keelingburgh	45456203z	walter.adah@example.org
11	Hank Cole	Kassulke	161329522	North Jenachester	63153970g	cormier.mellie@example.org
12	Kellie Green	O'Conner	297765953	Javierton	73362605s	kshlerin.hailey@example.net
13	Dr. Myles Cremin II	Hermann	620524346	East Bryana	42921549h	runolfsdottir.thomas@example.com
14	Chloe Walsh	Kuphal	373611521	East Monserrat	04142411n	lakin.daphne@example.net

Ilustración 7. Tabla clients

id	numCuenta	client_id	fechaAlta	saldo	fechaUMov	numMvtos	created_at	updated_at
1	96857485968574859685748 5	2	2018-04-22 00:00:00	658475.00	2018-04-22 00:00:00	6	NULL	NULL
2	96885968968574859685748 5	1	2018-04-12 00:00:00	8.00	2018-04-28 00:00:00	1	NULL	NULL
3	96857485968574859685748 5	11	2019-03-10 20:23:55	12.00	2019-03-10 20:23:55	0	2019-03-10 19:23:55	2019-03-10 19:23:55
4	12545214541254785452545 9	2	2019-03-10 20:27:14	5065718.00	2019-03-10 20:27:14	0	2019-03-10 19:27:14	2019-03-10 19:27:14

Ilustración 8. Tabla accounts

Veamos algunos ejemplos de FOREIGN KEY

```
USE TEST;

-- DefiniciónTabla clients
CREATE TABLE clients (
	id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
	nombre VARCHAR(20),
	apellidos VARCHAR(30),
	telefono CHAR(12),
	ciudad VARCHAR(30),
	dni CHAR(9) UNIQUE,
	email VARCHAR(50) UNIQUE
);

-- Definición Tabla accounts

-- Restricción de Columna
CREATE TABLE accounts(
	id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
	iban CHAR(22),
	fecha_alta TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
	fecha_mov TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
	num_movimientos INT,
	create_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
	update_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
	client_id INT UNSIGNED REFERENCES clients (id)
	ON DELETE RESTRICT ON UPDATE CASCADE
);
```

```
USE TEST;

-- Restricción de tabla
CREATE TABLE accounts(
      id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
      iban CHAR(22),
      fecha_alta TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
      fecha_mov TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
      num_movimientos INT,
      create_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
      update_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
      client_id INT UNSIGNED,
      FOREIGN KEY (client_id)REFERENCES clients (id)
      ON DELETE RESTRICT ON UPDATE CASCADE
);

-- Restricción mediante CONSTRAINT
CREATE TABLE accounts(
      id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
      iban CHAR(22),
      fecha_alta TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
      fecha_mov TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
      num_movimientos INT,
      create_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
      update_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
      client_id INT UNSIGNED,
      CONSTRAINT fk_client_id_accounts FOREIGN KEY
      (client_id)REFERENCES clients (id)
      ON DELETE RESTRICT ON UPDATE CASCADE
);
```

Motores de Almacenamiento

El motor de almacenamiento (storage-engine) se encarga de almacenar, manejar y recuperar información de una tabla. Los motores más conocidos son MyISAM e InnoDB. La elección de uno u otro dependerá mucho del escenario donde se aplique.

En la elección se pretende conseguir la mejor relación de calidad acorde con nuestra aplicación. Si necesitamos transacciones, claves foráneas y bloqueos, tendremos que escoger InnoDB. Por el contrario, escogeremos MyISAM en aquellos casos en los que predominen las consultas SELECT a la base de datos.

InnoDB dota a MySQL de un motor de almacenamiento transaccional (conforme a ACID) con capacidades de commit (confirmación), rollback (cancelación) y recuperación de fallos. InnoDB realiza bloqueos a nivel de fila y también proporciona funciones de lectura consistente sin bloqueo al estilo Oracle en sentencias SELECT. Estas características incrementan el rendimiento y la capacidad de gestionar múltiples usuarios simultáneos. No se necesita un bloqueo escalado en InnoDB porque los bloqueos a nivel de fila ocupan muy poco espacio. InnoDB también soporta restricciones FOREIGN KEY. En consultas SQL, aún dentro de la misma consulta, pueden incluirse libremente tablas del tipo InnoDB con tablas de otros tipos.

InnoDB es el motor por defecto. Para crear una tabla InnoDB se debe especificar la opción ENGINE = InnoDB en la sentencia SQL de creación de tabla:

```
CREATE TABLE NombreTabla (Columnas [,Columnas]) ENGINE=InnoDB;
```

Ventajas: MyISAM vs InnoDB

InnoDB:

- Soporte de transacciones
- Bloqueo de registros
- Nos permite tener las características ACID (Atomicity, Consistency, Isolation and Durability: Atomicidad, Consistencia, Aislamiento y Durabilidad en español), garantizando la integridad de nuestras tablas.
- Es probable que si nuestra aplicación hace un uso elevado de INSERT y UPDATE notemos un aumento de rendimiento con respecto a MyISAM.

MyISAM

- Mayor velocidad en general a la hora de recuperar datos.
- Recomendable para aplicaciones en las que dominan las sentencias SELECT ante los INSERT / UPDATE.
- Ausencia de características de atomicidad ya que no tiene que hacer comprobaciones de la integridad referencial, ni bloquear las tablas para realizar las operaciones, esto nos lleva como los anteriores puntos a una mayor velocidad.

¿Aún tienes dudas de qué motor es el que necesitas? Te ayudamos a tomar tu decisión

- ¿Tu tabla va a recibir INSERTs, UPDATEs y DELETES mucho más tiempo de lo que será consultada? Te recomendamos InnoDB
- ¿Necesitarás hacer búsquedas full-text? Tu motor ha de ser MyISAM
- ¿Prefieres o requieres diseño relacional de bases de datos? Entonces necesitas InnoDB
- ¿Es un problema el espacio en disco o memoria RAM? Decántate por MyISAM

Ejemplo:

```
USE TEST;

-- Definición Tabla Personas con Motor de Almacenamiento InnoDB
CREATE TABLE Personas (
      id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
      Apellido VARCHAR(40) NOT NULL,
      Nombre VARCHAR(15) NOT NULL,
      Direccion VARCHAR(50),
      Ciudad VARCHAR(20),
      Email VARCHAR(50) NOT NULL
) ENGINE=InnoDB;
```

5.2.2.3.4.5 Restricción DEFAULT

Cada columna de un registro debe contener un valor, aunque sea un valor NULL. Puede haber situaciones en las que deba cargar una fila de datos en una tabla, pero no conozca el valor de una columna o el valor ya no exista. Si la columna acepta valores NULL, puede cargar la fila con un valor NULL. Pero, dado que puede no resultar conveniente utilizar columnas que acepten valores NULL, una mejor solución podría ser establecer una definición DEFAULT para la columna siempre que sea necesario. Por ejemplo, es habitual especificar el valor cero como valor predeterminado para las columnas numéricas, o N/D (no disponible) como valor predeterminado para las columnas de cadenas cuando no se especifica ningún valor.

La restricción DEFAULT especifica un valor por defecto para un campo cuando no se inserta explícitamente en un comando INSERT.

Dicha restricción se puede asignar mediante comando CREATE TABLE o ALTER TABLE.

Esta restricción se asigna a nivel de columna, aunque también se puede añadir una nueva restricción DEFAULT medianteADD CONSTRAINT incluido en un ALTER TABLE.

Veamos los siguientes ejemplos de restricciones DEFAULT

```
USE TEST;

-- Definición restricción DEFAULT a Nivel Columna
CREATE TABLE Factura (
IdFactura INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
CodFacturaCHAR(20) DEFAULT '00000/2016',
Fecha timestamp DEFAULT CURRENT_TIMESTAMP,
Importe decimal(10,2) DEFAULT 0.00,
FormaPagoVarchar(20) DEFAULT 'Contado'
);
```

En otras ocasiones para establecer el valor por defecto podemos recurrir a definir nuestro propio literal mediante funciones o procedimientos almacenados, apartado que veremos más adelante, o recurrir a alguna de las siguientes funciones MySQL:

Funciones con DEFAULT	
Función	Descripción
{USER|CURRENT_USER}	Identificador del usuario actual
SESSION_USER	Identificador del usuario de esta sesión
SYSTEM_USER	Identificador del usuario del sistema operativo
CURRENT_DATE	Fecha actual
CURRENT_TIME	Hora actual
CURRENT_TIMESTAMP	Fecha y hora actual

5.2.2.3.4.6 Restricción CHECK

Las restricciones CHECK exigen la integridad del dominio mediante la limitación de los valores que puede aceptar una columna. Son similares a las restricciones FOREIGN KEY porque controlan los valores que se colocan en una columna. La diferencia reside en la forma en que determinan qué valores son válidos: las restricciones FOREIGN KEY obtienen la lista de valores válidos de otra tabla, mientras que las restricciones CHECK determinan los valores válidos a partir de una expresión lógica que no se basa en datos de otra columna.

Por ejemplo, es posible limitar el intervalo de valores para una columna SALARIO creando una restricción CHECK que sólo permita datos entre 15.000 y 100.000 €. De este modo se impide que se escriban salarios superiores al intervalo de salario normal.

Se puede aplicar varias restricciones CHECK a una columna. También puede aplicar una sola restricción CHECK a varias columnas si se crea en el nivel de la tabla.

Las restricciones CHECK se pueden establecer tanto al crear la tabla con CREATE TABLE o bien modificándola con ALTER TABLE.

CONDICIÓN DEL CHECK

Para poder crear una restricción CHECK tendremos que establecer una Condición o Expresión Lógica (boolean) que devuelva TRUE o FALSE, si devuelve TRUE admite el valor asignado y en caso contrario lo rechaza.

```
CHECK (condición)
```

Para construir la condición o expresión lógica puedo usar:

- Operadores de comparación
- Cláusula BETWEEN
- Cláusula IN
- Cláusula LIKE

La expresión lógica puede estar formada por varias condiciones unidas por los operadores lógicos AND, OR y NOT, pudiendo usar los paréntesis para establecer la prioridad de las operaciones.

Operadores de Comparación

Construye la condición basándose en los siguientes operadores de comparación:

- (>, <, >=, <=, =, IS NULL, IS NOT NULL)

Ejemplos:

```
USE TEST;

-- Definición restricción CHECK a nivel de columna
CREATE TABLE Pedido(
      NumPedido int AUTO_INCREMENT PRIMARY KEY,
      FechaPedido TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
      FormaPago varchar(25),
      Transporte Varchar(25),
      Observaciones TEXT,
      ImportePedido DECIMAL(10, 2) CHECK (ImportePedido> 0),
      Cliente Char(5) References Empresa (CodEmpresa)
      ON DELETE RESTRICT ON UPDATE CASCADE
);
```

En el ejemplo anterior no admitirá insertar aquellos pedidos cuyo importe sea cero.

Otros ejemplos aislados serían

```
TotalImporte DECIMAL(10,2) CHECK (TotalImporte> 400)
NBeneficiarios TINYINT UNSIGNED CHECK (NBeneficiarios< 5)
SalarioBase DECIMAL(10,2) CHECK (SalarioBase> 600)
NAsignaturas TINYINT UNISIGNED CHECK (NOT NAsignaturas> 10)
```

La restricción CHECK se puede definir también a nivel de Tabla en dicho caso podemos aplicar la restricción a varias columnas

```
USE TEST;

-- Definición restricción CHECK a nivel de tabla
CREATE TABLE Pedido(
      id int unsigned AUTO_INCREMENT PRIMARY KEY,
      FechaPedido TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
      FormaPagovarchar(25),
      Transporte Varchar(25),
      Observaciones TEXT,
      ImportePedido DECIMAL(10, 2),
      Cliente Char(5) References Empresa (CodEmpresa)
      ON DELETE RESTRICT ON UPDATE CASCADE,
      CHECK (ImportePedido> 0 AND FormaPago='CONTADO')
);
```

Definición de una restricción CHECK mediante CONSTRAINT

```
USE TEST;

-- Definición restricción CHECK a nivel de tabla
CREATE TABLE Pedido(
      id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
      FechaPedido TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
      FormaPago varchar(25),
      Transporte Varchar(25),
      Observaciones TEXT,
      ImportePedido DECIMAL(10, 2),
      Cliente Char(5) References Empresa (CodEmpresa)
      ON DELETE RESTRICT ON UPDATE CASCADE,
      CONSTRAINT CH_ImportePedido CHECK (ImportePedido> 0 AND
      ImportePedido< 12000)
);
```

Cláusula BETWEEN

La cláusula BETWEEN permite definir un rango de forma que sólo admitirá los valores comprendidos en dicho rango.

[NOT] BETWEEN ValorMínimo AND ValorMáximo

Ejemplo

```
CREATE TABLE IF NOT EXISTS LineaPedido(
      LineaPedidoint AUTO_INCREMENT,
      NumPedido INT,
      IdArticulo Char(7) NOT NULL,
      Cantidad INT,
      Precio DECIMAL(10, 2 ),
      TipoDescuent TINYINT UNSIGNED,
      Observaciones TEXT,
      CONSTRAINT pK_LineaPedido PRIMARY KEY (LineaPedido,NumPedido),
      CONSTRAINT fk_NumPedido FOREIGN KEY (NumPedido)
      REFERENCES Pedido (NumPedido),
      CONSTRAINT ct_TipoDescuent CHECK (TipoDescuent BETWEEN 1 AND 4)
);

/* TipoDescuent solo admitiría valores entre 1 y 4 ambos inclusives.
```

Cláusula IN

Permite definir un conjunto de forma que sólo admitirá los valores incluidos en dicho conjunto.

[NOT] IN (Valor1, Valor2, … , ValorN)

Veamos el siguiente ejemplo

```
CREATE TABLE Matricula(
      NumMatricula Int AUTO_INCREMENT PRIMARY KEY,
      FechaMatricula TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
      NombreAlumno Varchar(50),
      DniAlumno Char(9) UNIQUE,
      Email Varchar(50) CHECK (Email Like '%@%'),
      CiclEducativo Varchar(20)
      CHECK (CicloEducativo IN ('ESO','BACH', 'MEDIO', 'SUPERIOR',
      'ADULTOS')),
      Curso TINYINT UNSIGNED CHECK (Curso IN (1,2, 3, 4))
);
```

Cláusula LIKE

Permite definir un patrón de caracteres de forma que sólo admitirá aquellos valores que cumplan con dicho patrón.

[NOT] LIKE ‘Patrón’

Para declarar el Patrón se pueden usar dos caracteres comodines:

- '**%**' – sustituye a un conjunto de caracteres
- '**_**' – sustituye a un solo carácter

Algunos ejemplos aislados serían los siguientes

```
-- Nombre ha de comenzar por b
Nombre LIKE 'b%'
-- Nombre ha de terminar por fy
   Nombre LIKE '%fy'
-- Nombre ha de contener la letra s
   Nombre LIKE '%s%'
-- Email ha de contener @
Email LIKE '%@%'
-- Nombre donde el tercer carácter ha de ser una a
Nombre LIKE '__a%'
--
Email Varchar (50) CHECK (Email like '%@%')
CodArticuloCHAR(5) CHECK (CodArticulo Like '_AB__')
```

Un ejemplo completo integrado en la definición de una tabla

```
CREATE TABLE Matricula(
      NumMatricula INT AUTO_INCREMENT PRIMARY KEY,
      FechaMatricula TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
      NombreAlumno Varchar(50),
      DniAlumno Char(9) UNIQUE,
      Email Varchar(50) CHECK (Email Like '%@%'),
      CiclEducativo Varchar(20)
      CHECK (CicloEducativo IN ('ESO','BACH', 'MEDIO', 'SUPERIOR',
      'ADULTOS')),
      Curso TINYINT UNSIGNED CHECK (Curso IN (1,2, 3, 4))
);
```

5.2.2.4 Modificar Tablas: ALTER TABLE

Permite realizar modificaciones en la estructura o definición de una tabla permitiendo:

- Sobre las **columnas**: añadir, modificar y eliminar columnas
- Sobre las **restricciones**: añadir y eliminar restricciones

La sintaxis puede resultar algo compleja sabiendo el significado de las palabras reservadas la sentencia se aclara bastante; ADD (añade), ALTER (modifica), DROP (elimina), COLUMN (columna), CONSTRAINT (restricción).

```
ALTER TABLE tbl_nombreespecificar_alteracion [, especificar_alteracion] ...
```

5.2.2.4.1 Añadir Columnas: ADD COLUMN

Permite añadir una nueva columna a una tabla existente incluyendo el nombre de la columna, el tipo de dato y las restricciones si procede.

```
ALTER TABLE nombre_tabla ADD [COLUMN] nombre_columnatipoDeDato [restricciones]
```

Ejemplos

```
USE Test;
-- Creamos la tabla Factura
DROP TABLE IF EXISTS Factura;
CREATE TABLE IF NOT EXISTS Factura (
      IdFactura INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
      CodFactura CHAR(20) DEFAULT '00000/2016',
      Fecha timestamp DEFAULT CURRENT_TIMESTAMP,
      Importe decimal(10,2) DEFAULT 0.00,
      FormaPago Varchar(20) DEFAULT 'Contado'
);

-- Modificamos la tabla Factura añadiendo tres columnas
-- incluyendo restricciones en alguna de ellas
ALTER TABLE factura
ADD column Transportevarchar (30),
ADD COLUMN Cliente CHAR(5) DEFAULT '00000',
ADD COLUMN TiempoServicio TINYINT UNSIGNED CHECK (TiempoServicio>0);
```

5.2.2.4.2 ModificarTipoDato de una Columna: MODIFY

Permite cambiar el tipo de dato de una columna incluyendo el nombre de la columna, el tipo de dato y las posibles restricciones si procede, aunque esta cláusula no admite restricciones CHECK.

```
ALTER TABLE nombre_tabla MODIFY [COLUMN]nombre_columnatipoDeDato [Restricciones];
```

Veamos ahora un ejemplo a partir de la definición anterior de Factura

```
USE Test;
-- Creamos la tabla Factura
DROP TABLE IF EXISTS Factura;
CREATE TABLE IF NOT EXISTS Factura (
      IdFactura INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
      CodFactura CHAR(20) DEFAULT '00000/2016',
      Fecha timestamp DEFAULT CURRENT_TIMESTAMP,
      Importe decimal(10,2) DEFAULT 0.00,
      FormaPago Varchar(20) DEFAULT 'Contado'
);

-- Modificamos la tabla Factura añadiendo tres columnas
-- incluyendo restricciones en alguna de ellas
ALTER TABLE factura
ADD column Transportevarchar (30),
ADD COLUMN Cliente CHAR(5) DEFAULT '00000',
ADD COLUMN TiempoServicio TINYINT UNSIGNED CHECK   (TiempoServicio>0);
--
-- Modificamos alguno de los tipos de datos
-- añadiendo al final una nueva columna
--
ALTER TABLE Factura
MODIFY COLUMN Transporte varchar(20) NOT NULL,
    MODIFY COLUMN Importe DECIMAL(12,2) DEFAULT 0.0,
    MODIFY COLUMN CodFacturaCHAR(10) DEFAULT '00000/2016',
    ADD COLUMN Estado ENUM('Pedido', 'EnCurso', 'Entregado');
```

5.2.2.4.3 Modificar Nombre de una Columna: CHANGE

Permite modificar el nombre de una columna especificando el nombre antiguo, el nuevo nombre, el tipo de dato y las posibles restricciones si procede excluyendo a las de tipo CHECK.

Al cambiar el nombre de la columna puedo optar por modificar el tipo de dato y las posibles restricciones excepto las de tipo CHECK.

ALTER TABLE table_name CHANGE [COLUMN] OldNameColumnNewNameColumnTipoDeDato [Restricciones]

Veamos algunos ejemplos

```
USE TEST;
DROP TABLE IF EXISTS Factura;
      CREATE TABLE IF NOT EXISTS `factura` (
      `IdFactura` int(10) unsigned NOT NULL AUTO_INCREMENT,
      `CodFactura` char(10) DEFAULT '00000/2016',
      `Fecha` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
      `Importe` decimal(12,2) DEFAULT '0.00',
      `FormaPago` varchar(20) DEFAULT 'Contado',
      `Transporte` varchar(20) NOT NULL,
      `Cliente` char(5) DEFAULT '00000',
      `TiempoServicio` tinyint(3) unsigned DEFAULT NULL,
      `Estado` enum('Pedido','EnCurso','Entregado') DEFAULT NULL,
      PRIMARY KEY (`IdFactura`)
) ENGINE=InnoDB;

--
-- Al modificar el nombre de las columnas también puedo
-- optar por cambiar el tipo de dato y las restricciones.

ALTER TABLE Factura
    CHANGE COLUMN FechaFechaFactura TIMESTAMP DEFAULT
    CURRENT_TIMESTAMP,
    CHANGE COLUMN Importe ImporteFact DECIMAL(10,2) DEFAULT '0.0',
    CHANGE COLUMN Transporte FormaEnvio Varchar(25);
```

5.2.2.4.4 Eliminar Columna: DROP

Permite eliminar una columna de una tabla existente indicando sólo el nombre de la columna, sin necesidad de indicar el tipo de dato ni las posibles restricciones. Antes de ejecutar este comando es preciso asegurarse de su elección ya que supone una posible pérdida de datos, sin posibilidad de recuperación, a no ser que tengamos respaldo de seguridad.

ALTER TABLE NombreTabla DROP [COLUMN] NombreColumna

Veamos algunos ejemplos:

```
USE TEST;
DROP TABLE IF EXISTS Factura;
CREATE TABLE IF NOT EXISTS `factura` (
       `IdFactura` int(10) unsigned NOT NULL AUTO_INCREMENT,
       `CodFactura` char(10) DEFAULT '00000/2016',
       `Fecha` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
       `Importe` decimal(12,2) DEFAULT '0.00',
       `FormaPago` varchar(20) DEFAULT 'Contado',
       `Transporte` varchar(20) NOT NULL,
       `Cliente` char(5) DEFAULT '00000',
       `TiempoServicio` tinyint(3) unsigned DEFAULT NULL,
       `Estado` enum('Pedido','EnCurso','Entregado') DEFAULT NULL,
PRIMARY KEY (`IdFactura`)
) ENGINE=InnoDB;

--
-- En un mismo ALTER TABLE puedo incluir
-- modificaciones de varios tipos
--

ALTER TABLE Factura
   DROP COLUMN Fecha,
   DROP COLUMN Importe,
   ADD COLUMN FechaFactura TIMESTAMP DEFAULT CURRENT_TIMESTAMP;
```

5.2.2.4.5 Modificar Valor por Defecto: SET DEFAULT

Permite establecer, modificar o eliminar el valor por defecto asignado a una columna mediante la restricción DEFAULT.

ALTER TABLE *table_name*
ALTER *Nom_Column* {SET DEFAULT *Valor* | *DROP DEFAULT}*

Veamos el siguiente ejemplo

```
USE TEST;
DROP TABLE IF EXISTS Factura;
CREATE TABLE IF NOT EXISTS `factura` (
`IdFactura` int(10) unsigned NOT NULL AUTO_INCREMENT,
`CodFactura` char(10) DEFAULT '00000/2016',
`Fecha` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
`Importe` decimal(12,2) DEFAULT '0.00',
`FormaPago` varchar(20) DEFAULT 'Contado',
`Transporte` varchar(20) NOT NULL,
`Cliente` char(5) DEFAULT '00000',
`TiempoServicio` tinyint(3) unsigned DEFAULT NULL,
`Estado` enum('Pedido','EnCurso','Entregado') DEFAULT NULL,
PRIMARY KEY (`IdFactura`)
) ENGINE=InnoDB;

ALTER TABLE Factura
       ALTER Importe SET DEFAULT 0.01,
       ALTER FormaPago SET DAFAULT ‘Metálico,
       ALTER TiempoServicio DROP DEFAULT;
```

5.2.2.4.6 Añadir Restricciones: ADD CONSTRAINT

Permite añadir nueva restricción a una tabla

```
ALTER TABLE table_name
ADD CONSTRAINT NombreRestriccionDefiniciónRestricción
```

Veamos el siguiente ejemplo

```
USE TEST;
DROP TABLE IF EXISTS Factura;
CREATE TABLE IF NOT EXISTS `factura` (
`IdFactura` int(10) unsigned NOT NULL AUTO_INCREMENT,
`CodFactura` char(10) DEFAULT '00000/2016',
`Fecha` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
`Importe` decimal(12,2) DEFAULT '0.00',
`FormaPago` varchar(20) DEFAULT 'Contado',
`Transporte` varchar(20) NOT NULL,
`Cliente` char(5) DEFAULT '00000',
`TiempoServicio` tinyint(3) unsigned DEFAULT NULL,
`Estado` enum('Pedido','EnCurso','Entregado') DEFAULT NULL,
) ENGINE=InnoDB;

ALTER TABLE Factura
      ADD CONSTRAINT UN_CodFac Factura UNIQUE(CodFactura),
      ADD CONSTRAINT CH_Tiempo Factura CHECK(TiempoServicio> 20),
      ADD CONSTRAINT PK_Id Factura Factura PRIMARY KEY(IdFactura),
      ADD CONSTRAINT FK_Cliente Factura FOREIGN KEY(Cliente) REFERENCES
      CLIENTES(IdCliente) ON DELETE RESTRICT ON UPDATE
                  CASCADE;
```

5.2.2.4.7 Eliminar restricción: DROP CONSTRAINT

Sólo permite eliminar las restricciones que previamente han sido declaradas mediante CONSTRAINT

```
ALTER TABLE table_name
DROP Restriccion
```

Sobre las restricciones declaradas en los comandos anteriores veamos los siguientes ejemplos:

```
USE TEST;

ALTER TABLE Factura
      DROP CONSTRAINT UN_CodFacFactura,
      DROP CONSTRAINT PK_IdFacturaFactura,
      DROP CONSTRAINT FK_ClienteFactura;
```

5.2.2.5 Otras operaciones sobre las Tablas

5.2.2.5.1 Eliminar Tabla: DROP TABLE

Permite eliminar una Tabla de la base de datos activa. Una vez eliminada no puede volver a ser recuperada a no ser que tengamos un respaldo de seguridad de la Base de Datos.

DROP TABLE [IF EXISTS] NombreTabla [, NombreTabla]

Veamos el siguiente ejemplo:

```
USE TEST;

DROP TABLE IF EXISTS Factura;
DROP TABLE IF EXISTS Factura, Clientes, Proveedores, Albaranes;
```

5.2.2.5.2 Eliminar los Registros de una Tabla: TRUNCATE

La sentencia TRUNCATE elimina sin condiciones todos los registros de una Tabla.

Esta operación es peligrosa puesto que si se ejecuta erróneamente puede conllevar una pérdida de datos.

TRUNCATE TABLE table_name

Veamos algunos ejemplos

```
USE TEST;
TRUNCATE TABLE Factura;
```

En el siguiente tema veremos que existe una sentencia para el borrado selectivo de los registros de una tabla, dicha sentencia es DELETE. Con ella también podemos borrar todos los registros, pero a una velocidad de ejecución mucho más lenta que TRUNCATE.

5.2.2.5.3 Cambiar Nombre a una Tabla: RENAME

Permite cambiar el nombre a una tabla

RENAME TABLE *tbl_name* TO *new_tbl_name*

Veamos algunos ejemplos:

```
USE TEST;

RENAME TABLE Factura TO Facturacion;
RENAME TABLE Clientes TO Customers,
Factura TO Order,
Articulo TO Article;
```

5.2.3 Definición de Índices

Los índices son tablas de búsqueda especiales que utiliza el motor de búsqueda de la base de datos para agilizar el retorno de datos. También organizan la manera en la que la base de datos guarda los datos.

Los índices en una tabla son análogos a un catálogo en una biblioteca; si no se tiene un catálogo y hay que buscar un libro determinado en la Biblioteca Británica de Londres, con más de 150 millones de ejemplares, hay que buscar libro por libro y eso llevará bastante tiempo.

Sin embargo, con un catálogo, se busca el libro en el catálogo y se va directamente a la estantería indicada.

Sin un índice, la base de datos tiene que buscar en todos y cada uno de los registros de la tabla (este proceso se llama escaneo de tabla). Si tenemos un índice, la base de datos recorre el índice, encuentra donde están los datos y los va a buscar. Esta hace que el proceso sea mucho más rápido.

Un índice guarda un puntero al registro de la tabla

¿Cómo hace para encontrar los otros valores que están en ese mismo registro? Los índices de la base de datos almacenan punteros a los registros correspondientes de la tabla. Un puntero es una referencia al espacio en la memoria del disco donde están guardados los datos del registro correspondiente.

En el índice, por tanto guarda los valores de la columna de indexación así como un puntero al registro de la tabla donde están los demás valores de esa fila.

Imaginemos que nuestra tabla "LIbros" que tenemos a continuación, tiene miles de registros y que queremos encontrar el autor del libro "Android in 3 days". Podemos crear un índice con la columna "title". Este índice será como este;

idbooks	title	author	isbn	idpublisher
1	java for beginners	eli	2222	1
2	sql in 15 minutes	EDU	4444	2
3	android in 3 days	ELI	444	1
4	swing in 2 days	edu	333	0
5	spring	peter	4353	2
6	game development	rope	3563	2
NULL	NULL	NULL	NULL	NULL

INDEX	
idbooks	**Puntero**
3	0x82829
6	0x82840
1	0x82860
5	0x82880

2	0x82900
4	0x82820

Uno de los valores del índice para un "idbooks" podría ser algo como (“1), donde 0×82829 es la dirección en el disco (puntero), donde está la fila con los datos para “Android in 3 days”.

Uso de los índices

La utilización de los índices tiene un coste asociado. Los índices utilizan un espacio en el disco y harán que las operaciones INSERT, UPDATE, y DELETE vayan más lentas, ya que cada vez que una de estas operaciones se lleva a cabo, se actualizan los índices así como la base de datos.

5.2.3.1 Creación de Índice: CREATE INDEX

La creación de un índice se hace a través de una sentencia CREATE INDEX, que te permite nombrar al índice, especificar la tabla y la columnas o columnas para indexar.

```
CREATE INDEX NombreIndice
ON NombreTabla (NombreColumna [,NombreColumnas])
```

MySQL crea índices de forma automática para las siguientes restricciones:

- PRIMARY KEY: Crear un índice llamado PRIMARY cuyas columna o columnas de indexación se corresponden con la clave principal de la tabla.
- UNIQUE: Crea un índice cuya columna o columnas de indexación se corresponden con la columna o columnas que se han declarado como clave alternativa o secundaria. El nombre del índice se corresponde con el nombre de la columna.
- FOREIGN KEY: Crea un índice con el nombre de la columna sobre la que se ha aplicado este tipo de restricción

Veamos los siguientes ejemplos

```
USE TEST;
DROP TABLE IF EXISTS Matricula;
CREATE TABLE IF NOT EXISTS Matricula(
      NumMatriculaInt AUTO_INCREMENT PRIMARY KEY,
      FechaMatricula TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
      Apellidos Varchar(45),
      Nombre Varchar(20),
      DniAlumnoChar(9) UNIQUE,
      Email Varchar(50) CHECK (Email Like '%@%'),
      CiclEducativoVarchar(20),
      Curso TINYINT UNSIGNED REFERENCES Curso(Curso)
      ON DELETE RESTRICT ON UPDATE CASCADE,
)

-- Automáticamente se han creado los índices:
-- PRIMARY - de la restricción PRIMARY KEY
-- DniAlumno - de la restricción UNIQUE(DniAlumno)
-- Curso – procedente de la restricción FOREIGN KEY sobre Curso
--

--Además se crean los siguientes índices:

CREATE INDEX InNombreMatricula ON Matricula(Apellidos, Nombre);
CREATE INDEX InFechaMatricula ON Matricula(FechaMatricula);
```

5.2.3.2 Crear índice único: CREATE UNIQUE INDEX

Los índices también pueden ser "únicos". Este tipo de índices previene de la entrada de valores duplicados en la columna o combinación de columnas en donde existe índice. Ya sabemos que si en la creación o modificación de una columna asignamos la restricción UNIQUE dicho índice se crea automáticamente.

```
CREATE UNIQUE INDEX NombreIndice
ON NombreTabla (NombreColumna [,NombreColumnas])
```

Se pueden crear índices en cualquier momento; no hay que hacerlo justo después de ejecutar la sentencia CREATE TABLE. También se puede crear índices en las tablas que ya tienen datos. Lo que no se puede hacer, es crear un índice único en una tabla que ya contiene valores duplicados. MySQL lo identifica y no crea el índice. El usuario tiene que eliminar primero los valores duplicados antes de crearlo.

```
USE TEST;
DROP TABLE IF EXISTS Matricula;
CREATE TABLE IF NOT EXISTS Matricula(
      NumMatriculaInt AUTO_INCREMENT PRIMARY KEY,
      FechaMatricula TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
      Apellidos Varchar(40),
      Nombre Varchar(20),
      DniAlumno Char(9) UNIQUE,
      Email Varchar(50) CHECK (Email Like '%@%'),
      CiclEducativo Varchar(20),
      Curso TINYINT UNSIGNED,
)
--Además se crean los siguientes índices:

CREATE UNIQUE INDEX InEmailMatricula ON Matricula(Email);
```

5.2.3.3 Eliminar Índices: DROP INDEX

Permite eliminar índices de una tabla.

```
ALTER TABLE  NombreTabla
DROP INDEX ON NombreIndice [, Índices]
```

Veamos unos ejemplos

```
ALTER TABLE matricula DROP INDEX InNombreMatricula,
                      DROP INDEX InFechaMatricula;
```

5.2.4 Definición de Vistas

Como hemos observado, la arquitectura ANSI/SPARC distingue tres niveles, quese describen en el esquema conceptual, el esquema interno y los esquemas externos.

Hasta ahora, mientras creábamos las tablas de la base de datos, íbamos describiendo el esquema conceptual. Para describir los diferentes esquemas externos utilizamos el concepto de vista del SQL.

Para crear una vista es necesario utilizar la sentencia CREATE VIEW. Veamos su sintaxis:

```
CREATE VIEW NombreVista [(Columnas)]
AS SentenciaSELECT
[WITH CHECK POINT]
```

Lo primero que tenemos que hacer para crear una vista es decidir qué nombre le queremos poner. Si queremos cambiar el nombre de las columnas, o bien poner nombre a alguna que en principio no tenía, lo podemos hacer en Columnas. Y ya sólo nos quedará definir la consulta que formará nuestra vista.

Las vistas no existen realmente como un conjunto de valores almacenados en la base de datos, sino que son tablas ficticias, denominadas derivadas (no materializadas). Se construyen a partir de tablas reales (materializadas) almacenadas en la base de datos, y conocidas con el nombre de tablas básicas (otablas de base). La no-existencia real de las vistas hace que puedan ser actualizables o no.

Hay que tener cuidado con el uso excesivo de las vista ya que consumen demasiado procesador, pues cada vez que se soliciten tienen que ejecutar la sentencia SELECT que las define.

La definición de las vistas va ligada a la generación de consultas mediante la sentencia SELECT, capítulo que veremos en el siguiente tema.