

14

Creación de objetos en la base de datos

Introducción

Hasta el momento nos hemos dedicado a manipular tablas, es decir, a consultar datos de las tablas (SELECT), a insertar filas en ellas (INSERT), a eliminar filas (DELETE) y a modificar filas (UPDATE). Hemos llevado a cabo todas estas acciones con tablas que ya creábamos mediante scripts, pero no hemos llegado a crear ninguna tabla.

En esta unidad empezaremos a usar el lenguaje de descripción de datos o DDL (Data Description Language). Manejaremos las órdenes CREATE, DROP y ALTER. La orden CREATE sirve para crear objetos de base de datos: tablas, vistas, disparadores, etc.; DROP permite eliminar un objeto; y mediante la orden ALTER podemos modificar un objeto de base de datos.

Contenido

- 14.1. Definición de tablas
 - 14.2. Manipulación de vistas
 - 14.3. Triggers o disparadores
 - 14.4. El Editor de Tablas
- Ejercicios propuestos

Objetivos

- ▲ Manejar con fluidez las órdenes que permiten crear, modificar y suprimir tablas
- ▲ Crear y modificar tablas con restricciones
- ▲ Usar con fluidez las órdenes que posibilitan crear y suprimir vistas
- ▲ Manejar vistas
- ▲ Descubrir la importancia que tiene emplear restricciones al crear tablas
- ▲ Crear y manejar triggers de base de datos
- ▲ Utilizar manuales en línea para obtener información adicional

14.1 Definición de tablas

Antes de definir una tabla, es muy conveniente planificar ciertos aspectos:

- El nombre de la tabla. Debe ser un nombre que identifique su contenido. Por ejemplo, llamamos a una tabla ALUMNOS porque contendrá datos sobre alumnos.
- El nombre de cada columna de la tabla. Ha de ser un nombre autodescriptivo, que identifique su contenido. Por ejemplo, DNI, NOMBRE o APELLIDOS.
- El tipo de dato y el tamaño que tendrá cada columna.
- Las columnas obligatorias, los valores por defecto, las restricciones, etc.

14.1.1. Creación de una tabla

Para crear una tabla usamos la orden **CREATE TABLE**, cuyo formato más simple es el siguiente:

```
CREATE [TEMPORARY] TABLE NombreTabla
(
    Columna1 Tipo_dato [NOT NULL],
    Columna2 Tipo_dato [NOT NULL],
    .....
)
[ENGINE = Tipo_de_tabla];
```

Donde:

TEMPORARY indica que la tabla que se crea sólo existe durante la conexión actual. Columna1, Columna2 ... son los nombres de las columnas que contendrá cada fila de la tabla.

Tipo_dato indica el tipo de dato (VARCHAR, INT, CHAR, etc.) de cada columna. Los tipos de datos soportados por MySQL se vieron en el capítulo 10.

ENGINE = Tipo_de_tabla indica el tipo de almacenamiento para la tabla. NOT NULL indica que la columna debe contener alguna información; nunca puede ser nula.

MySQL permite varios tipos de almacenamiento. Por defecto, el almacenamiento es MyISAM que permite archivos de más de 4 gigabytes. Los datos se almacenan en un formato independiente, lo que permite pasar tablas entre distintas plataformas. Los índices se almacenan en un archivo con la extensión ".MYI" y los datos en otro archivo con extensión ".MYD". Ofrece la posibilidad de indexar campos BLOB y TEXT. Además, este tipo de tablas soportan el tipo de dato VARCHAR.

El tipo **InnoDB** proporciona soporte para trabajar con transacciones. Es el único formato que tiene MySQL para soportar claves ajena. Ofrece mejor rendimiento que el proporcionado por el tipo anterior.

Actividad de Aplicación 14.1

Abre MySQL Query Browser y crea un nuevo esquema de base de datos de nombre UNIDAD14 donde almacenaremos los objetos que creamos en este capítulo. Seleccionamos dicho esquema.

La orden **CREATE TABLE** permite definir distintos tipos de restricciones sobre una tabla: claves primarias, claves ajena, obligatoriedad, valores por defecto y verificación de condiciones. El objetivo de las restricciones es que las aplicaciones o los usuarios que van a manipular los datos tengan menos trabajo, y que sea MySQL el que realice la mayor parte de las tareas de mantenimiento de la integridad de la base de datos.

Para definir las restricciones en la orden **CREATE TABLE** usamos la cláusula **CONSTRAINT**. Una cláusula **CONSTRAINT** puede restringir una sola columna (restrictión de columna) o un grupo de columnas de una misma tabla (restrictión de tabla). Hay dos modos de especificar restricciones: como parte de la definición de columnas

Ejercicio Resuelto 14.1

Creamos una tabla llamada ALUMNOS con las siguientes columnas:

```
CREATE TABLE alumnos (
    numero_matricula INTEGER NOT NULL,
    nombre VARCHAR(15) NOT NULL,
    fecha_nacimiento DATE,
    direccion VARCHAR(30),
    localidad VARCHAR(15)
);
```

Observaciones:

- Esta sentencia crea una tabla de nombre ALUMNOS con cinco columnas, llamadas: NUMERO_MATRICULA, NOMBRE, FECHA_NACIMIENTO, DIRECCION y LOCALIDAD.
- Los tipos de datos para cada columna son: INTEGER, VARCHAR, DATE, VARCHAR y VARCHAR, respectivamente.
- La longitud de cada columna es: 15 para el NOMBRE, 30 para la DIRECCION y 15 para la LOCALIDAD. Para el tipo de datos DATE e INTEGER no se define longitud; MySQL la asigna automáticamente.
- En las columnas NUMERO_MATRICULA Y NOMBRE se ha definido la restricción NOT NULL, indicándose que siempre ha de darse algún valor al insertar una nueva fila.
- Dado que no se ha especificado el tipo de tabla, la tabla será de tipo MyISAM.
- Las definiciones individuales de columnas se separan mediante comas.
- No se pone coma después de la última definición de columna.
- Las mayúsculas y minúsculas son indiferentes a la hora de crear una tabla.
- La ejecución de la sentencia daría la siguiente salida: *Query returned no result.*

Si intentamos crear de nuevo la tabla, como ya existe, MySQL nos devolverá un mensaje de error. Véase la Figura 14.1.

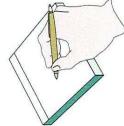
Figura 14.1. Mensajes de error tabla existente

14.1.1.1. Restricciones en CREATE TABLE

La orden **CREATE TABLE** permite definir distintos tipos de restricciones sobre una tabla: claves primarias, claves ajena, obligatoriedad, valores por defecto y verificación de condiciones. El objetivo de las restricciones es que las aplicaciones o los usuarios que van a manipular los datos tengan menos trabajo, y que sea MySQL el que realice la mayor parte de las tareas de mantenimiento de la integridad de la base de datos.

Para definir las restricciones en la orden **CREATE TABLE** usamos la cláusula **CONSTRAINT**. Una cláusula **CONSTRAINT** puede restringir una sola columna (restrictión de columna) o un grupo de columnas de una misma tabla (restrictión de tabla).

Hay dos modos de especificar restricciones: como parte de la definición de columnas



(*una restricción de columna*) o al final, una vez especificadas todas las columnas (*una restricción de tabla*).

A continuación aparece el formato de la orden CREATE TABLE donde algunas restricciones se pueden especificar en la definición de la columna y otras se especifican al final de la definición de todas las columnas:

```
CREATE TABLE nombre_tabla (
    Columna1 TIPO_DE DATO [NOT NULL] [DEFAULT valor] [AUTO_INCREMENT]
    [PRIMARY KEY] [UNIQUE] [CHECK (condición)],
    Columna2 TIPO_DE DATO [NOT NULL] [DEFAULT valor] [AUTO_INCREMENT]
    [PRIMARY KEY] [UNIQUE] [CHECK (condición)],
    ...
    [CONSTRAINT nombre_restricción] PRIMARY KEY (columnas),
    [CONSTRAINT nombre_restricción] UNIQUE (columnas),
    [CONSTRAINT nombre_restricción] FOREIGN KEY (columnas)
        REFERENCES Nombretabla (columnas) [Reglas],
    ...
    [CONSTRAINT nombre_restricción] CHECK (condición)
) [ENGINE = Tipo_de_tabla];
```

Por ejemplo, las siguientes órdenes CREATE TABLE crean una tabla de nombre PROVINCIAS, con dos columnas. La columna CODIGO es la clave primaria, y la columna NOMBRE tiene que tener obligatoriamente un valor, o lo que es lo mismo, no puede ser nula:

```
CREATE TABLE provincias (
    codigo TINYINT PRIMARY KEY,
    nombre VARCHAR(25) NOT NULL
);
```

A las restricciones de la orden CREATE TABLE que aparecen al final de la definición de las columnas se les puede asignar un nombre con la cláusula CONSTRAINT y pueden hacer referencia a varias columnas en una única restricción (por ejemplo, declarando dos columnas como clave primaria o ajena).

La orden anterior se puede escribir de varias formas. En una de ellas se da nombre a la restricción de clave primaria (PK); en la otra no se da nombre a la restricción:

```
CREATE TABLE provincias (
    codigo TINYINT,
    nombre VARCHAR(25) NOT NULL,
    CONSTRAINT PK PRIMARY KEY (codigo)
);
```

14.1.1.2. Obligatoriedad. La restricción NOT NULL

Esta restricción asociada a una columna significa que no puede tener valores nulos, es decir, que ha de tener obligatoriamente un valor. En caso contrario, causa un error. En ejemplos anteriores nos hemos ocupado de cómo se define una columna con la restricción NOT NULL.

14.1.1.3. Valores por defecto. La especificación DEFAULT

En el momento de crear una tabla podemos asignar valores por defecto a las columnas. Si especificamos la cláusula DEFAULT a una columna, le proporcionamos un valor

por omisión cuando el valor de la columna no se especifica en la cláusula INSERT. Un valor DEFAULT debe ser una constante; no puede ser una función o expresión. El siguiente ejemplo crea la tabla CIUDADES donde se especifica una columna con la cláusula DEFAULT:

```
CREATE TABLE ciudades (
    nombre VARCHAR(20),
    habitantes INT UNSIGNED,
    pais VARCHAR(20) DEFAULT "ESPAÑA"
);

Se insertan varias filas sin dar valor a la columna PAÍS:
```

```
INSERT INTO ciudades (nombre, habitantes)
VALUES ("GUADALAJARA", 85000);
INSERT INTO ciudades (nombre, habitantes)
VALUES ("TALAVERA DE LA REINA", 120000);
INSERT INTO ciudades (nombre, habitantes)
VALUES ("TOLEDO", 80000);
```

El contenido de la tabla se muestra en la Figura 14.2.

| nombre | habitantes | país |
|----------------------|------------|--------|
| GUADALAJARA | 85000 | ESPAÑA |
| TALAVERA DE LA REINA | 120000 | ESPAÑA |
| TOLEDO | 80000 | ESPAÑA |

Figura 14.2. Contenido de la tabla CIUDADES

14.1.1.4. Clave primaria. La restricción PRIMARY KEY

Una clave primaria dentro de una tabla es una columna o un conjunto de columnas que identifican únicamente a cada fila. Debe ser única, no nula y obligatoria. Como máximo, podemos definir una clave primaria por tabla. Esta clave se puede referenciar por una columna o columnas de otra tabla. Llamamos clave ajena a esta columna o columnas. Cuando se crea una clave primaria, automáticamente se crea un índice que facilita el acceso a la tabla, y se crea con la restricción NOT NULL. Para definir una clave primaria en una tabla usamos la restricción PRIMARY KEY. Ya se vio en los ejemplos anteriores como crear claves primarias.



Cuando se define una clave primaria mediante una restricción de tabla es necesario especificar en las columnas que forman parte de la clave primaria la restricción NOT NULL; de no hacerlo, no es obligatorio asignar valor a la clave primaria. MySQL asigna un valor por defecto (distinto de nulo) si no se da valor: 0 para los tipos numéricos y "" para las cadenas de caracteres. Para obligar a dar valores a la clave primaria es preciso indicar la restricción NOT NULL en la definición de la columna o columnas.

Actividad de Aplicación 14.2

Ejecuta las siguientes órdenes y analiza los resultados:

```
INSERT INTO provincias VALUES (1, "ALBAICETE");
INSERT INTO provincias (nombre) VALUES ("CÁCERES");
SELECT * FROM provincias;
INSERT INTO provincias (nombre) VALUES ("BADAJOZ");
```


| # | nombre |
|---|---------|
| 1 | GATO |
| 2 | FERRO |
| 3 | CABALLO |
| 4 | OCA |

Figura 14.5. Tabla ANIMALES

```
INSERT INTO animales (nombre) VALUES ("CABALLO");
INSERT INTO animales (nombre) VALUES ("OCA");
```

El contenido de la tabla se muestra en la Figura 14.5.

14.1.6. Claves ajena. La restricción FOREIGN KEY.

Una clave ajena está formada por una o varias columnas que están asociadas a una clave primaria de otra o de la misma tabla. Se pueden definir tantas claves ajena como sea preciso, y pueden estar o no en la misma tabla que la clave primaria. El valor de la columna o columnas que son claves ajena debe ser NULL o igual a un valor de la clave referenciada (regla de integridad referencial).

El siguiente ejemplo crea la tabla EDADES de dos maneras:

```
CREATE TABLE edades (
    nombre VARCHAR(25) PRIMARY KEY,
    edad TINYINT,
    cod_provincia TINYINT,
    FOREIGN KEY (cod_provincia)
        REFERENCES provincias (codigo)
        ON DELETE CASCADE
);
CREATE TABLE edades (
    nombre VARCHAR(25) PRIMARY KEY,
    edad TINYINT,
    cod_provincia TINYINT,
    CONSTRAINT FK FOREIGN KEY (cod_provincia)
        REFERENCES provincias (codigo)
        ON DELETE CASCADE
);
```

Se definen las siguientes restricciones:

Clave primaria: NOMBRE.

Clave ajena: COD_PROVINCIA, que referencia a la tabla PROVINCIAS creada anteriormente con la regla ON DELETE CASCADE.

Para definir claves ajena es necesario especificar la cláusula FOREIGN KEY; a la derecha se especifica la columna o columnas de la tabla que forman parte de la clave ajena. En la cláusula REFERENCES indicamos la tabla a la cual remite la clave ajena, y entre paréntesis se escribe el nombre de columna/s que es clave primaria en la tabla referenciada.

Las reglas que se pueden utilizar al definir claves ajena se muestran en la Tabla 14.1. En el ejemplo se ha utilizado ON DELETE CASCADE o borrado en cascada indicando que cuando se borren las filas con claves primarias, las filas con claves ajena que referencien a aquéllas, también se borrarán.

Reglas en las claves ajena:

| | |
|--|--|
| [ON DELETE CASCADE RESTRICT CASCADE SET NULL NO ACTION] | [ON UPDATE (RESTRICT CASCADE SET NULL NO ACTION)] |
| ON DELETE Regla de supresión. Determina la acción que se debe realizar cuando se elimina una fila padre (es decir, con clave primaria). | RESTRICT No permite eliminar o modificar la clave primaria si hay claves ajena referenciándola. Elimina o actualiza la clave primaria y automáticamente elimina o actualiza la clave ajena que la referencia. |
| ON UPDATE Regla de actualización. Determina la acción que se debe realizar cuando se actualiza una parte de la clave primaria de la fila padre. | SET NULL Elimina o actualiza la clave primaria y automáticamente asigna valor nulo a la columna o columnas que forman parte de la clave ajena, siempre y cuando no se hayan definido como NOT NULL. Es el valor por defecto. No se puede modificar o eliminar una clave primaria si es referenciada por una clave ajena. |
| NO ACTION | |

Tabla 14.1. Reglas para las claves ajena

Ejercicio Resuelto 14.3

Creamos las tablas ZONAS y PERSONAS. Las columnas de las tablas son las siguientes:

| Nombre columna | Tipo | Representa |
|--|--|--|
| TABLA ZONAS COD_ZONA NOMBRE | TINYINT VARCHAR(15) | Código de zona. Nombre de la zona. |
| TABLA PERSONAS DNI NOMBRE DIRECCION POBLACION CODZONA | VARCHAR(10) VARCHAR(30) VARCHAR(30) VARCHAR(30) | DNI de la persona. Nombre. Dirección. Población. Código de zona. |

- Donde:
- DNI es la clave primaria de la tabla PERSONAS.
 - COD_ZONA de la tabla ZONAS es clave primaria de esta tabla.
 - CODZONA de la tabla PERSONAS es clave ajena; se relaciona con la clave primaria de esta columna deben coincidir con la clave primaria de la tabla ZONAS. Podemos decir que ZONAS es la tabla maestra y PERSONAS es la tabla detalle.

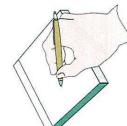
Hemos de crear, en primer lugar, la tabla ZONAS y, después, la tabla PERSONAS, ya que PERSONAS referencia a ZONAS. Si creamos primero la tabla PERSONAS y la tabla ZONAS no está creada, MySQL emitirá un mensaje de error. No se define ninguna regla para claves ajena. Se da un nombre a la restricción de clave ajena (FKPERSONAS):

```
CREATE TABLE personas (
    cod_zona TINYINT PRIMARY KEY,
    nombre VARCHAR(15),
    direccion VARCHAR(30),
    poblacion VARCHAR(30),
    codzona TINYINT NOT NULL,
    FOREIGN KEY (codzona)
        REFERENCES zonas (cod_zona)
        ON DELETE CASCADE
);
CREATE TABLE zonas (
    cod_zona TINYINT PRIMARY KEY,
    nombre VARCHAR(15),
    direccion VARCHAR(30),
    poblacion VARCHAR(30),
    codzona TINYINT NOT NULL,
    FOREIGN KEY (codzona)
        REFERENCES zonas (cod_zona)
        ON DELETE CASCADE
);
```

Actividad de Aplicación 14.4

Se han creado las tablas ZONAS y PERSONAS. Inserta filas en las tablas. Inserta filas en la tabla PERSONAS dando al código de zona un valor que no existe en la tabla ZONAS. ¿Qué ocurre? Analiza el resultado.





Ejercicio Resuelto 14.4

Partimos de una situación en que las dos tablas (ZONAS y PERSONAS) tienen datos. Vamos a borrar todas las filas de la tabla ZONAS:

```
DELETE FROM zonas;
```

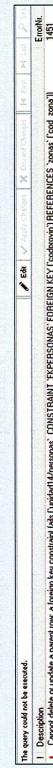


Figura 14.6. Error al borrar filas

Se produce un error, véase la Figura 14.6, debido a que no se pueden borrar filas en la tabla maestra o tabla padre (tabla ZONAS) si hay filas en la tabla detalle (PERSONAS) que las estén referenciando. Es decir, una fila con clave primaria no se puede borrar si es referenciada por alguna clave ajena. En el mensaje de error se puede ver el nombre de restricción que ha fallado: FKPERSONAS.

Si se añade la cláusula ON DELETE CASCADE en la opción REFERENCES de la tabla detalle (PERSONAS), se podrían eliminar las filas de la tabla maestra (ZONAS) y las filas correspondientes en la tabla detalle (PERSONAS) con esa zona. Borraremos la tabla PERSONAS desde el Navegador de Objetos y volvemos a crearla así:

```
CREATE TABLE personas (
    dni VARCHAR(10) PRIMARY KEY,
    nombre VARCHAR(30),
    direccion VARCHAR(30),
    poblacion VARCHAR(30),
    codzona TINYINT NOT NULL,
    CONSTRAINT FKPERSONAS FOREIGN KEY (codzona)
    REFERENCES zonas(id_zona) ON DELETE CASCADE
);
```

Una vez creada la tabla insertamos filas y borramos una fila de la tabla maestra (ZONAS). Automáticamente se borrarán las filas de la tabla detalle que se correspondan con las filas de la tabla maestra. Esta acción mantiene automáticamente la integridad referencial.

14.1.1.7. La restricción UNIQUE

La restricción UNIQUE evita valores repetidos en la misma columna. Puede contener una o varias columnas. Es similar a la restricción PRIMARY KEY, salvo que son posibles varias columnas UNIQUE definidas en una tabla. Admite valores NULL. Al igual que en PRIMARY KEY, cuando se define una restricción UNIQUE se crea un índice automáticamente.

El siguiente ejemplo crea la tabla UNICA definiendo una columna con la restricción UNIQUE de dos formas diferentes:

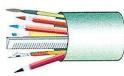
```
CREATE TABLE unica(
    dni VARCHAR(10) PRIMARY KEY,
    nom VARCHAR(30) UNIQUE,
    edad TINYINT,
    CONSTRAINT UNICA UNIQUE (nom)
);
```

Actividad de Aplicación 14.5

Ejecuta las siguientes órdenes y analiza los resultados:

```
INSERT INTO unica VALUES ('11111', 'PEPA', 20);
INSERT INTO unica VALUES ('11112', 'PEPA', 20);
```

Ejecuta la orden DESC unica; y observa la columna Key.



14.1.1.8. Verificación de condiciones. La restricción CHECK

Muchas columnas de tablas requieren valores limitados dentro de un rango o el cumplimiento de ciertas condiciones. Con una restricción de verificación de condiciones se puede expresar una condición que ha de cumplirse para todas y cada una de las filas de la tabla. La restricción CHECK actúa como una cláusula WHERE; puede controlar los valores que se colocan en una columna. En una cláusula CHECK no cabe incluir subconsultas.

Actualmente la cláusula CHECK no hace nada; MySQL la proporciona por compatibilidad, para simplificar la portabilidad de código desde otros servidores SQL y para arrancar aplicaciones que crean tablas con referencias.

El siguiente ejemplo crea la tabla CONDICIONES. Las columnas son: dni VARCHAR(10), nombre VARCHAR(30), edad TINYINT, curso TINYINT; y las restricciones siguientes:

- El DNI no puede ser nulo.
- La clave primaria es el DNI.
- La EDAD ha de estar comprendida entre 5 y 20 años.
- EL NOMBRE ha de estar en mayúsculas.
- El CURSO sólo puede almacenar 1, 2 o 3 y no puede ser nulo.

Las restricciones se pueden crear con nombre o sin nombre, o sin nombre o sin nombre; la descripción de la columna o al final de la descripción de todas las columnas:

```
CREATE TABLE condiciones (
    dni VARCHAR(10) NOT NULL,
    nombre VARCHAR(30),
    edad TINYINT NOT NULL,
    CURSO TINYINT NOT NULL,
    CONSTRAINT CLAVE_P PRIMARY KEY (dni),
    CONSTRAINT COMP_EDAD CHECK (edad BETWEEN 5 AND 20),
    CONSTRAINT NOMBRE_MAYUS CHECK (nombre = UPPER(nombre)),
    CONSTRAINT COMP_CURSO CHECK (curso IN(1, 2, 3))
);
```

También se puede crear de la siguiente manera:

```
CREATE TABLE condiciones (
    dni VARCHAR(10) NOT NULL PRIMARY KEY,
    nombre VARCHAR(30),
    edad TINYINT,
    curso TINYINT,
    CONSTRAINT COMP_EDAD CHECK (edad BETWEEN 5 AND 20),
    CONSTRAINT NOMBRE_MAYUS CHECK (nombre = UPPER(nombre)),
    CONSTRAINT COMP_CURSO CHECK (curso IN(1, 2, 3))
);
```

14.1.2. Creación de una tabla con datos recuperados en una consulta

La sentencia CREATE TABLE permite crear una tabla a partir de la consulta de otra tabla ya existente. La nueva tabla contendrá los datos obtenidos en la consulta. Se lleva a cabo esta acción con la cláusula AS colocada al final de la orden CREATE TABLE. El formato es el que sigue:

```
CREATE TABLE NombreTabla(
    Columna [, Columnas]
)
AS consulta;
```

No es necesario especificar tipos ni tamaño de las columnas, ya que vienen determinados por los tipos y los tamaños de las recuperadas en la consulta. La consulta puede contener una subconsulta, una combinación de tablas o cualquier sentencia SELECT válida. Las restricciones de claves primarias, ajenas o unicidad no se crean en una tabla desde la otra; sólo se crean aquellas restricciones que carecen de nombre (como, por ejemplo, NOT NULL).

El siguiente ejemplo crea la tabla EJEMPLO1 a partir de los datos de la tabla UNICA:

```
CREATE TABLE ejempl01 AS SELECT * FROM unica;
```

La tabla se crea con los mismos nombres de columnas e idéntico contenido de filas que la tabla UNICA.

En el siguiente ejemplo se crea una tabla con tres columnas (col1, col2 y col3), más las columnas de la tabla que se obtienen al hacer la SELECT. La descripción de la tabla se muestra en la Figura 14.7.

```
CREATE TABLE ejempl02
    (col1 VARCHAR(10), col2 VARCHAR(30), col3 TINYINT)
AS SELECT * FROM condiciones;
```

| Field | Type | Null | Key | Default |
|-------|-------------|------|-----|---------|
| col1 | varchar(10) | YES | | |
| col2 | varchar(30) | YES | | |
| col3 | tinyint(4) | YES | | |
| dbi | varchar(10) | NO | | |
| nombe | varchar(30) | YES | | |
| edad | tinyint(4) | YES | | |
| clase | tinyint(4) | NO | | 0 |

Figura 14.7. Tabla EJEMPLO2

14.1.3. Modificación de tablas

Se puede modificar una tabla mediante la orden ALTER TABLE. La modificación de tablas nos permitirá modificar, eliminar o añadir columnas a una tabla existente; añadir o eliminar restricciones; y renombrar una tabla. El formato es el siguiente:

```
ALTER TABLE nombreTabla
{
    [ADD [COLUMN] (columnas)]
    [MODIFY [COLUMN] declaracionColumna]
    [DROP [COLUMN] (columna)]
    [ADD PRIMARY KEY (columnas)]
```

```
[DROP PRIMARY KEY]
[ADD UNIQUE (nombre_index) (columnas)]
[DROP INDEX nombre_index]
[ADD [CONSTRAINT nombre_restriccion] FOREIGN KEY (columnas)
    REFERENCES definiciones]
[DROP FOREIGN KEY nombre_restriccion]
[ADD [CONSTRAINT nombre_restriccion] CHECK(condición)]
[RENAME (TQ) nombre_table_nuevo]
```

Ejercicio Resuelto 14.5

Partimos de la tabla EJEMPLO2. Añadimos dos columnas: SEXO con la restricción NOT NULL e IMPORTE. Las columnas se añaden al final de las que ya hay creadas:

```
ALTER TABLE ejempl02 ADD (sexo CHAR(1) NOT NULL, importe TINYINT);
1. Modificamos las columnas SEXO e IMPORTE de la tabla EJEMPLO2. El sexo lo definimos como VARCHAR(12) y el IMPORTE como INTEGER:
```

```
ALTER TABLE ejempl02 MODIFY sexo VARCHAR(12), importe INTEGER;
2. Se añade a la columna SEXO una restricción para que sólo pueda almacenar los valores "HOMBRE" o "MUJER":
```

```
ALTER TABLE ejempl02 ADD CHECK(sexo IN ("HOMBRE", "MUJER"));
3. Eliminamos las columnas SEXO e IMPORTE de la tabla EJEMPLO2:
```

```
ALTER TABLE ejempl02 DROP COLUMN sexo, DROP COLUMN importe;
```

```
4. Se añade la restricción de clave primaria a la columna DNI:
```

```
ALTER TABLE ejempl02 ADD PRIMARY KEY (dni);
5. Se borra la restricción de clave primaria:
```

```
ALTER TABLE ejempl02 DROP PRIMARY KEY;
```

```
6. Se añade la restricción UNIQUE con nombre INDICE a la columna NOM de la tabla EJEMPLO2, se añade la columna CODIG de tipo TINYINT y se añade la restricción de clave ajena de nombre CLAVE_AJENA a dicha columna que referencia a la tabla PROVINCIAS. Todas las modificaciones se pueden poner en una orden ALTER TABLE:
```

```
ALTER TABLE ejempl02 ADD UNIQUE indice (nom),
    ADD (codig TINYINT),
    ADD CONSTRAINT clave_ajena
        FOREIGN KEY (codig)
        REFERENCES provincias (codigo);
7. Se borra la restricción UNIQUE de nombre INDICE y la restricción de clave ajena de nombre CLAVE_AJENA:
```

```
ALTER TABLE ejempl02 DROP INDEX indice,
    DROP FOREIGN KEY clave_ajena;
8. Cambiamos el nombre de tabla EJEMPLO2. El nuevo nombre es TABLANUEVA:
```

```
ALTER TABLE ejempl02 RENAME tablanueva;
```

14.1.4. Borrado de tablas

La orden **DROP TABLE** suprime una o varias tablas de la base de datos. Al suprimir una tabla también se suprime los índices y los privilegios asociados a ella. Las vistas creadas a partir de esta tabla dejan de funcionar, pero siguen existiendo en la base de datos, por lo que habrá que eliminarlas. El formato de la orden **DROP TABLE** es:

```
DROP TABLE tabla1, tabla2, ...;
```

No se puede borrar una tabla que sea referenciada por claves ajena; por ejemplo, si intentamos borrar la tabla PROVINCIAS, MySQL nos devolverá un mensaje de error. Véase la Figura 14.8.



Figura 14.8. Mensaje de error al borrar un tabla

La siguiente orden borra las tablas UNICA y TABLANUEVA:

```
DROP TABLE unica, tablanueva;
```

14.2 Manipulación de vistas

A veces, para obtener datos de varias tablas hemos de construir una consulta compleja, y si en otro momento necesitamos realizar esa misma consulta, tenemos que construir de nuevo la sentencia **SELECT**. Sería muy cómodo obtener los datos de una consulta compleja con una simple sentencia **SELECT**.

Pues bien, las vistas solucionan este problema: mediante una consulta simple de una vista cabe la posibilidad de obtener datos de una consulta compleja. Una vista es una tabla lógica que permite acceder a la información de una o de varias tablas. No contiene información por sí misma, sino que su información está basada en la que contienen otras tablas, llamadas tablas base, y siempre refleja los datos de estas tablas; es, simplemente, una sentencia SQL. Si se suprime una tabla, la vista asociada se invalida. Las vistas tienen la misma estructura que una tabla: filas y columnas, y se tratan de forma semejante a una tabla. El formato básico para crear una vista es:

```
CREATE [OR REPLACE] VIEW NombreVista [ (columna [, columna]) ]  
AS consulta;
```

NombreVista es el nombre que damos a la vista.

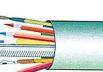
[(columna [, columna])] son los nombres de columnas que va a contener la vista. Si no se ponen, se asumen los nombres de columna devueltos por la consulta. AS consulta determina las columnas y las tablas que aparecerán en la vista. [OR REPLACE] crea de nuevo la vista si ya existía.

Actividad de Aplicación 14.6

Carga el script TABLAS_UNIDAD14.SQL que está en la página web del libro en el Editor de Scripts y ejecútalo.

14.2.2. Creación y uso de vistas complejas

Las vistas complejas se definen sobre más de una tabla, agrupan filas usando las cláusulas GROUP BY o DISTINCT, y contienen llamadas a funciones. Se pueden crear vistas usando funciones, expresiones en columnas y consultas avanzadas, pero únicamente se podrán consultar estas vistas.



Actividad de Aplicación 14.7

Crear una vista que contenga los datos de los empleados del departamento 10 con salario >1200. Después realiza operaciones INSERT, UPDATE y DELETE sobre la vista.



14.2.1. Creación y uso de vistas sencillas

Las vistas más sencillas son las que se crean a partir de una única tabla. Por ejemplo creamos la vista DEP30 que contiene el APELLIDO, el OFICIO y el SALARIO de los empleados de la tabla EMPLEADOS del departamento 30:

```
CREATE VIEW dep30  
AS SELECT apellido, oficio, salario FROM empleados  
WHERE dept_no=30;
```

Ahora la vista creada se puede usar como si se tratase de una tabla. Se puede consultar, se pueden borrar filas, actualizar filas siempre y cuando las columnas a actualizar no sean expresiones; y se puede insertar siempre y cuando **todas** las columnas obligatorias de la tabla asociada estén presentes en la vista. Todas las operaciones que se hagan en la vista repercuttirán sobre la tabla a partir de la cual se definió.

También podríamos haberla creado dando nombre a las columnas, por ejemplo, APE, OFI y SAL:

```
CREATE OR REPLACE VIEW dep30 (ape, ofi, sal)  
AS SELECT apellido, oficio, salario FROM empleados  
WHERE dept_no=30;
```

La siguiente inserción en la vista dará error ya que no se da valor a las columnas definidas como NOT NULL (EMP_NO y DEPT_NO):

```
INSERT INTO dep30 VALUES ('PEREZ', 'EMPLEADO', 1300);
```

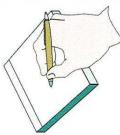
La siguiente modificación en la vista hará que se modifiquen los salarios de los empleados del departamento 30 de la tabla EMPLEADOS:

```
UPDATE dep30 SET sal = sal + 100;
```

La siguiente orden DELETE borraría todas las filas de la tabla EMPLEADOS del departamento 30:

```
DELETE FROM dep30;
```

Actividad de Aplicación 14.7



Ejercicio Resuelto 14.6

A partir de las tablas EMPLEADOS y DEPARTAMENTOS creamos una vista que contiene las columnas EMP_NO, APELLIDO, DEPT_NO y DNIOMBRE:

```
CREATE VIEW EMP_DEPT (emp_no, apellido, dept_no, dniombre) AS
SELECT emp_no, apellido, e.dept_no, dniombre
FROM empleados e, departamentos d
WHERE e.dept_no = d.dept_no;
```

1. Insertamos una fila en la vista:

```
INSERT INTO emp_dept VALUES (2222, 'SUELA', 20, 'INVESTIGACIÓN');
```

Pero se produce un error debido a que la vista se creó a partir de dos tablas, véase la Figura 14.9. Los borrados y las modificaciones también producirán errores.

| The query could not be executed. | Edits | Apply Changes | Discard Changes | Last | Search |
|---|-------|---------------|-----------------|------|--------|
| ! Description | | | | | |
| Can not insert into [on view 'Unidad14.emp_dept' without fields test] | | | | | |

Figura 14.9. Ejercicio Resuelto 14.6. Ejemplo1

2. Creamos una vista llamada PAGOS a partir de las filas de la tabla EMPLEADOS, cuyo departamento sea el 10. Las columnas de la vista se llamarán NOMBRE, SAL_MES, SAL_AN y DEPT_NO. El NOMBRE es la columna APELLIDO, SAL_MES la columna SALARIO*12; SAL_AN es la columna SALARIO. SAL_AN es la columna SALARIO*12;

```
CREATE VIEW pagos (nombre, sal_mes, sal_an, dept_no)
AS SELECT LOWER(apellido), salario, salario*12, dept_no
FROM empleados WHERE dept_no = 10;
```

3. Podemos modificar filas en la vista PAGOS siempre y cuando la columna que se va a modificar no sea la columna expresada en forma de cálculo (SAL_AN) o la que fue creada mediante la función LOWER();

```
UPDATE pagos SET sal_mes = 5000 WHERE nombre = 'muñoz';
```



Actividad de Aplicación 14.8

Crea la vista VMEDIA a partir de las tablas EMPLEADOS y DEPARTAMENTOS. Esta vista contendrá por cada departamento el número de departamento, el nombre, la media de salario y el máximo salario. Prueba a hacer inserciones, modificaciones y borrados en la vista.

14.2.3. Borrado de vistas

Es posible borrar las vistas con la orden DROP VIEW, cuyo formato es:

```
DROP VIEW nombrevista;
```

Por ejemplo, la siguiente orden borra la vista PAGOS:

```
DROP VIEW pagos;
```

14.3 Triggers o disparadores

Un trigger es un objeto de base de datos que se asocia con una tabla y se activa cuando ocurre un evento sobre dicha tabla. Se puede activar antes o después de que ocurra el evento sobre la tabla, es decir, antes o después de una inserción, un borrado o una actualización. Veámos cómo funcionan los triggers con un ejemplo.

Ejercicio Resuelto 14.7

El ejemplo mostrado en la Figura 14.10 crea dos tablas y un trigger asociado a una de las dos tablas.

```
SQL Query Area
1 CREATE TABLE test1(campo1 VARCHAR(30));
2 CREATE TABLE test2(campo2 VARCHAR(30), fecha DATE, hora TIME);
3
4 DELIMITER |
5 CREATE TRIGGER tri1 BEFORE INSERT ON test1
6 FOR EACH ROW BEGIN
7   INSERT INTO test2 VALUES (NEW.campo1, CURDATE(), CURTIME());
8 END;
9
10 DELIMITER ;
11
12 INSERT INTO test1 VALUES ('PRIMERA FILA');
13 INSERT INTO test1 VALUES ('SEGUNDA FILA');
14 SELECT * FROM test2;
```

Figura 14.10. Creación de un trigger

La tabla TEST1 contiene una única columna. La tabla TEST2 contiene tres columnas donde se almacenará el valor que se inserta en CAMPO1 y la fecha y hora en que se inserta una fila en la tabla TEST1. La inserción de datos en esta tabla se realiza en el trigger.

Al crear un trigger se escriben varias líneas de código, por lo que es necesario especificar la orden **DELIMITER** al principio para indicar cuál será el carácter utilizado para delimitar la última línea del trigger. En el ejemplo se usa la barra () al final de la última línea de código del trigger. Después de crear el trigger se vuelve a escribir la orden **DELIMITER** con el punto y coma al final indicando que el final de la orden SQL será el punto y coma a partir de ahora.

El disparador TRIG1 insertará una fila en la tabla TEST2 antes de insertar una fila en la tabla TEST1 (BEFORE INSERT ON test1), con la fecha y la hora de la inserción. Al visualizar el contenido de la tabla TEST2 se verá el valor insertado en TEST1 y la fecha y hora.