DISEÑO FÍSICO RELACIONAL

Tabla de Contenidos

1.- Introducción

El modelado de bases de datos sigue tres niveles de diseño: conceptual, lógico y físico.

El modelo conceptual es el más cercano al “mundo real”. Los conceptos que maneja son muy cercanos a como vemos la realidad, aunque elimina información que no es aplicable a lo que queremos modelar, y no se asocia con ningún modelo informático. El resultado del modelo conceptual se puede implementar con un ordenador o con un fichero tradicional por escrito. El modelo más utilizado para representar el diseño conceptual es el modelo entidad-relación.

El segundo nivel es el modelo lógico. En éste se parte del modelo conceptual y lo adapta a un modelo que puede ser implementado en un ordenador pero sin estar asociado a ninguna implementación en concreto, es decir, podemos utilizar el modelo lógico relacional para crear una base de datos en cualquier motor de bases de datos relacional, comercial o no.

El último nivel es el modelo físico. En éste nivel se adapta el modelo lógico, relacional en este caso, a un sistema de bases de datos concreto. Es en este nivel donde ya se obtiene un modelo que es utilizable físicamente en el ordenador y que permite almacenar y consultar datos reales.

En el diseño físico relacional se transforma un modelo lógico relacional a un sistema de base de datos en concreto. Para ello hay que:

- Mapear los dominios de los atributos a los tipos de datos que soporta el motor de base de datos elegido.
- Definir qué campos son opcionales y por tanto pueden valer NULL.
- Establecer las claves primarias de cada tabla.
- Establecer las claves ajenas, a qué claves primarias se refieren y la acción a realizar cuando se actualiza o elimina una clave primaria.

Asimismo conviene realizar un análisis de las operaciones que se realizan más frecuentemente sobre los datos a fin de optimizar el almacenamiento o acceso.

2.- Introducción al lenguaje SQL

SQL (pronunciado como las letras S-Q-L o como sequel) es el acrónimo inglés de Lenguaje de Consulta Estructurado (*Structured Query Languaje*). SQL es un lenguaje diseñado específicamente para comunicarse con las bases de datos relacionales.

SQL proviene del lenguaje SEQUEL que se diseñó originalmente para la base de datos relacional experimental de IBM System R. El nombre hubo que cambiarlo porque estaba sujeto a copyright.

A finales de los 70 la compañía Relational Software, que luego se convirtió en Oracle, vio el potencial comercial del lenguaje SQL y desarrolló su propia base de datos relacional incluyendo SQL como lenguaje de interacción.

El lenguaje SQL está estandarizado primero por el ANSI (*American National Standards Institute - Instituto Nacional de Estándares Americanos,* organización estadounidense formada para certificar los estándares desarrollados en las diversas industrias para que no sean influenciados por los intereses de una compañía o grupo) y posteriormente por ISO (*Internacional Organization for Standardization - Organización Internacional de Normalización*, cuya principal actividad es la elaboración de normas técnicas internacionales), pero, desafortunadamente, la implementación de la estandarización no ha ido tan bien, por varios motivos:

- Cuando aparecieron los primeros estándares ya existían algunas implementaciones de SQL. Los fabricantes no se han atrevido a perder compatibilidad con código SQL antiguo por lo que mantienen estructuras y palabras no estándar.
- Las nuevas características que intentan introducir los fabricantes no tienen reflejo en el estándar, por lo que cada uno lo implementa a su manera.
- La norma ha sido poco clara en algunos puntos y las implementaciones difieren entre fabricantes.

En resumen, que aunque la sintaxis y el funcionamiento es muy parecido entre fabricantes, hay diferencias, en algunos casos sustanciosas, entre las distintas implementaciones de SQL por lo que hay que suponer que sentencias escritas para un motor y versión concretos pueden no funcionar exactamente igual (o no funcionar en absoluto) en otros. Es conveniente revisar la documentación del motor que se está utilizando para conocer las diferencias.

3.- Clasificación de las sentencias de SQL

SQL nació como un lenguaje de consulta para expresar el álgebra relacional pero se le fueron añadiendo características y funcionalidades y hoy en día cubre todas las necesidades de interacción con el sistema de base de datos. Por esta razón, hoy en día se distinguen varios *subconjuntos* del lenguaje:

- **DDL** (*Data Definition Language – Lenguaje de Definición de Datos*)

  Parte del lenguaje que se encarga de crear y mantener la **estructura** de la base de datos.

- **DML** (*Data Manipulation Language – Lenguaje de Manipulación de Datos*)

  Parte del lenguaje que define como interactuar con los datos para añadir, modificar, borrar y consultar datos.

- **DCL** (*Data Control Language - Languaje de Control de Datos*)

  Parte del lenguaje que permite establecer permisos.

Inicialmente, en este bloque vamos a revisar la parte correspondiente al DDL.

SQL es un lenguaje de programación no procedimental, ya que se le especifica qué es lo que se quiere obtener sin entrar en el cómo. No se escriben instrucciones secuenciales y precisas, tan solo se indica qué acción se quiere realizar y es tarea del propio lenguaje encontrar el camino adecuado para ejecutarla.

SQL sólo sirve para ejecutar consultas sobre bases de datos. No dispone de estructuras de control (if, while, for).

4.- Características generales de SQL

SQL posee dos características muy apreciadas, que son potencia y versatilidad que contrastan con su facilidad para el aprendizaje, ya que utiliza un lenguaje bastante natural. Es por esto que las instrucciones son muy parecidas a órdenes humanas y por ello se le considera un *Lenguaje de Cuarta Generación*.

El lenguaje SQL está compuesto por comandos, cláusulas, operadores, funciones y literales. Todos estos elementos se combinan en las instrucciones y se utilizan para crear, actualizar y manipular bases de datos.

- **Comandos**

  El lenguaje SQL consta de sentencias que se pueden ver como comandos que realizan una tarea completa. Cada sentencia tiene una sintaxis que indica las opciones y datos que se pueden proporcionar a la sentencia.

- **Cláusulas**

  Llamadas también condiciones o criterios, son palabras especiales (reservadas) que permiten modificar el funcionamiento de un comando. El uso de cada palabra sólo puede ser el que le da la sintaxis SQL y no puede utilizarse para ninguna otra cosa, como por ejemplo nombre de tabla o columna.

  Las palabras reservadas no son sensibles a mayúsculas o minúsculas, aunque para mejorar la legibilidad de las sentencias se recomienda que se utilice siempre el mismo esquema de mayúsculas / minúsculas.

- **Operadores**

  Permiten crear expresiones complejas. Pueden ser aritméticos (+, -, *, /, ...) o lógicos (< , >, , < >, And, Or, etc.).

- **Funciones**

  Para conseguir valores complejos. Por ejemplo, la función promedio para obtener la media de un salario.

- **Literales**

  Llamadas también constantes, serán valores concretos, como por ejemplo un número, una fecha, un conjunto de caracteres, etc.

Se deben seguir las siguientes normas sencillas pero primordiales:

- Todas las instrucciones (sentencias) terminan con un signo de punto y coma.
- No se distingue entre mayúsculas y minúsculas.
- Cualquier comando puede separarse con saltos de línea o espacios para facilitar su lectura y comprensión.
- Los comentarios comienzan por /* y terminan con */ (excepto en algunos SGBD).
- Se pueden tabular líneas para facilitar la lectura si fuera necesario.

5.- Tipos de datos

Hasta ahora sabemos que vamos a guardar información relacionada en forma de filas y columnas.

Las columnas son los atributos o información que nos interesa incluir del mundo real que estamos modelando. Y esos atributos toman su valor dentro de un dominio, que formalmente es un conjunto de valores.

En los sistemas de bases de datos, al concepto de dominio se refiere como tipo de dato. Al crear la relación (tabla) hay que especificar tanto los atributos (campos) que se van a almacenar así como el tipo de datos que cada uno puede almacenar. Con la asignación del tipos de datos habremos seleccionado un dominio para un atributo.

El estándar SQL define los siguientes tipos de datos:

Cadenas de caracteres

- **CHAR (n)** o **CHARACTER (n)**

  Cadena de caracteres de longitud fija que se indica por el valor de n. Si no se especificara el valor es 1. Si el valor a almacenar es menor que n se rellenará con espacios en blanco por la derecha. Si es mayor se trunca a n.

- **VARCHAR (n)**

  Cadena de caracteres de longitud variable cuyo valor máximo se indica por el valor de n. Si el valor a almacenar es menor que n sólo se almacena la longitud del valor asignado. Si se asigna un valor mayor se trunca a n.

- **CLOB (n)**

  Cadena de caracteres de gran tamaño (Character Large Object) almacenada en un fichero distinto al de la tabla correspondiente. Algunos SGBDR limitan el uso de ciertas funcionalidades de trabajo con cadenas en este tipo de campos.

Números exactos

Se deben utilizar para almacenar códigos (al ser su acceso más rápido que el de un campo alfanumérico) y para valores numéricos operables.

- **INT (n)** o **INTEGER (n)**

  Número entero de n cifras almacenado en 4 bytes.

  Rango de valores entre -2.147.483.648 y 2.147.483.647.

- **SMALLINT**

  Número entero almacenado en 2 bytes.

  Rango de valores entre -32.768 y 32.767.

- **DEC (e,d)** o **NUMERIC (e,d)**

  Número con parte decimal de tamaño fijo. Se debe especificar el número de dígitos totales (e) y el de dígitos decimales (d).

Números aproximados

Ofrecen un rango mayor de representación de valores decimales, pero son menos precios que el tipo NUMERIC.

- **FLOAT (n)** o **REAL**

  Número real en coma flotante con n bits de precisión en la mantisa.

- **DOUBLE**

  Número real en coma flotante con precisión doble.

Fechas, horas e intervalos

- **DATE**

  Fecha expresada en la secuencia año - mes - día (“yyyy-mm-dd”).

- **TIME**

  Hora expresada en la secuencia hora - minuto - segundo (“hh:mm:ss”).

- **TIMESTAMP**

  Almacena la fecha y hora de inserción o modificación del registro en formato año - mes - día - hora - minuto - segundo (”yyyy-mm-dd hh:mm_ss”).

- **INTERVAL**

  Representa un intervalo de tiempo, bien en la forma años - meses o en la forma días - horas - minutos - segundos.

Valores lógicos

- **BOOLEAN**

  Incluye los valores “true” (verdadero) y “false (falso) y puede almacenar el valor NULL en algunas implementaciones.

Objetos binarios

- **BLOB**

  El tipo BLOB (Binary Large Object) se suele utilizar para almacenar imágenes, documentos y otros objetos, generalmente almacenados en ficheros distintos del de la tabla correspondiente.

Cada SGBD tiene su propia colección de tipos de datos con sus peculiaridades, por lo que hay que leer la documentación del sistema gestor concreto para conocer cómo describir un campo.

5.1.- Constantes en SQL

Una constante es un símbolo que representa un valor de datos específico. El formato de las constantes depende del tipo de dato del valor que representan.

En muchas sentencias SQL hay que utilizar valores constantes, por lo que es importante el determinar cómo se escriben los mismos. Por desgracia, también aquí hay cierta diferencia entre SGBD, por lo que habrá que consultar la documentación del que se vaya a utilizar. De todas formas y de manera general, la sintaxis es la siguiente:

Constantes de cadena

Se escriben entre comillas dobles (“) o simples (‘), aunque algunos SGBD sólo permiten una de las dos, e incluyen caracteres alfanuméricos (a-z, A-Z y 0-9) y caracteres especiales, como el signo de exclamación (!), la arroba (@) y el signo de número (#).

Dentro de una cadena se pueden utilizar los caracteres de escape para crear caracteres difíciles de conseguir con el teclado o que tienen un significado especial, aunque esto depende fuertemente del SGBD. Un carácter de escape comienza por el símbolo \ (backslash) y es seguido de un código numérico o una secuencia de caracteres que indican el símbolo que están representando.

Constantes binarias

Comienzan por el prefijo 0x seguido de números hexadecimales. No se incluyen entre comillas.

Constantes enteras

Se escriben con números y opcionalmente un signo menos delante.

Constantes numéricas reales

Constan de un signo opcional, parte entera, un punto, parte decimal. Si no hay parte decimal se puede omitir, así como el punto aunque la parte entera es obligatoria. También se pueden expresar en notación científica que es similar pero utilizando la letra e (E), un signo y el exponente. Por ejemplo -2.676E4 es lo mismo que -26760.

Constantes de fecha y hora

Aquí hay una amplia diversidad entre los SGBDs, tanto en la forma de separar los distintos componentes (utilizando / ó -) como en el orden de los mismos (día, mes y año; mes, día y año; año, mes y día,...). Lo mismo ocurre con las horas, aunque estas utilizan normalmente el símbolo (:) y suelen ir siempre en el mismo orden, aunque algunos SGBDs aumentan el tipo TIME para contener fracciones de segundo y esto influye en el formato de las constantes.

Constantes booleanas

Algunos SGBDs tienen las constantes TRUE y FALSE para representar los valores verdadero y falso, respectivamente.

Constante NULL

El valor NULL se representa con la palabra reservada NULL.

6.- Objetos que componen una base de datos

Un SGBD relacional contiene los siguiente objetos:

- **Diccionario de datos**

  Es una metabase de datos, es decir, una base de datos *sobre* la base de datos. Dicho en otras palabras es un lugar en el que se almacena las definiciones del resto de objetos que componen la base de datos. El sistema la utiliza de forma automática y transparente al realizar la mayoría de las operaciones que se le ordenan y muchas de ellas lo modifican de forma indirecta. Muchos SGBDs implementan el diccionario como otra base de datos más pero con un status especial y con un acceso mucho más restringido.

- **Usuarios y grupos**

  Para gestionar la seguridad de las bases de datos, muchos SGBDs implementan un sistema de permisos, similar al empleado en los sistemas de ficheros de los sistemas operativos. Se definen usuarios y grupos. Cada usuario tiene unas credenciales de acceso y tiene permiso para realizar ciertas operaciones sobre ciertos objetos de la base de datos. Los grupos permiten asignar permisos a múltiples usuarios relacionados al mismo tiempo, aunque no todos los sistemas los poseen. Algunos no poseen ni usuarios, no disponiendo de sistema de seguridad de acceso.

- **Base de datos**

  Una base de datos es una colección de tablas, índices, vistas, eventos y programas. Asimismo una base de datos define un espacio de nombres para cada uno de estos objetos, esto es, pueden existir dos tablas con el mismo nombre en dos bases de datos distintas pero no dos tablas con el mismo nombre en la misma base de datos. Además, una base de datos tiene ciertas propiedades físicas (forma de almacenamiento, forma de ordenar los campos de texto, etc.) que le son propias.

- **Tabla**

  Una tabla es un espacio de almacenamiento de una serie de registros (filas o tuplas). Una tabla se define con un nombre único entre las tablas y una serie de campos o columnas, cada uno con su nombre único entre los campos de una misma tabla y un tipo de datos. Las filas se añaden y quitan pero tienen que dar valores para cada uno de los campos y este valor debe cumplir las restricciones del tipo al que pertenece el mismo.

- **Índice**

  Un índice es un diccionario que se asocia con un campo o campos de una tabla, de forma que se accede más rápidamente a la fila que contiene un valor especificado de un campo. Algunos índices se crean de forma automática y otros se pueden crear explícitamente.

- **Vista**

  Una vista es una tabla “virtual” que se obtiene al restringir las columnas o filas de un conjunto de datos de la base de datos. Se utilizan para dar una visión restringida de la base de datos a determinados usuarios, aunque su uso tiene un coste y tiene algunas limitaciones, sobre todo con las operaciones que modifican datos.

- **Programa**

  Programa escrito en un lenguaje de programación adaptado a la base de datos y que puede actuar sobre los datos de ésta. Se utilizan para realizar funciones directamente sobre la base de datos, siendo muy rápidas y eficientes, aunque pueden producir efectos laterales indeseados.

- **Eventos**

  Son condiciones que hacen que se dispare un programa. Se pueden establecer sobre los datos de la base de datos de forma que se ejecute un programa cuando los datos cumplan ciertas condiciones, cuando se elimine o modifique un dato, etc. Su propósito inicial es ayudar a mantener las reglas de integridad de la información aunque se pueden utilizar para cualquier cosa.

Todos estos objetos se pueden administrar directa o indirectamente a través del LDD (Lenguaje de Definición de Datos). La creación del modelo físico de la base de datos consiste precisamente en la creación de una serie de sentencias SQL, adaptadas al SGBD destino, que creen o ajusten estos objetos.

7.- El Lenguaje de Definición de Datos (DDL)

El Lenguaje de Definición de Datos (LDD) es un subconjunto de SQL que permite administrar (crear, modificar y borrar) los objetos de la base de datos, en contraposición al Lenguaje de Manipulación de Datos (LMD) que permite la manipulación de los datos mantenidos por esos objetos.

En esta sección se va a describir el LDD de modo general.

7.1.- Convenciones

En este documento se van a a utilizar las siguientes convenciones a la hora de indicar los comandos SQL.

- Los comandos y sus parámetros se indican con fuente `monoespacio`.
- Las palabras reservadas del lenguaje se escriben con LETRAS MAYÚSCULAS. Estas palabras **no deben usarse como nombres para los elementos de las bases de datos**, como tablas o columnas.
- Las palabras escritas con letras minúsculas indican parámetros o datos que hay que proporcionar a los comandos.
- Las sentencias se incluyen en un recuadro con fondo amarillo claro.
- Cuando se vaya a proporcionar una lista, se utilizaran puntos suspensivos (...) para indicar los elementos que no se van a escribir por abreviar pero que deberían proporcionarse cuando se utilice el lenguaje en un entorno real.
- Las partes opcionales se colocan entre paréntesis cuadrados ([ ]). Estos paréntesis sólo sirven para delimitar en la explicación las partes opcionales y no se deben utilizar cuando se escriban comandos reales. Sólo se debe incluir, en caso de que se quiera usar la opción, el contenido de los paréntesis cuadrados.

Por ejemplo si se indica el comando:

```
COMANDO nombre [NULL]
```

esto indica que hay un comando llamado `COMANDO` que necesita como parámetro un `nombre` y puede opcionalmente ir seguido de la palabra `NULL`.

7.2.- Gestión de bases de datos

Hay tres operaciones que se pueden realizar sobre una base de datos: crearla, modificarla y borrarla.

7.2.1.- Creación de una base de datos

Una base de datos se crea utilizando la sentencia `CREATE DATABASE` con la siguiente sintaxis:

```
CREATE DATABASE [IF NOT EXITS] nombredb [opciones];
```

donde:

- **`nombredb`** es el nombre que se le desea dar a la nueva base de datos. No debe haber ya una base de datos con ese nombre o se producirá un error si no se indica la opción IF NOT EXITS..
- **`opciones`** son distintas opciones generales de configuración que se pueden dar a la base de datos. Estas opciones dependen fuertemente del motor de base de datos elegido.

EJEMPLO

Crea una base de datos con nombre `mybase`

```
CREATE DATABASE mybase;
```

NOTA: Enlace para crear BD con MySQL → https://conclase.net/mysql/curso/index.php?cap=007

7.2.2.- Modificación de una base de datos

La configuración general de la base de datos se puede modificar utilizando la sentencia `ALTER DATABASE` con la siguiente sintaxis:

```
ALTER DATABASE nombredb nuevasopciones;
```

donde:

- **`nombredb`** es la base de datos cuya configuración se desea modificar. Debe ser una base de datos existente.
- **`nuevasopciones`** son los nuevos valores de la configuración. Al igual que con `CREATE DATABASE`, estas opciones dependen del SGBD utilizado.

EJEMPLO (EN MARIADB)

Modifica la base de datos mybase para que utilice el juego de caracteres utf8 para las cadenas de caracteres.

```
ALTER DATABASE mybase CHARACTER SET utf8;
```

7.2.3.- Borrado de una base de datos

Se puede eliminar una base de datos completamente utilizando la sentencia `DROP DATABASE` con la siguiente sintaxis:

```
DROP DATABASE nombredb;
```

donde:

- **`nombredb`** es el nombre de la base de datos a eliminar. Si la base de datos no existe se produce un error.

Hay que tener mucha precaución en el uso de este comando porque **la base de datos se elimina completamente y con ella todos los objetos que contenga**: tablas, índices, vistas, etc., j**unto con los datos almacenados**.

EJEMPLO

Eliminar la base de datos `mybase`

```
DROP DATABASE mybase;
```

7.3.- Gestión de tablas

Las tablas son los objetos más importantes que contiene una base de datos pues son el sitio donde se almacena la información y sobre los que basan su operación el resto de los objetos de la base de datos.

Las operaciones que se pueden realizar sobre las tablas son: crearla, modificarla (su estructura, no sus datos) y eliminarla.

7.3.1.- Creación de tablas

Las tablas se crean utilizando la sentencia `CREATE TABLE` con la siguiente sintaxis:

```
CREATE TABLE nombretabla (definicioncol1, definicioncol2, ...,
definicioncoln) [opciones];
```

donde

- **nombretabla** es el nombre de la tabla a crear. No puede existir una tabla con ese nombre en la base de datos o el comando fallará.
- **definicioncol*X***. Cada una de ellas es, o bien una definición de columna (campo o atributo) o bien una restricción de tabla. Se elaborará mas adelante. El conjunto de las definiciones va encerrado entre paréntesis y las definiciones van separadas por comas.
- **opciones**. Opciones generales de la tabla. Dependen fuertemente del SGBD, pero pueden especificar desde el tipo de almacenamiento de la tabla como la ubicación en disco, etc.

La definición de una columna sigue la siguiente sintaxis:

```
nombrecol tipocol restricciones
```

donde:

- **nombrecol** es el nombre de la columna. Debe ser único entre los nombres de columna de la tabla, esto es, no puede haber dos columnas con el mismo nombre o se producirá un error.
- **tipocol** es el tipo de la columna. Uno de los tipos indicados en secciones anteriores o uno propio del SGBD utilizado.
- **restricciones** son restricciones adicionales que se pueden aplicar a la columna y que pueden ser:
  - **NULL**. Indica que la columna puede admitir valores nulos. Es la opción por defecto, por lo que no es necesario utilizarlo aunque nosotros vamos a utilizarlo siempre para que quede clara la intención.
  - **NOT NULL**. Indica que la columna **NO** puede admitir valores nulos. Esta opción hay que indicarla obligatoriamente cuando se necesite porque de lo contrario se aplicaría la opción por defecto que es permitir los valores nulos. Sólo se puede especificar uno de los dos (o `NULL` o `NOT NULL`).
  - **DEFAULT valorpordefecto**. Indica el valor por defecto que recibirá la columna si se crea una fila en la tabla y no se proporciona valor para la columna. Si no se da un valor por defecto, depende del SGBD el valor que se asignará.
  - **PRIMARY KEY**. Indica que la columna es la clave primaria de la tabla. Esto usualmente implica que en esta columna los valores deben ser únicos, esto es, que no puede haber dos filas con el mismo valor en esta columna y que se creará un índice sobre esta columna. Sólo se puede utilizar para una columna. Si se quiere crear una clave primaria que conste de más de una columna hay que utilizar restricciones (`CONSTRAINT`) (ver más adelante).
  - **UNIQUE**. Indica que la columna contendrá valores únicos por fila, esto es, que el valor de la columna debe ser distinto para cada fila y usualmente implica crear un índice sobre la columna.

EJEMPLO

Se desea crear una tabla llamada `consulta` con los siguientes campos:

- `cod_consulta`. Entero. Clave primaria.
- `id_med`. Entero. Código del médico que realiza la consulta. Es clave ajena a la tabla `medico` y el campo `id_med`.
- `fecha`. Fecha. No puede ser nulo.
- `diagnostico`. Carácter de hasta 200 caracteres. No puede ser nulo.
- `id_pac`. Entero. Código del paciente que ha recibido la consulta. Es clave ajena a la tabla `paciente` y el campo `id_pac`.

La sentencia de creación sería:

```
CREATE TABLE consulta (
    cod_consulta INTEGER PRIMARY KEY,
    id_med INTEGER NOT NULL,
    fecha DATE NOT NULL,
    diagnostico VARCHAR(200) NOT NULL,
    id_pac INTEGER NOT NULL
);
```

No se han definido las claves ajenas porque para ello hay que utilizar restricciones de tabla, que se verán a continuación.

Hay que hacer notar que las dos claves ajenas (`id_med` e `id_pac`) se han definido como `NOT NULL`. Esto indica que tanto la relación entre `consulta` y `medico` como entre `consulta` y `paciente` no es opcional. Si lo fuera debería de indicarse como `NULL` (o no poner nada y que tome el valor por defecto, que es `NULL`).

Anteriormente, cuando se ha hablado de la definición de la sentencia CREATE TABLE se han dejado sin comentar las restricciones de tabla. Esto ha sido así para poder introducir la sintaxis básica y ahora se van introducir.

Las restricciones de tabla son restricciones que se aplican a la tabla completa o a un conjunto de columnas de la tabla y que, por tanto, no se pueden definir como opción de una sola columna. Las restricciones de tabla pueden ser una de:

- **`CONSTRAINT PRIMARY KEY (col1, con2, …, coln)`**

  Define la clave primaria de la tabla que consta de varias columnas (`col1`, `col2`, …, `coln`).

- **`INDEX [nombre] (col1, col2, …, coln)`**

Define un índice sobre las columnas `col1`, `col2`, …, `coln`. Al índice se le denomina `nombre` (si se da, si no el sistema le proporciona un nombre automáticamente).

- **`CONSTRAINT UNIQUE (col1, col2, …, coln)`**

  Define una restricción de unicidad. La combinación de los valores de las columnas dadas debe ser único para cada fila de la tabla.

- **`CONSTRAINT FOREIGN KEY [nombre] (col1, col2, …, coln) REFERENCES`**

  Define una clave ajena con el `nombre` dado (si no se da, el sistema asigna uno). La definición de la parte REFERENCES es:

  - **`REFERENCES tablaref (columnaref) [ON DELETE opcion_ref1] [ON UPDATE opcion_ref2]`**. Indica que el campo es una clave ajena que referencia el campo `columnaref` de la tabla `tablaref` (que debería ser la clave primaria de la tabla, aunque esto no es obligatorio). Las opciones `opcion_ref1` y `opcion_ref2` indican la acción a tomar cuando se elimina o modifica, respectivamente, un valor en la columna `columnaref` de la tabla `tablaref` que coincide con el valor de esta columna en alguna fila de la tabla.
  - Las opciones pueden ser:
    - **`RESTRICT`**. No se permite la operación. Es la opción por defecto.
    - **`CASCADE`**. La fila o filas con el valor coincidente se elimina(n) o su valor se modifica para que sea igual al nuevo valor asignado en la tabla referenciada, de forma que se mantenga la referencia.
    - **`SET NULL`**. En la fila o filas con el valor coincidente se cambia(n) este por `NULL`. Es un error si se ha declarado el campo como `NOT NULL`.

EJEMPLO

Se va a hacer la misma definición del ejemplo anterior, pero utilizando restricciones de tabla, cuando se pueda:

```
CREATE TABLE consulta (
    cod_consulta     INTEGER,
    id_med           INTEGER          NOT NULL,
    fecha            DATE             NOT NULL,
    diagnostico      VARCHAR(200)     NOT NULL,
    id_pac           INTEGER          NOT NULL,
    CONSTRAINT PRIMARY KEY (cod_consulta),
    CONSTRAINT FOREIGN_KEY (id_med) REFERENCES medico(id_med),
    CONSTRAINT FOREIGN KEY (id_pac) REFERENCES paciente(id_pac)
);
```

7.3.2.- Modificar una tabla

Se realiza con la sentencia `ALTER TABLE` que tiene múltiples sintaxis porque puede ser utilizada para realizar muchas modificaciones distintas sobre una tabla, por lo tanto se van a definir por separado.

Aun así, todas comparten un inicio común. La sintaxis común es:

```
ALTER TABLE nombretabla modificacion;
```

donde:

- **`nombretabla`** es el nombre de la tabla que se va a modificar.
- **`modificacion`** es la modificación que se quiere realizar que se detallará en las secciones siguientes.

Añadir una columna

Sintaxis:

```
ADD COLUMN nombrecol defcol;
```

donde:

- **`nombrecol`** es el nombre de la nueva columna. Debe ser distinto a los nombres de columnas ya existentes en la tabla.
- **`defcol`**. Definición de columna (tipo, `NULL`, etc.). Como en `CREATE TABLE`.

EJEMPLO

Se desea añadir a la tabla consulta, ya creada, un nuevo campo que se ha olvidado, receta, que es un campo de texto con una longitud de 500 caracteres. Se podría añadir utilizando la sentencia:

```
ALTER TABLE consulta ADD COLUMN receta VARCHAR(500) NOT NULL;
```

Modificar una columna

Sintaxis:

```
MODIFY COLUMN nombrecol defcol;
```

donde el significado de **`nombrecol`** y **`defcol`** es el mismo que en la sintaxis anterior.

Hay que tener en cuenta que no siempre se puede cambiar el tipo de una columna ya que el nuevo tipo debe de poder ser convertido desde el anterior, por ejemplo, si se convierte un campo desde `VARCHAR` a `INTEGER` es posible que nos encontremos con algún valor que no tiene equivalencia. En ese caso se producirá un error.

EJEMPLO

Se desea modificar la longitud del campo `diagnostico` de la tabla consulta para que pueda contener 500 caracteres en lugar de los 200 actuales. Se podría hacer con la sentencia:

```
ALTER TABLE consulta MODIFY COLUMN diagnostico VARCHAR(500);
```

Modificar el nombre de una columna

Sintaxis:

```
CHANGE COLUMN nombrecolant nombrecolnuevo;
```

donde:

- **`nombrecolant`** es la columna cuyo nombre se desea cambiar
- **`nombrecolnuevo`** es el nuevo nombre que se le desea dar a la columna.

EJEMPLO

En la tabla consulta se desea modificar el nombre del campo `receta` a `prescripcion`. Para ello se utilizaría la sentencia:

```
ALTER TABLE CONSULTA CHANGE COLUMN receta prescripcion;
```

Modificar el valor por defecto de una columna

Sintaxis:

```
ALTER COLUMN nombrecol definicion_defecto;
```

donde:

- **`nombrecol`** es el nombre de la columna cuyo valor por defecto se desea modificar.
- **`definicion`**`_defecto` es el nuevo valor por defecto que puede ser uno de los siguientes:
  - **`SET DEFAULT valor_defecto`**. Cambia el valor por defecto de la columna a `valor_defecto`.
  - **`DROP DEFAULT`**. Elimina el valor por defecto. Si no se le da un valor al campo tomará `NULL`, si puede, o dará un error si no puede ser `NULL`.

EJEMPLO

Poner por defecto "`Ninguno`" a la columna `diagnostico` de la tabla consulta:

```
ALTER TABLE consulta ALTER COLUMN diagnostico SET DEFAULT "Ninguno";
```

Eliminar una columna

Sintaxis:

```
DROP COLUMN nombrecol;
```

donde **nombrecol** es la columna a eliminar. Se borrará tanto la columna como sus datos.

EJEMPLO

Se desea eliminar la columna `prescripcion` de la tabla `consulta`. Se podría utilizar:

```
ALTER TABLE consulta DROP COLUMN prescripcion;
```

Añadir una clave primaria

Sintaxis:

```
ADD CONSTRAINT PRIMARY KEY (col1, col2, ..., coln);
```

Añade una clave primaria a la tabla compuesta por las columnas `col1`, `col2`, ..., `coln`.

EJEMPLO

Para indicar la clave primaria de la tabla `consulta` caso de no hacerla en su creación:

```
ALTER TABLE consulta ADD CONSTRAINT PRIMARY KEY (cod_consulta);
```

Añadir una clave ajena

Sintaxis:

```
ADD CONSTRAINT FOREIGN KEY [nombre] (col1, col2, ..., coln) REFERENCES;
```

donde:

- **nombre** de la clave ajena. Si no se especifica, el sistema asigna un nombre automáticamente.
- **colX**. Columnas que forman parte de la clave ajena.
- **references**. Igual que la parte homónima en la restricción de tabla de `CREATE TABLE`.

EJEMPLO

Si en `consulta` se ha olvidado incluir que `id_pac` es una clave ajena a la clave primaria `id_pac` de la tabla `paciente`, se puede hacer más tarde utilizando la sentencia:

```
ALTER TABLE consulta ADD CONSTRAINT FOREIGN KEY id_pac_ext (id_pac)
REFERENCES paciente(id_pac);
```

7.3.3.- Eliminar una tabla

Una tabla se elimina utilizando la sentencia `DROP TABLE`. La sintaxis es la siguiente:

```
DROP TABLE nombre_tabla;
```

donde **`nombre_tabla`** es el nombre de la tabla a eliminar.

Cuidado al utilizar este comando porque **no se solicita confirmación** y elimina la tabla, su contenido y los índices y claves primarias y ajenas vinculadas a ella. **Esta operación no puede deshacerse.**

EJEMPLO

Si se quiere eliminar la tabla `consulta`, se podría hacer con la sentencia:

```
DROP TABLE consulta;
```

7.4.- Índices

Los índices son estructuras asociadas a uno o más columnas de una tabla y que permiten realizar búsquedas en esas columnas de forma más rápida. Funcionan como un diccionario en el que se ordenan los valores de las columnas que forman parte del índice y por cada una de ellas se almacena la dirección de la fila o filas en las que están contenidas y que contienen los datos correspondientes, permitiendo un acceso directo a la información en lugar de recorrer toda la tabla buscando coincidencias.

Por lo tanto son estructuras útiles si se van a realizar frecuentemente búsquedas o uniones de tablas utilizando los valores de dichos campos.

Sin embargo no todo son ventajas, ya que los índices hay que mantenerlos y cada vez que se inserta o elimina información en una tabla con índices o se modifica un valor de una columna que entra en un índice hay que actualizar los índices correspondientes para que sigan siendo válidos y esas operaciones tienen un costo.

Así, se recomienda el uso de los índices sobre todo en:

- Claves primarias de las tablas, ya que se suelen realizar muchas búsquedas sobre ellas, tanto para localizar una fila determinada como cuando se realizan uniones de tablas, ya que suele ser la columna o columnas que se utilizan para realizar la unión.
- Claves ajenas de tablas. Por la misma razón que las claves primarias respecto a la unión de tablas.
- Campos sobre los que se realizan muchas búsquedas. Por ejemplo, si se utiliza mucho la columna CodigoPostal para localizar a personas que viven en una determinada zona, quizá sería aconsejable crear un índice sobre ese campo.

7.4.1.- Crear un índice

Los índices se pueden crear de dos formas:

- **Implicitamente**

  Determinadas operaciones sobre la estructura de una tabla crean índices de manera implícita. Por ejemplo, muchos SGBDs crean un índice al definir la clave primaria o claves ajenas de una tabla de forma transparente.

- **Explicitamente**

  Se indica, mediante una sentencia SQL, que se desea crear un índice sobre una columna o columnas. Esta es la sintaxis que se va a describir en esta sección.

Para crear un índice de forma explícita se utiliza la siguiente sintaxis:

```
CREATE [UNIQUE] INDEX nombreindice ON nombretabla (col1, col2, ..., coln);
```

donde:

- **`nombreindice`** es el nombre del nuevo indice. El nombre se utiliza para poder gestionarlo desde SQL, ya que su uso es automático.
- **`nombretabla`**. Tabla sobre la que se va a crear el índice.
- **`colx`**. Columna o columnas que forman parte del índice.

Si se utiliza el modificador `UNIQUE`, el índice será único, esto es, no permitirá entradas duplicadas sobre los campos del índice. Para un índice sobre una sola columna, esto implica que no podrá haber valores repetidos en dicha columna. Para un índice sobre varias columnas, esto implica que, para cada fila, la combinación de los valores de las columnas que forman el índice debe ser única.

EJEMPLO

Índice sobre la columna `prescripcion` de la tabla `consulta`:

```
CREATE INDEX idxprescripcion ON consulta (prescripcion);
```

7.4.2.- Eliminar un índice

Un índice puede eliminarse si se considera que ya no es necesario. La sintaxis para eliminar un índice es:

```
DROP INDEX nombreindice ON nombretabla;
```

EJEMPLO

Para eliminar el índice `idxprescripcion` anteriormente creado se utilizaría:

```
DROP INDEX idxprescripcion ON consulta;
```

7.5.- Vistas

Una vista es el resultado de una consulta (sentencia SELECT) sobre una o varias tablas u otras vistas, que es compilada y almacenada como un objeto independiente en el sistema.

Las vistas son creadas en el diseño lógico específico para implementar el esquema externo de la arquitectura de los SGBD. Un usuario puede tener privilegios sobre una vista que afecta a ciertas filas de varias tablas y no tenerlos en las tablas en las que se basa dicha vista. De esta forma las vistas añaden un nivel de seguridad en el acceso a los datos del sistema. También son utilizadas para encapsular consultas que se realizan con mucha frecuencia.

Las vistas tienen las siguientes similitudes con las tablas:

- Están formadas por filas y columnas.
- Pueden invocarse en las mismas cláusulas donde lo hacen las tablas, por ejemplo, en la cláusula FROM de una sentencia SELECT.

Y son diferentes en los siguientes aspectos:

- Para crear una vista tienen que existir las tablas en las que se basa. Por esta razón se les denomina tablas virtuales.
- No contienen datos, sino la definición de cómo obtenerlos.
- Se pueden leer o consultar los datos de las vistas, aunque no todas las vistas son actualizables, es decir, pueden existir restricciones para realizas las operaciones de edición de datos (INSERT, UPDATE, DELETE).

Por ahora solo veremos la sintaxis de las sentencias de creación, modificación y eliminación de vistas. Volveremos a las vistas cuando finalicemos el siguiente tema dedicado al Lenguaje de Manipulación de Datos que permite realizar sentencias de consultas.

7.5.1.- Creación de vistas

```
CREATE VIEW nombre_vista AS sentencia_select;
```

donde:

- **nombre_vista**

  Es el nombre de la nueva vista a crear. No puede coincidir con el nombre de otra vista o tabla ya existente en la base de datos.

- **sentencia_select**

  Es la sentencia SELECT que va a proporcionar el esquema y los datos de la vista.

7.5.2.- Modificación de vista

Hay dos sintaxis distintas pero que realizan una función equivalente:

```
CREATE OR REPLACE VIEW nombre_vista AS sentencia_select;
```

o bien

```
ALTER VIEW nombre_vista AS sentencia_select;
```

7.5.3.- Eliminación de vista

```
DROP VIEW nombre_vista;
```

8.- El Lenguaje de Control de Datos (DCL)

Para garantizar la seguridad e integridad de los datos es necesario implementar un control de forma que los usuarios tengan acceso a los datos que necesitan, ni más ni menos. Considera lo siguiente:

- La mayoría de los usuarios necesitan leer y escribir datos de las tablas, pero pocos usuarios necesitarán crear y eliminar tablas.
- Algunos usuarios pueden necesitar leer tablas pero no necesitan actualizarlas.
- Puede que se quiera permitir a los usuarios añadir datos pero no borrarlos.
- Algunos usuarios (gerentes o administradores) podrían necesitar derechos para manipular cuentas de usuario, pero la mayoría no debería.
- Puede que se quiera que los usuarios accedan a los datos a través de procedimientos almacenados, pero nunca directamente.
- Puede que se quiera restringir el acceso a algunas funcionalidades basándose en el lugar donde el usuario se conecta.

Estos son sólo ejemplos, pero ayudan a demostrar un punto importante. Se debe proporcionar a los usuarios el acceso que necesitan y sólo el que necesitan. Esto se conoce como control de acceso, y la gestión del control de acceso requiere crear y gestionar cuentas de usuario.

La instalación del SGBD crea una cuenta de usuario llamada root que tiene un control completo y total sobre todo el servidor. Todas las pruebas que hemos estado realizando se han hecho con esta cuenta de superadministrador, pero en el mundo real nunca se debe usar root en el día a día.

A continuación veremos las sentencias del Lenguaje de Control de Datos para gestionar el acceso a la base de datos y sus datos.

8.1.- Usuarios

Tanto en MySQL como en MariaDB las cuentas de usuario y su información se almacenan en una base de datos llamada `mysql`. Esta base de datos contiene una tabla llamada `user` que contiene todas las cuentas de usuario.

8.1.1.- Gestión de cuentas de usuario

La sintaxis para crear una nueva cuenta de usuario es:

```
CREATE USER 'NombreUsuario'@'equipo' [IDENTIFIED BY 'ClaveAcceso'];
```

donde,

- **NombreUsuario@equipo**

  El nombre de la cuenta está formada por un nombre de usuario y un nombre de equipo (host) desde el cual el usuario puede conectarse. El nombre de usuario es case sensitive. El nombre de equipo:

  - Puede ser localhost (conexión de cliente local) u otro nombre
  - Puede ser una IP
  - Puede ser '%' que significa cualquier máquina

  Si no se proporciona nombre de equipo se asume que es '%'.
- **ClaveAcceso**

  Es la contraseña en forma de texto plano que el SGBD cifrará antes de guardarla en la tabla de usuarios.

EJEMPLOS

- usuario@localhost
- usuario@192.168.56.1
- usuario@'%'
- usuario@'iesbelen.org'
- usuario@'%.iesbelen.org'
- usuario@'192.168.56.0/255.255.255.0'
- usuario@'192.168.56.%'

Para eliminar o borrar un usuario la sintaxis es:

```
DROP USER 'NombreUsuario'@'equipo';
```

Para renombrar una cuenta de usuario, la sintaxis es:

```
RENAME USER usuario TO nuevoUsuario;
```

EJEMPLO

De creación, renombrado y eliminación de usuario:

```
CREATE USER alumno@localhost IDENTIFIED BY ‘bbdd’;
RENAME USER alumno@localhost TO student@’%’;
DROP USER student@’%’;
```

8.1.2.- Derechos de acceso y privilegios

Las cuentas de usuario recién creadas pueden iniciar sesión en pero no ven ningún dato y no pueden realizar ninguna operación en la base de datos. Por eso, una vez creada se le debe asignar derechos de acceso y privilegios.

Un privilegio es un permiso dado a un usuario para que realice ciertas operaciones, que pueden ser:

- **Permisos sobre objetos**, que permiten acceder y realizar cambios en las tablas de los esquemas de la base de datos, por ejemplo `INSERT`, `SELECT`, `UPDATE` y `DELETE`.
- **Permisos de sistema**, que permiten crear sesiones, estructuras o incluso ejecutar código contenido dentro de la base de datos. Por ejemplo las sentencias `CREATE USER`, `ALTER`, `CREATE` o `CREATE VIEW` corresponden a este tipo de permisos.

Para comprobar los permisos de un usuario se utiliza la siguiente sintaxis:

```
SHOW GRANTS FOR usuario;
```

Para asignar permisos se utiliza la sentencia `GRANT`, que como mínimo requiere que se especifique el privilegio que se concede, el objeto sobre el que se concede y el nombre de usuario al que se le concede.

```
GRANT privileges ON object TO user [WITH GRANT OPTION];
```

Para quitar derechos y permisos se utiliza la sentencia REVOKE.

```
REVOKE privileges ON object FROM user;
```

`GRANT` y `REVOKE` se pueden utilizar para controlar el acceso a varios niveles:

- Todo el servidor, usando `GRANT ALL` y `REVOKE ALL`
- Toda la base de datos, usando `ON database.*`
- Tablas específicas, usando `ON database.table`
- Columnas específicas
- Procedimientos almacenados específicos

Cuando se utilizan GRANT y REVOKE, la cuenta de usuario debe existir, pero los objetos a los que se hace referencia no tienen por qué existir. Esto permite a los administradores diseñar e implementar la seguridad antes de que se creen las bases de datos y las tablas.

Un efecto secundario de esto es que si se elimina una base de datos o una tabla (con una sentencia DROP) cualquier derecho de acceso asociado seguirá existiendo. Y si la base de datos o la tabla se vuelven a crear en el futuro, esos derechos se aplicarán a ellas.

La siguiente tabla enumera cada uno de los derechos y privilegios que se pueden conceder o revocar.

Privilege	Description
ALL	All privileges except GRANT OPTION.
ALTER	Use of ALTER TABLE.
ALTER ROUTINE	Use of ALTER PROCEDURE and DROP PROCEDURE.
CREATE	Use of CREATE TABLE.
CREATE ROUTINE	Use of CREATE PROCEDURE.
CREATE TEMPORARY TABLES	Use of CREATE TEMPORARY TABLE.
CREATE USER	Use of CREATE USER, DROP USER, RENAME USER, and REVOKE ALL PRIVILEGES.
CREATE VIEW	Use of CREATE VIEW.
DELETE	Use of DELETE.
DROP	Use of DROP TABLE.
EXECUTE	Use of CALL and stored procedures.
FILE	Use of SELECT INTO OUTFILE and LOAD DATA INFILE.
GRANT OPTION	Use of GRANT and REVOKE.
INDEX	Use of CREATE INDEX and DROP INDEX.
INSERT	Use of INSERT.
LOCK TABLES	Use of LOCK TABLES.
PROCESS	Use of SHOW FULL PROCESSLIST.
RELOAD	Use of FLUSH.
REPLICATION CLIENT	Access to location of servers.
REPLICATION SLAVE	Used by replication slaves.
SELECT	Use of SELECT.
SHOW DATABASES	Use of SHOW DATABASES.
SHOW VIEW	Use of SHOW CREATE VIEW.
SHUTDOWN	Use of mysqladmin shutdown (used to shut down MariaDB).
SUPER	Use of CHANGE MASTER, KILL, LOGS, PURGE MASTER, and SET GLOBAL. Also allows mysqladmin debug login.
UPDATE	Use of UPDATE.
USAGE	No access.

Para actualizar los permisos se utiliza la sentencia:

```
FLUSH PRIVILEGES;
```

EJEMPLOS

Otorga los permisos indicados sobre la tabla `articulo` al usuario local `vendedor` y permite que los ceda a otros usuarios.

```
GRANT SELECT, INSERT, UPDATE, DELETE ON articulo
TO 'vendedor'@'localhost' WITH GRANT OPTION;
```

Otorga todos los permisos sobre la tabla `articulo` al usuario local `vendedor`, excepto la posibilidad de otorgar también los permisos. Observa que el privilegio solo afecta a nivel de tabla.

```
GRANT ALL ON articulo TO 'vendedor'@'localhost';
```

Concede el permiso SELECT sobre la tabla `articulo` a cualquier usuario local:

```
GRANT SELECT ON articulo TO '*'@'localhost';
```

Permite al usuario local `vendedor` tener acceso a todas las bases de datos del servidor.

```
GRANT ALL PRIVILEGES ON *.* TO 'vendedor'@'localhost';
```

Permise consultar la tabla `pedido` de la base de datos `vehiculos` al usuario `vendedor`.

```
GRANT SELECT ON ‘vehiculos.pedido’ TO 'vendedor'@'localhost';
```

Otorga permiso para crear bases de datos y tablas en el sistema al usuario local `vendedor`.

```
GRANT CREATE ON *.* TO 'vendedor'@'localhost';
```

Da permiso al usuario local `vendedor` para eliminar tablas en la base de datos `vehiculos`.

```
GRANT DROP ON ‘vehiculos.*’ TO 'vendedor'@'localhost';
```

Da permiso al usuario local `vendedor` para consultar las columnas `nombre`, `apellidos` de la tabla `clientes` en la base de datos `vehiculos`.

```
GRANT SELECT(nombre,apellidos)
ON ‘vehiculo.clientes’
TO 'vendedor'@'localhost';
```

Para eliminar permisos las sentencias son iguales cambiando `GRANT` por `REVOKE` y `TO` por `FROM`.

Si el permiso revocado al usuario lo cedió con la opción `WITH GRANT OPTION` también a esos usuarios se le retira.

8.1.3.- Roles

Cuando el número de usuarios es elevado la gestión de permisos se puede optimizar mediante el uso de roles. Un rol es una agrupación de permisos que permite asociar permisos y cuentas de forma que cualquier cambio sobre el rol se traslada a las cuentas que tiene asignado el rol. Un usuario puede pertenecer a más de un rol.

Para crear un rol se utiliza la sintaxis básica:

```
CREATE ROLE nombre_rol;
```

Para asignar permisos al rol se utilizar la siguiente sentencia:

```
GRANT permiso ON objeto TO rol;
```

Posteriormente se asigna el rol al usuario:

```
GRANT nombre_rol TO usuario [IDENTIFIED BY contraseña] [WITH ADMIN OPTION];
```

Y como los roles no se aplican automaticamente, se deben activar mediante:

```
SET ROLE rol;
```

EJEMPLO

```
CREATE ROLE supervisor;
GRANT SELECT ON empresa.pedidos TO supervisor;
GRANT supervisor TO usuario;
SET ROLE supervisor;
```

Para eliminar un rol:

```
DROP ROLE nombre_rol;
```

SGBD MariaDB (MySQL)

En esta sección se describirá con más detalle el LDD concreto del SGBD MariaDB (fork de MySQL). La versión que se describe es la 10.6.5.

Tipos de datos

Los tipos de datos de MariaDB (tal y como se puede ver en la página del manual https://mariadb.com/kb/en/data-types/) son los siguientes:

- **Enteros**
  - `TINYINT`: 8 bits
  - `SMALLINT`: 16 bits
  - `MEDIUMINT`: 24 bits.
  - `INTEGER` ó `INT`: 32 bits
  - `BIGINT`: 64 bits.
- **Reales de punto fijo (número fijo de decimales)**
  - `DECIMAL(n, m)` ó `NUMERIC(n,m)`: Número de punto fijo con `n` dígitos en el número completo y `m` en la parte decimal. Dicho de otra manera, el número tiene `n` dígitos, de los que `m` forman la parte decimal y el resto (`n-m`) la parte entera.
- **Reales en punto flotante**
  - `FLOAT`: 32 bits
  - `DOUBLE`: 64 bits.
  - `FLOAT(T,D)`, `DOUBLE(T,D)`: Número real con `T` dígitos, de los cuales hasta `D` pueden ser decimales.

Adicionalmente los tipos enteros pueden contener el atributo `UNSIGNED` que hace que se almacene el número sin signo. También pueden tener el atributo `AUTO_INCREMENT` que genera una secuencia automática de números. Para ello, cada vez que se inserta una fila con un valor `NULL` ó `0` en esta columna, el SGBD genera el siguiente número de la secuencia y lo inserta en lugar de `NULL` ó `0`. (**para ello es importante utilizar el atributo `NOT NULL` en la declaración del campo al crear la tabla**). Si se da un valor numérico distinto de `0`, es ese valor el que se inserta y la secuencia se reinicializa a dicho valor, tomando la próxima vez el siguiente valor al proporcionado.

- **Fecha y hora**
  - `DATE`. Contiene una fecha, en formato `AAAA-MM-DD`, donde `AAAA` es el año, `MM` el mes y `DD` el día, todos en número.
  - `DATETIME`. Contiene una fecha y una hora dentro de esa fecha, en formato "`AAA-MM-DD HH:MM:SS.FFFFFF`" donde `AAAA`, `MM` y `DD` son como en `DATE`, `HH` son horas, `MM` minutos, `SS` segundos y `FFFFFF` fracciones de segundo.

- `TIMESTAMP`. Similar a `DATETIME` pero con un rango de fechas distinto a `DATETIME`.
- `TIME`. Contiene una hora en formato "`HHH:MM:SS.FFFFFF`". Formato similar a `DATETIME`. La hora puede tener tres dígitos si lo que se quiere representar una duración, en lugar de una hora del día.
- `YEAR(n)`. Representa un año. n puede ser 2 ó 4, aunque no se recomienda utilizar el valor 2 porque está marcado para desaparecer en futuras versiones.

- **Texto**
  - `CHAR(n)`. Cadena de tamaño fijo de n caracteres, donde n puede ser hasta 255. La cadena se almacena siempre con esa longitud. Si se almacena una cadena más corta se rellena con espacios, que se eliminan al consultar el dato.
  - `VARCHAR(n)`. Cadena de tamaño variable de hasta n caracteres, donde n puede ser hasta 65535. La cadena se almacena en formato `(longitud, datos)` por lo que sólo ocupa lo mínimo necesario más 2 bytes para almacenar la longitud.
  - `BINARY.(n)`. Como `CHAR` pero contiene bytes en lugar de caracteres, es decir, puede contener cualquier número binario.
  - `VARBINARY(n).` Como `VARCHAR` pero contiene bytes en lugar de caracteres.
  - `TEXT` y `BLOB`. Pueden contener texto (`TEXT`) o datos binarios (`BLOB`) de longitud arbitraria. Su uso no es muy recomendado porque puede afectar fuertemente el rendimiento del sistema.

Sintaxis CREATE DATABASE

```
CREATE [OR REPLACE] DATABASE [IF NOT EXISTS] nombredb
[CHARACTER SET nombrejuegocar]
[COLLATE nombreordenacion];
```

donde:

- La opción `OR REPLACE` comprueba si ya existe una base de datos con el nombre proporcionado y la elimina si éste es el caso. Si no existe una base de datos con el nombre proporcionado no tiene ningún efecto. **No puede ser utilizada junto a la opción `IF NOT EXISTS`.**
- La opción `IF NOT EXISTS` hace que se cree la base de datos sólo si no existe una ya con el nombre proporcionado. Si ya existe no se hace nada. Si no se especifica y ya existe una base de datos con ese nombre se producirá un error (y tampoco se hace nada). **No puede ser utilizada junto a la opción `OR REPLACE`.**
- `nombredb`. Nombre de la nueva base de datos a crear. Debe ser distinto a los nombres ya existentes o se producirá un error (si no se proporciona `IF NOT EXISTS`).
- `nombrejuegocar` es el nombre del juego de caracteres que se va a utilizar por defecto en la nueva base de datos para almacenar los valores de los campos de tipo texto. Es importante que sea adecuado al idioma que se va a utilizar en los valores para que no haya problemas con

determinadas letras, como la eñe o vocales acentuadas. La lista de juegos de caracteres es muy larga para incluirla aquí pero se puede consultar en la web de MariaDB.

- `nombreordenacion` es el nombre del sistema de ordenación que se va a utilizar al comparar cadenas en la base de datos. El sistema de ordenación establece cual es el orden alfabético entre los caracteres de un juego de caracteres. Por ejemplo, el sistema de ordenación español establece que la ñ va después de la n y antes de la o. Si se establece otro sistema es posible que la ñ esté en otra posición y al obtener una consulta ordenada las palabras que comienzan por Ñ no estén en la posición que deberían ocupar en la lista.

Página del manual: https://mariadb.com/kb/en/create-database/

Sintaxis ALTER DATABASE

```
ALTER DATABASE nombredb
[CHARACTER SET nombrejuegocar]
[COLLATE nombreordenacion];
```

La sintaxis y las opciones, junto con su significado son muy similares a las de `CREATE DATABASE`

Página del manual: https://mariadb.com/kb/en/alter-database/

Sintaxis DROP DATABASE

```
DROP DATABASE [IF EXISTS] nombredb;
```

donde:

- `IF EXISTS`, si se usa, previene que ocurra un error al intentar eliminar una base de datos que no existe. Si se usa y la base de datos no existe no ocurre nada. Si no se usa y la base de datos no existe se produce un error.
- `nombredb`. Nombre de la base de datos a eliminar.

Página del manual: https://mariadb.com/kb/en/drop-database/

Sintaxis CREATE TABLE

Además de lo ya indicado en la descripción general y al uso de los tipos propios de MariaDB, no hay mucho más que añadir, excepto que se pueden indicar una serie de opciones, entre ellas, por ejemplo:

- `OR REPLACE`. Si se utiliza la opción `OR REPLACE` (justo entre `CREATE` y `TABLE`), se comprueba si ya existe una tabla con dicho nombre en la base de datos y se elimina antes de realizar la creación. De esta forma se puede reemplazar una tabla por otra con una nueva definición y sin datos. **No se puede utilizar junto a la opción `IF NOT EXISTS`**.

- `IF NOT EXISTS`. Esta opción, que si se usa debe ir justo después de TABLE, indica que si la tabla ya existe, en lugar de dar un error, se termina silenciosamente sin hacer nada. No se puede utilizar junto a la opción `OR REPLACE`.
- `ENGINE nombremotor`. MariaDB posee la particularidad de que puede utilizar distintas implementaciones físicas o *motores* para almacenar la información de las tablas, cada uno con sus peculiaridades y adaptado a un tipo de tarea. De entre ellos, el más utilizado es el InnoDB que es más moderno y permite el uso de funcionalidades avanzadas, como transacciones y claves ajenas, que no son soportadas en otros motores, por lo que se recomienda su uso, si no está claro cual motor utilizar. Otra opción es el motor MyISAM, que carece de funcionalidad avanzada pero es eficiente en espacio y en tiempo y puede ser utilizado si las características avanzadas no se requieren. También hay motores especializados en interactuar con datos en formatos de otras aplicaciones o de sólo lectura. Hay que consultar la documentación oficial y la no oficial para determinar cual es el motor más adecuado a nuestras necesidades. En este curso se utilizará el motor InnoDB si no se especifica expresamente lo contrario. Para más información consultar (https://mariadb.com/kb/en/choosing-the-right-storage-engine/)
- `CHARACTER SET nombrejuegocaracteres`. Permite utilizar un juego de caracteres en esta tabla distinto al general especificado para la base de datos. Si no se especifica se utiliza el que se haya definido para la base de datos.
- `COLLATE nombreorden`. Igual que `CHARACTER SET`
- `AUTO_INCREMENT valor`. Valor inicial para las columnas `AUTO_INCREMENT` de la tabla.
- `COMMENT “comentario”`. Permite introducir un comentario sobre la tabla. Útil para almacenar descripciones o información para los administradores / desarrolladores.
- Para crear un índice no hay que utilizar la palabra clave `CONSTRAINT`, vista en la sintaxis general. El resto es idéntico.

Página del manual: https://mariadb.com/kb/en/create-table/

Sintaxis ALTER TABLE

Sin comentarios adicionales sobre la definición general.

Página del manual: https://mariadb.com/kb/en/alter-table/

Sintaxis DROP TABLE

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

Si se utiliza `IF EXISTS` y la tabla a borrar no existe no se produce error.

Página del manual: https://mariadb.com/kb/en/drop-table/

Uso de la consola de MariaDB

MariaDB (y MySQL) se pueden administrar mediante la consola de comandos.

Para ello hay que abrir un terminal y en el prompt ejecutar la consola mysql con el comando:

```
mysql -u root -p
```

La opción `-u usuario`, indica que se dese iniciar sesión como el usuario indicado. En este caso se utilizan el usuario `root`, que es el usuario administrador de la base de datos. Normalmente es una mala idea explotar una base de datos utilizando el usuario administrador, ya que un error puede provocar efectos a nivel de todo el SGBD pero hasta que se describa el sistema de permisos, es el que se va a utilizar. La opción `-p` indica que la consola debe solicitar la contraseña del usuario `root`.

Si todo va bien, se deberá ver un mensaje de MySQL y el prompt de la consola

```
MariaDB>
```

o bien

`MySQL>`

A partir de este momento se pueden comenzar a escribir comandos que se ejecutarán al pulsar la tecla ENTER.

Además de las sentencias SQL, MariaDB dispone de algunos comandos de control que pueden ser muy útiles:

- `show databases;`

  Muestra una lista de las bases de datos existentes en el sistema. Útil para comprobar si una base de datos existe ya o no.

- `use nombredb;`

  Hace que la base de datos con el nombre nombredb sea la base de datos activa o seleccionada. A partir de ese momento, las sentencias que se ejecuten lo harán sobre dicha base de datos.

- `show tables;`

  Cuando hay una base de datos seleccionada muestra las tablas que contiene.

- `quit`

  No necesita punto y coma al final. Termina la sesión de consola MySQL y sale al prompt del S.O.

Otro comando muy útil es `mysqldump`.

`mysqldump` hace un volcado de una base de datos ya existente en formato SQL, esto es, genera, a partir de una base de datos existente, un archivo SQL con las sentencias necesarias para crear la base de datos, sus objetos y rellenar las tablas con los datos que tenía.

La sintaxis básica que se va a utilizar de `mysqldump` es:

```
mysqldump -u root -p nombredb
```

donde `nombredb` es el nombre de la base de datos cuyo volcado se desea generar. Tras solicitar y obtener la contraseña del administrador, este comando volcará a consola las sentencias. Si se desea que estas se almacenen en un archivo para su uso posterior, habrá que redireccionar la salida estándar a un archivo con la siguiente forma:

```
mysqldump -u root -p nombredb >archivosalida
```

donde `archivosalida` es el nombre del archivo de salida que queremos generar. Si no existe, se creará y si ya existe, se sobreescribirá.

Por ejemplo, el comando:

```
mysqldump -u root -p hospital >hospital.sql
```

creará un archivo llamado `hospital.sql` con las sentencias necesarias para crear y rellenar la base existente hospital.