Modelo Físico: SQL - Instrucciones DDL y DML

Propuesta didáctica

Una vez conocido el modelo relacional, en esta unidad vamos a comenzar a trabajar el RA2 "**Crea bases de datos definiendo su estructura y las características de sus elementos según el modelo relacional**", además de continuar con el RA5 "**Modifica la información almacenada en la base de datos utilizando asistentes, herramientas gráficas y el lenguaje de manipulación de datos**".

Criterios de evaluación

Respecto al RA2:

- **CE2a**: Se ha analizado el formato de almacenamiento de la información.
- **CE2b**: Se han creado las tablas y las relaciones entre ellas.
- **CE2c**: Se han seleccionado los tipos de datos adecuados.
- **CE2d**: Se han definido los campos clave en las tablas.
- **CE2e**: Se han implantado las restricciones reflejadas en el diseño lógico.

Respecto al RA4:

- **CE4a**: Se han identificado las herramientas y sentencias para modificar el contenido de la base de datos.
- **CE4b**: Se han insertado, borrado y actualizado datos en las tablas.

Contenidos

Bases de datos relacionales:

- Lenguaje de descripción de datos (DDL).

Tratamiento de datos:

- Inserción, borrado y modificación de registros.
- Integridad referencial.

Cuestionario inicial

1. ¿Cuál es el propósito de las sentencias DDL?
2. ¿Y DML?
3. ¿Cómo averiguamos tanto las tablas de una base de datos como su estructura?
4. ¿Para qué sirve el comando CREATE TABLE?
5. ¿Qué diferencia CREATE TYPE de CREATE DOMAIN?
6. ¿Puedo eliminar las tablas en cualquier orden mediante DROP TABLE?
7. ¿Es obligatorio que una clave ajena siempre referencie a una clave primaria?
8. ¿Para qué sirve el comando INSERT INTO?
9. ¿Qué sucede si omitimos el WHERE en un DELETE FROM?

Programación de Aula (20h)

Esta unidad es la tercera, con lo que se imparte en la primera evaluación, a mediados del mes de noviembre, con una duración estimada de 11 sesiones lectivas:

Sesión	Contenidos	Actividades	Criterios trabajados
1	Modelo físico. Puesta en marcha	AC301	CE2a
2	DDL Bases de datos y tipos de datos		
3	DDL Tablas	AC303	CE2b, CE2c, CE2d
4	Restricciones	AC304	CE2e
5	Supuesto DDL	AC306	CE2b, CE2c, CE2d, CE2e
6	Cargando y exportando datos		CE4a
7	DML. Inserciones	AC309	CE4a, CE4b
8	DML. Actualizaciones y Borrados	AC310	CE4b
9	Supuesto DDL-DML I	AC313	CE2b, CE2c, CE2d, CE2e
10	Supuesto DDL-DML II	AC314	CE4b
11	Reto - Diseño físico	PR315	CE2a, CE2b, CE2c, CE2d, CE2e, CE4a, CE4b

Al finalizar esta unidad, comenzaremos un nuevo reto, centrado en el modelo físico, creando y cargando datos.

Modelo físico

El modelo físico es la traducción del modelo lógico a un modelo interno, dando lugar a un esquema físico interpretable por un SGBD concreto, paso final del diseño de las bases de datos.

Del diseño a los modelos

Para ello, transformaremos el modelo relacional al modelo físico utilizando un lenguaje específico de cada SGBD que nos permite definir los elementos, el cual se conoce como DDL (*Data Definition Language*).

SQL

SQL (*sequel* en inglés) es la *lingua franca* para interactuar con las bases de datos relacionales. Es un lenguaje declarativo, donde lo importante es definir qué se quiere hacer, en lugar de cómo se va a hacer.

Su origen se remonta a 1970 cuando *Edgar Frank Codd* publica el documento "*Un modelo relacional de datos para grandes bancos de datos compartidos*", donde se definieron las bases del modelo relacional.

Versiones

El lenguaje SQL ha ido evolucionando conforme lo han hecho los diferentes SGBD y tecnologías, desde el primer estándar en el archivo 1986 hasta la última versión de 2023:

- SQL-86: funcionalidad mínima para que un lenguaje se considere SQL, introduciendo las sentencia SELECT, CREATE TABLE, INSERT, ... y JOIN.
- SQL-89: añade instrucciones para gestionar las claves ajenas (reglas de integridad referencial).
- SQL-92: standard base. Introduce los tipos de datos para trabajar con fechas, como DATE, TIME y TIMESTAMP, la agregación de consultas con GROUP BY y sus operadores COUNT y SUM, así como la validación de los campos con CHECK.
- SQL:1999 (SQL3): se añaden soporte para la programación orientada objetos y el uso de *triggers* y procedimientos almacenados mediante PL/SQL.
- SQL:2003: añade características de XML y las funciones ventana, así como el uso de SEQUENCE para generar valores únicos.
- SQL:2008: introduce soporte para datos temporales, la instrucción MERGE para combinar datos y mejoras en las expresiones de tabla comunes (con WITH).
- SQL:2016: introduce soporte nativo para JSON y funciones para manejar datos geoespaciales.

SQL se divide a su vez en cuatro "sublenguajes" respecto a la funcionalidad que ofrecen:

- **DDL** (Data Definition Language): permite crear y manipular la estructura de una base de datos.
- **DML** (Data Manipulation Language): permite recuperar, almacenar, modificar, y eliminar datos de una BD
- **DCL** (*Data Control Language*): permite crear roles, permisos y usuarios, controlando el acceso a los elementos de nuestras bases de datos.
- **TCL** (*Transaction Control Language*): administra las modificaciones creadas con el DML.

Sintaxis

La sintaxis de SQL se define mediante instrucciones que tiene diferentes parámetros, unos opcionales, otros con diferentes valores, etc.. La nomenclatura que vamos a emplear al explicar la sintaxis de cada instrucción será la siguiente:

SELECT [ALL | DISTINCT] columna1 [,columna2, columna3,.....] | *

FROM tabla1 [tabla2, tabla3, ...]

[WHERE condición]

[ORDER BY expr1 [DESC | ASC] [, expr2 [DESC | ASC] ....]

Símbolo	Descripción
MAYÚSCULAS	Palabras reservadas (*keywords*) de SQL
minúsculas	variable que hay que sustituir por un elemento concreto
[]	opcional
|	se elige uno de los valores
[]	separa opciones alternativas
{}	obliga a elegir uno de los valores
...	número variable de datos

Case sensitive

En entornos Windows no se distingue entre mayúsculas y minúsculas, pero en entornos Linux sí.

Destacar que las instrucciones finalizan con el signo de punto y coma (😉 y que se emplea el doble guion (--) para introducir comentarios. Además, cualquier instrucción SQL puede ser partida por espacios o saltos de línea antes de finalizar la instrucción, facilitando su lectura. Así pues, posibles sentencias que cumplen la sintaxis serían:

-- Ejemplos de instrucciones SQL

SELECT nombre, dni FROM cliente WHERE salario > 100;

SELECT * FROM cliente WHERE salario > 100 ORDER BY salario;

SELECT DISTINCT nombre

FROM cliente

ORDER BY salario DESC;

DDL

Al empezar esta unidad habíamos comentado que DDL es el lenguaje encargado de definir las estructuras de las bases de datos, esto es, permite la creación, modificación y eliminación de los objetos de la base de datos (metadatos), como, por ejemplo, las bases de datos, tablas, índices o vistas.

TRUNCATE

Mientras que con DROP podemos borrar una tabla, con TRUNCATE vaciaremos las tablas de contenido. Es por ello, que no se considera una operación DDL, sino más bien DML.

Para ello, utilizaremos principalmente tres tipos de sentencias:

- CREATE: permite crear nuevas tablas, vistas e índices.
- ALTER: permite modificar las tablas agregando o eliminando campos, cambiar la definición de tipos, etc...
- DROP: empleado para eliminar las tablas y los índices.

Puesta en marcha

Una vez instalados los SGBD y conectados a los mismos, ya sean mediante la consola o la interfaz gráfica, debemos saber que aunque el 90% de la sintaxis de SQL es común, cada SGBD tiene sus particularidades. Es por ello, que en este curso vamos a trabajar con *MariaDB* y *Postgress* de manera indistinta.

Recuerda que todas las instrucciones terminan en ; y que vamos a dejar en MAYÚSCULAS las *keywords* para facilitar la lectura, aunque no es obligatorio escribirlas así.

Autoevaluación

En este momento deberías tener tanto *MariaDB* como *Oracle* instalado en tu máquina debian y ser capaz de conectarte desde la línea de comandos.

Una vez dentro, ejecuta los siguientes comandos y comprueba si entiendes qué realiza cada una de las siguientes sentencias:

![ref4]![ref1]

MariaDBPostgress

```
SHOW DATABASES;

USE mibd;

SHOW TABLES;

DESCRIBE mitabla;

select version();

SELECT datname FROM pg_database;
```

Comencemos con un caso muy sencillo, donde vamos a crear una tabla con un campo, insertar un registro y finalmente consultar la información recién almacenada:

```
CREATE TABLE ejemplo8a (
    id INT PRIMARY KEY,
    nombre VARCHAR(64) NOT NULL);
INSERT INTO ejemplo8a VALUES (1, "Pedro Picapiedra");
SELECT * FROM ejemplo8a;
```

Una vez conectados a nuestros SGBD y comprobado que podemos crear estructuras, insertar datos y consultarlos, vamos a profundizar en la definición de los esquemas relacionales.

Bases de datos

El principal elemento en un SGBD es la base de datos.

Cuando nos conectamos indicaremos siempre a qué base de datos lo hacemos. De todos modos, siempre podemos consultar las bases de datos existentes:

En el caso de *MariaDB* usaremos el commando `SHOW DATABASES`:

```
MariaDB [pruebas]> show databases;
```

```
| Database           |
|--------------------|
| information_schema |
| pruebas            |
|--------------------|

2 rows in set (0.001 sec)
```

Para crear una base de datos, usaremos la sentencia `CREATE DATABASE`:

```
CREATE [OR REPLACE] DATABASE [IF NOT EXISTS] nombreBD;
```

Por ejemplo:

```
create database midb;

create database if not exists severo;
```

Más información en la documentación oficial

Si necesitamos borrarla, porque ya no la necesitamos, usaremos la sentencia `DROP DATABASE` :

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

Por ejemplo:

```
drop database prueba;

drop database if exists severo;
```

Almacenamiento

A la hora de almacenar la información de una base de datos, tanto a nivel lógico como físico, necesitamos de diferentes elementos que conviene conocer.

Motores de almacenamiento

En los SGBD, y en concreto en *MariaDB*, los motores de almacenamiento (*storage engines*) son fundamentales para definir el formato de almacenamiento, es decir, cómo se almacenan y gestionan los datos en las bases de datos. *MariaDB* ofrece diferentes motores de almacenamiento que proporcionan distintas capacidades, los cuales influyen en cómo gestionan y organizan los datos en disco, afectando el rendimiento, la concurrencia y la eficiencia del espacio.

- ***MariaDB***

  Los motores más conocidos son:

  - ***InnoDB***: motor de almacenamiento predeterminado en *MariaDB*, desde la versión 10.2. Proporciona soporte completo para transacciones ACID. Maneja bloqueo a nivel de fila y cuenta con soporte para claves ajenas. Finalmente, es adecuado para aplicaciones donde la integridad y las transacciones sean críticas.
  - ***MyISAM***: Históricamente, fue el motor de almacenamiento por defecto en *MySQL* antes de *InnoDB*. Su uso ha caído al no ser transaccional y no soportar claves ajenas. Se utiliza principalmente en aplicaciones que priorizan el rendimiento de las lecturas sobre la integridad de los datos.

Existen otros motores más específicos como *TokuDB* (para grandes volúmenes de datos con alta concurrencia y compresión), *Aria* (utilizado para recuperación rápida y consultas de solo lectura), *ColumnStore* (en aplicaciones de análisis de datos) o *Memory* (para tablas temporales y cachés rápidas)

Además, es posible ajustar su comportamiento, por ejemplo, permitiendo acceder a datos externos a través de FDW (*Foreign Data Wrappers*), y optimizaciones internas mediante tablas particionadas y configuraciones adicionales.

Bloqueo a nivel de fila (*row-level locking*)

Técnica de control de concurrencia que permite a las bases de datos bloquear únicamente las filas que están siendo accedidas o modificadas por una transacción, en lugar de bloquear toda la tabla o bloques más grandes de datos.

Permite que múltiples transacciones trabajen en diferentes partes de una tabla sin interferir entre sí.

MVCC - *Multiversion Concurrency Control*

Cuando hay una modificación de datos, en lugar de sobrescribir una fila en actualizaciones, se almacena una nueva versión de la fila mientras se mantiene la versión anterior hasta que ya no sea necesaria (las transacciones que la usan han finalizado).

Las versiones anteriores de una fila se almacenan en el espacio de "deshacer" (*undo space*) para permitir transacciones y lecturas consistentes.

Jerarquía lógica de una BD*

Jerarquía lógica

A nivel lógico, los elementos que organizan la información de una base de datos se definen mediante la siguiente jerarquía:

- *Tablespace* o espacio de tablas: estructura física (normalmente una carpeta/directorio) para agrupar varias bases de datos o esquemas.
- Base de datos: elemento lógico de nivel superior
- Esquema: agrupación lógica de una o más tablas (en *MySQL*/*MariaDB* un esquema es una base de datos).
- Tabla: representación de una tabla relacional, que contiene datos organizados en filas y columnas. Cada tabla pertenece a un base de datos específica, y sus datos se almacenan en el *tablespace* asociado.
- Fila: cada uno de los registros de la tabla

En *MariaDB* no está soportado crear tablespaces

En cambio, a nivel físico, el formato de almacenamiento depende principalmente del *motor de almacenamiento* que se esté utilizando, ya que cada motor tiene su propia manera de organizar y almacenar los datos.

- ***MariaDB - InnoDB***
- Un bloque almacena los datos de tablas e índices mediante páginas (normalmente de 16 KB)
- Los bloques se almacenan en un archivo de tablaspace (generalmente ibdata1 para bases de datos que comparten un *tablespace*, o en un archivo por tabla).
- Respecto a las filas, se almacenan utilizando un formato compacto o redondeado, dependiendo de la configuración. Cada fila contiene una clave primaria (o un índice que actúa como clave primaria) y los datos correspondientes.

Juegos de caracteres

la colación, especifica cómo tratar el alfabeto del juego de caracteres, es decir, cómo se ordena

Entendemos como **juego de caracteres** (*character sets*) a la codificación que utiliza el SGBD para representar los datos, definiendo qué caracteres pueden almacenarse en la base de datos y cómo esos caracteres son codificados internamente.

Tal como ya habéis estudiado en el módulo profesional de *Sistemas Informáticos*, los juegos de caracteres más conocidos son ASCII (caracteres básicos en inglés), iso-8859-1 latin (expande ASCII para incluir los caracteres acentuados y símbolos de los lenguajes occidentales) y utf-8, el cual parte de la familia de estándares *Unicode*, que puede representar caracteres de casi todos los lenguajes del mundo, incluidos caracteres *multibyte*.

Además, podemos configurar la **colación** (*collation*), la cual define cómo se comparan y ordenan las cadenas de texto para un juego de caracteres específico (por ejemplo, la ñ después de la n y no conforme a la tabla de códigos ASCII), teniendo en cuenta cosas como el uso de mayúsculas y minúsculas o el tratamiento de acentos (por ejemplo, si Á es igual que á).

En *MariaDB* y *PostgreSQL*, los juegos de caracteres y las colaciones juegan un papel importante en cómo se almacenan y comparan las cadenas de texto. Cada sistema maneja los juegos de caracteres de manera diferente, y a continuación vamos a estudiar cómo funciona en cada uno:

- ***MariaDB***

> Admite múltiples juegos de caracteres y colaciones, que pueden establecerse a nivel de base de datos, tabla, columna o incluso a nivel de sesión.
>
> Así pues, ofrece una gran flexibilidad con varios juegos de caracteres (como utf8, utf8mb4, latin1, etc.) y permite especificar colaciones para determinar cómo se comparan y ordenan los textos.

La codificación y la colación se especifican al crear una base de datos mediante CREATE DATABASE con las opciones CHARACTER SET y COLLATE.

Además de utf8, latin1y ascii ya comentados previamente, una codificación muy empleado es **utf8mb4**, la cual es una extensión de utf8 que permite almacenar todos los caracteres de *Unicode*, incluidos emojis y otros símbolos que no se pueden representar en utf8.

Además, luego para cada juego de caracteres, tenemos sus colaciones. Por ejemplo, el juego de caracteres utf8 tiene varias colaciones como:

- utf8_general_ci: Colación insensible a mayúsculas/minúsculas (_case-insensitive*).
- utf8_bin: Colación binaria, sensible a mayúsculas/minúsculas y que compara los caracteres basándose en sus valores binarios.

```
CREATE DATABASE mi_base_de_datos

CHARACTER SET utf8mb4

COLLATE utf8mb4_unicode_ci;
```

Tipos de datos

Antes de entrar a ver cómo podemos crear las tablas, necesitamos conocer los tipos de datos que van a poder tener las columnas que definamos. *A grosso modo*, los tipos más empleados en MariaDB son:

Tipo de dato	*MariaDB*	Descripción
Numéricos		
TINYINT	Sí	Entero pequeño, de 1 byte
SMALLINT	Sí	Entero pequeño, de 2 bytes
MEDIUMINT	Sí	Entero de 3 bytes (*MariaDB*)
INT, INTEGER	Sí	Entero estándar de 4 bytes
BIGINT	Sí	Entero grande de 8 bytes
DECIMAL, NUMERIC	Sí	Número con precisión fija (exacta)
FLOAT	Sí	Número de coma flotante de precisión simple

Tipo de dato	*MariaDB*	Descripción
DOUBLE	Sí	Número de coma flotante de precisión doble
Cadenas		
CHAR(n)	Sí	Cadena de longitud fija
VARCHAR(n)	Sí	Cadena de longitud variable
TEXT	Sí	Cadena de longitud ilimitada
Fechas y Horas		
DATE	Sí	Fecha (año, mes, día)
TIME	Sí	Hora (sin zona horaria)
DATETIME	Sí	Fecha y hora
TIMESTAMP	Sí	Marca de tiempo (con o sin zona horaria)
Booleanos		
BOOLEAN	Sí (alias de TINYINT)	Tipo booleano

Algunos ejemplos de tipos serían:

```
BIGINT

CHAR(8)

VARCHAR(255)

DECIMAL(4,2) -- 5 dígitos en total, con 2 decimales, por ejemplo 123,45
```

Enumeraciones

Además de los tipos vistos, podemos crear enumeraciones como un conjunto de valores restringidos que puede tomar un determinado campo, haciendo uso de la keyword ENUM en MariaDB

```
curso ENUM ('0', '1', '2'),
```

```
horario ENUM ('mañana', 'tarde', 'noche')
```

Primero debemos definir un tipo y luego asociar ese tipo al campo:

```
CREATE TYPE animo AS ENUM ('feliz', 'triste', 'normal');

estado animo;
```

Tablas

Dentro de la definición de datos, la creación de tablas es la parte más importante ya que nos permite trasladar nuestro modelo relacional al modelo físico.

Para crear una tabla emplearemos la sentencia `CREATE TABLE`. La sintaxis básica compartida por todos los SGBD es la siguiente:

```
CREATE TABLE [OR REPLACE] [basededatos.]nombreDeTabla (

  columna1 tipoDato1 [propiedadesColumna],

  ...

  columnaN tipoDatoN [propiedadesColumna],

  [restriccionesTabla]

);
```

Conviene revisar la sintaxis de MariaDB para comprobar los detalles de cada SGBD.

Por ejemplo, para crear una tabla con un único atributo podríamos hacer:

```
CREATE TABLE USUARIO (

   nombre VARCHAR(25)

);
```

Algunas restricciones que hemos de tener en cuenta son:

- No puede haber nombres de tablas repetidas.
- El nombre de la tabla debe comenzar por un carácter alfabético y su longitud máxima es de 30 caracteres. Sólo se permiten letras del alfabeto inglés, dígitos o el signo de guión bajo.

- No podemos crear tablas cuyo nombre coincida con las palabras reservadas de SQL (por ejemplo, no podemos llamar a una tabla WHERE).
- Como convención, vamos a nombrar las TABLAS en MAYÚSCULAS y las columnas en minúsculas.
- En principio, respecto a los nombre de las tablas y las columnas, los SGBD no son *case sensitive* (no distinguen entre mayúsculas y minúsculas), aunque puede depender de la configuración del sistema operativo (sobre todo en sistemas Linux/Unix) o el uso de comillas dobles (no recomendado). En cambio, en los datos, depende de la configuración de la colación.

A continuación tenemos algunos ejemplos de creaciones de tablas con diferentes tipos de datos:

```
CREATE TABLE PROVEEDOR(nombre VARCHAR(32));

CREATE TABLE CLIENTE (

  nombre VARCHAR(32),

  localidad VARCHAR(30) DEFAULT 'Elche');

CREATE TABLE PRESTAMO (

  idPrestamo DECIMAL(8),

  fechaPrestamo DATE DEFAULT (CURRENT_DATE));

CREATE TABLE CIUDAD (

  nombre CHAR(20) NOT NULL,

  poblacion INT NULL,

  codigoPostal DECIMAL(6) NOT NULL DEFAULT 03203);

CREATE TABLE PROVINCIA (

  nombre CHAR(20) NOT NULL,

  poblacion INT DEFAULT 5000);
```

Con estos ejemplos, hemos visto que algunas de las propiedades de columna que podemos indicar son los valores no nulos (mediante NOT NULL) o valores por defecto (mediante DEFAULT). En el apartado Restricciones las estudiaremos en profundidad.

En todo momento, podemos obtener la estructura de una tabla mediante el comando `DESCRIBE nombreTabla`:

```
DESCRIBE CIUDAD;
```

Clave primaria

Una vez visto como crear la tablas y sus atributos, nos centramos en la creación de la clave primaria. Toda tabla debe tener una clave primaria (ya sea simple o compuesta). Para ello, si tenemos un atributo simple, podemos añadir la propiedad PRIMARY KEY al atributo, aunque por convenciones de código, es mejor definir la clave primaria después de todos los atributos, a modo de restricciones.

```
CREATE TABLE CIUDAD (
  nombre VARCHAR(20) PRIMARY KEY, -- clave primaria como propiedad
  poblacion INT DEFAULT 5000);
CREATE TABLE PROVINCIA (
  nombre VARCHAR(20),
  poblacion INT DEFAULT 5000,
  PRIMARY KEY(nombre)); -- clave primaria como restricción
CREATE TABLE EMPLEADO (
  codigo VARCHAR(9),
  departamento VARCHAR(15),
  nombre VARCHAR(40),
  PRIMARY KEY(codigo, departamento)); -- clave primaria compuesta
```

Un buena práctica es definir un atributo a modo de codigo (o id) asociado a un número entero autoincrementable. Para ello, tras la definición del atributo como entero (sin signo), añadiremos la propiedad AUTO_INCREMENT en la definición del atributo.

```
CREATE TABLE PERSONA (
  id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
  nombre VARCHAR(40),
  fecha DATE);
```

En *Oracle* se utilizan las secuencias en vez de los tipos autoincrementables. Tras cada inserción, la secuencia toma un nuevo valor.

```
CREATE SEQUENCE pid_seq AS integer;
```

```
CREATE TABLE PERSONAPS (
   id integer NOT NULL DEFAULT nextval('pid_seq')
);
```

Creando tablas con consultas

Un caso particular de la creación de tablas es hacerlo a partir del resultado de una consulta, de manera que contiene la estructura (tipos de datos) y los datos obtenido. Para ello, en vez de indicar los campos y restricciones, realizamos la consulta mediante SELECT:

```
CREATE TABLE nombreTabla AS SELECT…
```

Por ejemplo, para crear una tabla GERENTE con los datos de empleados catalogados como Gerentes, haríamos:

```
CREATE TABLE GERENTE AS
  SELECT e.nombre, e.apellido1, e.apellido2, e.email
  FROM EMPLEADO e
  WHERE e.puesto LIKE "%Gerente%";
```

Volveremos a esta posibilidad cuando aprendamos a realizar consultas mediante SQL:

Modificando tablas

Una vez creada una tabla, podemos modifica su estructura mediante las sentencias **ALTER TABLE**, en MariaDB

Para añadir columnas, usaremos la opción **ADD**:

```
ALTER TABLE nombreTabla ADD (
  nombreColumna tipoDatos [propiedades]
  ,columnaSiguiente tipoDatos [propiedades]…)
```

De este modo, si queremos añadir un campo para almacenar el teléfono de los empleados (sus campos estarán inicializados a NULL cada cada registro de la tabla EMPLEADO), haríamos:

```
ALTER TABLE EMPLEADO ADD (telefono VARCHAR(12));
```

En cambio, si queremos eliminar una columna, usaremos la opción DROP:

```
ALTER TABLE nombreTabla DROP (columna [,columnaSiguiente,...]);
```

Así pues, si queremos eliminar el teléfono recién creado:

```
ALTER TABLE EMPLEADO DROP (telefono);
```

Finalmente, si queremos modificar el tipo de una columna, usaremos la opción **MODIFY**:

```
ALTER TABLE nombreTabla MODIFY(
  columna tipo [propiedades]
  [,columnaSiguiente tipo [propiedades]]
```

Un caso particular de las operaciones de modificación es renombrar una tabla. Para ello, usaremos la opción RENAME TO:

```
ALTER TABLE nombreTablaViejo RENAME TO nombreNuevo;
```

Autoevaluación

Si tenemos las siguientes tablas:

```
CREATE TABLE CLIENTE (
dni VARCHAR(9) PRIMARY KEY,
cnombre VARCHAR(50),
direccion VARCHAR(60));
CREATE TABLE MASCOTA (
codigo INTEGER PRIMARY KEY,
nombre VARCHAR(50),
raza VARCHAR(50));
```

¿Sabes qué realizan las siguientes operaciones?

```
ALTER TABLE MASCOTA ADD especie VARCHAR(10) AFTER raza;

ALTER TABLE MASCOTA ADD cliente VARCHAR(9) AFTER nombre;

ALTER TABLE MASCOTA ADD CONSTRAINT fk_duenyo FOREIGN KEY (cliente) REFERENCES
CLIENTE(dni);

ALTER TABLE MASCOTA MODIFY codigo INT(3) AUTO_INCREMENT;
```

¿Sabes averiguar que realiza esta sentencia?

```
ALTER TABLE clientes RENAME nombre TO nomMascosta VARCHAR(50);
```

Eliminando tablas

Para eliminar tablas se emplea la instrucción `sql DROP TABLE nombreTabla`, tanto en MariaDB como en PostgreSQL. Hemos de tener en cuenta que cuando tenemos un base de datos con tablas relacionadas mediante claves ajenas, la integridad referencial restringe el borrado de una clave primaria referenciada por una clave ajena. Es por ello que el orden importa, y tenemos que ir eliminando primero las tablas que contienen las claves ajenas y no son referenciadas por otras tablas.

DROP TABLE CLIENTE;

Restricciones

Una vez hemos visto los casos más sencillos, hemos de saber que las tablas van a contener restricciones que aportan mayor integridad a los datos, en forma de claves ajenas, validaciones de campos, campos no nulos, únicos, etc... representando las restricciones estudiadas en el modelo relacional.

Así pues, si volvemos a la creación de tablas, y nos centramos en *MariaDB* podemos indicar las restricciones mediante diferentes parámetros:

```
CREATE [TEMPORARY] TABLE [db.]tabla (

campo1 tipo [(tamaño)]
  [NOT NULL | NULL]
  [DEFAULT valor] -- valor por defecto
  [UNIQUE [KEY] | PRIMARY KEY] -- clave única o primaria
  [REFERENCES tablaexterna [(campoexterno1, campoexterno2)] -- atributo clave
ajena
  [ON DELETE {CASCADE | SET NULL | NO ACTION}] -- propagación de borrados
  [ON UPDATE {CASCADE | SET NULL | NO ACTION}], -- propagación de actualizaciones
  ....,
  [CONSTRAINT de múltiples campos]) ENGINE = Innodb; -- motor de ejecución
```

Las restricciones las podemos definir a nivel de opciones dentro de cada campo, o tras la definición de todos los campos, mediante la *keyword* CONSTRAINT tanto en MariaDB .

Claves únicas

Cuando marcábamos en el modelo relación una propiedad como UK, es decir, clave única o alternativa, estamos indicando que dicho atributo no permite repetidos, y provocará la creación de un índice (los índices los estudiaremos en la unidad 9). Para ello, ahora usaremos la propiead **UNIQUE**.

Recuerda que las claves alternativas, normalmente, son una de las claves candidatas que no hemos elegido como clave primaria:

```
CREATE TABLE EMPLEADO(
  dni VARCHAR(9) PRIMARY KEY,
  nSegSocial VARCHAR(15) UNIQUE,
  nombre VARCHAR(40));
```

La principal diferencia entre una clave primaria y una clave única es que la clave única permite valores nulos.

Claves ajenas

Para indicar las claves ajena, aunque podemos hacerlo a nivel de campo, es mucho mejor acostumbrarse a definirlas siempre como restricciones. Para ello, usaremos la siguiente estructura, donde col es el campo que apunta a la clave primaria clave de la tabla tabla.

```
CONSTRAINT nombre FOREIGN KEY col REFERENCES tabla (clave)
```

A toda esta definición le pondremos un nombre por si más adelante necesitamos acceder a ella para modificarla, eliminarla, etc... Para definir este nombre, una convención es anteponer el tipo de restricción (PK, FK, CK, ...), y utilizando la barra de subrayado como separador, los nombres o siglas de los elementos involucrados.

Así pues, un ejemplo de tabla con clave primaria y claves ajenas definidas mediante CONSTRAINT sería:

```
CREATE TABLE ALQUILER (

 dni VARCHAR(9),

 codPelicula INT UNSIGNED,

 CONSTRAINT PK_ALQUILER PRIMARY KEY (dni, codPelicula),

 CONSTRAINT FK_ALQ_CLI FOREIGN KEY (dni) REFERENCES CLIENTE(dni),

 CONSTRAINT FK_ALQ_PEL FOREIGN KEY (codPelicula) REFERENCES PELICULA(cod)
```

```
);
```

Cuando tenemos una clave ajena compuesta, sólo hemos de indicar los atributos separándolos con comas:

```
CREATE TABLE EXISTENCIA (
  tipo CHAR(9),
  modelo DECIMAL(3),
  numAlmacen DECIMAL(1)
  cantidad DECIMAL(7),
  CONSTRAINT PK_EXISTENCIAS PRIMARY KEY(tipo, modelo, numAlmacen)
  CONSTRAINT FK_EXI_PIE FOREIGN KEY (tipo, modelo) REFERENCES PIEZA,
  CONSTRAINT FK_EXI_ALM FOREIGN KEY (numAlmacen) REFERENCES ALMACEN,
);
```

Propagación

Cuando se elimina (ON DELETE) o actualiza (ON UPDATE) un campo que está referenciado mediante una clave ajena, debemos indicar qué comportamiento de propagación debe realizar el SGBD con los datos en las tablas origen.

Vamos a retomar el ejemplo que vimos en la unidad 3, sobre un estudiante y las asignaturas que cursa, el cual se podría traducir en el siguiente DDL:

```
CREATE TABLE ESTUDIANTE (

  nif varchar(9),

  codigo int,

  nombre varchar(32),

  fMatricula date,

  direccion varchar(64),

  CONSTRAINT PK_ESTUDIANTE PRIMARY KEY(nif)

);

CREATE TABLE CURSAR (

  nifEstudiante varchar(9),

  asignatura int,

  anyo int,
```

```
  repetidor boolean,
  CONSTRAINT PK_CURSAR PRIMARY KEY(nifEstudiante, asignatura, anyo),
  CONSTRAINT FK_CUR_EST FOREIGN KEY (nifEstudiante) REFERENCES ESTUDIANTE(nif)
);
```

Y un ejemplo de las tablas con datos sería:

- ESTUDIANTE

nif	codigo	nombre	fMatricula	direccion
12345678A	1	Pedro Casas	1/9/24	Avenida de la libertad, 23
48123456B	2	Mireia Vidal	1/9/24	Porta de la Morera, 6

- CURSAR

nifEstudiante*	asignatura	anyo	repetidor
12345678A	1	2024	true
48123456B	1	2024	false
12345678A	2	2023	false

¿Cómo repercute el sistema si eliminamos el estudiante 12345678A de la tabla ESTUDIANTE? ¿O le cambiamos el nif (algo inusual, pero que puede pasar si nos hemos equivocado al teclearlo)? ¿Qué opciones tenemos para respetar la integridad referencial?

Las posibilidades son:

- `NO ACTION`: se impide la operación, rechazando el borrado o la actualización. Este es el comportamiento por defecto si no indicamos nada.
- `CASCADE`: la operación se propaga de la tabla origen a la destino. Esto es, si borramos el estudiante, borrará los registros de CURSAR. Si modificamos su nif, también modificará los nifEstudiante de CURSAR.
- `SET NULL`: la clave ajena se pone a NULL, de manera que se pierde la relación existente, provocando probablemente inconsistencia de datos.

Si volvemos a definir la tabla CURSAR indicando las restricciones de propagación, podríamos hacer:

```
CREATE TABLE CURSAR (
  nifEstudiante varchar(9),
  asignatura int,
  anyo int,
```

```
  repetidor boolean,
  CONSTRAINT PK_CURSAR PRIMARY KEY(nifEstudiante, asignatura, anyo),
  CONSTRAINT FK_CUR_EST FOREIGN KEY (nifEstudiante) REFERENCES ESTUDIANTE(nif)
    ON DELETE CASCADE
    ON UPDATE NO ACTION
);
```

Autoevaluación

¿Por qué en este caso concreto no podemos indicar la acción SET NULL?

Validaciones

Para poder incluir restricciones de validación y que los datos introducidos cumplan ciertas condiciones, emplearemos el atributo CHECK, tanto tras la definición de un atributo como una restricción con CONSTRAINT:

```
CREATE TABLE INGRESO1 (
  cod DECIMAL(5) PRIMARY KEY,
  concepto VARCHAR(40) NOT NULL,
  importe DECIMAL(11,2) CHECK (importe>0 AND importe<8000));

CREATE TABLE INGRESO2 (
  cod DECIMAL(5) PRIMARY KEY,
  concepto VARCHAR(40) NOT NULL,
  importe DECIMAL(11,2),
  CONSTRAINT CK_INGRESO CHECK (importe>0 AND importe<8000));
```

También podemos crear validaciones que comparen el valor de dos atributos de la tabla:

```
CREATE TABLE INGRESO3(
  cod DECIMAL(5),
  concepto VARCHAR(40) NOT NULL,
  importeMax DECIMAL(11,2),
  importe DECIMAL(11,2),
  CONSTRAINT PK_INGRESO (cod),
  CONSTRAINT CK_INGRESO CHECK (importe<importeMax));
```

Gestionando

Además de indicar las restricciones mientras creamos una tabla, podemos hacerlo mediante sentencia ALTER TABLE.

Así pues, para añadir una nueva restricción usaremos la opción ADD:

```
ALTER TABLE tabla ADD [CONSTRAINT nombre] tipoDeRestricción (columnas);
```

Para eliminar, la opción DROP:

```
ALTER TABLE tabla DROP {PRIMARY KEY | CONSTRAINT nombreRestricción} [CASCADE]
```

Respecto a las claves, si queremos deshabilitarlas para hacer algún tipo de modificación, haremos uso de las opciones ENABLE o DISABLE según convenga:

```
ALTER TABLE tabla ENABLE KEYS;

ALTER TABLE tabla DISABLE KEYS;
```

Otros elementos

Existen otras estructuras que forman parte de DDL pero que veremos más adelante son:

- **Vistas**: Permiten crear una representación externa (tabla virtual) a partir de una consulta, mediante CREATE VIEW. Las estudiaremos en la unidad 8.
- **Índices**: Se utilizan para acelerar las consultas, mediante CREATE INDEX. Los estudiaremos en la unidad 9.
- **Disparadores** (*triggers*): Se asocian a operaciones que se producen en el sistema (evento) para realizar otra acción. Los estudiaremos en la unidad 11.

Cargando datos

A la hora de cargar datos en una base de datos, además de las operaciones de inserción individual o múltiple de registros, todos los SGBD incorporan herramientas para la importación tanto de datos como recuperación de copias de seguridad que nos sirven para tener un entorno listo para trabajar.

Ejecutando un script

Si ya tenemos un *script*, por ejemplo, con las sentencias CREATE TABLE definidas o diferentes instrucciones que insertan o modifican datos, mediante la línea de comandos podemos ejecutarlas a través de la entrada estándar (**<**) o haciendo uso de parámetros:

```
mariadb -u usuario -p nombre_de_base_de_datos < archivo.sql

psql -U usuario -d nombre_de_base_de_datos < archivo.sql

psql -U usuario -d nombre_de_base_de_datos -f archivo.sql
```

Si en cambio lo queremos hacer dentro de cada cliente (una vez nos hemos autenticado con el usuario y contraseña):

En *MariaDB* usaremos el comando source:

```
source archivo.sql
```

Cargando datos desde archivos

Si en vez de un *script* tenemos los datos a cargar y ya tenemos nuestra estructura de datos creada (y probablemente con datos ya existentes), tenemos otras posibilidades:

En *MariaDB* usaremos la herramienta mariadb-import que permite cargar datos en formato texto, csv, etc...:

```
mariadb-import --local -u usuario -p nombre_de_base_de_datos /ruta/al/archivo.csv
```

Otra forma es hacer uso de datos almacenados en ficheros de texto y cargarlos directamente en una tabla es utilizar el comando LOAD DATA INFILE:

```
LOAD DATA INFILE '/ruta/al/archivo.csv'
INTO TABLE empleados
FIELDS TERMINATED BY ',' -- Separador de campos (coma)
LINES TERMINATED BY '\n' -- Fin de línea
IGNORE 1 LINES -- Ignora la primera línea (encabezados)
(nombre, puesto, salario); -- Especifica las columnas
```

Esta opción es mucho más eficiente que ejecutar múltiples sentencias INSERT.

En cambio, para exportar los datos, podemos crear un volcado de la base de datos mediante mariadb-dump:

```
mariadb-dump -u usuario -p nombre_de_base_de_datos > dump.sql
mariadb-dump -u usuario -p --all-databases > dump.sql
```

Usaremos la herramienta mariabackup, la cual es una herramienta de *MariaDB* para crear copias de seguridad físicas (binarias) de las bases de datos.

Para restaurar una copia de seguridad haremos:

```
mariabackup --copy-back --target-dir=/ruta/a/la/copia_de_seguridad
```

Y para realizar la copia de seguridad:

```
mariabackup --prepare --target-dir=/ruta/a/la/copia_de_seguridad
```

DML

Una vez que ya sabemos como crear las estructuras de datos para guardar la información, vamos a aprender a gestionarla.

CRUD

Las operaciones DML se asocian al acrónimo CRUD:

- **C**reate: insertar
- **R**ead: consultar
- **U**pdate: modificar
- **D**elete: eliminar

Sobre una tabla, las operaciones que vamos a poder realizar son:

- insertar datos en una tabla
- modificar datos de una tabla
- eliminar datos de una tabla
- consultar datos de una o más tablas.

Cuando hablamos de "datos de una tabla", podemos referirnos a registros completos, campos concretos o incluso elementos relacionados de varias tablas.

Para realizar estas operaciones, se utiliza el subconjunto DML (*Data Manipulation Language*) de SQL. En esta unidad nos vamos a centrar en la inserción, modificación y eliminación de datos, dejando las consultas para trabajarlas en profundidad en las siguientes unidades.

Para los siguientes ejemplos vamos a trabajar con la siguiente tabla:

```
CREATE TABLE dml8a (

  id INT PRIMARY KEY,

  nombre VARCHAR(64) NOT NULL DEFAULT "Chuck Norris",

  email VARCHAR(64));
```

Inserciones

La sentencia para insertar datos es INSERT INTO en MariaDB :

```
INSERT INTO tabla VALUES (valor1, valor2, ...);
```

Veamos unos ejemplos. Podemos insertar en la tabla dml8a utilizando:

```
INSERT INTO dml8a VALUES (1, "Aitor Medrano", "a.medrano@edu.gva.es");

INSERT INTO dml8a(id, nombre) VALUES (2, "Pedro Casas");
```

Si no ponemos los nombres de los campos detrás del nombre de la tabla, en los valores deberemos introducir todos los valores siguiendo el orden de los campos según fueron creados (siempre puedes consultar la estructura de una tabla mediante DESCRIBE nombre_tabla). Además, los tipos de los datos deben concordar, incluso en el tamaño o precisión, debiendo haber una correspondencia posicional uno a uno entre las columnas y los valores introducidos.

Si queremos insertar varios registros con una única instrucción, los indicaremos separados por comas:

```
INSERT INTO dml8a VALUES (3, "María Sánchez", "maria@gmail.com"),
  (4, "Eva Amaral", "eva@gmail.com");
```

Si id fuera un campo auto incrementable podríamos omitir su valor, y automáticamente le asignaría el siguiente valor de la secuencia:

```
INSERT INTO dml8a(nombre) VALUES ("José Manuel Pérez");
```

Si queremos insertar los valores por defectos asociados a una columna o un valor nulo (por ejemplo, a una clave ajena) usaremos DEFAULT y NULL respectivamente. Por ejemplo:

```
INSERT INTO dml8a VALUES (66, DEFAULT, NULL);
```

Gestión de errores

Si insertamos datos con tipos diferentes, dejamos campos no nulos en blanco, o insertamos datos en un campo que es clave ajena y no cumplimos la integridad referencial, el SBGD nos devolverá diferentes mensajes:

FIXME: completar - continuar

También podemos insertar datos con el resultado de una consulta mediante la instrucción INSERT SELECT, pero esta operación la estudiaremos en profundidad más adelante.

Modificaciones

La sentencia para actualizar datos es `UPDATE ... SET` en MariaDB:

```
UPDATE tabla SET campo=valor1 [,campo2=valor2];
```

Si queremos decidir qué elementos modificar, añadimos la expresión WHERE, ya que si no, actualizaremos todos los registros de la tabla:

```
UPDATE tabla SET campo=valor WHERE condicion;
```

Veamos unos ejemplos. Podemos modificar el email de un usuario concreto:

```
UPDATE dml8a SET email="pedro@gmail.com" WHERE id=2
```

O modificar varios campos de una sola vez:

```
UPDATE dml8a SET nombre="Pedro Casas García",
      email="pedro.casas@gmail.com"
  WHERE id=2
```

También podemos realizar cálculos al modificar un campo:

```
-- Incrementamos la población de Elche en un 10%

UPDATE CIUDAD SET poblacion = poblacion * 1.10 WHERE nombre = "Elche";
```

El filtado de los datos mediante la instrucción WHERE ofrece múltiples posibilidades, tanto con operadores relacionales, aritméticos, funciones de tratamiento de textos, fechas, etc... como comparación con el resultado de una consulta u operaciones de conjuntos. A partir de la próxima unidad profundizaremos en todas la alternativas del atributo WHERE.

Borrados

No te olvides...

Si no conoces este meme, hazte el favor y mira el siguiente video.

Hice un delete...

La sentencia para actualizar datos es `DELETE ... FROM` , en MariaDB:

```
DELETE campo FROM tabla;
```

Si queremos decidir qué elementos eliminar, añadimos la expresión WHERE, ya que si no, eliminaremos todos los registros de la tabla, lo cual es muy peligroso:

```
DELETE campo FROM tabla WHERE condicion;
```

Integridad referencial

Recuerda que si eliminamos un registro referenciado por una clave ajena, se ejecutará la regla descrita en la tabla de origen, de manera, que es posible que no podamos eliminar ciertos datos al estar referenciado desde otra tabla (si estaban configuradas con NO ACTION o RESTRICT).

En cambio, si la clave ajena se configuró con CASCADE, al borrar el registro, se borrarán todos los registros de las tabla origen que referenciaban a éste.

Finalmente, si se configuró con NULL, la tabla origen modificará el valor de su clave ajena se pondrá a nulo.

Conviene destacar que si queremos eliminar todos los datos de una tabla, es mucho más eficiente utilizar el comando TRUNCATE TABLE, tanto en MariaDB

TRUNCATE TABLE dml8s

Por debajo, cuando truncamos una tabla, realmente está eliminando la tabla y volviéndola a crear, eliminando todos los ficheros de datos y reinicinado los campos autonuméricos.

Reto II - Creamos

En el reto anterior diseñamos una base de datos. En este reto nos vamos a centrar en la creación del modelo físico, creando y cargando datos, para posteriormente definir una serie de informes y KPIs que generaremos mediante consultas utilizando SQL.

KPI

Un KPI (*key performance indicator*) es una métrica cuantitativa que muestra un valor importante para el negocio, permitiendo comparar el desempeño de alguna característica.

Ejemplos de KPIs podrian ser la cantidad de pedido realizados durante el último més, beneficios obtenido en el último año, comparación *Year-over-year (YoY)* de las ventas de este mes respecto a las ventas del mismo mes pero en el año pasado, etc...

Así pues, inicialmente crearemos la estructura de datos necesaria, la cargaremos con datos ficticios, y deberemos pensar qué información queremos extraer de ella.

Referencias

- Sintaxis SQL oficial de MariaDB.
- Materiales sobre el módulo de BD:

- *DDL _y DML - Institut Obert de Catalunya*
- Creación de bases de datos en MySQL de *José Juan Sánchez*
- Introducción a SQL, DDL y DML de *Jorge Sánchez*
- Tratamiento de datos de *Javier Gutiérrez*
- Diseño físico de bases de datos - DDL y Modificación de bases de datos - DML de gestionbasesdatos.readthedocs.io
- Introducción a SQL, por *Luis Valencia* y *David Orellana*, de la *Universidad de Sevilla*.

Actividades

- **AC301**. (RABD.2 // CE2a // 3p) En *MariaDB*, descarga la base de datos world.sql.zip, y tras descomprimirla y cárgala mediante *PhpMyAdmin*. A continuación, averigua:
  - ¿Cuántas tablas tiene?
  - ¿Cuántos y de qué tipos son los campos de la tabla city?
  - ¿Cuál es el formato de almacenamiento de la base de datos? ¿Qué motor de almacenamiento emplea?
- **AP302**. (RABD.2 // CE2a // 3p) En *PostgreSQL*, mediante SQL, crea una base de datos con un *tablespace* definido previamente y el juego de caracteres que permita el uso de emojis y que no sea *case sensitive*.

En dicha tabla, crea una tabla cuya clave sea un valor autonumérico y tenga un atributo que permita introducir cadenas de texto.

A continuación, mediante *PgAdmin*, obtén la información tanto de la base de datos como de la tabla, e inserta una frase que contenga algún emoji. Comprueba que la información se ha almacenado correctamente.

Debes entregar en el aula_virtual:

- Comandos SQL empleados.
- Capturas de pantalla de *PgAdmin* con la información solicitada.
- **AC303**. (RABD.2 // CE2b, CE2c, CE2d // 3p) Crea el siguiente modelo relacional en *MariaDB* y en *PostgreSQL*, y adjunta tanto las instrucciones DDL como capturas de pantalla con la estructura de las tablas:
- Tabla FABRICANTE
  - Campos:
    - codFabricante: entero autoincrementable
    - nombre: cadena 32
    - pais: cadena 32
  - Restricciones:
    - La clave primaria es codFabricante
    - El nombre no puede ser nulo
- Tabla ARTICULO
  - Campos:
    - codigo: cadena 32

        - codFabricante: entero
        - peso: numérico con dos decimales
        - categoria: cadena 16
        - precioVenta: numérico con dos decimales
        - precioCompra: numérico con dos decimales
        - existencias: entero sin signo
    - Restricciones:
        - La clave primaria es: codigo, codFabricante, peso, categoría.
        - La categoria ha de ser primera, segunda o tercera.

1. Modifica el atributo pais de la tabla FABRICANTE para que el valor por defecto sea España.
2. Añade una columna antes de pais en la tabla FABRICANTE para guardar la provincia.
3. Renombra la tabla ARTICULO a PRODUCTO.

- **AC304**. (RABD.2 // CE2e // 3p) A partir del ejercicio AC303, vuelve a indicar la instrucción CREATE TABLE de la tabla PRODUCTO de manera que:
    - codFabricante sea una clave ajena que apunta a FABRICANTE.
    - al eliminar un fabricante concreto, si hay artículos de dicho fabricante, se deben prohibir la operación. En cambio, si modificamos el código de la tabla FABRICANTE, la clave ajena también cambiará su valor.
    - los atributos precioVenta y precioCompra deben de ser mayores de 0.

      A continuación, mediante operaciones ALTER TABLE añade las siguientes restricciones:
    - La columna PRODUCTO.codFabricante no puede admitir valores nulos.
    - El peso debe ser superior o igual a 1.00.

- **AR505**. (RABD.2 // CE2e // 3p) Mediante *MariaDB* o *PostgreSQL*, crea las tablas necesarias para representar que un empleado tiene diferentes números de teléfono que queremos almacenar, teniendo en cuenta que:

- Los teléfonos se presentan mediante un prefijo de tres caracteres y número de 9 dígitos.

- Si eliminamos a un empleado, también eliminaremos todos sus teléfonos.

Para ello, haciendo uso de la herramienta gráfica apropiada, inserta datos (varios empleados y varios teléfonos para cada empleado) y posteriormente prueba a eliminar un empleado. Adjunta tanto los scripts DDL como capturas de las tablas antes y después de realizar la operación de borrado.

- **AC306**. (RABD.2 // CE2b, CE2c, CE2d // 3p) Crea el siguiente modelo relacional en *MariaDB* y en *PostgreSQL*:

  DEPARTAMENTO (codD, nombre, direcc) · PK: (codD) · VNN: (nombre)

  EMPLEADO (dni, nombrec, salario, direcc, departamento) · PK: (dni) · UK: (nombrec) · VNN: (nombrec) · VNN: (salario) · VNN: (departamento) · FK: (departamento) → DEPARTAMENTO

  Además, el salario de cada empleado debe ser igual o superior a 900, y su valor por defecto es de 1000.

A continuación, mediante haciendo uso de ALTER TABLE:

- Elimina la clave ajena de la tabla EMPLEADO.
- Vuelve a añadir la clave ajena, de manera que al modificar un departamento, el cambio se propague en cascada, pero que al eliminarlo, no realice ningún acción.

Finalmente, elimina las tablas creadas.

- **AP307**. (RABD.2 // CE2b, CE2c, CE2d, CE2e // 3p) A partir de la actividad AC407 (que a su vez se basa en la AC203), crea las instrucciones DDL necesarias para crear el modelo físico.
- **AP308**. (RABD.4 // CE4a // 3p) Haciendo uso de *MariaDB* y del cliente mariabd, a partir del siguiente *script* con las instrucciones para crear una base de datos, carga los datos que tienes disponibles en AP308.zip haciendo uso de las instrucciones LOAD DATA. Finalmente, mediante mysqldump realiza una exportación de toda la base de datos (con los datos cargados), y adjunta tanto los comandos empleados como el resultado de la exportación.
- **AC309**. (RABD.4 // CE4a, CE4b // 3p) A partir de la actividad AC304, inserta los siguientes datos utilizando las herramientas gráficas, tanto en *MariaDB* como en *PostgreSQL*:
  - FABRICANTE

codFabricante	nombre	provincia	pais
1	Sony	Kantō	Japón
2	Microsoft	Nuevo México	USA
3	Nintendo	Kantō	Japón

  - PRODUCTO

codigo	codFabricante*	peso	categoria	precioVenta	precioCompra	existencias
PS5	1	4.5	primera	500.00	550.00	123
PS4	1	2.8	primera	300.00	400.00	234
XSX	2	4.4	primera	500.00	550.00	345
NSW	3	0.3	segunda	225.00	330.00	456

  - A continuación, indica las sentencias SQL para vaciar las tablas como para introducir los datos.
- **AC310**. (RABD.4 // CE4b // 3p) Carga el siguiente script de datos sobre las tablas de la AC306, y realiza las siguentes modificaciones:
- Añade un nuevo departamento llamado Formación.
- Renombra el departamento IT por Informática.
- Añade el campo email al final de la tabla DEPARTAMENTO, y rellenalos todos con email@AC310.com.
- Incrementa el salario de todos los empleados del departamento de informática en 300€.

- Inserta un nuevo empleado con tus datos en el departamento Formación.
- Cambia al empleado 12345678A al departamento de Formación.
- Modifica la clave del departamento Formación a 666.
- Elimina el departamento Legal. ¿Has podido? ¿Qué ha pasado con sus empleados?
- **AP311**. (RABD.4 // CE4b // 3p) Sobre la actividad AC310, añade una nueva tabla DIETAS donde podamos anotar los gastos ocasionados por las dietas de los empleados, como son la fecha, el coste y una descripción de la misma. A continuación, inserta 10 dietas de diferentes empleados. Debes tener en cuenta que si eliminamos un empleado, no debe permitir realizar la operación ni eliminar las dietas, ya que si no, luego no nos cuadraría la contabilidad. ¿Cómo solucionamos dar de baja a un empleado pero no eliminarlo del sistema?
- **AR512**. (RABD.4 // CE4b // 3p) Carga el *script* ar512.sql con el DDL sobre una base de datos de libros y préstmos, y realiza las siguientes operaciones:
- Inserta los siguientes valores en las tablas:
  - LIBRO

codigo	titulo	autor	stock	genero
1	Historia de España	J. Pérez	5	HIS
2	Reina Roja	J. Gómez-Jurado	33	NOV

  - PRESTAMO

codLibro	codSocio	fentrega	fdevolucion
1	1	15/10/24	
2	1	25/10/24	

- Realiza las siguientes operaciones:
  1. Cambia la clave primaria de libro al título. Realiza los cambios necesarios para que la tabla libros y prestamos estén relacionadas.
  2. Eliminar aquellos libros que sean del genero de historia. ¿Se elimina el registro de la tabla LIBRO? Razona la respuesta.
     1. Si no lo has conseguido, cambia la restricción de la clave ajena para que la operación se propague.
     2. Vuelve a realizar la operación de borrado de libros.
     3. ¿Cuantos registros tiene ahora la tabla PRESTAMO?
  3. Inserta los siguientes libros (*Loba Negra*, *MariaDB Essentials*), buscando en Internet la información que necesites.
  4. Reduce el stock de Reina Roja en 3 unidades.
  5. Haz que el libro prestado tenga como fecha de devolución el 31/12/24.
  6. Cambia el nombre del autor de Reina Roja por Juan Gómez Jurado.

- **AC313**. (RABD.2 // CE2a, CE2b, CE2c, CE2d, CE2e // 3p) A partir del siguiente modelo relacional, genera el script DDL (AC313.sql) para la creación de las tablas mediante *MariaDB* en una base de datos denominada AC313 (el *script* debe poder ejecutarse una y otra vez, de manera que si ya existe la base de datos debe borrarla, y para crear cada tabla, debe comprobar que no exista previamente).

El diccionario de datos es el siguiente:

TABLA	CAMPO	TIPO	RESTRICCIONES
ALUMNOS	id	Entero sin signo autoincrementable	PK
	dni	Carácter (9)	UK
	nombre	Texto (64)	obligatorio
	apellidos	Texto (64)	obligatorio
	sexo	{H, M}	obligatorio
	dirección	Texto (128)	
	telefono	Carácter (12)	
	fnac	Fecha	
ASIGNATURAS	id	Entero sin signo autoincrementable	PK
	nombre	Texto (64)	obligatorio
	aula	entero	Entre 1 y 10, por defecto se asigna el aula 1
	duracion	Entero	Entre 1 y 12, por defecto su duración es 3
NOTAS	idAlumno	Entero sin signo	PK, FK → ALUMNOS
	idAsignatura	Entero sin signo	PK, FK → ASIGNATURAS
	fecha	Fecha	obligatorio
	calificación	Real	Entre 0.0 y 10.0, obligatorio

A continuación, modifica la estructura de la tablas mediante ALTER TABLE:

- Cambia el nombre de las tablas y ponlas en singular.
- Añade una columna recuperacion a NOTA para almacenar la posible calificación de un examen de recuperación. Dicha calificación debe estar comprendida entre 0 y 10.
- Modifica la columna sexo de ALUMNO para añadir un tercer valor (indeterminado - I).
- Modifica la clave primaria de NOTA para incluir también la fecha.

- Modifica la clave ajena de NOTA hacia ALUMNO para que al realizar un borrado de un alumno, se eliminen todas sus notas.

- **AC314**. (RABD.4 // CE4b // 3p) Continua la actividad anterior, y crea el script AC314.sql donde insertes los siguientes datos para el alumnado y los cursos mediante instrucciones INSERT:

- ALUMNO

DNI	Nombre	Apellidos	Sexo	Dirección	Teléfono	Fecha de nacimiento
12345678A	Johnny	Mentero	H	Debajo del puente, -1		25/12/80
23456789B	María	Gracia	M		636112233	
34567890C	Armando	Casas	I	Villarriba, 33	636223344	31/12/90
45678901D	Mario	Neta	H	Circo del Sol, 22	636334455	
56789012E	Susana	Oria	M	Calle La Granja		3/1/00

- ASIGNATURA

id	Nombre	Aula	Duración
	SQL	3	9
666	Humor		12
	Cocina	6	

A continuación, inserta los siguientes datos de calificaciones mediante LOAD DATA, creando previamente el archivo AC314.csv con la siguiente información:

Alumno	Asignatura	Fecha	Nota	Recuperación
1	1	20/12/20	7,25	
1	666	20/12/20	9,99	
2	2	10/1/21	5	
2	666	10/1/21	7	
3	1	10/1/21	3	6
3	666	10/1/21	3	6

Finalmente, realiza las siguientes operaciones de modificación y borrado de datos:

1. Modificar:
    1. El teléfono del alumno 1 es 666111222
    2. La dirección de todos los alumnos hombres es Donde me lleve la vida.
    3. El apellido de todos alumnos que se llamen Armando ahora es Bullas.

    4. Todos los cursos tendrán una unidad más de duración
    5. Los alumnos que no tengan dirección se les asignará Internet.
2. Eliminar:
    1. Borra las calificaciones inferiores a 4. ¿has podido? ¿cuantas calificaciones quedan?
    2. Borra la información de Johnny Mentero. ¿has podido? ¿cuantos calificaciones quedan?
    3. Elimina el curso de cocina ¿has podido? ¿cuantas asignaturas quedan?
    4. Elimina los alumnos que no tengan teléfono ¿has podido? ¿cuantos alumnos quedan? ¿cuantas calificaciones quedan?
    5. Elimina todos los alumnos ¿has podido? ¿cuantos alumnos quedan? ¿cuantas calificaciones quedan?

Recuerda que para comprobar los datos que contiene una tabla puedes ejecutar en el cliente SELECT * FROM tabla;

- **pr315**. (RABD.2, RABD.4 // CE2a, CE2b, CE2c, CE2d, CE2e, CE4a, CE4b // 10p) A partir del modelo diseñado en la actividad PY414 asociada al primer reto de diseño de la base de datos, ahora nos vamos a centrar en crear el modelo físico y cargar datos para trabajar con él. Para ello, como equipo, deberéis decantaros por un SGBD en concreto.

  Para ello, se pide:

    - Informe con el modelo ER, el modelo MR y justificacion de las decisiones tomadas respecto a las restricciones incluidas en el modelo físico.
    - Script SQL con las sentencias DDL.
    - Script SQL con las sentencias DML para cargar datos.
    - Script SQL para vaciar todas las tablas.

  Se utilizará una rúbrica para su evaluación en base a la siguiente lista de cotejo:

    - Limpieza y calidad de los scripts.
    - Documentación de los scripts.
    - El informe entregado no contiene faltas de ortografía.
    - El informe entregado tiene un formato adecuado (portada, apartados, autores, etc...).
    - Cada tabla contiene un mínimo de 10 registros, salvo justificación.
    - En el caso de existir relaciones entre tablas, cada registro debe relacionarse con un mínimo de tres registros más.

- **AR516**. (RABD.6 // CE6b, CE6c, CE6d, CE6e // 3p) Una vez finalizada la unidad, responde todas las preguntas del cuestionario inicial, con al menos un par de líneas para cada una de las cuestiones.