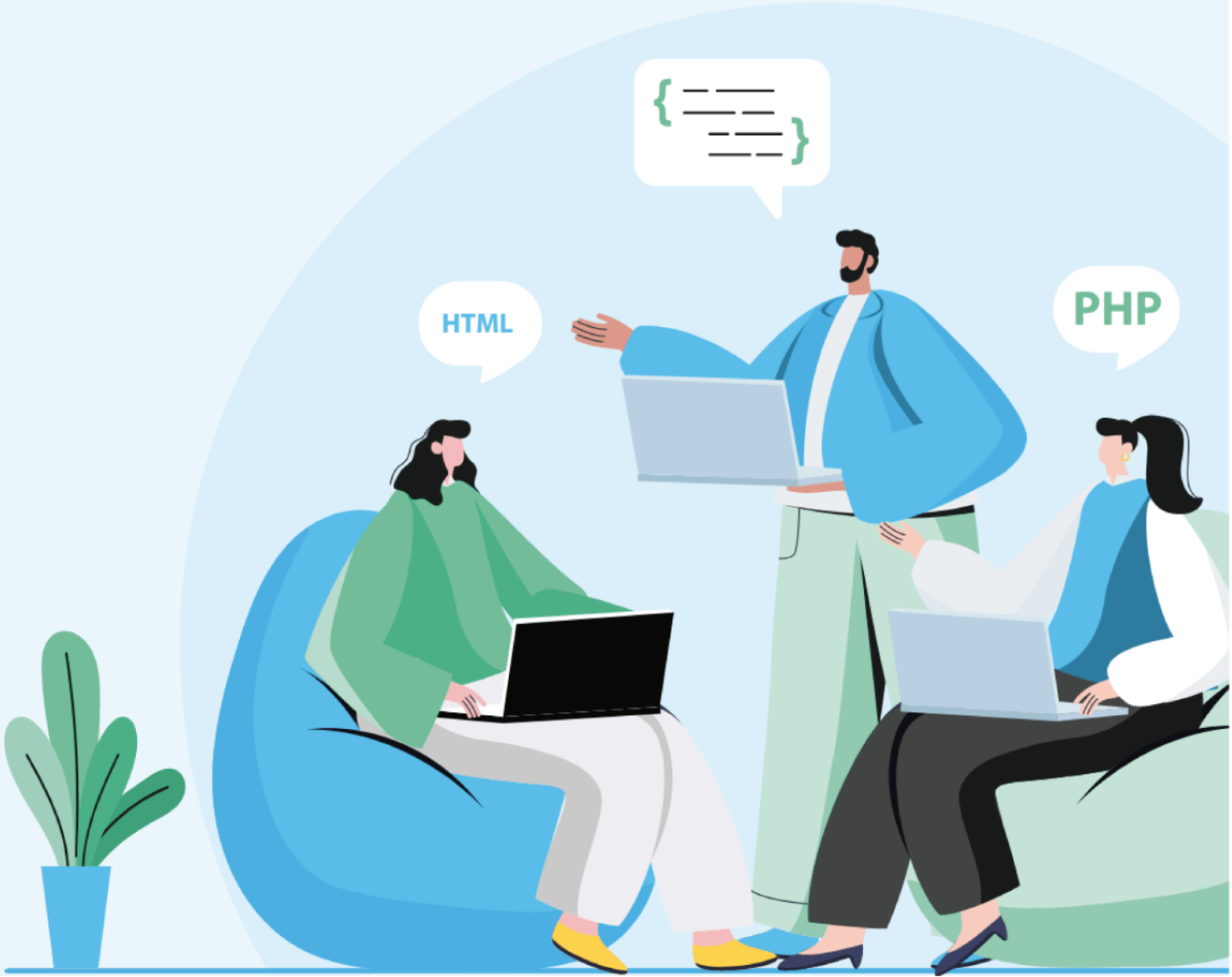
alkemy

SQL I



Bienvenidos al módulo de SQL I. Durante el desarrollo de este módulo, nos adentraremos en los principios básicos de SQL. Para ello, buscaremos hacer foco en conceptos claves como los componentes de una base de datos, estructura de tablas, manipulación de datos y tipos de JOIN.

1. Tema 1: Manipulación

Cómo usar SQL para acceder, crear y actualizar datos almacenados en una base de datos

Bases de datos relacionales

Antes de empezar a hablar de bases de datos relacionales, es necesario saber qué es una base de datos.

> *Una base de datos es un conjunto de datos pertenecientes a un mismo contexto y almacenados sistemáticamente para su posterior uso. Una aplicación web de e-commerce almacena sus datos de clientes, proveedores, ventas, envíos, etc. en una base de datos.*

Una base de datos **relacional** es aquella cuya organización de datos cumple con el **modelo relacional**.

El modelo relacional es el modelo más utilizado para modelar los datos de una base de datos. Está compuesto por **tablas, atributos, tuplas y relaciones**.

El principal concepto del modelo relacional es la **tabla**.

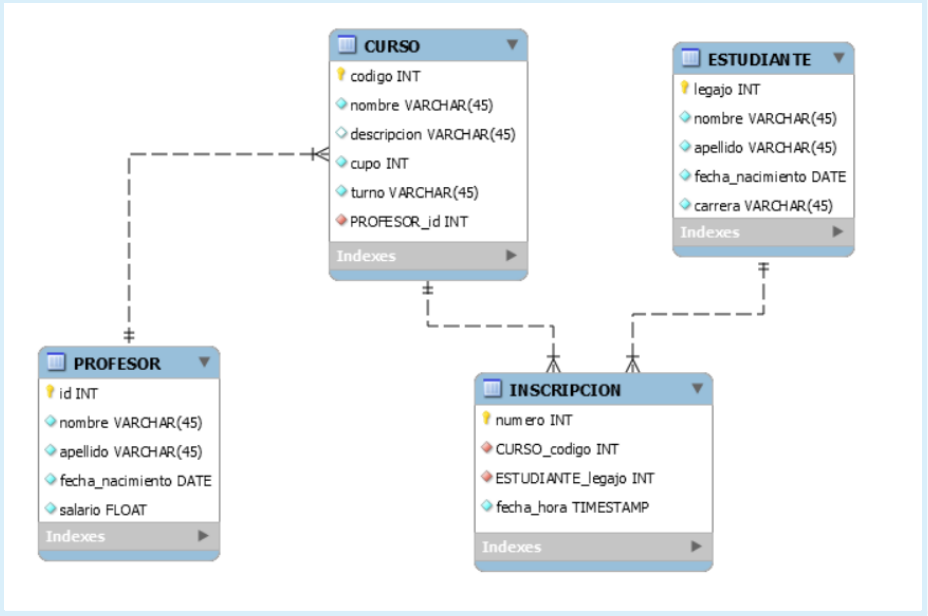
Considerada esta como un conjunto de columnas y filas reunidas en una estructura que representa una **entidad**. Una entidad es una persona, lugar, cosa, evento o concepto sobre el cual los datos son recolectados.
Cada tabla comprende uno o más **atributos** (columnas). Un atributo es un hecho simple que describe o caracteriza una entidad.

Los datos se almacenan dentro de una tabla en **tuplas** (filas). Una tupla es un conjunto de datos cuyos valores hacen una instancia de cada atributo definido por esa relación.

En el modelo relacional, cada tabla tiene al menos un atributo que se considera como clave primaria. La clave primaria es aquel atributo o conjunto de ellos que identifica unívocamente a una tupla (registro). Además, las tablas se relacionan a través de claves foráneas: la clave foránea se coloca como atributo en la tabla hija y su valor es el mismo que la clave primaria del registro de la tabla padre. De esa manera se relacionan las tablas.

Una base de datos relacional debe estar normalizada, esto es, en simples palabras, minimizar los datos redundantes y preservar la integridad de los datos almacenados y su mantenimiento.
En la siguiente imagen, podemos ver un ejemplo sencillo de modelo relacional.

Figura 1: Modelo relacional



Fuente: [Imagen sin título sobre modelo relacional]. (s.f.).

Indicación del elemento: hay dos tablas, una que representa a la entidad “PROFESOR” y otra que representa a la entidad “CURSO”. Cada tabla tiene sus propios atributos. Cada una tiene su clave primaria: “id” para la tabla PROFESOR y “código” para la tabla CLASE. Ambas entidades se relacionan entre sí, de forma que un

profesor puede dictar muchas clases, pero para este caso, una clase puede ser dictada solo por un profesor. De esta forma, la tabla CLASE incluye entre sus campos una clave foránea (PROFESOR_id) que hace referencia a la clave primaria de PROFESOR para identificar al profesor que dicta la clase.

Para leer más sobre normalización y las tres formas normales más conocidas, puedes dirigirte a la siguiente publicación.

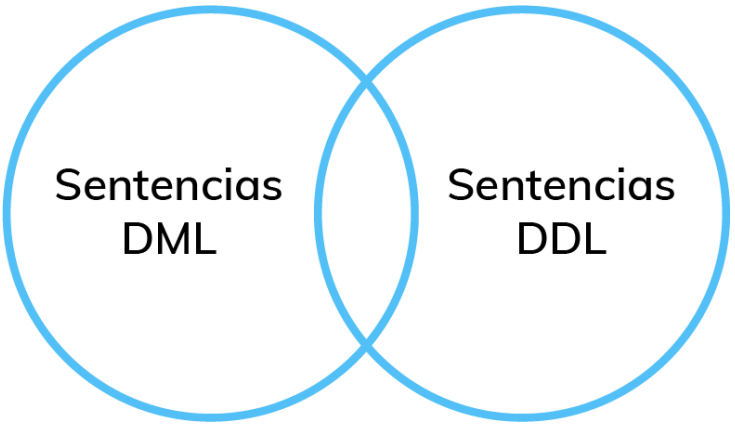
Fuente: Digital Guide Ionos, (2018). Normalización: evita las redundancias en las bases de datos. Recuperado de https://www.ionos.es/digitalguide/hosting/cuestiones-tecnicas/normalizacion/

Links de interés:
SQL insert into select. Statement: - https://www.w3schools.com/sql/sql_insert_into_select.asp

Sentencias SQL

Las sentencias de SQL son los diferentes comandos que nos sirven para operar sobre los datos. Estos se dividen en diferentes tipos. Los dos tipos principales son los que se muestran a continuación.

Figura 2: Principales sentencias de SQL



Fuente: elaboración propia

Sentencias DML – Lenguaje de manipulación de datos: permiten acceder y manipular datos. Por ejemplo:

- SELECT
- INSERT

- UPDATE
- DELETE

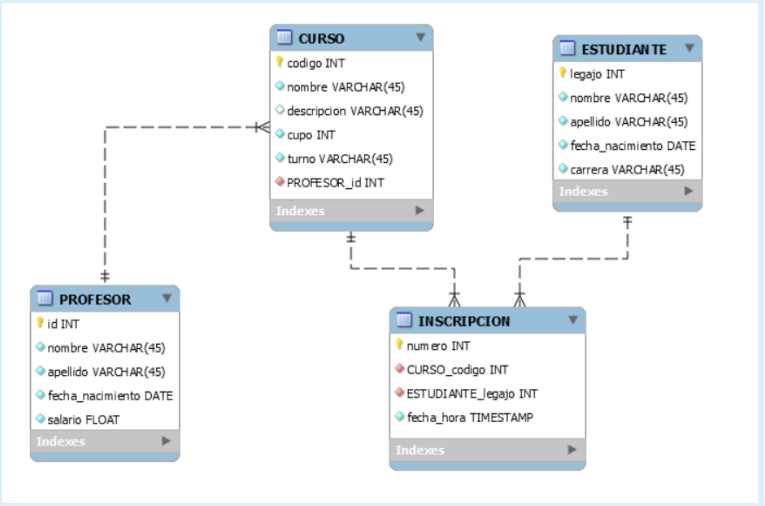
Sentencias DDL – Lenguaje de definición de datos: permiten crear, modificar o eliminar objetos de la base de datos. Por ejemplo:

- CREATE
- DROP
- ALTER

Existen otros tipos de sentencias como las DCL (lenguaje de control de datos) que están relacionadas con los privilegios para acceder a determinados objetos o las TCL (lenguaje de control de transacción) para el control de las transacciones. Sin embargo, exceden el alcance de esta cápsula, por lo que solo serán nombradas.

En las siguientes secciones, vamos a profundizar más sobre las sentencias DDL y DML. Para todos los ejemplos nos basaremos en el modelo relacional de la sección anterior.

Figura 3: Modelo relacional



Fuente: [Imagen sin título sobre modelo relacional]. (s.f.).

CREATE

La sentencia CREATE puede ser utilizada para crear diferentes objetos: bases de datos, tablas, constraints, etc.

Para crear una base de datos:

CREATE DATABASE *nombre_base_de_datos;*
Por ejemplo:
CREATE DATABASE mydb;

Para crear una tabla:

CREATE TABLE *nombre_tabla (*
nombre_columna_1 tipo_de_dato,
nombre_columna_2 tipo_de_dato,
...);
Por ejemplo:
CREATE TABLE profesor (
id INT NOT NULL AUTO_INCREMENT,
nombre VARCHAR(45) NOT NULL,
apellido VARCHAR(45) NOT NULL,

fecha_nacimiento DATE NOT NULL,
salario FLOAT NOT NULL,
PRIMARY KEY(id)
);
CREATE TABLE curso (
codigo INT NOT NULL,
nombre VARCHAR(45) NOT NULL,
descripcion VARCHAR(45),
cupo INT NOT NULL DEFAULT 20,
turno VARCHAR(45) NOT NULL,
PRIMARY KEY(codigo)
);

Un campo puede ser definido como entero, cadena de caracteres, fecha, flotante, entre otros. Las palabras reservadas para indicar el tipo de dato dependerán de la variante de SQL que se utilice.

También es posible definir si el campo no admite valores nulos, si el campo es autoincremental y otras restricciones.

- Si el campo no admite valores nulos utilizando las palabras reservadas: NOT NULL. Esto es similar a los formularios web. En ocasiones, hay campos marcados con (*) para indicar que son obligatorios. Si no ingresamos datos en esos campos, la página web no nos permitirá enviar el formulario. Pasa lo mismo con los campos definidos con NOT NULL, si no insertamos datos en ese campo, no podremos insertar el registro a menos que el campo tenga definido un valor por defecto.

Es posible agregar un valor por defecto con la siguiente sintaxis:

```
CREATE TABLE curso (
codigo INT NOT NULL,
 nombre VARCHAR(45) NOT NULL,
 descripcion VARCHAR(45),
 cupo INT NOT NULL DEFAULT 20,
 turno VARCHAR(45) NOT NULL,
 PRIMARY KEY(codigo)
);
```

De esta forma, cada vez que se inserte un registro sin valor para el campo “cupo”, se grabará el número 20 en ese campo.

- Si el campo es autoincremental, es decir, que se genera automáticamente con valores secuenciales, se puede indicar con la palabra reservada “AUTO_INCREMENT” (dependiendo de la variante SQL).

```
CREATE TABLE profesor (
  id INT NOT NULL AUTO_INCREMENT,
  nombre VARCHAR(45) NOT NULL,
  apellido VARCHAR(45) NOT NULL,
  fecha_nacimiento DATE NOT NULL,
  salario FLOAT NOT NULL,
  PRIMARY KEY(id)
 );
```

Cuando definimos que un campo es autoincremental el motor de base de datos se encarga de asignarle valores secuenciales.

- Otras restricciones como: PRIMARY KEY, FOREING KEY, UNIQUE, etc. Las restricciones pueden ser creadas al momento de definir la tabla o, como veremos en la sección “*Contraints*”, individualmente.

Nota: también es posible crear una tabla a partir de otra. Para obtener más información se sugiere visitar el siguiente sitio web: https://www.w3schools.com/sql/sql_create_table.asp

INSERT

La sentencia INSERT se utiliza para insertar datos en una tabla creada (también se utiliza la frase "poblar una tabla con datos"). Hay varias formas de usar esta sentencia.

1) Especificando los campos en los que se quiere insertar datos y su valor respectivo:

   INSERT INTO *nombre_tabla* (*nombre_columna_1*, *nombre_columna_2*,...)

   VALUES(*valor_col_1*, *valor_col_2*, ...);

   Por ejemplo, a continuación, le decimos a SQL que en los campos nombre, apellido, fecha_nacimiento y salario de la tabla profesor, inserte los valores 'Juan', 'Pérez','1990-06-06', 45000 respectivamente.

   INSERT INTO profesor (nombre, apellido, fecha_nacimiento, salario) VALUES ('Juan','Pérez','1990-06-06',45000);

Tabla 1: VALUES - ejemplo 1

	id	nombre	apellido	fecha_nacimiento	salario
▶	1	Juan	Pérez	1990-06-06	45000

Fuente: elaboración propia

En este caso, no fue necesario especificar un valor para el campo id porque es autoincremental.
Tener en cuenta que, si no especificamos el valor que queremos asignar a uno o más campos, se les asignará NULL o el valor por defecto que se haya determinado. Si no tiene un valor por defecto y el campo no permite nulos, no se insertarán datos y la sentencia retornará error. Por ejemplo:

INSERT INTO curso (codigo,nombre,descripcion,turno,profesor_id)

values (101,'Algoritmos', 'Algoritmos y estructuras de datos','Mañana',1);

Tabla 2: VALUES - ejemplo 2

	codigo	nombre	descripcion	cupo	turno	PROFESOR_id
▶	101	Algoritmos	Algoritmos y estructuras de datos	20	Mañana	1

Fuente: elaboración propia

En la sentencia anterior, no especificamos ningún valor para el campo “cupo”, sin embargo, podemos ver que el registro que insertamos tiene el valor 20. Esto se debe a que ese campo tiene definido el valor 20 como valor por defecto.

2) Omitir la especificación de los campos y escribir solamente los valores. Para estos casos, debemos indicar valores para todos los campos de la tabla, en el mismo orden de definición de la tabla.

INSERT INTO *nombre_tabla* VALUES (*valor_col_1*, *valor_col_2*, ...)

Por ejemplo, para insertar datos con este método en la tabla “profesor”, podemos ejecutar:

INSERT INTO profesor VALUES (null,'María Emilia','Paz','1984-07-15',60000);

Tabla 3: VALUES - ejemplo 3

	id	nombre	apellido	fecha_nacimiento	salario
▶	1	Juan	Pérez	1990-06-06	45000
	2	María Emilia	Paz	1984-07-15	60000

Fuente: elaboración propia

Así, el motor de base de datos toma los campos de la tabla en el orden de creación: id, nombre, apellido, fecha_nacimiento, salario y les asigna uno a uno los valores especificados en el INSERT.

Si bien el campo id es autoincremental, para este caso debemos especificarle un valor. Si lo que queremos es que se asigne automáticamente, podemos asignarle el valor NULL.

3) Insertar registros a partir de otra tabla:

```
INSERT INTO tabla2 (col_1, col_2, col_3)

        SELECT col_1_t1, col_2_t1, col_3_t1
        FROM tabla1
        WHERE condición;
```

En este caso, también es posible especificar los campos en los que se quieren insertar datos, u omitir esa parte de la sentencia y especificar valores para todos los campos.

SELECT

Para consultar los datos de una tabla se utiliza la sentencia SELECT.

Con SELECT podemos obtener los datos de todos los campos de la tabla, por ejemplo: SELECT * FROM profesor; nos devolverá todos los datos que tenga la tabla en todas sus columnas. Haciendo una analogía entre tabla y planilla, podemos decir que el resultado será todos los profesores que estén anotados en la planilla de profesores.

Tabla 4: VALUES - ejemplo 4

	id	nombre	apellido	fecha_nacimiento	salario
▶	1	Juan	Pérez	1990-06-06	45000
	2	María Emilia	Paz	1984-07-15	60000

Fuente: elaboración propia

En algunos casos, no es necesario o no interesan todas las columnas por una cuestión de visibilidad (tener en cuenta que una tabla puede tener **muchas** columnas), para esos casos, especificamos cuáles son las columnas que nos interesan. Por ejemplo:

SELECT apellido, fecha_nacimiento FROM profesor;

Siguiendo con la analogía entre tabla y planilla, ejecutando esa consulta obtendremos el apellido y la fecha de nacimiento de cada profesor que se encuentre en la planilla de profesores.

Tabla 5: VALUES - ejemplo 5

	apellido	fecha_nacimiento
▶	Pérez	1990-06-06
	Paz	1984-07-15

Fuente: elaboración propia

También se puede especificar un orden distinto. Tal vez quiero ver, primero, la fecha de nacimiento y luego el apellido del profesor, entonces ejecuto:

SELECT fecha_nacimiento, apellido FROM profesor;

Tabla 6: VALUES - ejemplo 6

	fecha_nacimiento	apellido
▶	1990-06-06	Pérez
	1984-07-15	Paz

Fuente: elaboración propia

ALTER

Para modificar la definición de un objeto de la base de datos se utiliza ALTER.
ALTER permite modificar la definición de una tabla, ya sea sus columnas (agregar, modificar o eliminarlas) o sus *constraints*.

1) Agregar una columna a una tabla:

   ALTER TABLE *nombre_tabla* ADD nueva_columna *tipo_de_datos*;

   Por ejemplo:
   ALTER TABLE profesor ADD direccion VARCHAR(100);

   Si ejecutamos “select * from profesor” podemos ver la nueva columna agregada.

Tabla 7: VALUES - ejemplo 7

	id	nombre	apellido	fecha_nacimiento	salario	direccion
▶	1	Juan	Pérez	1990-06-06	45000	NULL
	2	María Emilia	Paz	1984-07-15	60000	NULL

Fuente: elaboración propia

2) Modificar una columna de una tabla:

   ALTER TABLE *nombre_tabla* MODIFY *nombre_columna nuevo_tipo_datos*;

   Por ejemplo:
   ALTER TABLE profesor MODIFY direccion VARCHAR(150);

3) Eliminar una columna de una tabla:

   ALTER TABLE *nombre_tabla* DROP COLUMN *nombre_columna*;

   Por ejemplo:
   ALTER TABLE profesor DROP direccion;

UPDATE

Para modificar los datos que tiene almacenados una tabla, utilizamos la sentencia UPDATE con la siguiente sintaxis:

```
UPDATE nombre_tabla
SET columna_1 = valor_nuevo1, columna_2 = valor_nuevo2
WHERE condicion;
```

Si bien la sentencia WHERE se explicará más adelante, es necesario que sepamos que el WHERE permite filtrar los registros según las condiciones indicadas. Si al momento de hacer un UPDATE a la tabla, no especificamos ninguna condición en la sección del WHERE, la actualización de campos se realizará en todos los registros de la tabla.

Ejemplo: si queremos incrementar el salario de todos los profesores en un 20 % más, ejecutamos:

UPDATE profesor SET salario = salario * 1.2;

Esto actualizará en un 20 % el salario de todos los profesores.

Tabla 8: Antes del UPDATE (ejemplo 1)

	id	nombre	apellido	fecha_nacimiento	salario
▶	1	Juan	Pérez	1990-06-06	45000
	2	María Emilia	Paz	1984-07-15	60000

Fuente: elaboración propia

Tabla 9: Después del UPDATE (ejemplo 1)

	id	nombre	apellido	fecha_nacimiento	salario
▶	1	Juan	Pérez	1990-06-06	54000
	2	María Emilia	Paz	1984-07-15	72000

Fuente: elaboración propia

Si solo queremos cambiar el salario de Juan, deberíamos ejecutar:
UPDATE profesor SET salario = 55000

WHERE id = 1;

Esta consulta obtiene los registros de la tabla profesor, realiza un filtro buscando el id = 1 y le actualiza el salario a 55 000.

Tabla 10: Antes del UPDATE (ejemplo 2)

	id	nombre	apellido	fecha_nacimiento	salario
▶	1	Juan	Pérez	1990-06-06	54000
	2	María Emilia	Paz	1984-07-15	72000

Fuente: elaboración propia

Tabla 11: Después del UPDATE (ejemplo 2)

	id	nombre	apellido	fecha_nacimiento	salario
▶	1	Juan	Pérez	1990-06-06	55000
	2	María Emilia	Paz	1984-07-15	72000

Fuente: elaboración propia

DELETE

Usaremos la sentencia DELETE cuando queramos eliminar registros de una tabla (solo elimina registros, no sirve para eliminar la estructura de la tabla). La sintaxis es la siguiente:

```
DELETE FROM nombre_tabla WHERE condiciones;
```

Al igual que el UPDATE, si no se incluyen condiciones de filtrado, la sentencia DELETE eliminará todos los registros de la tabla.

Tomaremos como ejemplo la tabla curso.

Tabla 12: Tabla curso

codigo	nombre	descripcion	cupo	turno	PROFESOR_id
101	Algoritmos	Algoritmos y estructuras de datos	20	Mañana	1
102	Matemática Discreta	NULL	20	Tarde	2

Fuente: elaboración propia

Si ejecutamos: DELETE FROM curso; la tabla curso quedará vacía. En cambio, si ejecutamos: DELETE FROM curso WHERE codigo = 102; solo se eliminará el segundo registro de la tabla, quedando como se muestra a continuación.

Tabla 13: DELETE FROM curso WHERE codigo = 102;

codigo	nombre	descripcion	cupo	turno	PROFESOR_id
101	Algoritmos	Algoritmos y estructuras de datos	20	Mañana	1

Fuente: elaboración propia

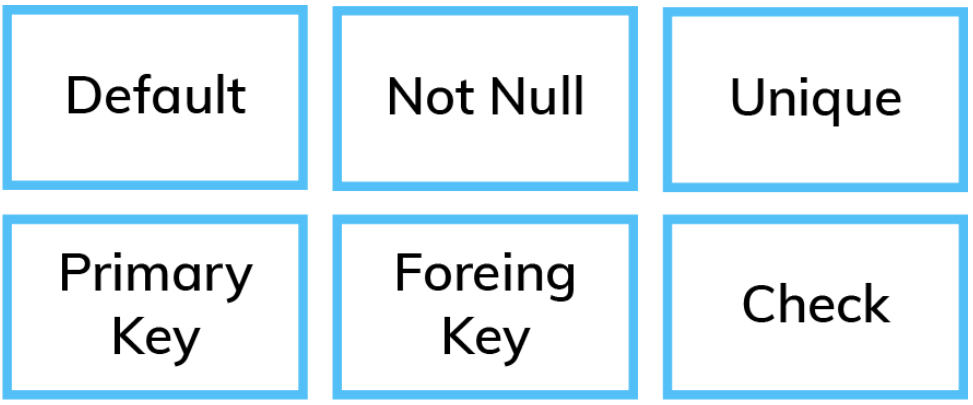
CONSTRAINTS

Las ***constraints*** son reglas que permiten definir cuáles son las reglas para insertar o actualizar datos en una tabla, cuáles son las condiciones con las que debe cumplir el dato para ser insertado.

Estas reglas se pueden especificar durante la creación de la tabla o después, agregándolas o modificándolas. Las *constraints* afectan a los campos de la tabla.

Hay varios tipos de *constraints* posibles como lo muestra la imagen a continuación.

Figura 4: *Constraints*



Fuente: elaboración propia

DEFAULT: indica el valor por defecto que se insertará en un campo cuando no se especifiquen datos para él.

Sintaxis:

ALTER TABLE *nombre_tabla*
ADD CONSTRAINT *nombre_contraint* DEFAULT '*valor_default*'
FOR *nombre_campo*;

Por ejemplo:

ALTER TABLE profesor
ADD CONSTRAINT default_salario DEFAULT 30000 FOR salario;

La sintaxis siempre depende del motor de base de datos donde se están ejecutando las sentencias. Si queremos agregar un valor por defecto en MySQL Server usamos:

ALTER TABLE persona
ALTER TABLE profesor alter salario set DEFAULT 30000;

NOT NULL: indica que un campo no puede tener valor nulo. Esta *constraint* puede combinarse con la del valor por defecto. De modo que cuando se intente insertar un registro con un valor nulo en un campo que no los admite, se asigne un valor por defecto.

ALTER TABLE *nombre_tabla* MODIFY *nombre_campo tipo_dato* NOT NULL;
ALTER TABLE curso MODIFY cupo INT NOT NULL DEFAULT 20;

UNIQUE: especifica que el valor de un campo o un conjunto de campos debe ser único en toda la tabla.

Sintaxis para una *constraint* de un solo campo:

ALTER TABLE *nombre_tabla* ADD CONSTRAINT *nombre_constraint* UNIQUE (campo);

Sintaxis para una *constraint* con más de un campo:

ALTER TABLE *nombre_tabla*
ADD CONSTRAINT *nombre_constraint* UNIQUE (campo1, campo2);

Si la tabla profesor tuviera el campo “DNI”, podríamos especificar ese campo con una *constraint* de tipo UNIQUE, ya que no debería haber números de documento repetidos entre personas.

PRIMARY KEY: identifica cada registro en la tabla. Puede ser de uno o varios campos (compuesta), no puede ser nulo y su valor (o conjunto de valores) no puede repetirse.

Sintaxis:

ALTER TABLE *nombre_tabla*
ADD CONSTRAINT *nombre_constraint* PRIMARY KEY (*campo*);
ALTER TABLE *nombre_tabla*
ADD CONSTRAINT *nombre_constraint* PRIMARY KEY (*campo1*, *campo2*);

Por ejemplo, nuestra tabla profesor tiene como PRIMARY KEY al campo id. Ese campo es el que identifica unívocamente a un profesor en la tabla. Si lo analizamos mejor, ninguno de los otros campos (nombre, apellido, fecha_nacimiento, salario) podría ser candidato a PRIMARY KEY porque en todos los casos puede haber valores repetidos: dos profesores pueden tener el mismo nombre, o el mismo apellido, o haber nacido el mismo día o cobrar la misma cantidad de dinero en su sueldo.

FOREING KEY: sirve para relacionar dos tablas. Una FOREING KEY o clave foránea en la tabla curso, es una PRIMARY KEY en la tabla profesor. A partir del campo clave foránea, las tablas curso y profesor se relacionan.

Sintaxis:

ALTER TABLE *nombre_tabla1*

ADD CONSTRAINT *nombre_constraint*
FOREIGN KEY (*campo_tabla1)* REFERENCES
nombre_tabla2(*campo_tabla2*);

Por ejemplo, se puede especificar que un profesor está relacionado con un curso:

ALTER TABLE curso
ADD CONSTRAINT *FK_profesor_curso*
FOREING KEY(PROFESOR_id) REFERENCES *profesor*(*id*);

Con esa sentencia estamos indicando que, para el campo PROFESOR_id de la tabla curso, será una clave foránea que hará referencia al campo id de la tabla profesor.

CHECK: sirve para realizar otras validaciones lógicas sobre el valor de un campo. Pueden utilizarse los operadores =, <, >, etc.

Debido a que la sintaxis para definir una *constraint* es parecida en todos los casos, para este caso solo mostraremos un ejemplo:

ALTER TABLE *profesor*
ADD CONSTRAINT *CHK_Salario*
CHECK *salario* >=15000;

Si lo que desea es eliminar una *constraint*, debe usarse la siguiente sintaxis:

ALTER TABLE *nombre_tabla*
DROP CONSTRAINT *nombre_constraint*;

Ejercicio 1

Objetivo: en este ejercicio crearemos una tabla, especificando los campos; luego agregaremos un campo a la tabla creada anteriormente. Para finalizar, utilizaremos sentencias de manipulación de datos

Ingresa al sitio web: https://sqliteonline.com/

Selecciona el editor de SQLite, en el centro podrás ver un editor de SQL.

1) Crea una tabla llamada CURSO con los atributos:
   a. Código de curso (clave primaria, entero no nulo);
   b. Nombre (varchar no nulo);
   c. Descripción (varchar);
   d. Turno (varchar no nulo).

1) Agrega un campo “cupo” de tipo numérico.

2) Inserta datos en la tabla:
   a. (101, “Algoritmos”,”Algoritmos y Estructuras de Datos”,”Mañana”,35)
   b. (102, “Matemática Discreta”,””,”Tarde”,30)

3) Crea un registro con el nombre nulo y verifica que no funciona.

4) Crea un registro con la clave primaria repetida y verifica que no funciona.

5) Actualiza, para todos los cursos, el cupo en 25.

6) Elimina el curso “Algoritmos”.

Documentación

Más sobre el modelo relacional:

Fuente: **Campus MVP**, (s.f.). Diseñando una base de datos en el modelo relacional. Recuperado de https://www.campusmvp.es/recursos/post/Disenando-una-base-de-datos-en-el-modelo-relacional.aspx?source=post_page----------

Resumen de sentencias SQL:

Fuente: **Khan Academy**, (s.f.). Desarrollo de SQL. Recuperado de https://es.khanacademy.org/computing/computer-programming/sql-documentation

Teoría y ejercicios SQL: https://www.w3schools.com/sql/default.asp

Fuente: **Oppel, A.; Sheldon, R.** (2010). Fundamentos de SQL (3.ra. ed.). México: Pearson Educación.

Videos

Modelo relacional

Fuente: **Danisable** [Danisable]. (s.f.). Bases de datos desde Cero | Modelo Relacional | Parte 7 [YouTube]. Recuperado de https://www.youtube.com/watch?v=SAp0NFqOQ3w&list=PLAzlSdU-KYwWodXmd8PeJY6zpv0wEwY0J&index=7&ab_channel=Danisable

Sentencias DDL

Fuente: **Píldoras Informáticas** [pildorasinformáticas]. (28 de diciembre de 2015). Curso SQL. DDL Creación de tablas. Vídeo 16 [YouTube]. Recuperado de https://www.youtube.com/watch?v=XJb6qflbsx4&ab_channel=pildorasinformaticas

Sentencias DDL – ALTER y DROP

Fuente: **Píldoras Informáticas** [pildorasinformáticas]. (11 de enero de 2016). Curso SQL. Agregar, eliminar y modificar campos. Vídeo 17 [YouTube]. Recuperado de https://www.youtube.com/watch?v=XJb6qflbsx4&ab_channel=pildorasinformaticas