TEMA 3. CREACIÓN DE TABLAS

Paulina Barthelemy

1 - Consideraciones previas a la creación de una tabla

Antes de escribir la sentencia para crear una tabla debemos pensar una serie de datos y requisitos que va a ser necesario definir en la creación.
Es importante pensar en la tabla que se quiere crear tomando las decisiones necesarias antes de escribir el formato de la creación.

1.1 – Definición de la tabla

Por una parte unas consideraciones básicas como:

- El nombre de la tabla.
- El nombre de cada columna.
- El tipo de dato almacenado en cada columna.
- El tamaño de cada columna.

1.2 - Restricciones en las tablas

Por otra parte, información sobre lo que se pueden almacenar las filas de la tabla. Esta información serán las restricciones que almacenamos en las tablas. Son una parte muy importante del modelo relacional pues nos permiten relacionar las tablas entre sí y poner restricciones a los valores de que pueden tomas los atributos (columnas) de estas tablas.

Restricciones, llamadas **`CONSTRAINTS`**, son condiciones que imponemos en el momento de crear una tabla para que los datos se ajusten a una serie de características predefinidas que mantengan su integridad.

Se conocen con su nombre en inglés y se refieren a los siguientes conceptos:

a) **`NOT NULL.`** Exige la existencia de valor en la columna que lleva la restricción.

b) **`DEFAULT.`** Proporciona un valor por defecto cuando la columna correspondiente no se le da valor en la instrucción de inserción.
Este valor por defecto debe ser una constante. No se permiten funciones ni expresiones.

c) **`PRIMARY KEY.`** Indica una o varias columnas como dato o datos que identifican unívocamente cada fila de la tabla. Sólo existe una por tabla y en ninguna fila puede tener valor `NULL`, por definición.
Es obligatoria su existencia en el modelo relacional.

d) **FOREIGN KEY.** Indica que una determinada columna de una tabla, va a servir para referenciar a otra tabla en la que está definida la misma columna (columna o clave referenciada). El valor de la clave ajena deberá coincidir con uno de los valores de esta clave referenciada o ser NULL. No existe límite en el número de claves ajenas que pueda tener una tabla. Como caso particular, una clave ajena puede referenciar a la misma tabla en la que está. Para poder crear una tabla con clave ajena deberá estar previamente creada la tabla maestra en la que la misma columna es clave primaria.

e) **UNIQUE.** Indica que esta columna o grupo de columnas debe tener un valor único. También admite valores nulos. Al hacer una nueva inserción se comprobará que el valor es único o NULL. Algunos sistemas gestores de bases de datos relacionales generan automáticamente índices para estas columnas

f) **CHECK.** Comprueba si el valor insertado en esa columna cumple una determinada condición.

2 - Formato genérico para la creación de tablas.

La sentencia SQL que permite crear tablas es CREATE TABLE.

2.1 - Formato básico para la creación de tablas

Comenzaremos con un formato básico de creación de tabla al que iremos añadiendo, posteriormente, otras informaciones.

```
CREATE TABLE [IF NOT EXISTS] NombreTabla
( NombreColumna TipoDato [ , NombreColumna TipoDato ].... );
```

donde **NombreTabla .....** es el identificador elegido para la tabla
NombreColumna ..... es el identificador elegido para cada columna
TipoDato..... indica el tipo de dato que se va a almacenar en esa columna. (Ver apartado 3 del Tema 2).

El nombre de la tabla debe ser único en la base de datos. Los nombres de columnas deben ser únicos dentro de la tabla.

Para el nombre de la tabla y de las columnas se elegirán identificadores de acuerdo con las reglas del gestor de la base de datos (Ver apartado 2.1 del Tema 2). Estos identificadores no pueden coincidir con palabras reservadas.

Existirán tantas definiciones de columna como datos diferentes se vayan a almacenar en la tabla que estamos creando, todas ellas separadas por comas.

La cláusula IF NOT EXISTS previene el posible error generado si existiese una tabla con ese nombre.

2.2 – Ejemplos básicos de creación de tablas

Los siguientes ejemplos de creación de tablas se van a utilizar siempre nuevas tablas para no alterar las tablas anteriormente utilizadas.

Realizaremos con un ejemplo de una biblioteca queremos guardar los datos de los socios en una tabla socios y los préstamos que se realizan en una tabla prestamos. Empezaremos con los formatos básicos e iremos añadiendo cláusulas. En cada apartado le daremos un nombre diferente a las tablas para evitar problemas (si en la instrucción e creación el nombre de la tabla ya existe se produce un error)

1-. Crear una tabla socios con los datos de los socios:

- Numero de socio — Número entero de 4 dígitos
- Apellidos del socios — Cadena de 14 caracteres máximo
- Teléfono — Cadena de 9 caracteres
- Fecha de alta como socio — Fecha
- Dirección — Cadena de 20 carateres máximo

- Codigo postal Número entero de 5 dígitos

```
mysql> CREATE TABLE socios_0
    ->  (socio_no INT(4),
    ->  apellidos VARCHAR(14),
    ->  telefono CHAR(9),
    ->  fecha_alta DATE,
    ->  direccion VARCHAR(20),
    ->  codigo_postal INT(5));
Query OK, 0 rows affected (0.30 sec)
```

El campo `telefono` lo creamos tipo `CHAR` en lugar de `VARCHAR` porque siempre tendrá 9 caracteres

2. Crear una tabla `prestamos` para guardar los préstamos hechos a los socios con los datos:
   - Número del préstamo Número entero de 2 dígitos
   - Código del socio Número entero de 4 dígitos

```
mysql> CREATE TABLE prestamos_0
    ->   (num_prestamo INT(2),
    ->    socio_no INT(4));
Query OK, 0 rows affected (0.08 sec)
```

2.3 – Formatos para la creación de tablas con la definición de restricciones

Los valores por defecto pueden ser: constantes, funciones `SQL` o las variables `USER` o `SYSDATE`.
La restricciones pueden definirse de dos formas, que llamaremos

1. **Restriccion1** Definición de la restricción a nivel de columna
2. **Restriccion2** Definición de la restricción a nivel de tabla

Estas restricciones , llamadas `CONSTRAINTS` se pueden almacenar con o sin nombre. Si no se lo damos nosotros lo hará el sistema siguiendo una numeración correlativa, que es poco representativa.

Es conveniente darle un nombre, para después podernos referirnos a ellas si las queremos borrar o modificar. Estos nombres que les damos a las `CONSTRAINTS` deben ser significativos para hacer mas fácil las referencias. Por ejemplo:

- *pk_NombreTabla* para `PRIMARY KEY`
- *fk_NombreTabla1_NombreTabla2* `FOREIGN KEY` donde NombreTabla1 es la tabla donde se crea y NombreTabla2 es la tabla a la que referencia.
- *uq_NombreTabla_NombreColumna* para `UNIQUE`

2.3.1 - Formato para la creación de tablas con restricciones definidas a nivel de columna

Definimos cada restricción al mismo tiempo que definimos la columna correspondiente.

```
CREATE TABLE [IF NOT EXISTS] NombreTabla
  ( NombreColumna TipoDato [Restriccion1]
      [ , NombreColumna TipoDato  [Restriccion1] ..... ] );
```

donde ***Restriccion1.....*** es la definición de la restricción a nivel de columna

Las restricciones solo se pueden definir de esta forma si afectan a una sola columna, la que estamos definiendo es ese momento.

Definición de los diferentes tipos de `CONSTRAINTS` a nivel de tabla (Restriccion1)

a) **CLAVE PRIMARIA**
```
PRIMARY KEY
```
b) **POSIBILIDAD DE NULO**
```
NULL | NOT NULL
```
c) **VALOR POR DEFECTO**
```
DEFAULT ValorDefecto
```
d) **UNICIDAD**
```
UNIQUE
```
e) **COMPROBACION DE VALORES**
```
CHECK (Expresion)
```
Nota: Esta cláusula de SQL estándar, en MySQL en la versión 5 está permitida pero no implementada

f) **CLAVE AJENA**
```
REFERENCES NombreTabla [( NombreColumna )]
```
Notación: el nombre de tabla referenciada es el nombre de la tabla a la que se va a acceder con la clave ajena. Si la columna que forma la clave referenciada en dicha tabla no tiene el mismo nombre que en la clave ajena, debe indicarse su nombre detrás del de la tabla referenciada y dentro del paréntesis. Si los nombres de columnas coinciden en la clave ajena y en la primaria, no es necesario realizar esta indicación.

g) **AUTO INCREMENTO**
```
AUTO_INCREMENT
```
Aunque no es propiamente una restricción, pero como parte de la definición de una columnas podemos indicar que sea `AUTO_INCREMENT.` De esta forma el sistema gestor irá poniendo valores en esta columna incrementándolos de 1 en 1 respecto al anterior y empezando por 1 (opción por defecto que se puede modificar cambiando las opciones de las tabla en la creación lo que queda fuera de este curso). Esta definición sólo se puede aplicar sobre columnas definidas como y enteras y que sean claves.

2.3.2 - Ejemplos de definición de restricciones a nivel de columna

Veremos un ejemplo de cada caso donde se van definiendo las columnas con las correspondientes restricciones.

a) **PRIMARY KEY.** El numero de socio en la *tabla socios*

```
mysql> CREATE TABLE socios_1a
    ->  (socio_no INT(4) PRIMARY KEY,
    ->  apellidos VARCHAR(14),
    ->  telefono CHAR(9),
    ->  fecha_alta DATE,
    ->  direccion VARCHAR(20),
    ->  codigo_postal INT(5));
Query OK, 0 rows affected (0.08 sec)
```

b) **NOT NULL.** La columna teléfono es obligatoria en la tabla `socios`, nunca irá sin información.

```
mysql> CREATE TABLE socios_1b
    ->  (socio_no INT(4) PRIMARY KEY,
    ->  apellidos VARCHAR(14),,
    ->  telefono CHAR(9) NOT NULL,
    ->  fecha_alta DATE,
    ->  direccion VARCHAR(20),
    ->  codigo_postal INT(5));
Query OK, 0 rows affected (0.05 sec)
```

c) **DEAFAULT.** En ausencia de valor el campo fecha _ alta tomará el valor de 1 de enero de 2000.

```
mysql> CREATE TABLE socios_1c
    ->  (socio_no INT(4) PRIMARY KEY,
    ->  apellidos VARCHAR(14),
    ->  telefono CHAR(9) NOT NULL,
    ->  fecha_alta DATE DEFAULT '2000-01-01',
    ->  direccion VARCHAR(20),
    ->  codigo_postal INT(5));
Query OK, 0 rows affected (0.11 sec)
```

d) **UNIQUE.** La columna apellido será única en la tabla `socios`.

```
mysql> CREATE TABLE socios_1d
    ->  (socio_no INT(4) PRIMARY KEY,
    ->  apellidos VARCHAR(14) UNIQUE,
    ->  telefono CHAR(9) NOT NULL,
    ->  fecha_alta DATE DEFAULT '2000-01-01',
    ->  direccion VARCHAR(20),
    ->  codigo_postal INT(5) );
Query OK, 0 rows affected (0.06 sec)
```

e) **CHECK.** Se comprobará que la columna cdigo_postal corresponde a Madrid (valores entre 28000 y 28999)

```
mysql> CREATE TABLE socios_1e
    ->  (socio_no INT(4) PRIMARY KEY,
    ->  apellidos VARCHAR(14) UNIQUE,
    ->  telefono CHAR(9) NOT NULL,
    ->  fecha_alta DATE DEFAULT '2000-01-01',
    ->  direccion VARCHAR(20),
    ->  codigo_postal INT(5)
```

```
            CHECK (codigo_postal BETWEEN 28000 AND 28999) );
Query OK, 0 rows affected (0.06 sec)
```

f) **FOREIGN KEY.** El campo socio_num de la tabla *prestamos* tendrá que tener valores existentes en el campo num_socio de la tabla *socios* o valor nulo.

```
mysql> CREATE TABLE prestamos
    ->  (num_prestamo INT(2) PRIMARY KEY,
    ->  socio_no INT(4) REFERENCES socios_1e(socio_no));
Query OK, 0 rows affected (0.17 sec)
```

g) **AUTO INCREMENTO.** Crearemos una tabla con una columna AUTO_INCREMENT y posteriormente (siguiente tema) insertaremos valores.

```
mysql> CREATE TABLE inventario
    -> (num INT(2) AUTO_INCREMENT PRIMARY KEY,
    ->  descripcion VARCHAR(15));
Query OK, 0 rows affected (0.49 sec)
```

2.3.3 - Formato para la creación de tablas con restricciones definidas a nivel de tabla

En este caso definimos todas las restricciones al final de la sentencia, una vez terminada la definición de las columnas.

CREATE TABLE [IF NOT EXISTS NombreTabla
(NombreColumna TipoDato [, NombreColumna TipoDato.....]
[Restriccion2 [, Restrccion2].....);

donde ***Restriccion2.....*** es la definición de la restricción a nivel de tabla

Las restricciones siempre se pueden definir de esta forma tanto si afectan a una sola columna como a varias columnas y puede darse un nombre a cada una de las restricciones.

Definición de los diferentes tipos de `CONSTRAINTS` a nivel de tabla (Restriccion2)

a) **PRIMARY KEY**

```
[CONSTRAINT [NombreConstraint]]
          PRIMARY KEY (Nombrecolumna [,NombreColumna....] )
```

b) **UNIQUE**

```
[CONSTRAINT [NombreConstraint]] UNIQUE (NombreColumna
                                        [,NombreColumna...] )
```

c) **CHECK**

```
[CONSTRAINT [NombreConstraint]] CHECK (Expresion)
```

d) **FOREIGN KEY**

```
[CONSTRAINT [NombreConstraint ]]
```

```
FOREIGN KEY (NombreColumna[, NombreColumna...])
REFERENCES ( NombreTabla [NombreColumna [, NombreColumna.....]])
```

Notación: los nombres de columna o columnas que siguen a la cláusula FOREIGN KEY es aquella o aquellas que están formando la clave ajena. Si hay más de una se separan por comas. El nombre de tabla referenciada es el nombre de la tabla a la que se va a acceder con la clave ajena. Si la columna o columnas que forman la clave referenciada en dicha tabla no tienen el mismo nombre que en la clave ajena, debe indicarse su nombre detrás del de la tabla referenciada y dentro de paréntesis. Si son más de una columna se separan por comas. Si los nombres de columnas coinciden en la clave ajena y en la primaria, no es necesario realizar esta indicación.

2.3.4 - Ejemplos de definición de restricciones a nivel de tabla

Veremos un ejemplo de cada caso donde se van definiendo la columna con la correspondiente restricción. Algunos ejemplos con nombre de constraint y otros sin él.
En un ejemplo, igual que el apartado anterior, de una biblioteca queremos guardar los datos de los socios en una tabla *socios* y los préstamos que se realizan en una tabla *prestamos*

a)**PRIMARY KEY.** El numero de socio será la clave primaria en la *tabla socios.* Será obligatoriamente no nulo y único.

```
mysql> CREATE TABLE socios_2a
    ->  (socio_no INT(4),
    ->  apellidos VARCHAR(14),
    ->  telefono CHAR(9),
    ->  fecha_alta DATE,
    ->  direccion VARCHAR(20),
    ->  codigo_postal INT(5),
    ->  CONSTRAINT PK2_DEPARTAMENTOS PRIMARY KEY (socio_no));
Query OK, 0 rows affected (0.08 sec)
```

b) **UNIQUE.** El campo apellido es único. Tendrá valores diferentes en cada fila o el valor nulo

```
mysql> CREATE TABLE socios_2b
    ->  (socio_no INT(4),
    ->  apellidos VARCHAR(14),
    ->  telefono CHAR(9),
    ->  fecha_alta DATE,
    ->  direccion VARCHAR(20),
    ->  codigo_postal INT(5),
    ->  CONSTRAINT PK_SOCIOS PRIMARY KEY(socio_no),
    ->  CONSTRAINT UQ_UNIQUE UNIQUE (apellidos));
Query OK, 0 rows affected (0.06 sec)
```

c)**CHECK.** La columna codigo_postal no admitirá como válidas aquellas filas en las que el código postal no tenga valores entre 28.000 y 28.999 (correspondientes a Madrid)

```
mysql> CREATE TABLE socios_2c
    ->  (socio_no INT(4),
    ->  apellidos VARCHAR(14),
    ->  telefono CHAR(9),
```

```
    ->  fecha_alta DATE,
    ->  direccion VARCHAR(20),
    ->  codigo_postal INT(5),
    ->  CONSTRAINT PK_DEPARTAMENTOS PRIMARY KEY(socio_no),
    ->  CONSTRAINT UQ_UNIQUE UNIQUE(apellidos),
    ->  CONSTRAINT CK_CODIGO
    ->                    CHECK (codigo_postal BETWEEN 28000 AND
28999) );
Query OK, 0 rows affected (0.17 sec)
```

d) **FOREIGN KEY.** El número de socio en una *tabla prestamos* será clave ajena referenciando a la columna correspondiente de la tabla `socios.`

```
mysql> CREATE TABLE prestamos_2
    ->  (num_prestamo INT(2),
    ->  socio_no INT(4) ,
    ->  CONSTRAINT PK_PRESTAMOS PRIMARY KEY(num_prestamo),
    ->  CONSTRAINT FK_SOCIO_PRESTAMOS   FOREIGN KEY (socio_no)
        REFERENCES socios_2c(socio_no) );
Query OK, 0 rows affected (0.13 sec)
```

2.4 - Integridad referencial

La definición de claves ajenas nos permiten mantener la integridad referencial en una base de datos relacional. Hemos dicho que la columna o columnas definidas como clave ajena deben tomar valores que se correspondan con un valor existente de la clave referenciada. La pregunta es: ¿qué sucede si queremos borrar o modificar un valor de la clave primaria refenciada? Pues que el sistema debe impedirnos realizar estas acciones pues dejaría de existir la integridad referencial.

Por ejemplo si tenemos

```
CREATE TABLE departamentos
      (dep_no INT(4)
             CONSTRAINT pk_departamentos PRIMARY KEY.....);

CREATE TABLE empleados
      (....dep_no INT(4) CONSTRAINT fk_empleados_departamentos
                         REFERENCES departamentos(dep_no).... );
```

En este caso empleados.dep_no solo puede tomar valores que existan en departamentos.dep_no pero ¿que sucede si queremos borrar o modificar un valor de departamentos.dep_no.? El sistema no nos lo permitirá si existen filas con ese valor en la tabla de la clave ajena.

Sin embargo, en ocasiones, será necesario hacer estas operaciones. Para mantener la integridad de los datos, al borrar `(DELETE),` modificar `(UPDATE)` una fila de la tabla referenciada, el sistema no nos permitirá llevarlo a cabo si existe una fila con el valor referenciado. También se conoce como `RESTRICT.` Es la opción por defecto, pero existen las siguientes opciones en la definición de la clave ajena:

- **CASCADE.** El borrado o modificación de una fila de la tabla referenciada lleva consigo el borrado o modificación en cascada de las filas de la tabla que contiene la clave ajena. Es la más utilizada.
- **SET NULL**. El borrado o modificación de una fila de la tabla referenciada lleva consigo

poner a NULL los valores de las claves ajenas en las filas de la tabla que referencia.

- **SET DEFAULT**. El borrado o modificación de una fila de la tabla referenciada lleva consigo poner un valor por defecto en las claves ajenas de la tabla que referencia.
- **NO ACTION**. El borrado o modificación de una fila de la tabla referenciada solo se produce si no existe ese valor en la tabla que contiene la clave ajena. Tiene el mismo efecto que RESTRICT.

Formato de la definición de clave ajena con opciones de referencia

```
REFERENCES NombreTabla [ ( NombreColumna [ , NombreColumna ] ) ]
      [ON DELETE {CASCADE| SET NULL|NO ACTION | SET DEFAULT| RESTRICT}]
      [ON UPDATE {CASCADE| SET NULL|NO ACTION | SET DEFAULT| RESTRICT}]
```

por ejemplo

```
CREATE TABLE empleados
      (....
       dep_no NUMBER(4) CONSTRAINT FK_EMPLEADOS_DEPARTAMENTOS
                        REFERNCES departamentos(dep_no)
                        ON DELETE SET NULL
                        ON UPDATE CASCADE .... );
```

Esto quiere decir que el sistema pondrá nulos en dep_no de la tabla empleados si se borra el valor correspondiente en departamentos y modificará el valor de dep_no en la tabla empleados con el nuevo valor si se modifica la columna dep_no en la tabla departamentos.

En el tema siguiente veremos con más detalle los borrados y modificaciones en los casos en los que existan estas cláusulas en las definiciones de las tablas, realizando algún ejemplo.

2.5 - Formato completo de la creación de tablas

```
CREATE TABLE [IF NOT EXISTS] NombreTabla
       (  NombreColumna TipoDato [Restriccion1 ]
              [ , NombreColumna TipoDato [Restriccion1..... ]
        [Restriccion2 [ , Restrccion2....]    );
```

Notación: los diferentes formatos de definición de restricciones, restricción1 y restricción2, están entre corchetes porque son opcionales, pudiéndose elegir entre ambos sólo si la restricción afecta a una sola columna.

Vamos a crear la tabla socios y prestamos con el formato completo. Algunas CONSTRAINTS las creamos a nivle de columna y otras de tabla:

```
mysql> CREATE TABLE socios
    ->      (socio_no INT(4),
    ->       apellidos VARCHAR(14),
    ->       telefono CHAR(9) NOT NULL,
    ->       fecha_alta DATE DEFAULT '2000-01-01',
    ->       direccion VARCHAR(20),
```

```
    ->       codigo_postal INT(5),
    ->       CONSTRAINT PK_DEPARTAMENTOS PRIMARY KEY(socio_no),
    ->       CONSTRAINT UQ_UNIQUE UNIQUE(apellidos),
    ->       CONSTRAINT CK_CODIGO
    ->                     CHECK (codigo_postal BETWEEN 28000 AND
28999) );
Query OK, 0 rows affected (0.53 sec)

mysql> CREATE TABLE prestamos
    ->     (num_prestamo INT(2) PRIMARY KEY,
    ->      socio_no INT(4) ,
    ->      CONSTRAINT FK_SOCIO_PRESTAMOS FOREIGN KEY (socio_no)
    ->         REFERENCES socios(socio_no) );
Query OK, 0 rows affected (0.17 sec)
```

Utilizaremos estas tablas en el tema siguiente para realizar inserciones, modificaciones y borrados de filas.

3 - Modificación de la definición de una tabla.

Una vez que hemos creado una tabla, a menudo se presenta la necesidad de tener que modificarla. La sentencia SQL que realiza esta función es **ALTER TABLE.**

3.1 - Formato general para la modificación de tablas

La especificación de la modificación es parecida a la de la sentencia CREATE pero varía según el objeto SQL del que se trate.

ALTER TABLE NombreTabla
EspecificacionModificacion [, EspecificacionModificacion.....]

donde **NombreTabla.....** nombre de la tabla se desea modificar.
EspecificacionModificacion..... las modificaciones que se quieren realizarse sobre la tabla

Las modificaciones que se pueden realizar sobre una tabla son:

- Añadir una nueva columna
- Añadir un nueva restricción
- Borrar una columna
- Borrar una restricción
- Modificar una columna sin cambiar su nombre
- Modificar una columna y cambiar su nombre
- Renombrar la tabla

a) AÑADIR UNA NUEVA COLUMNA

```
ADD [COLUMN] NombreColumna TipoDato  [ Restriccion1 ]
```

Puede añadirse una nueva columna y todas las restricciones, salvo NOT NULL. La razón es que esta nueva columna tendrá los valores NULL al ser creada.

b) AÑADIR UNA CONSTRAINT

```
ADD [CONSTRAINT [NombreConstraint] ]
      PRIMARY KEY (NombreColumna [, NombreColumna...] )
ADD [CONSTRAINT [NombreConstraint] ]
      FOREIGN KEY (NombreColumna [, NombreColumna...] )
      REFERENCES NombreTabla[(NombreColumna[,NombreColumna...] )]
ADD [CONSTRAINT [NombreConstraint] ]
      UNIQUE (NombreColumna [, NombreColumna...  ] )
```

c) BORRAR UNA COLUMNA

```
DROP [COLUMN] NombreColumna
```

d) BORRAR UNA CONSTRAINT

```
DROP PRIMARY KEY
```

```
DROP FOREIGN KEY NombreConstraint
```

e) MODIFICAR UNA COLUMNA SIN CAMBIAR SU NOMBRE

```
MODIFY [COLUMN] NombreColumna TipoDato  [Restriccion1]
```

f) MODIFICAR LA DEFINCION DE UNA COLUMNA Y SU NOMBRE

```
CHANGE [COLUMN] NombreColumnaAntiguo
                NombreColumnaNuevo TipoDatos [Restriccion1]
```

f) RENOMBRAR LA TABLA

```
RENAME [TO] NombreTablaNuevo
```

4 - Eliminación de una tabla.

Para borrar una tabla y su contenido de la base de datos se utiliza la sentencia `DROP TABLE.`

4.1 - Formato para eliminar una tabla.

DROP TABLE [IF EXISTS] NombreTabla
[CASCADE | RESTRICT] ;

La cláusula `IF EXISTS` previene los errores que puedan producirse si no existe la tabla que queremos borrar.
Nota: las cláusulas `CASCADE Y RESTRICT` para borrado en cascada y borrado de las restricciones están permitidas pero no implementadas en esta versión.

4.2 – Ejemplos de borrado de una tabla

1- Borraremos las tablas `departamentos2 y empleados2` enlazadas con una clave ajena. Hay que tener cuidado con el orden:

```
mysql> DROP TABLE  departamentos2;
ERROR 1217 (23000): Cannot delete or update a parent row: a
foreign key constraint fails
```

Debemos borrar primero `empleados2:`

```
mysql> DROP TABLE empleados2;
Query OK, 0 rows affected (0.10 sec)
mysql> DROP TABLE  departamentos2;
Query OK, 0 rows affected (0.06 sec)
```

2 - Borraremos una tabla `empleados7,` que no existe, con la cláusula `IF EXISTS` y vemos que no nos da error.

```
mysql> DROP TABLE IF EXISTS empleados7;
Query OK, 0 rows affected, 1 warning (0.00 sec)
```

5 – Renombrado de una tabla

Permite cambiar de nombre una tabla, asignándole un nombre nuevo y el nombre antiguo desaparece.

5.1 - Formato para renombrar una tabla

RENAME TABLE NombreTablaAntiguo TO NombreTablaNuevo
[, NombreTablaAntiguo TO NombreTablaNuevo .....]

Notación: se pueden renombrar varias tablas en una sentencia por eso van entre corchetes las siguientes tablas a renombrar.

donde **NombreTablaAntiguo .....** es el nombre antiguo de la tabla, que debe existir
NombreTablaNuevo..... es el nombre nuevo de la tabla, que no debe existir.

Permite renombrar en la misma instrucción una o varias tablas.

5.2 – Ejemplos de renombrado de una tabla

1- Renombrar la tabla *socios* para que su nuevo nombre sea socios2

```
mysql> RENAME TABLE socios TO socios2;
Query OK, 0 rows affected (0.05 sec)

mysql> select * from socios;
ERROR 1146 (42S02): Table 'test.socios' doesn't exist
```

TEMA 4. ACTUALIZACION DE TABLAS

Paulina Barthelemy

1 - Introducción

Como ya hemos explicado el lenguaje SQL incluye un conjunto de sentencias que permiten modificar, borrar e insertar nuevas filas en una tabla. Este subconjunto de comandos del lenguaje SQL recibe la denominación de *Lenguaje de Manipulación de Datos* (DML)

Las órdenes que no permiten actualizar los valores de las tablas son:

- INSERT para la inserción de filas
- UPDATE para la modificación de filas
- DELETE para el borrado de filas.

2 - Inserción de nuevas filas en la base de datos

Para añadir nuevas filas a una tabla utilizaremos la sentencia INSERT.

2.1 – Formato de la inserción una fila

La sentencia para inserción de filas es INSERT cuyo formato típico es:

```
INSERT INTO NombreTabla [( NombreColumna [,NombreColumna...] ) ]
VALUES (Expresion [, Expresión ...] );
```

Notación: la lista de columnas en las que insertamos va es opcional, por lo que va entre corchetes. Si no se especifica se esperan valores para todas las columnas.

Donde **NombreTabla**..... es el nombre de la tabla en la que queremos insertar una fila.
NombreColumna..... incluye las columnas en las que queremos insertar.
Expresion..... indica las expresiones cuyos valores resultado se insertarán, existiendo una correspondencia con la lista de columnas anterior

Como se ve en el formato también se pueden insertar filas en una tabla sin especificar la lista de columnas pero en este caso la lista de valores a insertar deberá coincidir en número y posición con las

columnas de la tabla. El orden establecido para las columnas será el indicado al crear la tabla (el mismo que aparece al hacer una descripción de la tabla DESC NombreTabla*)*.

2.2 – Ejemplos de inserción de filas

Por ejemplo, para insertar una nueva fila en la tabla *departamentos* introduciremos la siguiente instrucción:

```
mysql> INSERT INTO departamentos(dep_no, dnombre, localidad)
    -> VALUES (50, 'MARKETING','BILBAO');
Query OK, 1 row affected (0.08 sec)

mysql> select * from departamentos;
+--------+---------------+-----------+
| DEP_NO | DNOMBRE       | LOCALIDAD |
+--------+---------------+-----------+
|     10 | CONTABILIDAD  | BARCELONA |
|     20 | INVESTIGACION | VALENCIA  |
|     30 | VENTAS        | MADRID    |
|     40 | PRODUCCION    | SEVILLA   |
|     50 | MARKETING     | BILBAO    |
+--------+---------------+-----------+
5 rows in set (0.00 sec)
```

Este formato de inserción tiene, además, las siguientes características:

- En la lista de columnas se pueden indicar todas o algunas de las columnas de la tabla. En este último caso, aquellas columnas que no se incluyan en la lista quedarán sin ningún valor en la fila insertada, es decir, se asumirá el valor `NULL` o el valor por defecto para las columnas que no figuren en la lista. En aso de una columna definida como `NOT NULL` tomará el valor 0 si es numérica o blancos sino.
- Los valores incluidos en la lista de valores deberán corresponderse posicionalmente con las columnas indicadas en la lista de columnas, de forma que el primer valor de la lista de valores se incluirá en la columna indicada en primer lugar, el segundo en la segunda columna, y así sucesivamente.
- Se puede utilizar cualquier expresión para indicar un valor siempre que el resultado de la expresión sea compatible con el tipo de dato de la columna correspondiente.

Algunos ejemplos de inserciones válidas son:
1- Inserción de un socio con socio_no=1000

```
mysql> INSERT INTO Socios
(socio_no,apellidos,telefono,fecha_alta,direccion,codigo_postal)
    ->    VALUES ( 1000,'LOPEZ SANTOS','916543267', '2005-01-08',
    ->            'C. REAL 5',28400);
Query OK, 1 row affected (0.05 sec)
```

2 - Inserción de un socio con socio_no=1001

```
mysql> INSERT INTO Socios
 (socio_no,apellidos,telefono,fecha_alta,direccion,codigo_postal)
```

```
    ->     VALUES ( 1001,'PEREZ CERRO','918451256', '2005-01-12',
    ->            'C. MAYOR 31',28400);
Query OK, 1 row affected (0.06 sec)
```

3 - Inserción de un socio con socio_no=1002

```
mysql> INSERT INTO Socios
 (socio_no,apellidos,telefono,fecha_alta,direccion,codigo_postal)
    ->     VALUES ( 1002,'LOPEZ PEREZ','916543267', '2005-01-18',
    ->            'C. REAL 5',28400);
Query OK, 1 row affected (0.06 sec)
```

4 - Inserción de un socio con socio_no=1003 sin valor en teléfono (como es un campo `NOT NULL` en lugar de nulos pondrá blancos)

```
mysql> INSERT INTO socios
    -> (socio_no,apellidos,fecha_alta,direccion,codigo_postal)
    ->     VALUES ( 1003,'ROMA LEIVA', '2005-01-21',
    ->            'C. PANADEROS 9 ',28431);
Query OK, 1 row affected (0.06 sec)
```

5- Inserción de un préstamo para el socio con socio_no=1000

```
mysql> INSERT INTO prestamos
    ->     (num_prestamo, socio_no)
    ->     VALUES ( 1,1000);
Query OK, 1 row affected (0.05 sec)
```

6 - Inserción de un préstamo para el socio con socio_no=1002

```
mysql> INSERT INTO prestamos
    ->     (num_prestamo, socio_no)
    ->        VALUES ( 2,1002);
Query OK, 1 row affected (0.07 sec)
```

7 - Inserción de un socio con socio_no=1004, con una instrucción INSERT sin lista de columnas:

```
mysql> INSERT INTO socios
    -> VALUES ( 1004,'GOMEZ DURUELO','918654329', '2005-01-31',
    ->            'C. REAL 15',28400);
Query OK, 1 row affected (0.43 sec)
```

8 - Inserción de un socio con socio_no=1005, con una instrucción INSERT sin valor en el campo fecha que tiene un valor por defecto (pondrá ese valor 2000-01-01)

```
mysql> INSERT INTO socios
    ->   (socio_no,apellidos,telefono,direccion,codigo_postal)
    ->      VALUES ( 1005,'PEÑA MAYOR','918515256','C. LARGA 31',
    ->                 28431);
Query OK, 1 row affected (0.07 sec)
```

9 - Inserción de un socio con socio_no=1004, con una instrucción INSERT sin lista de columnas:

```
mysql> INSERT INTO prestamos
    ->      VALUES ( 4,1004);
Query OK, 1 row affected (0.40 sec)
```

Algunos intentos de inserciones erróneas:

1 – Inserción en la tabla `socios` en la que falta un valor para la columna dirección:

```
mysql> INSERT INTO Socios
    ->   (socio_no,apellidos,telefono,fecha_alta,direccion)
    ->    VALUES ( 1005,'LOPEZ SANTOS','916543267', '2005-01-08',
    ->            'C. REAL 5',28400);
ERROR 1136(21S01):Column count doesn't match value count at row 1
```

2 – Inserción en la tabla `socios` de una columna con clave primaria duplicada:

```
mysql> INSERT INTO Socios
 (socio_no,apellidos,telefono,fecha_alta,direccion,codigo_postal)
    ->    VALUES ( 1000,'LOPEZ SANTOS','916543267', '2005-01-08',
    ->            'C. REAL 5',28400);
ERROR 1062 (23000): Duplicate entry '1000' for key 1
```

3 - Inserción en la tabla `socios` de una fila con valores duplicados en la columna apellidos:

```
mysql> INSERT INTO Socios
 (socio_no,apellidos,telefono,fecha_alta,direccion,codigo_postal)
    ->    VALUES ( 1009,'LOPEZ SANTOS','916543267', '2005-01-18',
    ->            'C. REAL 5',28400);
ERROR 1062 (23000): Duplicate entry 'LOPEZ SANTOS' for key 2
```

4 - Inserción de una fila en la tabla `prestamos` con un valor en la columna socios_no que no existe en la tabla `socios`

```
mysql> INSERT INTO prestamos
    ->     (num_prestamo, socio_no)
    ->        VALUES ( 3,1009);
ERROR 1216 (23000): Cannot add or update a child row: a foreign key
constraint fails
```

Comprobación de los valores insertados:

```
mysql> SELECT *
    -> FROM socios;
```

socio_no	apellidos	telefono	fecha_alta	direccion	codigo_postal
1000	LOPEZ SANTOS	916543267	2005-01-08	C. REAL 5	28400
1001	PEREZ CERRO	918451256	2005-01-12	C. MAYOR 31	28400
1002	LOPEZ PEREZ	916543267	2005-01-18	C. REAL 5	28400
1003	ROMA LEIVA		2005-01-21	C. PANADEROS 9	28431
1004	GOMEZ DURUELO	918654329	2005-01-31	C. REAL 15	28400
1005	PEÑA MAYOR	918515256	2000-01-01	C. LARGA 31	28431

```
   6 rows in set (0.00 sec)

SELECT *
FROM prestamos;

       mysql> SELECT *
           -> FROM prestamos;
       +--------------+----------+
       | num_prestamo | socio_no |
       +--------------+----------+
       |            1 |     1000 |
       |            2 |     1002 |
       |            4 |     1004 |
       +--------------+----------+
       3 rows in set (0.00 sec)
```

Ejemplo de inserción con un campo autonumérico, de paso creamos una tabla con una columna autonumérica.

```
       mysql> CREATE TABLE inventario
           -> (num INT(2) AUTO_INCREMENT PRIMARY KEY,
           ->  descripcion VARCHAR(15));
       Query OK, 0 rows affected (0.49 sec)
```

1 – Inserción sin enumerar la columna autonumérica

```
       mysql> INSERT INTO inventario (descripcion)
           -> VALUES ('ARMARIO BLANCO');
       Query OK, 1 row affected (0.42 sec)

       mysql> INSERT INTO inventario (descripcion)
           -> VALUES ('MESA MADERA');
       Query OK, 1 row affected (0.05 sec)
```

2 – Inserción de toda la fila (en este caso debe ponerse NULL en la columna correspondiente)

```
       mysql> INSERT INTO inventario
           -> VALUES (NULL,'ORDENADOR');
       Query OK, 1 row affected (0.07 sec)

       mysql> INSERT INTO inventario
           -> VALUES (NULL,'SILLA GIRATORIA');
       Query OK, 1 row affected (0.06 sec)
```

Comprobación de los valores insertados:

```
       mysql> SELECT *
           -> FROM inventario;
       +-----+-----------------+
       | num | descripcion     |
```

```
+-----+-----------------+
|   1 | ARMARIO BLANCO  |
|   2 | MESA MADERA     |
|   3 | ORDENADOR       |
|   4 | SILLA GIRATORIA |
+-----+-----------------+
4 rows in set (0.00 sec)
```

3 - Modificación de filas.

En ocasiones necesitaremos modificar alguno de los datos de las filas existentes de una tabla. Por ejemplo, cambiar el salario o el departamento de uno o varios empleados, etcétera. En estos casos utilizaremos la sentencia UPDATE.

3.1 – Formato de la modificación de filas

La sentencia de modificación de filas es UPDATE cuyo formato genérico es el siguiente:

```
UPDATE NombreTabla
SET NombreColumna = Expresion  [, NombreColumna = Expresion....]
[WHERE Condición];
```

Notación: puede actualizarse una o varias columnas, por lo que la segunda actualización va entre corchetes La cláusula WHERE aparece entre corchetes porque es opcional. En el caso de que no se utilice, la actualización afectará a toda la tabla.

Donde **NombreTabla……** indica la tabla destino donde se encuentran las filas que queremos borrar.
NombreColumna……..... indica el nombre de la columna cuyo valor queremos modificar
Expresión…… es una expresión cuyo valor resultado será el nuevo valor de la columna
Condición…… es la condición que deben cumplir las filas para que les afecte la modificación.

Recordamos que una expresión es un conjunto de datos, constantes y variables, operadores y funciones. Y una condición es una expresión cuyo resultado es `verdadero/falso/nulo`
Para aquellas filas en las que al evaluar la condición el resultado es `verdadero` se modificará el valor de la columna poniendo como nuevo valor el resultado de evaluar la expresión.
Si se quiere modificar más de una columna de la misma tabla se indicarán separadas por comas.

3.2 – Ejemplos de modificación de filas

Partimos de la tabla socios

```
mysql> SELECT  * FROM socios;
+----------+--------------+-----------+------------+----------------+---------------+
| socio_no | apellidos    | telefono  | fecha_alta | direccion      | codigo_postal |
+----------+--------------+-----------+------------+----------------+---------------+
|     1000 | LOPEZ SANTOS | 916543267 | 2005-01-08 | C. REAL 5      |         28400 |
|     1001 | PEREZ CERRO  | 918451256 | 2005-01-12 | C. MAYOR 31    |         28400 |
|     1002 | LOPEZ PEREZ  | 916543267 | 2005-01-18 | C. REAL 5      |         28400 |
|     1003 | ROMA LEIVA   |           | 2005-01-21 | C. PANADEROS 9 |         28431 |
|     1004 | GOMEZ DURUELO | 918654329 | 2005-01-31 | C. REAL 15     |         28400 |
|     1005 | PEÑA MAYOR   | 918515256 | 2000-01-01 | C. LARGA 31    |         28431 |
+----------+--------------+-----------+------------+----------------+---------------+
6 rows in set (0.01 sec)
```

1 - Supongamos que se desea cambiar la dirección del socio de número 1000 y la nueva dirección es :

'C.CUESTA 2'.

```
mysql> UPDATE socios
    -> SET direccion = 'C.CUESTA 2'
    -> WHERE socio_no = 1000;
Query OK, 1 row affected (0.08 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

2 - Supongamos que se desea cambiar el teléfono del socio de número 1000 y el nuevo teléfono es 918455431

```
mysql> UPDATE socios
    -> SET telefono = '918455431'
    -> WHERE socio_no = 1000;
Query OK, 1 row affected (0.06 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

También podíamos haber modificado las dos columnas con una sola sentencia:

```
mysql> UPDATE socios
    -> SET telefono = '918455431',direccion='C.CUESTA 2'
    -> WHERE socio_no = 1000;
Query OK, 0 rows affected (0.08 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

Las actualizaciones anteriores afectan a una única fila pero podemos escribir comandos de actualización que afecten a varias filas:

```
mysql> UPDATE socios
    -> SET codigo_postal = 28401
    -> WHERE codigo_postal = 28400;
Query OK, 4 rows affected (0.07 sec)
Rows matched: 4  Changed: 4  Warnings: 0
```

Si no se incluye la cláusula WHERE la actualización afectará a todas las filas. El siguiente ejemplo modifica la fecha de alta de todos los empleados al valor 1 de enero de 2005.

```
mysql> UPDATE socios
    -> SET fecha_alta = '2005-01-01';
Query OK, 6 rows affected (0.06 sec)
Rows matched: 6  Changed: 6  Warnings: 0
```

Podemos comprobar las modificaciones:

```
mysql> SELECT *
    -> FROM socios;
+----------+--------------+-----------+------------+----------------+--------------+
| socio_no | apellidos    | telefono  | fecha_alta | direccion      | codigo_postal |
+----------+--------------+-----------+------------+----------------+--------------+
|     1000 | LOPEZ SANTOS | 918455431 | 2005-01-01 | C.CUESTA 2     |         28401 |
|     1001 | PEREZ CERRO  | 918451256 | 2005-01-01 | C. MAYOR 31    |         28401 |
|     1002 | LOPEZ PEREZ  | 916543267 | 2005-01-01 | C. REAL 5      |         28401 |
|     1003 | ROMA LEIVA   |           | 2005-01-01 | C. PANADEROS 9 |         28431 |
|     1004 | GOMEZ DURUELO | 918654329 | 2005-01-01 | C. REAL 15     |         28401 |
|     1005 | PEÑA MAYOR   | 918515256 | 2005-01-01 | C. LARGA 31    |         28431 |
+----------+--------------+-----------+------------+----------------+--------------+
6 rows in set (0.00 sec)
```

4 - Eliminación de filas.

La sentencia `DELETE` es la que nos permite eliminar una o más filas de una tabla.

4.1 – Formato de la eliminación de filas

Para eliminar o suprimir filas de una tabla utilizaremos la sentencia `DELETE`. Su formato genérico es el siguiente:

```
DELETE FROM NombreTabla
[WHERE Condicion];
```

Notación: la cláusula WHERE es opcional por eso aparece entre corchetes. Si no se especifica se borraran todas las filas de la tabla

donde **NombreTabla**..... indica la tabla destino donde se encuentran las filas que queremos borrar
Condición..... es la condición que deben cumplir las filas a borrar

4.2 – Ejemplos de la eliminación de filas

A continuación se muestran algunos ejemplos de instrucciones DELETE

```
mysql> DELETE
    -> FROM socios
    -> WHERE socio_no = 1001;
Query OK, 1 row affected (0.08 sec)
```

Hay que tener cuidado con las claves ajenas. Como `prestamos` tiene una clave ajena que referencia a `socios` si pretendemos borrar un socio que tiene préstamos nos encontraremos con un error.

```
mysql> DELETE
    -> FROM socios
    -> WHERE socio_no = 1002;
ERROR 1217 (23000): Cannot delete or update a parent row: a
foreign key constraint fails
```

Podemos comprobar las modificaciones:

```
mysql> SELECT *
    -> FROM socios;
+----------+--------------+-----------+------------+----------------+--------------+
| socio_no | apellidos    | telefono  | fecha_alta | direccion      | codigo_postal |
+----------+--------------+-----------+------------+----------------+--------------+
|     1000 | LOPEZ SANTOS  | 918455431 | 2005-01-01 | C.CUESTA 2     |         28401 |
```

```
|     1002 | LOPEZ PEREZ   | 916543267 | 2005-01-01 | C. REAL 5      |         28401 |
|     1003 | ROMA LEIVA    |           | 2005-01-01 | C. PANADEROS 9 |         28431 |
|     1004 | GOMEZ DURUELO | 918654329 | 2005-01-01 | C. REAL 15     |         28401 |
|     1005 | PEÑA MAYOR    | 918515256 | 2005-01-01 | C. LARGA 31    |         28431 |
+----------+---------------+-----------+------------+----------------+---------------+
6 rows in set (0.00 sec)
```

5 - Restricciones de integridad y actualizaciones.

Como ya hemos vistos los gestores de bases de datos relacionales permiten especificar ciertas condiciones que deban cumplir los datos contenidos en las tablas, como por ejemplo:

- Restricción de clave primaria (PRIMARY KEY) : columnas que identifican cada fila
- Restricción de clave ajena (FOREIGN KEY): columnas que hacen referencia a otras de la misma o de otras tablas
- Restricción de comprobación de valores (CHECK): conjunto de valores que puede o debe tomar una columna.
- Restricción de no nulo (NOT NULL): columnas que tienen que tener necesariamente un valor no nulo.
- Restricción de unicidad (UNIQUE): columnas que no pueden tener valores duplicados.

5.1 - Control de las restricciones de integridad referencial

Estas restricciones dan lugar a que no podamos hacer cualquier modificación en los datos de las tablas. No nos estará permitido hacer inserciones ni modificaciones con valores no permitidos en las columnas ni borrar filas a las que referencien otras filas de la misma o de otra tabla

En nuestras tablas de ejemplo están definidas las siguientes restricciones:
a) **Claves primarias** (PRIMARY KEY). Sirven para referirse a una fila de manera inequívoca. No se permiten valores repetidos.

TABLA DEPARTAMENTOS: PRIMARY KEY (DEP_NO)
TABLA EMPLEADOS: PRIMARY KEY (EMP_NO)
TABLA CLIENTES: PRIMARY KEY(CLIENTE_NO)
TABLA PRODUCTOS: PRIMARY KEY (PRODUCTO_NO)
TABLA PEDIDOS: PRIMARY KEY (PEDIDO_NO)

Como ya hemos visto no podemos hacer inserciones con valores repetidos de las claves primarias.

b) **Claves ajenas** (FOREIGN KEY): Se utilizan para hacer referencia a columnas de otras tablas. Cualquier valor que se inserte en esas columnas tendrá su equivalente en la tabla referida. Opcionalmente se pueden indicar las acciones a realizar en caso de borrado o cambio de las columnas a las que hace referencia .

```
TABLA EMPLEADOS: FOREIGN KEY (DEP_NO)
                        REFERENCES DEPARTAMENTOS(DEP NO)
TABLA CLIENTES: FOREIGN KEY (VENDEDOR_NO)
                        REFERENCES EMPLEADOS(EMP_NO)
TABLA PEDIDOS: FOREIGN KEY (PRODUCTO_NO)
                        REFERENCES PRODUCTOS(PRODUCTO NO)
TABLA PEDIDOS: FOREIGN KEY (CLIENTE_NO)
                        REFERENCES CLIENTES(CLIENTE NO)
```

Las claves ajenas sirven para relacionar dos tablas entre sí y limitan los valores que puede tomar esa columna a los valores existentes en ese momento en la columna a la que referencian, pudiendo tomar valores existentes en esa columna o nulos.

- Esto nos limita los valores de las claves ajenas no podrán tomar valores que no existan en las columnas referenciadas
- Los valores de las claves primarias no podrán actualizarse si existen filas que los referencian.

Lo vemos con un ejemplo:
La tabla EMPLEADOS tiene una clave ajena DEP_NO que referencia la clave primaria DEP_NO de la tabla DEPARTAMENTOS.
Si existe un departamento con dep_no = 20 y existen filas de empleados con este valor de dep_no

1. No podremos borrar ni modificar el valor de dep_no = 20 de la tabla DEPARTAMENTOS
2. No podremos modificar el valor del campo dep_no de la tabla EMPLEADOS a un valor que no exista en la tala DEPARTAMENTOS.

Estas condiciones se determinaron al diseñar la base de datos y se especificaron e implementaron mediante el lenguaje de definición de datos (DDL). Por lo tanto, todos los comandos de manipulación de datos deberán respetar estas restricciones de integridad ya que en caso contrario el comando fallará y el sistema devolverá un error.

5.2 - Ejemplos de borrados y modificaciones en cascada

Vamos a ver con un ejemplo como funciona el borrado en cascada.
Creamos las tablas `departamentos2` y `empleados2` con una clave ajena con borrado en cascada e insertamos los valores desde las tablas `departamentos` *y* `empleados`

```
mysql> CREATE TABLE departamentos2
    ->  (dep_no INT(4),
    ->    dnombre VARCHAR(14),
    ->    localidad VARCHAR(10),
    ->  CONSTRAINT PK2_DEP PRIMARY KEY (DEP_NO)
    ->  );
Query OK, 0 rows affected (0.09 sec)

mysql> INSERT INTO departamentos2
    ->  SELECT dep_no, dnombre, localidad
    ->          FROM departamentos;
Query OK, 4 rows affected (0.03 sec)
Records: 4  Duplicates: 0  Warnings: 0

mysql> CREATE TABLE empleados2
    -> (emp_no          INT(4),
    ->   apellido       VARCHAR(8),
    ->   oficio         VARCHAR(15),
    ->   director       INT(4),
    ->   fecha_alta     DATE,
    ->   dep_no         INT (2),
    -> CONSTRAINT PK_EMPLEADOS_EMP_NO2 PRIMARY KEY (emp_no),
    -> CONSTRAINT FK_EMP_DEP_NO2 FOREIGN KEY (dep_no)
    ->      REFERENCES departamentos2(dep_no) ON DELETE CASCADE
    -> );
Query OK, 0 rows affected (0.06 sec)

mysql> INSERT INTO empleados2
```

```
    -> SELECT emp_no, apellido, oficio, director,
    ->           fecha_alta, dep_no
    -> FROM empleados;
Query OK, 9 rows affected (0.03 sec)
Records: 9  Duplicates: 0  Warnings: 0

mysql> SELECT * FROM departamentos2;
+--------+---------------+-----------+
| dep_no | dnombre       | localidad |
+--------+---------------+-----------+
|     10 | CONTABILIDAD  | BARCELONA |
|     20 | INVESTIGACION | VALENCIA  |
|     30 | VENTAS        | MADRID    |
|     40 | PRODUCCION    | SEVILLA   |
+--------+---------------+-----------+
4 rows in set (0.00 sec)

SELECT * FROM empleados2;

mysql> SELECT * FROM empleados2;
+--------+----------+------------+----------+------------+--------+
| emp_no | apellido | oficio     | director | fecha_alta | dep_no |
+--------+----------+------------+----------+------------+--------+
|   7499 | ALONSO   | VENDEDOR   |     7698 | 1981-02-23 |     30 |
|   7521 | LOPEZ    | EMPLEADO   |     7782 | 1981-05-08 |     10 |
|   7654 | MARTIN   | VENDEDOR   |     7698 | 1981-09-28 |     30 |
|   7698 | GARRIDO  | DIRECTOR   |     7839 | 1981-05-01 |     30 |
|   7782 | MARTINEZ | DIRECTOR   |     7839 | 1981-06-09 |     10 |
|   7839 | REY      | PRESIDENTE |     NULL | 1981-11-17 |     10 |
|   7844 | CALVO    | VENDEDOR   |     7698 | 1981-09-08 |     30 |
|   7876 | GIL      | ANALISTA   |     7782 | 1982-05-06 |     20 |
|   7900 | JIMENEZ  | EMPLEADO   |     7782 | 1983-03-24 |     20 |
+--------+----------+------------+----------+------------+--------+
9 rows in set (0.00 sec)
```

Ahora vamos a borrar una fila en la tabla departamentos. Si no existiese borrado en cascada, debido a la integridad referencial, no podríamos borrar ningún departamento que tuviese empleados. De esta forma al borrar un departamento se borrarán todos los empleados de ese departamento.

```
mysql> DELETE FROM departamentos2
    -> WHERE dep_no=10;
Query OK, 1 row affected (0.05 sec)

SELECT * FROM departamentos2;

mysql> SELECT * FROM departamentos2;
+--------+---------------+-----------+
| dep_no | dnombre       | localidad |
+--------+---------------+-----------+
|     20 | INVESTIGACION | VALENCIA  |
|     30 | VENTAS        | MADRID    |
|     40 | PRODUCCION    | SEVILLA   |
+--------+---------------+-----------+
3 rows in set (0.00 sec)
```

```
SELECT * FROM empleados2;

mysql> SELECT * FROM empleados2;
+--------+----------+----------+----------+------------+--------+
| emp_no | apellido | oficio   | director | fecha_alta | dep_no |
+--------+----------+----------+----------+------------+--------+
|   7499 | ALONSO   | VENDEDOR |     7698 | 1981-02-23 |     30 |
|   7654 | MARTIN   | VENDEDOR |     7698 | 1981-09-28 |     30 |
|   7698 | GARRIDO  | DIRECTOR |     7839 | 1981-05-01 |     30 |
|   7844 | CALVO    | VENDEDOR |     7698 | 1981-09-08 |     30 |
|   7876 | GIL      | ANALISTA |     7782 | 1982-05-06 |     20 |
|   7900 | JIMENEZ  | EMPLEADO |     7782 | 1983-03-24 |     20 |
+--------+----------+----------+----------+------------+--------+
6 rows in set (0.00 sec)
```

6 - Control de transacciones: COMMIT y ROLLBACK.

Los gestores de bases de datos disponen de dos comandos que permiten confirmar o deshacer los cambios realizados en la base de datos:

- COMMIT: confirma los cambios realizados haciéndolos permanentes .
- ROLLBACK: deshace los cambios realizados.

Cuando hacemos modificaciones en las tablas estas no se hacen efectivas (llevan a disco) hasta que no ejecutamos la sentencia `COMMIT`. Cuando ejecutamos comandos `DDL` (definición de datos) se ejecuta un `COMMIT` automático, o cuando cerramos la sesión.

El comando `ROLLBACK` no permite deshacer estos cambios sin que lleguen a validarse. Cuando ejecutamos este comando se deshacen todos los cambios hasta el último `COMMIT` ejecutado.

Hay dos formas de trabajar con `AUTO_COMMIT` activado (validación automática de los cambios) o desactivado. Si está activado se hace `COMMIT` automáticamente cada sentencia y no es posible hacer `ROLLBACK`. Si no lo está, tenemos la posibilidad de hacer `ROLLBACK` después de las sentencias `DML (INSERT, UPDATE, DELETE)` dejando sin validar los cambios. Es útil para hacer pruebas.

Existe una variable, `AUTO_COMMIT`, que indica la forma de trabajo y tiene el valor 1 si está en modo `AUTO_COMMIT` y 0 si no lo está. Por defecto el MySQL está en modo `AUTO_COMMIT`. Su valor se puede cambia con la sentencia:

```
mysql> SET AUTOCOMMIT = 0;
Query OK, 0 rows affected (0.41 sec)
```

Vamos a verlo con un ejemplo:

```
mysql> SELECT *
    -> FROM inventario;
+-----+----------------+
| num | descripcion    |
+-----+----------------+
|   1 | ARMARIO BLANCO |
|   2 | MESA MADERA    |
|   3 | ORDENADOR      |
|   4 | SILLA GIRATORIA|
+-----+----------------+
4 rows in set (0.00 sec)

mysql> SET AUTOCOMMIT = 0;
Query OK, 0 rows affected (0.41 sec)

mysql> SELECT * FROM inventario;
+-----+----------------+
| num | descripcion    |
+-----+----------------+
|   1 | ARMARIO BLANCO |
|   2 | MESA MADERA    |
|   3 | ORDENADOR      |
|   4 | SILLA GIRATORIA|
+-----+----------------+
```

```
4 rows in set (0.00 sec)

mysql> INSERT INTO inventario
    -> VALUES (NULL,'IMPRESORA');
Query OK, 1 row affected (0.00 sec)

mysql> SELECT * FROM inventario;
+-----+----------------+
| num | descripcion    |
+-----+----------------+
|   1 | ARMARIO BLANCO |
|   2 | MESA MADERA    |
|   3 | ORDENADOR      |
|   4 | SILLA GIRATORIA|
|   5 | IMPRSORA       |
+-----+----------------+
4 rows in set (0.00 sec)

mysql> ROLLBACK;
Query OK, 0 rows affected (0.09 sec)

mysql> SELECT * FROM inventario;
+-----+----------------+
| num | descripcion    |
+-----+----------------+
|   1 | ARMARIO BLANCO |
|   2 | MESA MADERA    |
|   3 | ORDENADOR      |
|   4 | SILLA GIRATORIA|
+-----+----------------+
4 rows in set (0.00 sec)
```

Para hacer los ejercicios y los ejemplos puede ser interesante modificar esta variable y así disponer de la posibilidad de hacer `ROLLBACK`. Cada vez que se inicie una sesión estará en modo `AUTO_COMMIT` que es el valor por defecto y el `ROLLBACk` no será aplicable.

```
mysql> SET AUTOCOMMIT = 1;
Query OK, 0 rows affected (0.41 sec)

mysql> INSERT INTO inventario
    -> VALUES (NULL,'ARCHIVADOR');
Query OK, 1 row affected (0.06 sec)

mysql> SELECT * FROM inventario;
+-----+----------------+
| num | descripcion    |
+-----+----------------+
|   1 | ARMARIO BLANCO |
|   2 | MESA MADERA    |
|   3 | ORDENADOR      |
|   4 | SILLA GIRATORIA|
|   6 | ARCHIVADOR     |
+-----+----------------+
```

```
5 rows in set (0.00 sec)

mysql> ROLLBACK;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT * FROM inventario;
+-----+----------------+
| num | descripcion    |
+-----+----------------+
|   1 | ARMARIO BLANCO |
|   2 | MESA MADERA    |
|   3 | ORDENADOR      |
|   4 | SILLA GIRATORIA|
|   6 | ARCHIVADOR     |
+-----+----------------+
5 rows in set (0.00 sec)
```

3 - Vistas

Podemos definir una vista como una consulta almacenada en la base de datos que se utiliza como una tabla virtual. Se define asociadas a una o varias tablas y no almacena los datos sino que trabaja sobre los datos de las tablas sobre las que está definida, estando así en todo momento actualizada.

3.1 - ¿Qué son las vistas y para qué sirven?.

Se trata de una perspectiva de la base de datos o ventana que permite a uno o varios usuarios ver solamente las filas y columnas necesarias para su trabajo.

Entre las ventajas que ofrece la utilización de vistas cabe destacar:

- Seguridad y confidencialidad: ya que la vista ocultará los datos confidenciales o aquellos para los que el usuario no tenga permiso.
- Comodidad: ya que solamente muestra los datos relevantes, permitiendo, incluso trabajar con agrupaciones de filas como si se tratase de una única fila o con composiciones de varias tablas como si se tratase de una única tabla.
- Independencia respecto a posibles cambios en los nombres de las columnas, de las tablas, etcétera.

Por ejemplo, la siguiente consulta permite al departamento de VENTAS realizar la gestión de sus empleados ocultando la información relativa a los empleados de otros departamentos.

```
SELECT *
FROM EMPLEADOS
WHERE dep_no=30;
```

La siguiente consulta permite a cualquier empleado de la empresa obtener información no confidencial de cualquier otro empleado ocultando las columnas SALARIO y COMISION:

```
SELECT emp_no, apellido, oficio, director, fecha_alta, dep_no FROM
empleados;
```

Para ello crearemos vistas y permitiremos a los usuarios tener acceso a las vistas sin tenerlo de la tabla completa.

3.2 - Creación y utilización de vistas

Como hemos dicho son tablas virtuales resultado de una consulta realizadas sobre tablas ya existentes.
Las vistas no ocupan espacio en la base de datos ya que lo único que se almacena es la definición de la vista. El gestor de la base de datos se encargará de comprobar los comandos SQL que hagan referencia a la vista, transformándolos en los comandos correspondientes referidos a las tablas originales, todo ello de forma transparente para el usuario.

3.2.1 – Formato de la creación de vistas.

Para crear una vista se utiliza el comando CREATE VIEW según el siguiente formato genérico:

```
CREATE VIEW NombreVista
            [(DefiniciónColumna [,DefiniciónColumna....] )]
AS Consulta;
```

Notación: la lista de columnas que define las columnas de la vista es opcional.

donde **NombreVista……** es el nombre que tendrá la vista que se va a crear.
DefinicionColumnas…… permite especificar un nombre y su correspondiente tipo para cada columna de la vista.
Si no se especifica, cada columna quedará con el nombre o el alias correspondiente y el tipo asignado por la consulta.
Consulta…… en una consulta cuyo resultado formará la vista.

El siguiente ejemplo crea la vista emple_dep30 para la gestión de los empleados del departamento 30 mencionada en el apartado anterior.

```
mysql> CREATE VIEW emple_dep30 AS
    ->  SELECT * FROM EMPLEADOS
    ->  WHERE DEP_NO = 30;
Query OK, 0 rows affected (0.55 sec)

CREATE VIEW emple_dep30 AS
SELECT * FROM EMPLEADOS
WHERE DEP_NO = 30;
```

A continuación se muestra la sentencia que crea la vista `datos_emple` que contiene información de todos los empleados ocultando la información confidencial.

```
mysql> CREATE VIEW datos_emple AS
    -> SELECT emp_no, apellido, oficio, director,
                fecha_alta, dep_no
    -> FROM empleados;

Query OK, 0 rows affected (0.00 sec)
```

Las vistas pueden a su vez definirse sobre otras vistas. Si ya tenemos creada las vista `datos_emple`, podríamos crear otra vista sobre ella:

```
mysql> CREATE VIEW datos_emple_10 AS
    ->  SELECT *
    ->  FROM datos_emple
    ->  WHERE dep_no=10;
Query OK, 0 rows affected (0.00 sec)
```

3.2.2 – Utilización de vistas

Una vez definida puede ser utilizada para consultas de igual forma que una tabla.
Con algunas restricciones, todos los formatos de selección vistos para las tablas son aplicables para la selección de filas en las vistas.

Por ejemplo:

```
mysql> SELECT * FROM datos_emple;

+--------+----------+-----------+----------+------------+--------+
| emp_no | apellido | oficio    | director | fecha_alta | dep_no |
```

```
+--------+----------+-----------+----------+------------+--------+
|   7499 | ALONSO   | VENDEDOR  |     7698 | 1981-02-23 |     30 |
|   7521 | LOPEZ    | EMPLEADO  |     7782 | 1981-05-08 |     10 |
|   7654 | MARTIN   | VENDEDOR  |     7698 | 1981-09-28 |     30 |
|   7698 | GARRIDO  | DIRECTOR  |     7839 | 1981-05-01 |     30 |
|   7782 | MARTINEZ | DIRECTOR  |     7839 | 1981-06-09 |     10 |
|   7839 | REY      | PRESIDENTE|     NULL | 1981-11-17 |     10 |
|   7844 | CALVO    | VENDEDOR  |     7698 | 1981-09-08 |     30 |
|   7876 | GIL      | ANALISTA  |     7782 | 1982-05-06 |     20 |
|   7900 | JIMENEZ  | EMPLEADO  |     7782 | 1983-03-24 |     20 |
+--------+----------+-----------+----------+------------+--------+
9 rows in set (0.01 sec)
```

También podemos seleccionar solo algunas columnas y poner una condición

```
mysql> SELECT apellido, director
    ->  FROM datos_emple
    ->  WHERE oficio = 'VENDEDOR';

+----------+----------+
| apellido | director |
+----------+----------+
| ALONSO   |     7698 |
| MARTIN   |     7698 |
| CALVO    |     7698 |
+----------+----------+
3 rows in set (0.00 sec)
```

Pero debe tenerse en cuenta que si al definir la vista hemos indicado nuevos nombres para columnas y expresiones si podremos hacer referencia a ellos en las sentencias de selección, pero si hemos omitido la definición de las columnas y en la sentencia de creación hemos realizado la selección de todas las columnas (creada con select *) solo puede hacerse una selección de todas las columnas de la vista (con *)

La vista `emple_dep30` la creamos sin especificar nuevo nombre para las columnas de la vista y con una sentencia select *. Podemos obtener los datos de la vista si escribimos:

```
mysql> SELECT * FROM emple_dep30;

+-------+----------+---------+---------+------------+---------+--------+--------+
|EMP_NO | APELLIDO | OFICIO  |DIRECTOR | FECHA_ALTA | SALARIO |COMISION| DEP_NO |
+-------+----------+---------+---------+------------+---------+--------+--------+
|  7499 | ALONSO   | VENDEDOR|    7698 | 1981-02-23 | 1400.00 | 400.00 |     30 |
|  7654 | MARTIN   | VENDEDOR|    7698 | 1981-09-28 | 1500.00 |1600.00 |     30 |
|  7698 | GARRIDO  | DIRECTOR|    7839 | 1981-05-01 | 3850.12 |   NULL |     30 |
|  7844 | CALVO    | VENDEDOR|    7698 | 1981-09-08 | 1800.00 |   0.00 |     30 |
+-------+----------+---------+---------+------------+---------+--------+--------+
4 rows in set (0.06 sec)
```

Pero no podemos referenciar las columnas en la selección al recuperar datos de la vista:

```
mysql> SELECT apellido FROM emple_dep30;
ERROR 1054 (42S22): Unknown column 'apellido' in 'field list'
```

3.2.3 - Restricciones para la creación y utilización de vistas

No se puede usar la cláusula `ORDER BY` en la creación de una vista ya que las filas de una tabla no están ordenadas (la vista es una tabla virtual). No obstante, si se puede utilizar dicha cláusula a la hora de recuperar datos de la vista.

Es obligatorio especificar la lista de nombres de columnas de la vista o un alias cuando la consulta devuelve funciones de agrupamiento como `SUM`, `COUNT`, etcétera y posteriormente quiere hacerse referencia a ellas.

Pueden utilizarse funciones de agrupación sobre columnas de vistas que se basan a su vez en funciones de agrupación lo que permite resolver los casos en los que un doble agrupamiento que no está permitido por el estándar. Así creamos una vista con una primera función de agrupación y sobre ella aplicamos la segunda función de agrupación, obteniendo el resultado deseado.

3.2.4 - Ejemplos creación y utilización de vistas

Como hemos dicho una vez creada la vista se puede utilizar como si se tratase de una tabla (observando las restricciones anteriores). Veamos lo que podemos hacer con las vistas con los ejemplos.

1- El siguiente ejemplo crea la vista *datos_vendedores* que muestra solamente las columnas *emp_no, apellido, director, fecha_alta, dep_no,* de aquellos empleados cuyo oficio es VENDEDOR.

```
mysql> CREATE VIEW datos_vendedores
    ->  (num_vendedor, apellido, director, fecha_alta, dep_no) AS
    ->  SELECT emp_no, apellido, director, fecha_alta, dep_no
    ->  FROM empleados
    ->  WHERE oficio = 'VENDEDOR';
Query OK, 0 rows affected (0.00 sec)
```

Los datos accesibles mediante la vista creada serán:

```
mysql> SELECT  *
    ->  FROM datos_vendedores;
+-------------+----------+----------+------------+--------+
| num_vendedor | apellido | director | fecha_alta | dep_no |
+-------------+----------+----------+------------+--------+
|         7499 | ALONSO   |     7698 | 1981-02-23 |     30 |
|         7654 | MARTIN   |     7698 | 1981-09-28 |     30 |
|         7844 | CALVO    |     7698 | 1981-09-08 |     30 |
+-------------+----------+----------+------------+--------+
3 rows in set (0.02 sec)
```

2- También se pueden crear vistas a partir de consultas que incluyen agrupaciones, como en el siguiente ejemplo:

```
mysql> CREATE VIEW resumen_dep1
    ->  (dep_no, num_empleados, suma_salario, suma_comision) AS
    ->  SELECT dep_no, COUNT(emp_no), SUM(salario),
              SUM(IFNULL(comision,0))
    ->  FROM empleados
    ->  GROUP BY dep_no;
Query OK, 0 rows affected (0.01 sec)
```

En estos casos, cada fila de la vista corresponderá a varias filas en la tabla original tal como se puede comprobar en la siguiente consulta:

```
mysql> SELECT *
    ->  FROM resumen_dep1;
+--------+---------------+--------------+---------------+
| dep_no | num_empleados | suma_salario | suma_comision |
+--------+---------------+--------------+---------------+
|     10 |             3 |      9800.50 |          0.00 |
|     20 |             2 |      4750.00 |          0.00 |
|     30 |             4 |      8550.12 |       2000.00 |
+--------+---------------+--------------+---------------+
3 rows in set (0.02 sec)
```

Normalmente la mayoría de las columnas de este tipo de vistas corresponden a funciones de columna tales como SUM, AVERAGE, MAX, MIN, etcétera. Por ello el estándar SQL establece en estos casos la obligatoriedad de especificar la lista de columnas o de alias si posteriormente quiere hacerse referencia a ellas. Aunque algunos gestores de bases de datos permiten saltar esta restricción. No es aconsejable ya que las columnas correspondientes de la vista quedarán con nombres como COUNT(EMP_NO), *SUM(SALARIO), SUM(COMISION)* lo cual no resulta operativo para su posterior utilización.

```
mysql> SELECT dep_no, num_empleados, suma_salario
    -> FROM resumen_dep1;
+--------+---------------+--------------+
| dep_no | num_empleados | suma_salario |
+--------+---------------+--------------+
|     10 |             3 |      9800.50 |
|     20 |             2 |      4750.00 |
|     30 |             4 |      8550.12 |
+--------+---------------+--------------+
3 rows in set (0.14 sec)
```

3- Sobre esta vista, con una función de agrupación, podemos hacer otra función de agrupación, por ejemplo obtener el departamento con mayor suma de salarios para sus empleados:

Creamos la vista resumen_dep2 con dos totales resultado e aplicar funciones de grupo

```
mysql> CREATE VIEW resumen_dep2
    -> AS SELECT dep_no, COUNT(emp_no) "num_empleados",
          SUM(salario) "suma_salario",
    -> FROM empleados
    -> GROUP BY dep_no;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT dep_no, num_empleados, suma_salario
    -> FROM resumen_dep2;
+--------+---------------+--------------+
| dep_no | num_empleados | suma_salario |
+--------+---------------+--------------+
|     10 |             3 |      9800.50 |
```

```
|     20 |             2 |      4750.00 |
|     30 |             4 |      8550.12 |
+--------+---------------+--------------+
3 rows in set (0.00 sec)
```

Y sobre ella volvemos a aplicar una función de grupo para hallar el máximo de las sumas de los salarios

```
mysql> SELECT MAX(suma_salario)
    -> FROM resumen_dep2;
+-------------------+
| MAX(suma_salario) |
+-------------------+
|           9800.50 |
+-------------------+
1 row in set (0.00 sec)
```

Y también podemos obtener el departamento el que pertenece este salario máximo

```
mysql> SELECT dep_no, dnombre
    -> FROM departamentos
    -> WHERE dep_no = (SELECT dep_no
    ->                 FROM resumen_dep2
    ->                 WHERE suma_salario=(SELECT MAX(suma_salario)
    ->                                     FROM resumen_dep2));

+--------+--------------+
| dep_no | dnombre      |
+--------+--------------+
|     10 | CONTABILIDAD |
+--------+--------------+
1 row in set (0.00 sec)
```

4 –Así mismo, se pueden crear vistas que incluyan todas o varias de las posibilidades estudiadas. Por ejemplo la siguiente vista permite trabajar con datos de dos tablas, agrupados y seleccionando las filas que interesan (en este caso todos los departamentos que tengan más de dos empleados):

```
mysql> CREATE VIEW resumen_emp_dep
    ->  (departamento,  num_empleados, suma_salario) AS
    ->   SELECT dnombre, COUNT(emp_no), SUM(salario)
    ->   FROM empleados, departamentos
    ->   WHERE empleados.dep_no = departamentos.dep_no
    ->   GROUP BY empleados.dep_no,dnombre
    ->   HAVING COUNT(*) > 2;
Query OK, 0 rows affected (0.00 sec)


mysql> SELECT  departamento, num_empleados
    -> FROM resumen_emp_dep;
+--------------+---------------+
| departamento | num_empleados |
+--------------+---------------+
| CONTABILIDAD |             3 |
| VENTAS       |             4 |
+--------------+---------------+
2 rows in set (0.02 sec)
```

3.3 - Eliminación de vistas

3.3.1 - Formato de borrado de vistas

La sentencia DROP VIEW permite eliminar la definición de una vista.

```
DROP VIEW   [IF EXISTS] NombreVista
 [RESTRICT | CASCADE ]
```

La cláusula IF EXISTS previene los errores que puedan producirse si no existe la tabla que queremos borrar
La cláusula `CASCADE Y RESTRICT` están permitidas pero no implementada en la versión 5.

3.3.2 - Ejemplos de borrado de vistas

```
mysql> DROP VIEW IF EXISTS resumen_emp_dep;
Query OK, 0 rows affected, 1 warning (0.00 sec)
```

3.3.3 - Borrado de las tablas o vistas asociadas a una vista

Si se borran las tablas a las vistas sobre las que están definidas las vistas la vista se queda invalida (no se borra su definición pero no está utilizable).

```
mysql> SELECT *
    -> FROM datos_emple_10;

+--------+----------+------------+----------+------------+--------+
| emp_no | apellido | oficio     | director | fecha_alta | dep_no |
+--------+----------+------------+----------+------------+--------+
|   7521 | LOPEZ    | EMPLEADO   |     7782 | 1981-05-08 |     10 |
|   7782 | MARTINEZ | DIRECTOR   |     7839 | 1981-06-09 |     10 |
|   7839 | REY      | PRESIDENTE |     NULL | 1981-11-17 |     10 |
+--------+----------+------------+----------+------------+--------+
3 rows in set (0.05 sec)
```

Si borramos `datos_emple` sobre la que está definida `datos_emple_10` esta pasará a estar inválida.

```
mysql> DROP VIEW IF EXISTS datos_emple;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT *
    -> FROM datos_emple_10;
ERROR 1356 (HY000): View 'test.datos_emple_10' references invalid
table(s) or column(s)
```