Tema 3 Creación de bases de datos.

1 Creación de bases de datos.

1.1 SQL: Structured Query Language.

El lenguaje **SQL (Structured Query language, en español Lenguaje de consulta Estructurado)** es un lenguaje para el acceso a bases de datos relacionales. SQL se ha convertido en la actualidad en el lenguaje estándar de bases de datos, soportándolo la mayoría de los sistemas de bases de datos.

Este lenguaje va a permitir consultar, actualizar, insertar y borrar filas de una tabla (**tuplas**), crear modificar y borrar objetos, también nos va a permitir controlar accesos a las bases de datos y a sus objetos, además de garantizar la integridad de los datos.

Nació en los laboratorios de investigación de IBM, que lo adoptó y lo popularizó internacionalmente. Gracias al éxito, numerosas compañías empezaron a comercializar productos basados en SQL, convirtiéndose en el estándar de facto, aunque a partir del 86 se hizo de jure. Desafortunadamente, cada compañía ha desarrollado su propio "dialecto" SQL, por lo que SQL puede variar de un sistema a otro. Dichos cambios son menores para el programador, pero aún así suelen requerir la reescritura de muchas sentencias SQL si deseamos trasladar nuestra base de datos entre sistemas.

Actualmente, el último estándar aprobado por los comités ANSI e ISO es SQL3 o SQL-99, que incorpora nuevos tipos de datos, soporte de expresiones regulares y disparadores (triggers).

Las sentencias SQL se clasifican por su finalidad dando origen a los siguientes sublenguajes:

- LMD (Lenguaje de Manipulación de Datos).
- LDD (Lenguaje de Definición de Datos).
- LCD (Lenguaje de Control de Datos).

En este curso empezaremos por las sentencias LDD y LMD. Los comandos LCD son más propios de los capítulos de administración de MySQL, que se verán al final del curso.

1 1 1 Sentencias DDL.

DDL: Data Description Language. En español: Lenguaje de Definición de Datos, LDD.

Este tipo de comandos definen, modifican o borran las tablas en las que se almacenan los datos y las relaciones entre éstas.

Comando	Descripción
CREATE	Utilizado para crear nuevas tablas, campos e índices.
DROP	Empleado para eliminar tablas e índices.
ALTER	Utilizado para modificar las tablas agregando campos o cambiando la definición de los campos.

1 1 2 Sentencias DML.

DML: Data Manipulation Language. En español: Lenguaje de Manipulación de Datos. Estas se conocen como consultas, LMD.

Existen tres tipos: consultas de selección, consultas de acción y consultas de control de concurrencia.

- Las **consultas de selección** devuelven información de la base de datos sin realizar modificaciones en la misma.
- Las **consultas de acción** modifican los datos de las tablas de la base de datos.
- Las **consultas de control** de concurrencia permiten mantener la integridad de los datos obligando a la ejecución de varias consultas de forma íntegra e inseparable.

Comando	Descripción
SELECT	Utilizado para consultar registros de la base de datos que satisfagan un criterio determinado.
INSERT	Utilizado para cargar lotes de datos en la base datos en una única operación.
UPDATE	Utilizado para modificar los valores de los campos y registros especificados.
DELETE	Utilizado para eliminar registros una tabla de una base de datos.
COMMIT	Graba permanentemente en la base de datos las últimas modificaciones.
ROLLBACK	Permite volver al estado en el que estaba la base de datos tras el último COMMIT.

1 1 3 Sentencias DCL.

DCL: Data Control Language. En español: Lenguaje de Control de Datos, LCD.

Este tipo de comandos se encargan de la seguridad de nuestros datos y el control de usuarios.

Comando	Descripción
GRANT	Se emplea para dar permisos a los usuarios.
REVOKE	Con él podemos eliminar permisos a los usuarios.

1.2 Base de datos.

En MySQL una base de datos es físicamente un conjunto de tablas y sus relaciones entre ellas. Por tanto, antes de crear tablas y rellenarlas es preciso crear antes una base de datos.

1.3 Tipos de Datos.

Es en los tipos de datos donde más se diferencian las diversas versiones de SQL, puesto que dependen de la implementación del gestor. En MySQL hay tres tipos principales: numéricos, de cadenas y de fecha. Aunque dentro de cada uno de ellos hay mucha variedad, que veremos en breve, todos los datos se pueden dividir en estas tres grandes categorías.

Lo mejor será siempre intentar escoger el tipo de dato más pequeño posible para representar nuestros datos, porque reducirá el espacio ocupado por nuestras tablas y acelerará el acceso a las mismas. No obstante, escogerlo demasiado pequeño puede llevar a perder datos.

1 3 1 Numéricos.

Los tipos de datos numéricos sirven para almacenar precios, edades, cantidades, etc. Tienen dos clases: los enteros (sin decimales) y los reales. Todos los tipos admiten dos opciones: UNSIGNED (que prohíbe los negativos) y ZEROFILL(que además de marcar como UNSIGNED rellena el espacio sobrante con ceros en lugar de espacios). A los tipos enteros se les puede indicar que se incrementen en 1 automáticamente en cada nuevo registro con la opción AUTO_INCREMENT.

Tipo	Longitud	Descripción
TINYINT[(M)]	1 byte	Un entero muy pequeño, cuyo rango va de -128 a 127 (o de 0 a 255).
BIT, BOOL	1 byte	Sinónimos de TINYINT(1)
SMALLINT[(M)]	2 bytes	Un valor entero entre -32.768 y 32.767 (o de 0 a 65.535 si está sin signo).
MEDIUMINT[(M)]	3 bytes	Entero entre -8.388.608 y 8.388.607 (o de 0 a 16.777.215).
INT[(M)], INTEGER[(M)]	4 bytes	Entero entre -2.147.483.648 y 2.147.483.647 (o de 0 a 4.294.967.295).
BIGINT[(M)]	8 bytes	Un entero verdaderamente grande, entre -9.223.372.036.854.775.808 y 9.223.372.036.854.775.807. Elmáximo para campos sin signo es de 18.446.744.073.709.551.615.
FLOAT[(M,D)]	4 bytes	Un valor en punto flotante con un rango de -3,402823466E+38 a -1,175494351E-38, y de 1,175494351E-38 a 3,402823466E+38.
DOUBLE[(M,D)], DOUBLE PRECISION[(M,D)], REAL[(M,D)]	8 bytes	Un valor en punto flotante con un rango de -1,7976931348623157E+308 a -2,2250738585072014E-308, y de 2,2250738585072014E-308 a 1,7976931348623157E+308.
DECIMAL[(M[,D)]], DEC[(M[,D)]], NUMERIC[(M[,D)]]	1 byte por cifra	Un número decimal almacenado como una cadena. M indica el númerototal de cifras (eso excluye el signo y el punto decimal) y D los dígitossituados detrás del punto decimal, que por defecto será 0.

1 3 2 Cadenas.

Las columnas tipo cadena sirven para almacenar cualquier dato de tipo carácter, como nombres, direcciones o artículos.

Tipo	Longitud	Descripción
CHAR(M) [BINARY]	M bytes	Almacena M caracteres, como máximo 255.
VARCHAR(M) [BINARY]	1 byte + tamaño del contenido	Almacena tantos caracteres como se le asigne hasta un tamaño máximo de M. No puede superar los 255 caracteres.
TINYTEXT	1 byte + tamaño	Equivalente a VARCHAR(255), aunque más lento.
TEXT	2 bytes + tamaño	Como TINYTEXT, pero con un máximo de 65.535 caracteres.
MEDIUMTEXT	3 bytes + tamaño	Máximo de 16.777.215 caracteres.
LONGTEXT	4 bytes + tamaño	Máximo de 4.294.967.295 caracteres
ENUM('valor1',...)	-	Enumeración. El campo sólo puede tener uno de los valores especificados, NULLo "".
SET('valor1',...)	-	Conjunto de 0 a 64 de los valores especificados.

Existe una serie de campos (TINYBLOB, BLOB, MEDIUMBLOB y LONGBLOB) equivalentes a sus versiones TEXT excepto en que la búsqueda se realiza en ellos diferenciando entre mayúsculas y minúsculas, cosa que no sucede en los campos de texto. Suelen emplearse para almacenar datos binarios (imágenes, por ejemplo).

1 3 3 Fechas.

Las columnas tipo fecha están diseñadas para trabajar con datos sobre el tiempo, como la hora del día o el día de nacimiento.

Tipo	Descripción
DATETIME	Formato YYYY-MM-DD HH:MM:SS. El rango va desde las 00:00:00 del 1 de enero del año 1000 a las 23:59:59 del 31 de diciembre de 9999. No le afecta el efecto 2000, claro.
DATE	Formato YYYY-MM-DD, con el mismo rango que el anterior pero sin horas.
TIME	Formato HH:MM:SS.
YEAR	Solo almacena el año en formato YYYY.
TIMESTAMP	Formato YYYYMMDDHHMMSS.

Este último tipo admite emplear como parámetro un número par, que indicará el número de cifras que se visualizarán con las que se trabajarán, aunque internamente almacene las 14. Así, por ejemplo, TIMESTAMP(6) almacenaría YYYYMM.

1.4 Creación de tablas.

La sentencia CREATE TABLE sirve para crear la estructura de una tabla, no para rellenarla con datos. Nos permite definir las columnas que tiene y ciertas restricciones que deben cumplir esas columnas o campos. Su sintaxis es la siguiente:

```
CREATE TABLE nbtabla (nbcol tipo restricciones) tipo
```

nbtabla	Nombre de la tabla que estamos definiendo
nbcol	Nombre de la columna que estamos definiendo
tipo	Tipo de dato de la columna, todos los datos almacenados en la columna deberán ser de ese tipo
restricciones	Lista de reglas que deberá cumplir la columna
tipo	Tipo de tabla, por defecto MyISAM; nosotros emplearemos InnoDB.

Ejemplo:

Creamos una tabla llamada departamentos compuesta por:

- Una columna llamada departamento.
- Una columna nombre.
- Una columna localidad.

Nota: Para ejecutar comandos SQL es necesario estar conectado a la base de datos sobre la que estamos trabajando. En MySQL eso se consigue con el comando `CONNECT basededatos;`.

```
CREATE TABLE departamentos (departamento INTEGER,
  nombre CHAR,
  localidad CHAR) TYPE=InnoDB;
```

Ejercicio 5
Ahora que ya sabemos como introducir SQL en MyCC vamos a abrir el fichero de ejemplo y a ejecutar el SQL que contiene en MyCC.

```
DROP TABLE IF EXISTS departamentos;
CREATE TABLE departamentos (departamento INT NOT NULL,
  nombre CHAR(15),
  localidad CHAR(15),
  PRIMARY KEY (departamento)) ENGINE=InnoDB;

INSERT INTO departamentos VALUES(10, "Contabilidad", "Salamanca");
INSERT INTO departamentos VALUES(20, "Busqueda", "Avila");
INSERT INTO departamentos VALUES(30, "Ventas", "Madrid");
INSERT INTO departamentos VALUES(40, "Operaciones", "Barcelona");


DROP TABLE IF EXISTS empleados;
CREATE TABLE empleados (
  codigo INT NOT NULL,
  apellido VARCHAR(20) NOT NULL,
  trabajo VARCHAR(20) NOT NULL,
  depende INT,
  fechaingreso DATE NOT NULL,
  salario INT NOT NULL,
  comision INT,
  departamento INT,
  PRIMARY KEY (codigo),
  INDEX (departamento),
  FOREIGN KEY (departamento) REFERENCES departamentos(departamento) ON DELETE
SET NULL ON UPDATE CASCADE,
  INDEX (depende),
  FOREIGN KEY (depende) REFERENCES empleados(codigo) ON DELETE RESTRICT ON
UPDATE RESTRICT
) ENGINE=InnoDB;

INSERT INTO empleados VALUES
  (7839, "King", "presidente", NULL, "1981/11/17", 5000,NULL , 10);
INSERT INTO empleados VALUES
  (7566, "Janeiro", "programador", 7839, "1981/04/02", 4016, NULL, 20);
INSERT INTO empleados VALUES
  (7902, "Fraga", "analista", 7566, "1981/12/03", 3000, NULL, 20);
INSERT INTO empleados VALUES
  (7698, "Blasco", "programador", 7839, "1981/05/01", 3848, NULL, 30);
INSERT INTO empleados VALUES
  (7369, "Sanchez", "administrativo", 7902, "1980/12/17", 1080, NULL, 20);
INSERT INTO empleados VALUES
  (7499, "Almendro", "vendedor", 7698, "1981/02/20", 2160, 300, 30);
INSERT INTO empleados VALUES
  (7521, "Willians", "vendedor", 7698, "1981/02/22", 1688, 500, 30);
INSERT INTO empleados VALUES
  (7654, "Martin", "vendedor", 7698, "1981/09/28", 1688, 1400, 30);
INSERT INTO empleados VALUES
  (7782, "Carrera", "programador", 7839, "1981/06/09", 3308, NULL, 10);
INSERT INTO empleados VALUES
  (7788, "Santos", "analista", 7566, "1982/12/09", 3000, NULL, 20);
INSERT INTO empleados VALUES
  (7844, "Turner", "vendedor", 7698, "1981/09/08", 2025, 0, 30);
INSERT INTO empleados VALUES
  (7876, "Armentero", "administrativo", 7788, "1983/01/12", 1485,NULL , 20);
INSERT INTO empleados VALUES
  (7900, "Jordan", "administrativo", 7698, "1981/12/03", 1282, NULL, 30);
INSERT INTO empleados VALUES
  (7934, "Morales", "administrativo", 7782, "1982/01/23", 1755, NULL, 10);
```

1 4 1 Restricciones: NOT NULL y UNIQUE.

En las instrucciones ALTER TABLE (que veremos más adelante) y CREATE TABLE se pueden definir restricciones que se podrán eliminar más tarde, cuando queramos, sin tener que borrar la columna. Como restricciones tenemos la de clave primaria (clave principal), la de índice único (sin duplicados), la de valor no nulo, y la de clave foránea. Podemos poner varias restricciones, tantas como sean necesarias.

La restricción NOT NULL.

La restricción NOT NULL indica que la columna no podrá contener un valor nulo, es decir, que se deberá rellenar obligatoriamente y con un valor válido.

Sintaxis:

```
campo tipo NOT NULL
```

La restricción UNIQUE.

La restricción UNIQUE sirve para definir un índice único sobre la columna. Un índice único no permite valores duplicados, de modo que si una columna tiene definida una restricción de este tipo no podrá haber dos filas con el mismo valor en esa columna. Se suele emplear para que el sistema se asegure de que no se añaden valores que ya existen.

Sintaxis:

UNIQUE [nombre] (campo)

Esto se añade en el CREATE TABLE detrás de todas las definiciones de campos. Se le puede poner nombre a la restricción, lo que facilita la tarea de cambiar o eliminar posteriormente la restricción, haciendo referencia a la misma por su nombre. No obstante, si no le especificamos ninguno, MySQL le pondrá como nombre por defecto el nombre del campo. Si una restricción afecta a varios campos, cogerá el nombre del primero.

1 4 2 Claves primarias y extranjeras. PRIMARY KEY y FOREING KEY.

La cláusula PRIMARY KEY se utiliza para definir la columna como clave principal (o primaria) de la tabla. Esto supone que la columna no puede contener valores nulos ni puede haber valores duplicados en esa columna, es decir que dos filas no pueden tener el mismo valor en esa columna. En una tabla no puede haber varias claves principales, por lo que no podemos incluir la cláusula PRIMARY KEY más de una vez.

Sintaxis:

```
PRIMARY KEY (campo)
```

La cláusula FOREIGN KEY es una columna o conjunto de columnas que contiene un valor que hace referencia a una fila de otra tabla. Genera un índice externo (toma como valor del índice campos contenidos en otras tablas). Normalmente el campo externo al que hace referencia es una clave primaria de otra tabla.

Sintaxis:

```
FOREIGN KEY (campol, campo2, …)
  REFERENCE5 tabla_externa (campo_externol, campo_externo2,...)
```

Lo normal, no obstante, es que sólo haya un campo externo. Este tipo de restricciones tiene más opciones, que veremos más adelante en el apartado de integridad referencial, y sólo funcionan en tablas InnoDB.

1.4.2.1 Ejemplo:

Creamos una tabla llamada clientes compuesta por:

- Una columna llamada codcliente de tipo entero definida como clave principal.
- Una columna nombre que puede almacenar hasta 20 caracteres alfanuméricos y no puede contener valores nulos.
- Una columna apellido de hasta 25 caracteres que no podrá contener valores repetidos.
- Una columna edad de tipo entero sin ninguna restricción.
- Una columna departamento de tipo entero clave foránea que hace referencia a valores de la clave principal de la tabla departamentos.

```
CREATE TABLE clientes (
  codcliente INT N0T NULL,
  nombre CHAR(20) NOT NULL,
  apellido CHAR(25),
  edad INTEGER,
  departamento INT,
  PRIMARY KEY (codcliente),
  UNIQUE (apellido),
  INDEX (departamento),
  FOREIGN KEY (departamento) REFERENCES departamentos(departamento)
) TYPE=InnoDB;
```

1 4 3 Ejercicios.

Ejercicio: 6.

Creamos una tabla llamada estudiantes compuesta por:

• Una columna llamada id que será de tipo autonumérico y ademas será la clave principal de la tabla.

• Una columna nombre de tipo carácter.

• Una columna apellido de tipo carácter.

• Una columna fnacimiento de tipo fecha.

• Una columna observaciones que será de tipo texto.

• Una columna cuota que será de tipo entero pequeño.

```
CREATE TABLE estudiantes (
 id INT NOT NULL AUTO_INCREMENT,
 nombre VARCHAR(20),
 apellido VARCHAR(20),
 fnacimiento DATETIME,
 observaciones TEXT,
 cuota TINYINT,
 PRIMARY KEY (id)
) TYPE=InnoDB;
```

Ejercicio: 7.

Creamos una tabla llamada formasenvio compuesta por:

- Una columna llamada idenvio que será de tipo autonumérico y además será la clave principal de la tabla.
- Una columna descripción de tipo carácter que no admitirá valores repetidos.
- Una columna forma de tipo carácter y no podrá contener valores nulos.
- Una columna nombreservicio de tipo texto.

```
CREATE TABLE formasenvio (
 idenvio INT NOT NULL AUTO_INCREMENT,
 descripcion VARCHAR (100),
 forma VARCHAR(20) NOT NULL,
 nombreservicio TEXT,
 PRIMARY KEY (idenvio),
 UNIQUE (descripcion)
) TYPE=InnoDB;
```

1.5 Borrado. DROP TABLE.

La sentencia DROP TABLE sirve para eliminar una tabla. La tabla y todos sus registros serán borrados de la base de datos. Algunos gestores de bases de datos exigen que la tabla se encuentre vacía para poder realizar esta operación, pero no es lo más habitual y, de hecho, MySQL no lo hace.

No se puede eliminar una si el borrado infringe las reglas de integridad referencial que veremos más adelante (si es referenciada como clave extranjera por otra tabla y tiene registros relacionados). La sintaxis es la siguiente:

```
DROP TABLE [IF EXISTS] tabla;
```

La cláusula IF EXISTS permite evitar que MySQL de un error si la tabla existe ya. Esto es especialmente útil a la hora de hacer copias de seguridad en formato SQL.

Hay que tener cuidado con esta sentencia, ya que si se elimina una tabla no podrá ser recuperada. Hay que estar seguro de lo que vamos a hacer.

Ejemplo:

En el ejemplo se elimina de la base de datos la tabla clientes. Una vez comprobado que se borra, aconsejo volver a ejecutar el ejercicio en que se creaba dicha tabla pues la utilizaremos en el futuro.

```
DROP TABLE clientes;
```

Ejercicio 8:

Borrar la tabla formasenvio que hemos creado anteriormente.

```
DROP TABLE formasenvio;
```

De nuevo, recordad volver a crear la tabla una vez halláis comprobado que se borra.

1.6 Modificación. ALTER TABLE.

La sentencia **ALTER TABLE** sirve para modificar la estructura de una tabla existente. Mediante esta instrucción podemos añadir columnas nuevas o eliminarlas. También nos permite añadir nuevas restricciones o borrar las existentes.

1 6 1 Añadir columnas o restricciones.

Para añadir una columna se utiliza **`ADD [COLUMN]`** y nombre de la columna que queremos añadir seguido por el tipo de dato y las restricciones.

Ejemplo:

Añade a la tabla empleados una nueva columna llamada edad de tipo Byte.

```
ALTER TABLE empleados ADD Edad TINYINT;
```

Para añadir una nueva restricción en la tabla podemos utilizar la cláusula **`ADD restriccion`**.

Ejemplo:

Añadir a la tabla empleados un índice único (sin duplicados) sobre la columna apellidos.

```
ALTER TABLE empleados ADD UNIQUE (apellido);
```

Para borrar una columna se utiliza **`DROP [COLUMN]`** y el nombre de la columna que queremos borrar. Se perderán todos los datos almacenados en la columna.

Ejemplo:

Eliminar la columna edad de la tabla clientes.

```
ALTER TABLE clientes DROP edad;
```

Para modificar la definición entera de una columna se utiliza **`MODIFY [COLUMN]`**.

Ejemplo:

Modificar el campo edad de la tabla empleados para que sea un entero corto.

```
ALTER TABLE empleados MODIFY Edad SMALLINT;
```

1 6 2 Eliminar restricciones.

El modo de eliminar una restricción depende del tipo de restricción. Para eliminar una clave primaria, como sólo debe haber una, basta con emplear **DROP** PRIMARY KEY, sin más parámetros.

Ejemplo:

Elimina la clave primaria de la tabla clientes.

```
ALTER TABLE clientes DROP PRIMARY KEY;
```

En otros casos basta con utilizar la cláusula **DROP INDEX** y el nombre de la restricción que queremos borrar.

Recordad que, si no hemos puesto ninguno, coincidirá con el nombre de la columna. En este caso los datos almacenados ni se modifican ni se pierden.

Ejemplo:

Con esta sentencia borramos el índice único sobre la columna apellidos.

```
ALTER TABLE empleados DROP INDEX apellido;
```

Aún no se pueden modificar claves extranjeras por limitaciones de InnoDB. Pero podemos borrarlas y añadirlas de nuevo:

Ejemplo:

Elimina la clave extranjera de la tabla clientes. Para averiguar qué nombre tiene hay que hacer antes un SHOW CREATE TABLE clientes; y emplear el nombre interno que viene tras CONSTRAINT.

```
ALTER TABLE clientes DROP FOREIGN KEY O_243;
```

Ejercicio: 9.

Añadir a la tabla estudiantes una columna llamada comentario de tipo texto.

```
alter table estudiantes add comentario observaciones text;
```

Ejercicio: 10.

Borrar la columna observaciones de la tabla estudiantes.

1.7 Tipos de tabla.

MySQL es una suerte de contenedor. Interpreta el SQL y le pasa las peticiones correspondientes a módulos que trabajan con las tablas. Es debido a esta estructura modular que dispone de una característica única: soporta varios tipos de tablas. MySQL llamará a un módulo u otro dependiendo del tipo concreto de la tabla con la que necesite interactuar. Hay dos tipos de tabla que soportan transacciones pero son más lentas (InnoDB y BDB) y cuatro de menores prestaciones pero que son más rápidas.

1 7 1 ISAM.

Las tablas ISAM (Indexed Sequential Access Method) son el tipo de tabla más antiguo de MySQL. Aunque seguirán siendo incluidas en los binarios de MySQL hasta las versiones 4,1.x, están siendo

sustituidas por las tablas MyISAM, cuyos índices ocupan menos espacio. Eso significa que emplean menos memoria en los SELECT, pero más potencia de procesador.

Tienen las siguientes características:

- Almacenan en el fichero `nombretabla.isd` los datos y en `nombretabla.ism` los índices.
- Esos ficheros no se pueden copiar de un sistema operativo a otro y que funcionen, de modo que ese sistema de backup no se puede emplear.

Dado que este tipo de tablas va a dejar de existir en breve, se aconseja convertir a MyISAM todas las tablas de este tipo:

```
ALTER TABLE nombretabla TYPE=MyISAM
```

1 7 2 MyISAM.

Las tablas MyISAM tienen una estructura similar a las tablas ISAM, aunque las extensiones de los ficheros son `myd` para los datos y `myi` para los índices. Existen tres subtipos de tablas: estáticas, dinámicas o comprimidas. MySQL decide automáticamente entre las dos primeras. Si en la definición de los campos hay alguno de longitud variable (como VARCHAR, BLOB o TEXT) la tabla será dinámica.

En las tablas estáticas (o de longitud fija) todos los registros tienen el mismo tamaño. Eso las hace muy rápidas (el gestor siempre sabrá que el segundo registro está en el carácter número 11), fácilmente reconstruibles tras algún desastre, pero requieren mucho espacio (en este caso usan 30 caracteres cuando sólo harían falta 19).

En las tablas dinámicas todas las columnas de tipo carácter son de anchura variable (a menos que tengan menos de 4 bytes). Normalmente ocupan mucho menos que las fijas. No obstante, de vez en cuando hay que desfragmentarlas, cual disco duro de Windows y no son fáciles de recuperar tras un desastre (especialmente si estaban muy fragmentadas en ese momento).

Tablas comprimidas

Son tablas de sólo lectura que emplean muy poco espacio. Para emplearlas necesitaremos tener instalado `myisampack` y emplear el parámetro `ROW_FORMAT="compressed"` detrás del tipo de tabla. No resultan demasiado útiles.

1 7 3 MERGE.

Las tablas MERGE son un tipo especial de tablas, pues se forman a base de la unión de dos tablas MyISAM idénticas.

Ejemplo:
Creamos una tabla MERGE. Si después de hacerlo probamos a insertar registros en las dos primeras tablas con MyCC y examinamos el contenido de `unidas`, veremos que tiene los registros insertados en ambas tablas.

```
CREATE TABLE aunir1 (
 id INT NOT NULL,
 campo CHAR(10),
 PRIMARY KEY (id)
) TYPE = MyISAM;

CREATE TABLE aunir2 (
 id INT NOT NULL,
 campo CHAR(10),
 PRIMARY KEY (id)
) TYPE = MyISAM;

CREATE TABLE unidas (
 id INT NOT NULL,
 campo CHAR(10),
 PRIMARY KEY (id)
) TYPE = MERGE UNION=(aunir1, aunir2);
```

Principalmente se utilizan para tablas muy grandes, que superen el tamaño máximo de archivo permitido por el sistema operativo. También pueden emplearse por seguridad, dejando en una tabla de sólo lectura los contenidos antiguos y arriesgándose a que falle en una inserción o modificación otra, más pequeña. No obstante, las consultas pueden ser más lentas.

Los registros estarán físicamente en las tablas MyISAM, de modo que podemos realizar modificaciones y borrados en la tabla MERGE con la tranquilidad de que se realizarán sobre la tabla correcta. No obstante, al insertar MySQL no tiene manera de saber en cual de las dos tablas deseamos insertar físicamente el registro. Esto se soluciona añadiendo detrás de UNION la cláusula INSERT_METHOD que puede ser FIRST (las inserciones se realizarán en la primera tabla de la unión), LAST (en la última) o NO (no se harán). La definición anterior quedaría:

```
CREATE TABLE unidas (
 id INT NOT NULL,
 campo CHAR(10),
 PRIMARY KEY (id)
) TYPE = MERGE
UNION=(aunir1, aunir2)
INSERT_METHOD=LAST;
```

1 7 4 HEAP.

Las tablas HEAP se almacenan en memoria y tienen índices de tipo hash. Son, por tanto, las más rápidas pero si se produce un problema o, simplemente, se necesitan parar MySQL, los datos que contiene se perderán. Normalmente no almacenarán muchos datos, a no ser que tengamos una cantidad ingente de RAM.

El uso normal de este tipo de tablas es la creación de tablas de acceso rápido a las que pediremos los datos, mientras las inserciones y las modificaciones se realizan en otra tabla algo más seguro que se almacene en el disco duro. Sirven como una suerte de tablas caché. Para poder crearlas emplearemos una sintaxis específica:

```
CREATE TABLE prueba_heap TYPE=HEAP SELECT * FROM departamentos;
```

Ahora podemos ver con MyCC que tiene la misma estructura y las mismas columnas que la tabla original.

1 7 5 InnoDB.

Las tablas InnoDB **soportan transacciones** (lo que significa que podremos utilizar en ellas COMMIT y ROLLBACK). Las tablas MyISAM son completamente bloqueadas cuando se inserta.

Durante la fracción de segundo que dura la inserción no se puede realizar nada más en la tabla. InnoDB emplea bloqueo a nivel de registro, de modo que sólo ese registro estará bloqueado y no toda la tabla.

Las tablas InnoDB no se almacenan en ficheros divididos por directorios, cada uno con el nombre de la base de datos a la que pertenecen: todas las tablas e índices se almacenan en el llamado espacio de tabla, para el que se usa uno o varios ficheros, dependiendo del tamaño total de todas las tablas InnoDB. De ese modo. Estos ficheros tendrán el nombre ibdata1, ibdata2, etc..

Las tablas InnoDB son también las únicas que, de momento, disponen de la posibilidad de emplear claves extranjeras.

1 7 6 BDB.

BDB son las siglas de Berkeley Database (pues este tipo de tablas se creó en esa Universidad). Soporta transacciones, como InnoDB, pero no claves extranjeras, al igual que MyISAM. Actualmente, el soporte de BDB en MySQL todavía se considera en estado beta. Ambos productos son ya antiguos, es el interfaz entre ellos el que es relativamente nuevo. No obstante, comprueba la documentación de tu versión de MySQL por si dicho problema ya está solucionado. Las tablas BDB se crean de esta manera:

```
CREATE TABLE prueba_bdb ( id INT NOT NuLL, campo cHAR(1o), PRIMARY KEY (id) )
TYPE = BDB;
```

Esto creará un fichero prueba_bdb.db donde se almacenarán tanto los datos como los índices.

1 7 7 Escoger tipo de tabla.

Seguramente no existe un tipo de tabla universal para emplear en todas las situaciones y proyectos. Por sus características, seguramente nos veremos tentados a emplear siempre InnoDB, no obstante también la rapidez es algo que tendremos que tener en cuenta. Estas son unas medidas de rendimiento basadas en tablas simples:

	Inserciones por segundo	Consultas por segundo
HEAP	4150	2395
MyISAM	2968	2079
InnoDB	3042	2051
BDB	2602	2107

Por prestaciones, vemos que las tablas HEAP son las mejores, pero no parecen suficientemente rápidas como para justificar las complicaciones que debemos afrontar al usarlas. Las demás presentan un comportamiento parecido, pese a que los manuales de MySQL recomiendan tablas MyISAM cuando hay muchas más consultas que inserciones e InnoDB cuando está la cosa más equilibrada o hay más inserciones.

1.8 Integridad referencial.

La **integridad referencial** se define como el conjunto de mecanismos para conservar y garantizar de forma automática la integridad de todos los datos de los sistemas de gestión de bases de datos. Define las *reglas que claves primarias y claves foráneas* deben cumplir.

Las tablas que forman una base de datos pueden ser de diferentes tipos, dependiendo de las relaciones que tengan con otras tablas:

Tabla referenciada: su clave primaria es referenciada por una o más claves foráneas de la misma o diferente tabla.

Tabla dependiente: Tiene al menos una clave foránea.

Tabla independiente: No tiene ninguna clave primaria referenciada ni foránea.

Tabla autorreferenciada: Su clave primaria está referenciada por una clave foránea definida en esta misma tabla.

Al diseñar el modelo conceptual de una base de datos se definen las relaciones existentes entre las entidades. Durante el diseño a nivel físico del modelo relacional se definen físicamente las relaciones de forma que el gestor de bases de datos asegure la integridad de los datos de acuerdo a dichas relaciones. En estas relaciones se establecen las restricciones referenciales. Es decir, establecemos las acciones que el gestor debe tomar al efectuar las validaciones entre la clave primaria y la clave foránea referente como consecuencia de una operación "contra" la base de datos.

Las primeras restricciones que deberemos tener en cuenta son el requisito de que *la clave primaria debe ser única* y *no puede contener valores nulos* y que *una clave foránea debe tener el mismo tipo de datos que la clave primaria a la que referencia*.

1 8 1 Reglas y opciones.

Los sistemas gestores de bases de datos siguen una serie de *reglas* para soportar la integridad de los datos mediante la integridad referencial:

- **Insert.**
- **Delete.**
- **Update.**

Estas reglas se aplican al realizar las operaciones SQL con el mismo nombre. Dentro de cada una de las reglas existen unas *opciones* de integridad, impuestas por el SGBD en algunos casos:

- **Restrict.**
- **Set NulI.**
- **Cascade.**

Estas operaciones se aplican en operaciones realizadas sobre claves primarias y foráneas. No todos los SGDB admiten todas las opciones en todas las reglas. MySQL emplea **restrict** sobre **insert** y **update**, y admite las tres en **delete**.

1.8.1.1 Opciones.

Opción RESTRICT.

Con esta opción se impide que se propague la operación a otras tablas relacionadas con la afectada. Si la operación se aplica sobre:

- **La tabla referenciada**: Esta opción sólo es aplicable a las operaciones (reglas de integridad) de *actualización* y *borrado*. No se permitirá actualizar o borrar una clave primaria que exista en la tabla dependiente como clave foránea.
- **La tabla dependiente**: Esta opción sólo es aplicable a las operaciones (reglas de integridad) de *actualización* e *inserción*. Sólo se permitirá insertar o modificar una clave foránea si existe previamente en la tabla referenciada como clave primaria.

Opción SET NULL.

Se aplica siempre sobre la clave foránea. Con esta opción se ponen a valor *nulo* los valores de las claves foráneas que coincidan con la clave primaria que va a ser *modificada* o *borrada*.

Opción CASCADE.

Con esta opción se permite la propagación de la operación a las tablas relacionadas con la tabla afectada. Se aplica sobre las tablas dependientes, nunca sobre las tablas referenciadas. Esta opción se aplica sobre las operaciones (reglas de integridad) de *actualización* y *borrado*.

Cuando se actualiza o borra una fila de la tabla referenciada, se actualizan o borran de forma automática todas las filas de la tabla dependiente cuya clave foránea coincida con el valor de la clave primaria de la fila sobre la que se realizó la operación. Hay que tener "cuidado" con esta opción, ya que se pueden producir borrados masivos no deseados.

1.8.1.2 Reglas.

Regla INSERT.

Sólo se aplica en las filas que se insertan en una tabla dependiente. Se aplica automáticamente, por lo que no se puede aplicar ninguna de las opciones de integridad. Cuando se inserta una fila en una tabla dependiente, la clave foránea debe ser:

- □ Nula si está permitido en alguno de los atributos componentes de dicha clave
- □ Igual a un valor de una clave primaria a la que referencia
- □ Puede ser no única

Regla DELETE.

En esta regla son aplicables todas las opciones de integridad vistas anteriormente. Se aplica en las operaciones de borrado efectuadas sobre una tabla referenciada. Al intentar borrar una fila de una tabla referenciada, se comprobarán todas las condiciones impuestas por la opción de integridad definida, y sólo se efectuará el borrado si dichas condiciones se cumplen. A continuación veremos las diferentes opciones que se aplican a esta regla.

Regla UPDATE.

Se aplica en operaciones realizadas sobre tablas referenciadas y tablas dependientes. Si se trata de una tabla referenciada, se aplica automáticamente la opción RESTRICT, y no se puede aplicar ninguna de las otras opciones de integridad. Sin embargo, si se trata de una tabla dependiente, se pueden aplicar todas las opciones de integridad definidas.

1.8.1.3 DELETE.

DELETE RESTRICT.

No se permite borrar una fila de una tabla referenciada si existen filas en una tabla dependiente de la anterior, cuya clave foránea coincida con el valor de la clave primaria de la fila que se quiere borrar.

Para poder realizar esta operación hay 2 opciones:

- □ Borrar las filas dependientes y después la fila referenciada
- □ Actualizar los valores de la clave foránea con otros diferentes a los de la clave primaria a borrar y después borra la fila referenciada

DELETE CASCADE.

Al eliminar una fila de la tabla referenciada se produce un borrado en cadena en la tabla dependiente, es decir, se borran todas las filas dependientes (aquellas cuya clave foránea coincida con la clave primaria de la tabla a la que referencian). Hay que tener cuidado con esta opción ya que se pueden producir borrados masivos en las tablas implicadas.

DELETE SET NULL.

Al borrar una fila de una tabla referenciada con una restricción referencial de este tipo se producen las siguientes acciones:

- □ Borrado de la fila de la tabla referenciada.
- □ Se pone a nulo el valor de las claves foráneas de las filas dependientes.

Una vez realizadas estas operaciones, las filas afectadas de la tabla dependiente no sufren validaciones de integridad hasta que el valor de su clave foránea se actualice con un valor que corresponda al de alguna clave primaria de la tabla referenciada.

Cuando una operación depende de varias restricciones referenciales falla si no se cumple alguna de ellas. Las opciones de integridad SET NULL y CASCADE no provocan fallo, pero RESTRICT sí que puede incumplirse. Por lo tanto, esta última opción es la que 'decide' si la operación se realiza o no.

Nota: El campo.

1.8.1.4 UPDATE.

UPDATE RESTRICT.

No se permite actualizar el valor de la clave primaria de una tabla referenciada mientras existan claves foráneas con el mismo valor en tablas dependientes.

Para realizar esta operación hay 2 opciones:

- □ Borrar las filas dependientes con dicho valor de clave foránea.
- □ Actualizar el valor de las claves foráneas con un valor diferente al de la clave primaria y después actualizar el valor de dicha clave primaria.

UPDATE CASCADE.

Al actualizar el valor de una clave primaria de una tabla referenciada, se propaga el cambio a las claves foráneas de las tablas dependientes.

UPDATE SET NULL.

Al actualizar la clave primaria de una fila referenciada, el funcionamiento es el mismo que el de la regla DELETE con la opción SET NULL, excepto que en lugar de borrar la fila de la tabla referenciada, se actualiza la clave primaria con un nuevo valor.

1 8 2 SQL.

Por defecto se asumen las opciones RESTRICT tanto para actualización como para borrado. Si deseamos cambiarlas deberemos indicarlo al final de la especificación de clave extranjera:

```
FOREIGN KEY (campo)
  REFERENCES tabla_externa (campo_externo)
    [ON DELETE {CASCADE | SET NULL | NO ACTION
      | RESTRICT}]
    [ON UPDATE {CASCADE | SET NULL | NO ACTION
      | RESTRICT}]
```

NO ACTION tiene el efecto de eliminar cualquier comprobación de integridad referencial y es similar a no definir clave extranjera. En MySQL 4.0.13, las claves autorreferenciadas no parecían poder emplear nada más que RESTRICT y NO ACTION en las modificaciones, aunque admiten todas las opciones también CASCADE en los borrados.

Ejemplo:

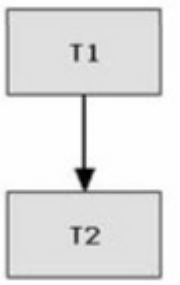
Cambiamos la definición de la clave extranjera de la tabla clientes. Como el nombre es interno, debemos averiguarlo pidiendo la estructura de la tabla. Suponemos que la clave extranjera está definida con CONSTRAINT 'O_290' FOREIGN KEY ('departamento') REFERENCES 'departamentos' ('departamento'). Podéis insertar datos en MyCC para probar las nuevas restricciones.

```
SHOW CREATE TABLE clientes;
ALTER TABLE clientes DROP FOREIGN KEY 0_290;
ALTER TABLE clientes ADD FOREIGN KEY (departamento)
   REFERENCES departamentos(departamento)
   ON DELETE SET NULL ON UPDATE CASCADE;
```

1 8 3 Tipos de estructuras referenciales.

Una **estructura referencial** es un conjunto de tablas y relaciones entre ellas de modo que cada tabla es referenciada o dependiente de otra u otras del mismo conjunto. Hay 3 tipos de estructuras:

- □ Lineal.
- □ Cíclica.
- □ Cíclica-autorreferencial.

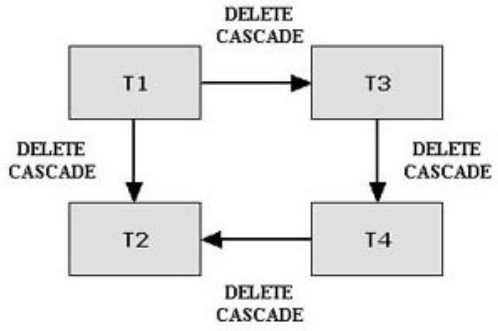


Vamos a ver estas estructuras centrándonos en la regla DELETE. Una tabla está conectada por borrado con otra cuando el borrado de las filas de una implica el borrado de las filas de otra. Es decir, una tabla referenciada está conectada con una tabla dependiente, ya que ésta tiene una clave foránea que referencia a la clave primaria de la tabla referenciada,

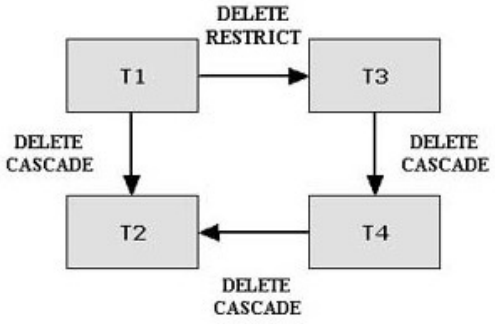
En el gráfico de la derecha, la tabla T2 es una tabla dependiente de la tabla T1, es decir, que la tabla T2 tiene una clave foránea que referencia a la clave primaria de la tabla T1. se diría que la tabla T2 está conectada por borrado con la tabla T1.

1.8.3.1 Estructura lineal.

Esta estructura está formada por un conjunto de tablas relacionadas entre sí sin cerrar ningún ciclo. Varias tablas relacionadas entre sí mediante una estructura lineal deben tener todas la misma regla DELETE definida. Por ejemplo:



Estructura Válida



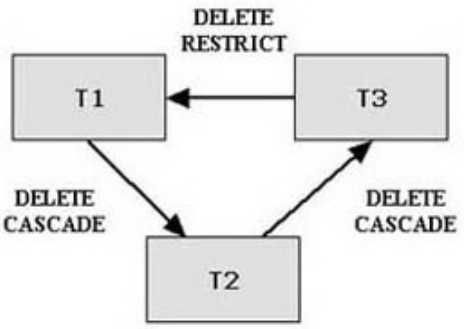
Estructura Inválida

1.8.3.2 Estructura cíclica.

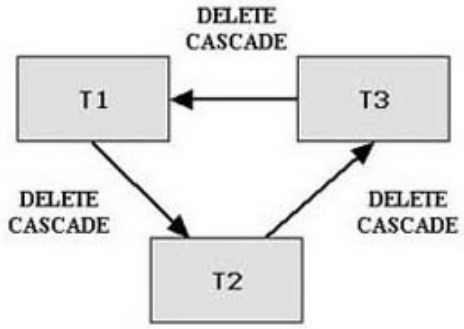
En esta estructura, una o varias tablas acaban conectándose consigo mismas a través de las relaciones que tienen con otras tablas. Hay que tener en cuenta las siguientes precauciones a la hora de enfrentarse a una estructura de este tipo:

- Una tabla no puede estar conectada por borrado consigo misma.
- El camino de borrado se debe bloquear con una opción SET NULL o RESTRICT.

Con estas medidas se consigue evitar situaciones en las que el borrado de una fila de una tabla referenciada provoque el borrado masivo de filas de la tabla dependiente y éstas provoquen, a su vez, el borrado masivo de filas de la tabla referencial, con lo que se produce un borrado indeseado de las tablas.

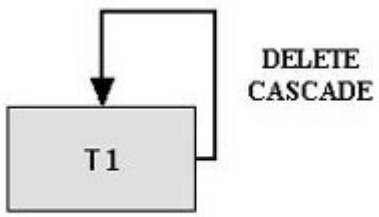


Estructura Válida



Estructura Inválida

1.8.3.3 Estructura cíclica-autorreferenciada.



Estructura Válida

Este tipo de estructuras se dan cuando una tabla se relaciona consigo misma directamente. Se crea un ciclo en el que interviene una sola tabla. En este tipo de estructuras la regla DELETE siempre tiene que tener la opción CASCADE, ya que de lo contrario se pueden dar resultados diferentes según el gestor de bases de datos, y no se puede permitir que una misma operación produzca resultados diferentes.

Ejercicio: 11.

Vamos a coger el diseño y el paso a tablas de la base de datos de comercio electrónico, tal y como lo resolvimos en el ejercicio correspondiente, y vamos a crear las tablas correspondientes. Las tablas deberán ser de tipo InnoDB para poder tener claves extranjeras.

Solución:

La solución propuesta incorpora antes de la creación de tablas un borrado "si existe", Es lo ideal para hacer pruebas. También incluye, detrás del tipo de tabla, la cláusula CQMMENT, que sirve para explicar qué contiene una tabla.

```
DROP TABLE IF EXISTS categorias;
CREATE TABLE categorias (
  id int NOT NULL auto_increment,
  nombre varchar(50) NOT NULL,
  descripcion varchar(200) NOT NULL,
  PRIMARY KEY  (id)
) TYPE=InnoDB COMMENT='Categorías de productos';

DROP TABLE IF EXISTS productos;
CREATE TABLE productos (
  id int NOT NULL auto_increment,
  idcategoria int NOT NULL,
  nombre varchar(50) NOT NULL,
  descripcion varchar(150) NOT NULL,
  precio double NOT NULL,
  enportada tinyint(1) NOT NULL,
  importancia mediumint,
  PRIMARY KEY  (id),
  INDEX (idcategoria),
  FOREIGN KEY (idcategoria) REFERENCES categorias(id) ON DELETE CASCADE ON UPDATE CASCADE
) TYPE=InnoDB COMMENT='productos de la tienda';

DROP TABLE IF EXISTS usuarios;
CREATE TABLE usuarios (
  id varchar(15) NOT NULL,
  passwd varchar(32) NOT NULL,
  nombre varchar(25) NOT NULL,
  apellidos varchar(50) NOT NULL,
  email varchar(50) NOT NULL,
  esadmin tinyint(1) NOT NULL,
  PRIMARY KEY  (id)
) TYPE=InnoDB COMMENT='Usuarios de la tienda';

DROP TABLE IF EXISTS pedidos;
CREATE TABLE pedidos (
  id int NOT NULL auto_increment,
  idusuario varchar(15),
  fecha datetime NOT NULL,
  total double NOT NULL,
  finalizado tinyint(1) NOT NULL,
  PRIMARY KEY  (id),
  INDEX (idusuario),
  FOREIGN KEY (idusuario) REFERENCES usuarios(id) ON DELETE SET NULL ON UPDATE CASCADE
) TYPE=InnoDB COMMENT='Pedidos de los usuarios';

DROP TABLE IF EXISTS productos_pedidos;
CREATE TABLE productos_pedidos (
  idpedido int NOT NULL,
  idproducto int NOT NULL,
  precio double NOT NULL,
  cantidad mediumint NOT NULL,
  PRIMARY KEY  (idpedido,idproducto),
  INDEX (idpedido),
  FOREIGN KEY (idpedido) REFERENCES pedidos(id) ON DELETE CASCADE ON UPDATE CASCADE,
  INDEX (idproducto),
  FOREIGN KEY (idproducto) REFERENCES productos(id) ON DELETE CASCADE ON UPDATE CASCADE
) TYPE=InnoDB COMMENT='Productos correspondientes a un pedido';
```

2 Diario 1.

¿Qué tipos de Tablas soporta MySQL?
Detalle brevemente las características de cada una.

A continuación paso a enumerar y describir los siguientes tipos de tablas soportadas por MySQL:

MyISAM: Es el motor de almacenamiento por defecto. Están basadas en el código ISAM, por lo que tienen una estructura muy similar. Hay que tener en cuenta que MySQL 5.0 ya no soporta ISAM. Hay tres subtipos de tablas: estáticas, dinámicas o comprimidas. `MyISAM` soporta tres formatos de almacenamiento distintos. Las dos primeras se eligen automáticamente en función del tipo de columnas que esté usando, si hay alguna de longitud variable (como VARCHAR, BLOB o TEXT) la tabla será dinámica. El formato comprimido puede ser creado sólo con la utilidad `myisampack` y son tablas de solo lectura.

MERGE: Es un tipo especial de tablas que se forman a base de la unión de dos tablas MyISAM idénticas. Las modificaciones y borrados se pueden realizar con tranquilidad sobre la tabla MERGE pues estas operaciones se realizarán sobre la tabla MyISAM que corresponda. Respecto a las inserciones, se plantea la cuestión de en qué tabla se realizará la operación. Esto se soluciona indicando el método de inserción que puede ser FIRST (en la primera tabla de la unión), LAST (en la última) o NO (no se harán inserciones).

HEAP/MEMORY: Son tablas cuyo contenido se almacena en memoria y tienen índices de tipo hash, lo que las hace muy rápidas pero hay mayor probabilidad de pérdida de datos.

InnoDB: Las tablas InnoDB soportan transansacciones, es decir, podremos utilizar COMMIT y ROLLBACK para recuperación de fallos. Permiten el bloqueo a nivel de registro y tienen la posibilidad de utilizar claves extranjeras.

BDB (Berkeley DataBase): Soporta transacciones pero no claves extranjeras (al igual que MyISAM).