

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 creamos 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.

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.*

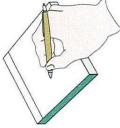
Si intentamos crear de nuevo la tabla, como ya existe, MySQL nos devolverá un mensaje de error. Véase la Figura 14.1.

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

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.

14.1.2. Obligatoriedad. La restricción NOT NULL

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

14.1.4. 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. Esta clave se puede referenciar a otra tabla. Cuando se define una clave primaria 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 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 (nombre) VALUES ('ALBAEITE');
INSERT INTO provincias (nombre) VALUES ('CÁRERES');
SELECT * FROM provincias;
INSERT INTO provincias (nombre) VALUES ('BADAJOZ');
```

14.1.3. 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

Ejercicio Resuelto 14.2

Creamos la tabla BLOQUESPIOS. Las columnas son las siguientes:

Nombre columna	Representa	Tipo
CALLE	Calle donde está el bloque.	VARCHAR(30)
NUMERO	Número donde está el bloque.	TINYINT
PISO	Número de planta.	CHAR(1)
PUERTA	Puerta.	TINYINT
CODIGO_POSTAL	Código postal.	VARCHAR(60)
METROS	Metros de la vivienda.	TINYINT
COMENTARIOS	Otros datos de la vivienda.	VARCHAR(60)
COD_ZONA	Código de zona donde está el bloque.	VARCHAR(10)
DNI	DNI del propietario.	VARCHAR(10)

La clave primaria estará formada por las columnas CALLE, NUMERO, PISO y PUERTA, y es necesario asignarle un valor (por lo que se definirá la restricción NOT NULL). Como son cuatro columnas, definiremos la clave primaria al final de la definición de todas las columnas:

```
CREATE TABLE bloquespisos (
    calle VARCHAR(30) NOT NULL,
    numero TINYINT NOT NULL,
    piso TINYINT NOT NULL,
    puerta CHAR(1) NOT NULL,
    codigo_postal TINYINT,
    metros TINYINT,
    comentarios VARCHAR(60),
    cod_zona VARCHAR(10),
    dni VARCHAR(10),
    CONSTRAINT PK_VIV PRIMARY KEY (calle, numero, piso, puerta)
);
```

La última sentencia se podría haber puesto de la siguiente manera: `PRIMARY KEY (calle, numero, piso, puerta)`, sin dar nombre a la restricción. Cuando en la orden CREATE TABLE aparece la cláusula PRIMARY KEY sólo se debe especificar una vez. Al visualizar la descripción de la tabla comprobamos que estas cuatro columnas tienen la restricción NOT NULL (la columna Null es NO). Véase la Figura 14.3.

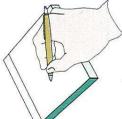


Figura 14.3. Descripción de la tabla BLOQUESPIOS

Field	Type	Null	Key	Default	Extra
calle	varchar(30)	NO	PRI		
numero	tinyint(4)	NO	PRI	0	
piso	tinyint(4)	NO	PRI	0	
puerta	char(1)	NO	PRI		
codigo_postal	tinyint(4)	YES			
metros	tinyint(4)	YES			
comentarios	varchar(60)	YES			
cod_zona	tinyint(4)	YES			
dni	varchar(10)	YES			

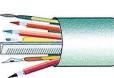
Figura 14.3. Descripción de la tabla BLOQUESPIOS

La orden SHOW TABLES; nos muestra las tablas creadas en el esquema actual. Véase la Figura 14.4.



Figura 14.4. Orden SHOW TABLES

Tables in_unidad14
alumnos
bloquespisos
provincias



Actividad de Aplicación 14.3

Borra la tabla PROVINCIAS desde el Navegador de Objetos y créala de nuevo de tal forma que tengamos que dar siempre un valor a la clave primaria. Si no la das en el esquema de la base de datos haz clic con el botón derecho del ratón en el esquema de base de datos y clíck en la opción Refresh del menú contextual.

14.1.15. El atributo AUTO_INCREMENT

Se utiliza para columnas con valores enteros y definidas como claves primarias. Solo puede haber una columna AUTO_INCREMENT para cada tabla.

Cuando se inserta un valor NULL o un 0 en una columna AUTO_INCREMENT, la columna tomará el valor mayor actual para la columna +1. Las secuencias de AUTO_INCREMENT se inicien en 1. Si eliminamos la fila que contiene el valor máximo para la columna AUTO_INCREMENT, el valor se reutilizará con una tabla ISAM o BDB, pero no con una MySQL o InnoDB. Si se borran todas las filas, la secuencia no se reinicia.

El siguiente ejemplo crea una tabla con una columna AUTO_INCREMENT y clave primaria:

```
CREATE TABLE animales (
    id TINYINT NOT NULL AUTO_INCREMENT,
    nombre CHAR(30) NOT NULL,
    PRIMARY KEY (id)
);
```

Se insertan varias filas:

```
INSERT INTO animales (nombre) VALUES ("GATO");
INSERT INTO animales (nombre) VALUES ("PERRO");
```

#	nombre
1	GATO
2	PERRO
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.
```

Ejercicio Resuelto 14.3

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

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

```
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
);

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
);
```

El siguiente ejemplo crea la tabla EDADES de dos maneras:

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 ajenas 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 ajenas 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:

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 ajenas 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 ajenas 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 ajenas que

referencien a aquéllas, también se borrarán.

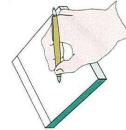
Clave primaria: NOMBRE.

Clave ajena: COD_ZONA, que referencia a 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 ajenas. Se da un nombre a la restricción de clave ajena (FKPERSONAS).

```
CREATE TABLE personas (
    dni VARCHAR(10) PRIMARY KEY,
    nombre VARCHAR(15),
    direccion VARCHAR(30),
    poblacion VARCHAR(30),
    codzona TINYINT NOT NULL,
    CONSTRAINT FKPERSONAS
        FOREIGN KEY (codzona)
            REFERENCES zonas (cod_zona)
);
```

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 exista en la tabla ZONAS. ¿Qué ocurre? Analiza el resultado.



Reglas en las claves ajenas:		
[ON DELETE {RESTRICT CASCADE SET NULL NO ACTION}]	[ON UPDATE {RESTRICT CASCADE SET NULL NO ACTION}]	No permite eliminar o modificar la clave primaria si hay claves ajenas referenciándola.
ON DELETE	RESTRICT	Elimina o actualiza la clave primaria y automáticamente elimina o actualiza la clave ajena que la referencia.
Regla de supresión. Determina la acción que se debe realizar cuando se elimina una fila padre (es decir, con clave primaria).	CASCADE	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.
ON UPDATE	SET NULL	Es el valor por defecto. No se puede modificar o eliminar una clave primaria si es referenciada por una clave ajena.
Regla de actualización. Determina la acción que se debe realizar cuando se actualiza una parte de la clave primaria de la fila padre.	NO ACTION	

Tabla 14.1. Reglas para las claves ajenas

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;
```

The query will be executed.

Description:
1 Caracteres o código a partir now, la foreign key constraint das 'unidades' apunta a 'zonas'.
1 Constrains FOREIGN KEY('codzonas') REFERENCES 'zonas' ('cod_zona')

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 podrán 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(cod_zona) ON DELETE CASCADE
);
```

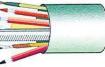
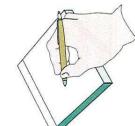
Una vez creada la tabla insertaremos filas y borraremos una fila de la tabla maestra (ZONAS). Automáticamente se borrarán las filas de la tabla detalle que se correspondan con las filas de la tabla maestra. Esta acción mantiene automáticamente la integridad referencial.

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:

<pre>CREATE TABLE unica(dni VARCHAR(10) PRIMARY KEY, nom VARCHAR(30) UNIQUE, edad TINYINT);</pre>	<pre>CREATE TABLE unica(dni VARCHAR(10) NOT NULL PRIMARY KEY, nombre VARCHAR(30) CHECK (nombre = UPPER(nombre)), edad TINYINT, curso TINYINT CONSTRAINT UNICA UNIQUE (nom));</pre>
---	--



Actividad de Aplicación 14.5

Ejecuta las siguientes órdenes y analiza los resultados:

```
INSERT INTO unica VALUES ('111111', 'PEPA', 20);
INSERT INTO unica VALUES ('111122', '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, formando parte de 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,
    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) CHECK (nombre = UPPER(nombre)),
    edad TINYINT,
    curso TINYINT CONSTRAINT UNICA UNIQUE (nom)
);
```

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 viene 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	NO	
col2	varchar(30)	YES	NO	
col3	tinyint(4)	YES	NO	
dni	varchar(10)	NO	NO	
nombre	varchar(30)	YES	NO	
edad	tinyint(4)	YES	NO	
curso	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] declaracióncolumna]
    [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 [TO] nombre_tabla_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 suprimen 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 ajenas; 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 una 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.

14.2.1. 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 repercutirán 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 ('PÉREZ', '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:

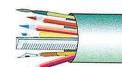
```
DELETE FROM dep30;
```

Actividad de Aplicación 14.7

Crea 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.

14.2.2. 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.



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 DNAME:

```
CREATE VIEW EMP_DEPT (emp_no, apellido, dept_no, dname) AS
SELECT emp_no, apellido, e.dept_no, dname
FROM empleados e, departmentos 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.	Edit	Apply Changes	Discard Changes	File	User	Search
! Description ! Can not insert into [join view] 'vincula14.emp_dept' without fields 'dept_no'						

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, a la que aplicamos la función LOWER(). SAL_MES 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;
```

Ejercicio Resuelto 14.7

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 QueryArea
1 CREATE TABLE test1(campo1 VARCHAR(30));
2 CREATE TABLE test2(campo2 VARCHAR(30), fecha DATE, hora TIME);
3 DELIMITER |
4 CREATE TRIGGER trig1 BEFORE INSERT ON test1
5 FOR EACH ROW BEGIN
6     INSERT INTO test2 VALUES (NEW.campo1, CURDATE(), CURTIME());
7 END;
8 DELIMITER ;
9 |
10 DELIMITER ;
11 INSERT INTO test1 VALUES ('PRIMERA FILA');
12 INSERT INTO test2 VALUES ('PRIMERA FILA');
13 INSERT INTO test1 VALUES ('SEGUNDA FILA');
14 SELECT * FROM test2;
15
```

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.

14.3.1. Manejo de Triggers

El formato básico para crear un trigger es el siguiente:

```
CREATE TRIGGER nombre_trigger
  {BEFORE | AFTER} {INSERT | UPDATE | DELETE}
  ON nombre_tabla
  FOR EACH ROW BEGIN
    Sentencias;
  END;
```

Dónde:

BEFORE | AFTER indican cuándo se activará el trigger. Con BEFORE el trigger se activa antes de ejecutar la sentencia. Con AFTER el trigger se activa después de ejecutar la sentencia.

INSERT | UPDATE | DELETE indican el evento que activará el trigger. Si se escribe INSERT, el trigger se activa cuando se inserte una nueva fila en la tabla. Con UPDATE el trigger se activará cuando se modifique una fila y con DELETE cuando se elimine una fila. Para una misma tabla no puede haber dos triggers con el mismo evento asociado.

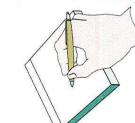
Sentencias _ tabla indica el nombre de la tabla sobre la que actúa el trigger. Para acceder a los valores de las columnas de la tabla sobre las que actúa el trigger se utilizan los valores NEW y OLD.

- En un trigger INSERT sólo se pueden usar los valores NEW.columna.
- En un trigger DELETE sólo se pueden usar los valores OLD.columna.
- En un trigger UPDATE se puede acceder a los valores NEW.columna que son los nuevos valores que se dan a las columnas y OLD.columna que son los antiguos valores de las columnas antes de la actualización.

Ejercicio Resuelto 14.8

Creamos la tabla AUDITATEST1. En esta tabla se irán almacenando los valores antiguos y nuevos de la columna CAMPO1 de la tabla TEST1, dependiendo del tipo de operación que se haga sobre la tabla, la operación (INSERCIÓN, MODIFICACIÓN o BORRADO), la fecha y la hora de la operación. Las columnas de esta tabla son las siguientes:

```
CREATE TABLE auditatest1 (
  antiguovalor VARCHAR(30),
  nuevovalor VARCHAR(30),
  operacion VARCHAR(30),
  fecha DATE,
  hora TIME
);
```



```
DELIMITER |
CREATE TRIGGER trig2 AFTER UPDATE ON test1
FOR EACH ROW BEGIN
  INSERT INTO auditatest1 VALUES
  (OLD.campo1, NEW.campo1, "MODIFICACIÓN", CURDATE(), CURTIME());
END;
```

Después se insertan varias filas en la tabla TEST1 y se visualiza el contenido de la tabla AUDITATEST1. Véase la Figura 14.11.

SQL Query Area			
6 hora TIME			
7;			
8 DELIMITER			
9 CREATE TRIGGER trig2 AFTER UPDATE ON test1			
10 FOR EACH ROW BEGIN			
11 INSERT INTO auditatest1			
12 VALUES (OLD.campo1, NEW.campo1, "MODIFICACIÓN", CURDATE(), CURTIME());			
13 END;			
14			
15 DELIMITER :			
16 UPDATE test1 SET campo1 = "Fila 1" WHERE campo1 = "PRIMERA FILA";			
17 UPDATE test1 SET campo1 = "Fila 2" WHERE campo1 = "SEGUNDA FILA";			
18 UPDATE test1 SET campo1 = "Fila 3" WHERE campo1 = "TERCERA FILA";			
19 SELECT * FROM auditatest1;			

Figura 14.11 Creación del trigger TRIG2

2. Creamos un trigger asociado a la operación de borrado sobre la tabla TEST1; se activará después de hacer un borrado sobre la tabla TEST1 (AFTER DELETE ON test1). Por cada operación DELETE sobre la tabla TEST1 se insertará una fila en la tabla AUDITATEST1 con el antiguo valor del CAMPO1 antes del borrado; (como se trata de borrado, no hay nuevo valor, por lo que damos a esta columna el valor NULL), la operación, en este caso BORRADO, la fecha y la hora de la operación. Eliminaremos una fila y visualizaremos el contenido de la tabla AUDITATEST1. Véase la Figura 14.12.

```
DELIMITER |
CREATE TRIGGER trig3 AFTER DELETE ON test1
FOR EACH ROW BEGIN
  INSERT INTO auditatest1 VALUES
  (OLD.campo1, NULL, "BORRADO", CURDATE(), CURTIME());
END;
```

|

antiguovalor	nuevovalor	operación	fecha	hora
PRIMERA FILA	Fila 1	MODIFICACIÓN	2006-06-22	00:12:41
SEGUNDA FILA	Fila 2	MODIFICACIÓN	2006-06-22	00:12:43
Fila 1		BORRADO	2006-06-22	00:16:49

Figura 14.12 Contenido de la tabla AUDITATEST1

5. Hacemos clic en el botón *Apply Changes*. Se visualiza un cuadro de diálogo donde aparece la orden que se ha generado al modificar la tabla. Se hace clic en el botón *Execute* para aplicar los cambios sobre la tabla. Y después clic en el botón *Close* para cerrar el Editor de Tablas.

Desde el Editor de Tablas editamos la tabla EMPLEADOS. Hacemos clic en el icono que acompaña a la columna que queremos que sea clave primaria, en este caso la columna EMP_NO, para que cambie y muestre una pequeña llave.

Para añadir restricción de clave ajena a la columna DEPT_NO hacemos lo siguiente:

6. Desde la pestaña *Foreign Keys* hacemos clic en el signo + (véase la Figura 14.18). Se abre un cuadro de diálogo desde el que damos nombre a la clave ajena; por ejemplo *FK_empleados* (véase la Figura 14.18). Hacemos clic en el botón OK.

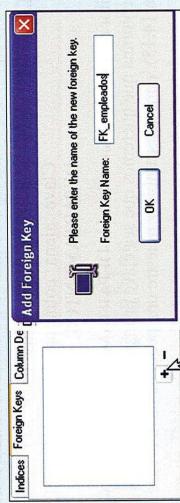


Figura 14.18. Ejercicio Resuelto 14.9, Paso 6

7. Desde la lista *Ref Table* elegimos la tabla a la que va a referenciar la clave ajena, en este caso la tabla DEPARTAMENTOS. Véase la Figura 14.19. Desde el área *Foreign Key Settings* se pueden elegir las reglas de claves ajenas. En el ejemplo se dejan las opciones por defecto.

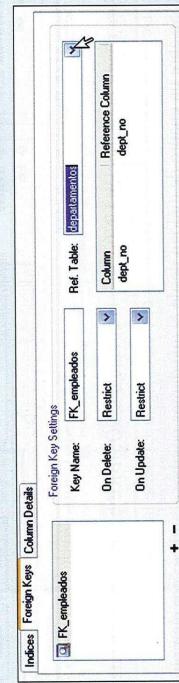


Figura 14.19. Ejercicio Resuelto 14.9, Paso 7

8. Hacemos clic en el botón *Apply Changes*. Se visualiza un cuadro de diálogo donde aparece la orden que se ha generado al modificar la tabla, véase la Figura 14.20. Se hace clic en el botón *Execute* para aplicar los cambios sobre la tabla. Y después clic en el botón *Close* para cerrar el Editor de Tablas.

- 14.3 Añade a la tabla FABRICANTES una nueva columna de tipo VARCHAR(10) para que almacene la clase de fabricante, y la restricción UNIQUE a la columna NOMBRE.

- 14.4 Modifica las columnas PRECIO_VENTA y PRECIO_COSTO de la tabla ARTICULOS a tipo de dato FLOAT(7,2).

- 14.5 Añade la restricción de clave ajena a la columna COD_FABRICANTE de la tabla ARTICULOS para que refiera a la tabla FABRICANTES. Realiza un borrado en cascada.



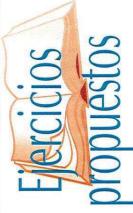
Direcciones Web de interés:

Zona de descarga de documentación sobre MySQL:
<http://dev.mysql.com/doc/>

Manual de referencia de MySQL en castellano:
<http://dev.mysql.com/doc/refman/5.0/es/index.html>

Fuente de consulta y referencia para el aprendizaje de MySQL: artículos, noticias, eventos, etc.
<http://www.mysql Hispano.org/>

Tutorial básico de MySQL:
http://www.programacion.com/bbdd/tutorial/mysql_basico/



14.1 Crea la tabla FABRICANTES con las siguientes columnas y restricciones:

cod_fabricante	TINYINT (3)
nombre	VARCHAR (15)
país	VARCHAR (15)

- La clave primaria es COD_FABRICANTE. No puede ser nula. Se define el atributo AUTO_INCREMENT.
- Las columnas NOMBRE y PAÍS han de almacenarse en mayúscula.
- Valor por defecto para el país "ESPAÑA".

14.2 Crea la tabla ARTICULOS con las siguientes columnas y restricciones:

articulo	VARCHAR (20)
cod_fabricante	TINYINT (3)
peso	TINYINT (3)
categoria	VARCHAR (10)
precio_venta	FLOAT (6, 2)
costo	FLOAT (6, 2)
existencias	INT

- La clave primaria está formada por las columnas ARTICULO, COD_FABRICANTE y CATEGORIA. No pueden tener valor nulo.
- PRECIO_VENTA, PRECIO_COSTO y PESO han de ser > 0.
- CATEGORIA ha de ser 'Primera', 'Segunda' o 'Tercera'.

- 14.3 Añade a la tabla FABRICANTES una nueva columna de tipo VARCHAR(10) para que almacene la clase de fabricante, y la restricción UNIQUE a la columna NOMBRE.

- 14.4 Modifica las columnas PRECIO_VENTA y PRECIO_COSTO de la tabla ARTICULOS a tipo de dato FLOAT(7,2).

- 14.5 Añade la restricción de clave ajena a la columna COD_FABRICANTE de la tabla ARTICULOS para que refiera a la tabla FABRICANTES. Realiza un borrado en cascada.

Figura 14.20. Ejercicio Resuelto 14.9, Paso 8

Figura 14.20. Ejercicio Resuelto 14.9, Paso 8

Tablas PERSONAL, PROFESORES Y CENTROS

14.6 Crea una vista que se llame CONSERJES que contenga el nombre del centro y el nombre de sus conserjes.

14.7 Añade a la tabla PROFESORES una columna llamada COD_ASIG con dos posiciones numéricas.

14.8 Crea la tabla TASIG con las siguientes columnas: COD_ASIG numérico, 2 posiciones y NOM_ASIG cadena de 20 caracteres.
14.9 Añade la restricción de clave primaria a la columna COD_ASIG de la tabla TASIG.

14.10 Añade la restricción de clave ajena a la columna COD_ASIG de la tabla PROFESORES. Visualiza el nombre de las restricciones y las columnas, y las columnas afectadas para las tablas TASIG y PROFESORES.

14.11 Crea una vista de nombre CENTROSPROFESORES que contenga el nombre del centro, los apellidos y la especialidad del profesor.

14.12 Crea una vista de nombre PROFESORESCENTROS que contenga por cada código de centro el número de profesores que hay.

14.13 Crea una tabla de nombre AUDITORIA con tres columnas: una contendrá datos de tipo fecha, otra contendrá la hora y la tercera columna contendrá una cadena de caracteres.

14.14 Crea un trigger asociado a la tabla DEPARTAMENTOS que se active después de insertar una fila. La acción del trigger será insertar en la tabla AUDITORIA una fila con la fecha y hora de la inserción y el nombre de departamento que se inserta. Después inserta filas en DEPARTAMENTOS y comprueba el contenido de la tabla AUDITORIA.

15

Páginas Web con MySQL y PHP

Objetivos

- Instalar y configurar Apache y PHP
- Utilizar PHP para conectarse a MySQL y a una base de datos MySQL
- Utilizar PHP para realizar operaciones con las tablas de una base de datos MySQL
- Utilizar sesiones PHP

Contenido

- 15.1. Instalación del servidor Apache**
 - 15.2. ¿Qué es PHP?**
 - 15.3. Primeros pasos con PHP**
 - 15.4. Consulta de registros**
 - 15.5. Altas, Bajas y Modificaciones de Registros**
 - 15.6. Sesiones en PHP**
 - 15.7. Otras funciones MySQL para PHP**
- Ejercicio propuesto

En este capítulo veremos cómo se utiliza el lenguaje SQL para el diseño de páginas web dinámicas a través del lenguaje PHP. Veremos cómo algunas sentencias SQL se pueden escribir embedidas en código PHP y HTML para realizar páginas web que pueden acceder a datos almacenados en una base de datos basada en MySQL.

Introducción