Diseño físico – DDL de SQL

¿Qué es SQL?

SQL (Structured Query Language) es un lenguaje de programación que se utiliza para comunicarse con las bases de datos relacionales y poder manipular tanto la estructura de almacenamiento como los datos almacenados en ella. En resumen SQL es el lenguaje que permite a los usuarios crear, modificar y consultar bases de datos.

¿Qué es DDL?

En el lenguaje SQL podemos agrupar las consultas en 3 grupos según su funcionalidad y ámbito de uso:

- DDL (Lenguaje de Definición de Datos): Se utiliza para definir (crear y modificar) la estructura de las bases de datos.
- DML (Lenguaje de Manipulación de Datos): Con este tipo de consultas, podremos manipular los datos en sí (consultar datos, añadir, modificar o eliminar dichos datos), pero no la estructura de almacenamiento como ocurría en el caso anterior.
- DCL (Lenguaje de Control de Datos): Emplearemos este subgrupo de consultas para controlar los privilegios de acceso y manipulación en la base de datos.

Sentencias del DDL en SQL

Crear una base de datos

En primer lugar, vamos a conocer la sentencia necesaria para crear una base de datos:

CREATE DATABASE nombre_de_bbdd;

donde sustituiremos *nombre_de_bbdd* por el nombre que deseamos asignar a nuestra base de datos.

Eliminar una base de datos

En cambio, si deseamos crear una base de datos existente (y por tanto toda la información que almacena), emplearemos la sentencia:

DROP DATABASE nombre_de_bbdd;

donde *nombre_de_bbdd* será el nombre de la base de datos a eliminar.

Crear una tabla

En este caso la sentencia a emplear tendrá el siguiente esquema:

```
CREATE TABLE nombre_de_tabla (nombre_campo1 tipo1,
                              nombre_campo2 tipo2,
                              … ,
                              nombre_campoN tipoN,
                              CONSTAINT nombre_restricción TIPO);
```

donde *nombre_de_tabla* representa el nombre de la tabla a crear, seguido de un listado con los nombres de los campos a crear y el tipo de cada uno de ello, finalmente se añaden las restricciones necesarias, como son claves primarias, claves alternativas, claves ajenas o cualquier otro tipo de restricción. A continuación se añade un ejemplo para facilitar aún más su comprensión:

```
CREATE TABLE pilotos (
        id INT2,
        nombre VARCHAR(50) NOT NULL,
        apodo VARCHAR(20),
        fecha_nacimiento DATE,
        nacionalidad VARCHAR(30),
        CONSTRAINT pk_pilotos PRIMARY KEY(id),
        CONSTRAINT fk_piloto_nacionalidad FOREIGN KEY (nacionalidad)
                REFERENCES nacionalidades);
```

Esta sentencia nos creará una tabla llamada *pilotos* con 5 campos (id, nombre, apodo, fecha_nacimiento, nacionalidad). De estos 5 campos, *id* será la clave primaria , el campo *nombre* será valor no nulo (es decir, debe contener un valor) y el campo *nacionalidad* es una clave ajena a una tabla llamada nacionalidades (cabe mencionar que si sólo indicamos la tabla referenciada, como hemos hecho en el ejemplo, la clave ajena apuntará a la clave primaria de dicha tabla, si se desea apuntar a otro campo o campos, estos deberán ser clave alternativa y se indicará dicho campo o campos entre paréntesis tras el nombre de la tabla, por ejemplo: *REFERENCES nacionalidades (pais)*).

Eliminar una tabla

En este caso, para eliminar una tabla, la instrucción a emplear es muy sencilla:

```
DROP TABLE nombre_de_tabla;
```

Modificar una tabla

En este caso, la consulta puede ser más o menos compleja en función de qué elemento deseamos modificar, ya que SQL permite modificar desde el nombre de una

tabla, hasta el tipo de un campo, eliminar o añadir un campo, o una restricción; vamos a ver alguno de los casos más relevantes:

Modificar el nombre de una tabla:

ALTER TABLE nombre_actual_tabla **RENAME TO** nuevo_nombre_tabla;
Así si por ejemplo queremos modificar el nombre de la tabla *pilotos* para llamarla *drivers,* la sentencia quedaría del siguiente modo:

ALTER TABLE pilotos
RENAME TO drivers;

Modificar el nombre de una campo:

La sentencia es parecida a la mostrada en el apartado anterior, pero indicando la columna a renombrar:

ALTER TABLE tabla_donde_modificar
RENAME COLUMN columna_a_renombrar **TO** nuevo_nombre_de_columna;

Por tanto, si se deseara modificar el nombre de la columna nacionalidad, para que se llamara pais_nacimiento, ejecutaríamos la consulta:

ALTER TABLE pilotos
RENAME COLUMN nacionalidad **TO** pais_nacimiento;

Añadir un nuevo campo:

En este caso, indicaríamos tanto el nombre de la tabla a modificar como el nombre y el tipo del campo o campos a añadir:

ALTER TABLE tabla_donde_modificar
ADD COLUMN nombre_nuevo_campo **TIPO**;

Así, si deseáramos añadir un campo de tipo fecha, llamado fecha_defunción en la tabla pilotos, lo haríamos mediante la instrucción:

ALTER TABLE pilotos
ADD COLUMN fecha_defunción **DATE**;

Modificar tipo de dato de un campo:

En este caso, la sentencia seguiría un patrón del tipo:

ALTER TABLE tabla_donde_modificar
MODIFY COLUMN nombre_campo TYPE **NUEVO_TIPO**;

Siguiendo con el ejemplo anterior, imaginemos que deseamos almacenar la fecha y hora de defunción, entonces la sentencia a emplear sería:

ALTER TABLE pilotos
ALTER COLUMN fecha_defunción **TYPE TIMESTAMP**;

Eliminar un campo:

Para eliminar un campo, la sentencia es algo más sencilla:

ALTER TABLE tabla_donde_modificar
DROP COLUMN nombre_campo_a_eliminar;

Por lo tanto, para eliminar el campo fecha_defunción, ejecutaríamos:

ALTER TABLE pilotos
DROP COLUMN fecha_defunción**;**

Añadir una nueva restricción:

En el caso de querer añadir una nueva restricción, dependerá de la restricción a añadir, así si por ejemplo deseamos añadir una clave alternativa, la sintaxis de la sentencia sería:

ALTER TABLE tabla_donde_modificar
ADD CONSTRAINT nombre_restriccion **UNIQUE** (campo_clave_alternativa);

Si a la tabla pilotos deseamos convertir el campo *nombre* en clave alternativa, lo haríamos del siguiente modo:

ALTER TABLE pilotos
ADD CONSTRAINT UK_nombre_piloto **UNIQUE (**nombre**);**

Otras modificaciones:

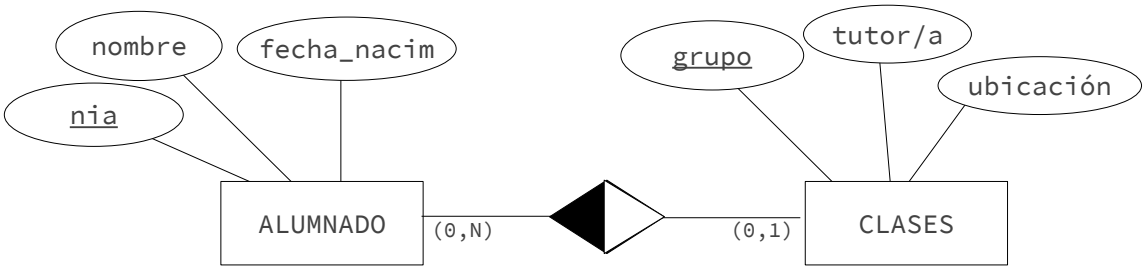
Como se puede observar, las posibles modificaciones son tantas y tan variadas, para evitar extendernos demasiado en el siguiente documento, se añade enlace a la documentación oficial de POSTGRESQL.

Restricciones en las claves ajenas

Las bases de datos tratan de preservar la integridad referencial y de este modo evitar incoherencia de datos. Pero ¿qué es esta integridad referencial y como la protegen los SGBD?

Como sabemos, en una base de datos, las tablas suelen estar relacionadas una con otras mediante las claves ajenas; cuando hablamos de integridad referencial, estamos hablando de estas claves ajenas y los datos implicados; y de la coherencia que debe haber entre el dato de una clave ajena y el dato al que hace referencia. Para proteger la integridad referencial, deberemos controlar ciertas acciones realizadas sobre las claves ajenas. Entenderemos esto mejor mediante un ejemplo:

Imaginemos que tenemos 2 tablas: ALUMNADO y CLASES, donde todo alumno/a es miembro de una clase:



Esto traducido a tablas y con datos de ejemplo, lo veríamos del siguiente modo:

ALUMNADO			
nia	nombre	fecha_nacim	grupo
111111	María Lloret	01/01/2000	1DAW
222222	José Sellés	03/05/2000	2DAW
333333	Patricia Orts	07/07/2001	1DAW
444444	Empar Llinares	11/11/2000	1DAW
555555	Roser Bella	01/05/2003	1DAW
666666	Miguel López	07/06/2002	2DAW

CLASES		
grupo	tutor	ubicación
1DAW	Kiko R.	1-001
2DAW	Francisco O.	1-002

Pero ¿qué sucedería si ahora eliminamos de la tabla CLASES el registro de 1DAW? Si permitimos dicha eliminación, tendríamos 4 alumnos/as matriculados en un grupo que no existe. Una posible solución es evitar que se eliminen datos referenciados.

Y si en lugar de eliminar un dato referenciado, lo modificamos ¿qué sucedería? Si cambiamos en la tabla CLASES el nombre del grupo, y éste pasara a llamarse 1rDAW en lugar de 1DAW, tendríamos el mismo problema. Sin embargo, en esta situación, tal vez en lugar de prohibir los cambios, una mejor estrategia sería propagar el cambio, es decir cambiar en cascada el dato, en todos los registros afectados, de este modo el cambio se produciría de manera automática y los 4 alumnos/as estarían matriculados en el grupo 1rDAW de manera automática.

Y ¿cómo podemos controlar y/o restringir dichos cambios en los campos referenciados? Esto lo haremos añadiendo unas clausula en el momento de definir la clave ajena de cada tabla.

Siguiendo con el ejemplo del alumnado y los grupos, si deseamos que un grupo no pueda ser eliminado, pero que los cambios se produzcan en cascada, crearíamos la tabla con la siguiente sentencia:

```
CREATE TABLE alumnado(
        nia                     VARCHAR(10),
        nombre          VARCHAR(50),
        fecha_nac       DATE,
        grupo           VARCHAR(10),
        CONSTRAINT PK_alumnado PRIMARY KEY (nia),
        CONSTRAINT FK_alumnado_clases FOREIGN KEY (grupo) REFERENCES clases
                ON DELETE RESTRICT
                ON UPDATE CASCADE);
```

Entre las diferentes opciones que podemos indicar junto al ON DELETE y/o junto al ON CASCADE encontramos:

- ✔ **CASCADE**: Si se borra o actualiza una fila de la tabla principal, se borran o actualizan automáticamente las filas correspondientes en la tabla secundaria. Por ejemplo, si tienes una tabla de "Pedidos" y una tabla de "Productos", y un producto se elimina, entonces todas las entradas de ese producto en la tabla de "Pedidos" también se eliminarán.
- ✔ **SET NULL**: Si se borra o actualiza una fila de la tabla principal, se establecen en NULL los valores de la clave ajena en las filas de la tabla secundaria. Esto es útil cuando quieres mantener el registro en la tabla secundaria, pero indicar que ya no está asociado con un registro en la tabla principal.
- ✔ **RESTRICT**: Esta es la opción predeterminada. Impide que se borre o actualice una fila de la tabla principal si tiene filas relacionadas en la tabla secundaria. Es decir, no puedes borrar un registro de la tabla principal si hay registros relacionados en la tabla secundaria.
- ✔ **NO ACTION**: Similar a RESTRICT, no realiza ninguna acción si hay una violación de la integridad referencial. Es decir, si intentas borrar o actualizar un registro de la tabla principal que tiene registros relacionados en la tabla secundaria, la base de datos no te lo permitirá.
- ✔ **SET DEFAULT**: Si se borra o actualiza una fila de la tabla principal, se establecen en su valor por defecto los valores de la clave ajena en las filas de la tabla secundaria. Esto es útil cuando tienes un valor por defecto definido para la clave ajena en la tabla secundaria.

Restricciones check

Además de las restricciones inherentes al modelo relacional, la mayoría de gestores de datos permiten establecer condiciones para la validación de datos introducidos, cosa que podremos hacer mediante la clausula **CHECK** .

Para entender como añadir una restricción en un campo de una tabla, imaginemos que tenemos una tabla de usuario donde tenemos, entre otros datos, la edad de los usuarios. Nuestra plataforma sólo permitirá usuarios con una edad igual o superior a 16 años, en este casos la sentencia para crear la tabla sería:

```
CREATE TABLE usuarios(
        nick    VARCHAR(20),
        nombre          TEXT,
        email   TEXT,

        edad    NUMERIC(3),
        CONSTRAINT PK_usuarios PRIMARY KEY (nick),
        CONSTRAINT CK_edad16 CHECK (edad >= 16));
```