<codoacodo/>

Desarrollo Fullstack

BA

Les damos la bienvenida

Vamos a comenzar a grabar la clase

BASES DE DATOS

Sintaxis

Agencia de Aprendizaje a lo largo de la vida

Buenos Aires Ciudad
Educación

Repaso 🤔

La clase pasada, aprendimos que es una base de datos y cómo montar un servidor MySQL, visualizando nuestros datos a través de un Sistema Gestor de Base de Datos.

Además conocimos sus tipos de datos, restricciones y hablamos del lenguaje DDL para crear o manipular nuestras bases de datos y tablas.

Vamos a crear una base de datos y una tabla de ejemplo...

Ahora que ya tenemos nuestra base de datos, es momento de empezar a llenarla pero antes, hablemos sobre DML.

¿Qué es DML?

Al igual que el lenguaje DDL, es un conjunto de palabras claves utilizadas para manipular la estructura que almacena los datos.

Las sentencias DML son aquellas utilizadas para insertar, borrar, modificar y consultar los datos de una tabla.

Sentencias DML

INSERT

Añade nuevos datos a la tabla de la base de datos existente

SELECT

Consulta datos de la tabla o de varias tablas

UPDATE

Cambia o actualiza los valores de la tabla.

DELETE

Elimina registros o filas de la tabla

INSERTAR

```
INSERT INTO empleados (nombre,
apellido, genero, edad) VALUES
(“Pepe”, “Grillo”, “M”, 32);

// Múltiples valores
INSERT INTO empleados (nombre,
apellido, genero, edad) VALUES
    (“Natasha”, “Romanoff”, “F”, 35),
    (“Tony”, “Stark”, “M”, 43),
    (“Clark”, “Kent”, “M”, 37)
;
*id se genera automáticamente
```

INSERT nos permite agregar registros a nuestras tablas.

Es necesario indicar los campos que vamos a agregar y luego de la sentencia **VALUES** poner en el mismo orden los valores correspondientes a esos campos.

También es posible agregar múltiples registros al mismo tiempo, separando cada declaración mediante una coma.

SELECCIONAR

Con **SELECT** en cambio, vamos a consultar los datos de una tabla, trayendo solo los valores solicitados dependiendo la consulta realizada.

Con el ***** traemos todos los valores y registros desde (**FROM**) la tabla seleccionada.

También podemos traer solo algunos campos determinados, para eso indicamos los campos buscados antes del FROM.

```
// Todos los campos de todos los empleados.
SELECT * FROM empleados;


// Solo los campos nombre y edad de todos los empleados
SELECT nombre, edad FROM empleados;
```

SELECCIONAR

```
// Todos los campos de los empleados mayores de 35 años.
SELECT * FROM empleados
    WHERE edad > 35;

// Todos los campos de los primeros 5 empleados mayores de 35 años.
SELECT * FROM empleados
    WHERE edad > 35 LIMIT 5;
```

Otra forma muy útil es colocar una sentencia **WHERE** para generar una condición a cumplir y de esa manera filtrar el resultado.

Asimismo, con la sentencia **LIMIT** podemos limitar la cantidad de registros obtenidos.

SELECCIONAR

Por otra parte, SQL nos brinda la posibilidad de traer los datos de forma ordenada.

Ese ordenamiento puede ser en forma ascendente o descendente.

```
// Todos los campos de los empleados ordenados por edad ascendente.
SELECT * FROM empleados ORDER BY edad asc;

// Todos los campos de los empleados ordenados por edad descendente.
SELECT * FROM empleados ORDER BY edad desc;
```

Vamos a probar estas nuevas sentencias.

ACTUALIZAR

```
// Seleccionamos el registro con ID 3
y modificamos su atributo name.
UPDATE empleados
     SET name = “Antonio”
     WHERE id = 3;

*¡NO OLVIDES EL WHERE!
```

Mediante la sentencia **UPDATE** podemos actualizar los atributos de nuestros registros. Es importante asignarle un nuevo valor al atributo seleccionado mediante **SET**.

En caso que nos olvidemos la sentencia **WHERE** cambiaremos el nombre de TODOS los registros, por eso nunca debemos olvidarla.

ELIMINAR

Posiblemente la sentencia más conflictiva, culpable de incontables errores irreparables para principiantes y no tan principiantes.

La sentencia **DELETE**, al igual que la anterior **UPDATE**, necesita un **WHERE** para saber que registro eliminar. Si no se lo pasamos, entonces **borraremos todos los registros de la tabla.**

```
// Seleccionamos el registro con nombre “Pepe” y lo eliminamos de la tabla.
DELETE FROM empleados
      WHERE nombre like “Pepe”;

*¡AQUÍ SÍ, POR FAVOR, NO OLVIDES EL WHERE! 🙏
```

Relaciones SQL

En una base de datos las tablas pueden estar relacionadas entre sí. Esto permite poder seleccionar datos de varias tablas en una misma consulta.

Tipos de Relaciones

Las bases de datos relacionales tienen diversos “tipos de relaciones” utilizadas para vincular nuestras tablas

Este “vínculo” depende de la cantidad de ocurrencias que tiene un registro de una tabla dentro de otra tabla (esto se conoce como cardinalidad).

uno a uno - 1:1

Cuando un registro de una tabla sólo está relacionado con un registro de otra tabla, y viceversa.

uno a muchos / muchos a uno - 1:N - N:1

Una relación de uno a muchos existe cuando un registro de la tabla A está relacionado con ninguno o muchos registros de la tabla B, pero este registro en la tabla B sólo está relacionado con un registro de la tabla A.

muchos a muchos - N:N

Cuando muchos registros de una tabla se relacionan con muchos registros de otra tabla. Vamos a verlo en un ejemplo.

Claves primarias y foráneas

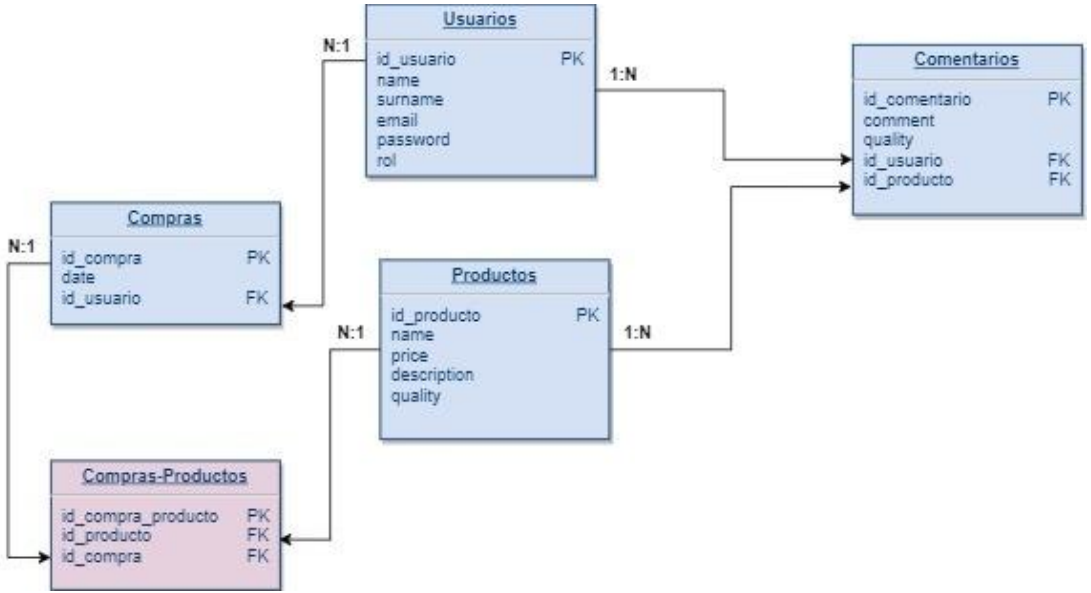
Para que una relación entre dos tablas exista, la tabla que deseas relacionar debe poseer una clave primaria, mientras que la tabla donde estará el lado dependiente de la relación debe poseer una clave foránea.

PRIMARY KEY

Campo cuyos valores identifican de forma única cada registro dentro de la tabla. Este campo tiene la cláusula.

FOREIGN KEY

Campo dentro de la tabla cuyos valores hacen referencia a «claves primarias» en otra tabla. Este campo viene acompañado de la cláusula.



Ejemplo de relaciones

A nuestra base de datos le agregamos una tabla “salarios”, que va a estar relacionada a la tabla de empleados en un tipo de uno a muchos, es decir, muchos empleados poseen un salario.

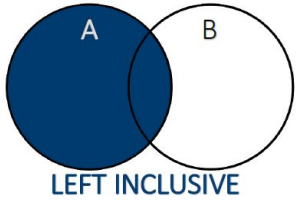
```
CREATE TABLE salarios (
    id_salario  INT  NOT NULL,
    id_empleado INT  NOT NULL,
    salario     INT  NOT NULL,
    FOREIGN KEY (id_empleado) REFERENCES
empleados (id) ON DELETE CASCADE,
    PRIMARY KEY (id_salario)
);
```

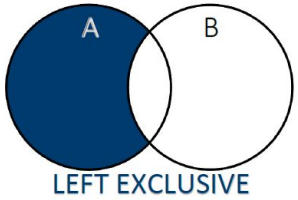
Nuestra tabla posee un campo **id_salario** que funciona como PK de cada registro y un campo **id_empleado** que funciona como FK que apunta al ID de un empleado en esa tabla.

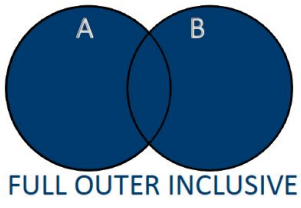
Las palabras **ON DELETE CASCADE**, indican que si se elimina un empleado de la tabla, debe eliminarse el registro de salario relacionado a ese empleado.

JOINS

Es una de las sentencias que nos permiten concatenar en una sola consulta datos relacionados de diferentes tablas.





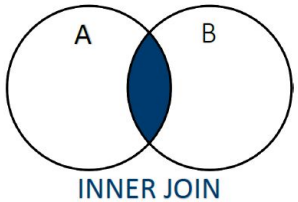


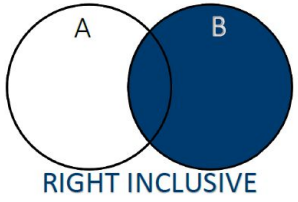
SQL JOINS

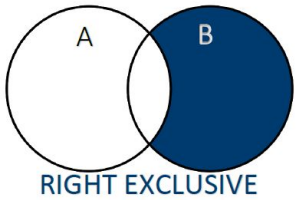
LEFT INCLUSIVE	RIGHT INCLUSIVE
SELECT *[Select List]* FROM TableA A LEFT OUTER JOIN TableB B ON A.Key= B.Key	SELECT *[Select List]* FROM TableA A RIGHT OUTER JOIN TableB B ON A.Key= B.Key
LEFT EXCLUSIVE	**RIGHT EXCLUSIVE**
SELECT *[Select List]* FROM TableA A LEFT OUTER JOIN TableB B ON A.Key= B.Key WHERE B.Key IS NULL	SELECT *[Select List]* FROM TableA A LEFT OUTER JOIN TableB B ON A.Key= B.Key WHERE A.Key IS NULL
FULL OUTER INCLUSIVE	**FULL OUTER EXCLUSIVE**
SELECT *[Select List]* FROM TableA A FULL OUTER JOIN TableB B ON A.Key = B.Key	SELECT *[Select List]* FROM TableA A FULL OUTER JOIN TableB B ON A.Key = B.Key WHERE A.Key IS NULL OR B.Key IS NULL

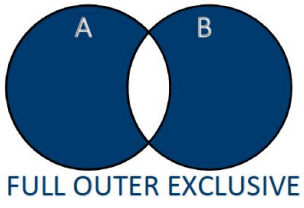
INNER JOIN

SELECT *[Select List]*
FROM TableA A
INNER JOIN TableB B
ON A.Key = B.Key









No te olvides de dar el presente

Recordá:

- Revisar la Cartelera de Novedades.
- Hacer tus consultas en el Foro.

Todo en el Aula Virtual.

Gracias