



INSTITUTO TECNOLÓGICO DE TIJUANA



TECNOLOGIAS DE BASE DE DATOS

Unidad 1
Sistemas Gestores de Base de Datos

Practica 1

Introducción a MySQL

PRESENTA:
Camacho Lopez Valeria

C22211475

DOCENTE:
Fortunato Ramírez Arzate

INTRODUCCIÓN

El objetivo de la primera práctica es empezar a familiarizarse con los distintos comandos (query) básicos para interactuar en MySQL a través de la consola, comprendiendo desde la instalación y configuración hasta la creación y manipulación de bases de datos, principalmente tablas y datos (registros) para la realización de nuevos conceptos.

CONCEPTOS BASICOS DE MySQL

Una base de datos relacional es un sistema que organiza la información en tablas relacionadas entre sí. Cada tabla almacena datos en filas y columnas, donde las filas son registros y las columnas son atributos o campos.

SQL (Structured Query Language) es el lenguaje estándar utilizado para interactuar con estas bases de datos. Permite realizar consultas, insertar, actualizar y eliminar datos, además de administrar la estructura de las tablas.

Las tablas son esenciales porque organizan los datos de manera estructurada. Un registro es una fila dentro de una tabla que representa una entrada específica de datos.

La clave primaria es un identificador único para cada registro en una tabla, lo que garantiza que cada fila sea distinta y facilita la búsqueda y la relación de datos entre tablas.

OBJETIVO GENERAL:

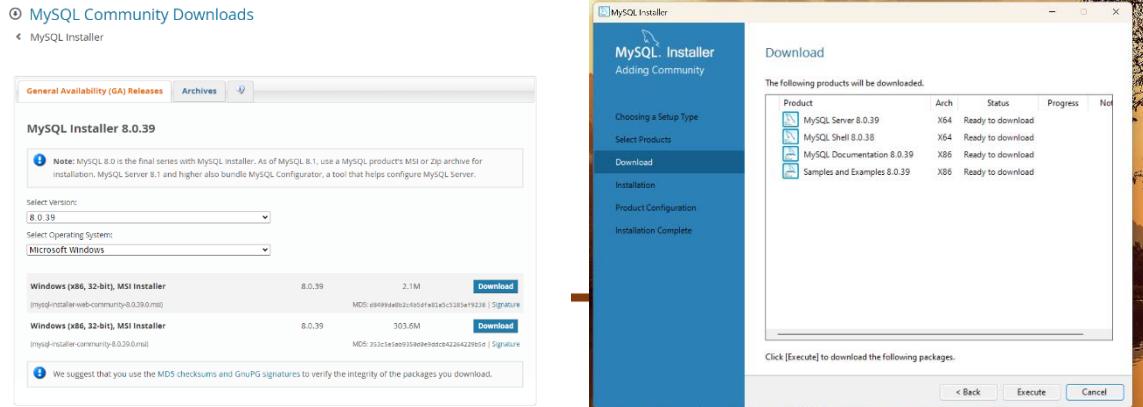
Familiarizar al estudiante con los conceptos básicos y las operaciones fundamentales en MySQL, desde la instalación y configuración hasta la creación y manipulación de bases de datos, tablas, y datos.

Sección 1: Instalación y Configuración

Objetivo: Asegurarse de que MySQL esté instalado y configurado correctamente en el sistema del estudiante.

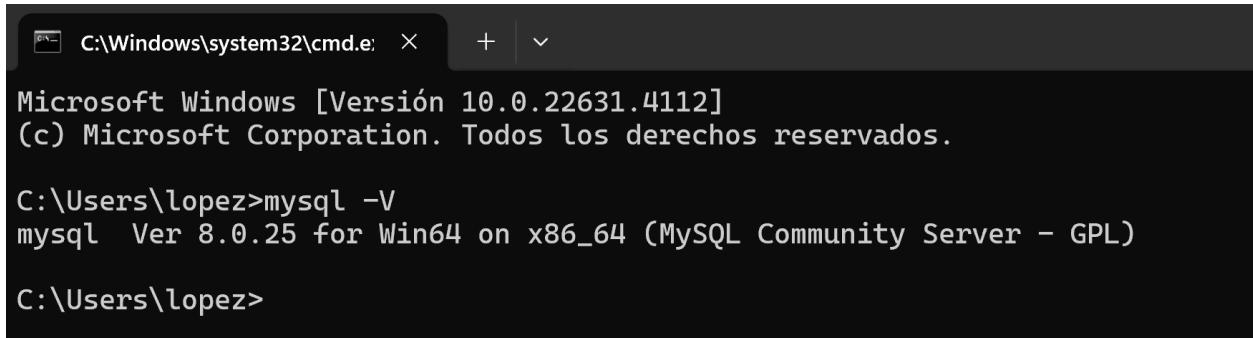
1. Instalación de MySQL:

- **Windows:** El estudiante debe descargar el instalador de MySQL desde la página oficial. Debe seguir las instrucciones del instalador para la



2. Verificación de la instalación:

- El estudiante debe verificar que MySQL está instalado ejecutando el siguiente comando en la terminal:



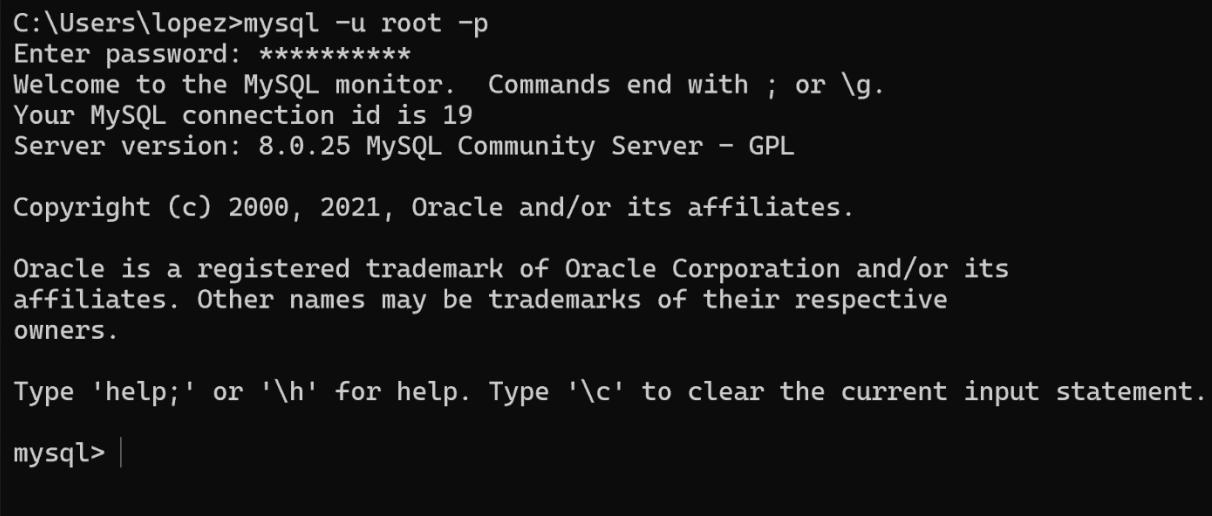
```
C:\Windows\system32\cmd.e: + v
Microsoft Windows [Versión 10.0.22631.4112]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\lopez>mysql -V
mysql Ver 8.0.25 for Win64 on x86_64 (MySQL Community Server - GPL)

C:\Users\lopez>
```

- Luego, deberá iniciar sesión en MySQL como root para confirmar que todo funciona correctamente:

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



```
C:\Users\lopez>mysql -u root -p
Enter password: *****
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 19
Server version: 8.0.25 MySQL Community Server - GPL

Copyright (c) 2000, 2021, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> |
```

Sección 2: Conceptos Básicos de MySQL

Objetivo: Introducir al estudiante en los conceptos fundamentales de bases de datos relacionales y el lenguaje SQL.

1. Bases de Datos Relacionales:

- Una **base de datos** es una colección organizada de datos.
- Una **tabla** es una estructura dentro de una base de datos que almacena datos en filas y columnas.
- **SQL (Structured Query Language)** es el lenguaje estándar para interactuar con bases de datos relacionales.

2. Componentes Clave:

- **Filas (Registros):** Cada fila representa una entrada única en la tabla.
- **Columnas (Campos):** Cada columna representa un atributo de los datos, como nombre, edad, etc.
- **Claves Primarias:** Un campo (o conjunto de campos) que identifica de manera única cada registro en la tabla.

Sección 3: Iniciando Sesión y Gestión de Usuarios

Objetivo: Aprender a iniciar sesión en MySQL y manejar usuarios y permisos.

1. Iniciar sesión en MySQL:

- El estudiante debe iniciar sesión como el usuario root con el siguiente comando:

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

2. Crear un nuevo usuario:

- El estudiante creará un usuario llamado nuevo_usuario utilizando la siguiente instrucción SQL:

```
CREATE USER 'nuevo_usuario'@'localhost' IDENTIFIED BY 'contraseña_segura';
```

```
mysql> CREATE USER 'nuevo_usuario'@'localhost' IDENTIFIED BY 'contraseña_segura';
Query OK, 0 rows affected (0.01 sec)
```

```
mysql>
```

3. Asignar permisos:

- Se deben otorgar todos los privilegios al nuevo usuario sobre todas las bases de datos:

```
GRANT ALL PRIVILEGES ON *.* TO 'nuevo_usuario'@'localhost' WITH GRANT OPTION;
```

```
mysql> GRANT ALL PRIVILEGES ON *.* TO 'nuevo_usuario'@'localhost' WITH GRANT OPTION;
Query OK, 0 rows affected (0.01 sec)

mysql>
```

4. Aplicar cambios:

- El estudiante debe asegurarse de que los cambios en los permisos se apliquen correctamente:

```
FLUSH PRIVILEGES;
```

```
mysql> FLUSH PRIVILEGES;
Query OK, 0 rows affected (0.01 sec)

mysql>
```

Sección 4: Creación y Gestión de Bases de Datos

Objetivo: Aprender a crear, modificar y eliminar bases de datos.

1. Crear una base de datos:

- El estudiante debe crear una base de datos llamada mi_base_de_datos con el siguiente comando:

```
CREATE DATABASE mi_base_de_datos;
```

```
mysql> CREATE DATABASE mi_base_de_datos;
Query OK, 1 row affected (0.01 sec)

mysql> S
```

2. Mostrar bases de datos existentes:

- Se deben listar todas las bases de datos disponibles en el sistema:

```
SHOW DATABASES;
```

```
mysql> SHOW DATABASES;
+-----+
| Database |
+-----+
| empresa |
| information_schema |
| mi_base_de_datos |
| mysql |
| performance_schema |
| sakila |
| sys |
| world |
+-----+
8 rows in set (0.00 sec)
```

3. Seleccionar una base de datos:

- El estudiante debe usar la base de datos recién creada con el siguiente comando:

```
USE mi_base_de_datos;
```

```
mysql> USE mi_base_de_datos;
Database changed
mysql>
```

4. Eliminar una base de datos:

- Finalmente, se eliminará la base de datos usando el siguiente comando:

```
DROP DATABASE mi_base_de_datos;
```

```
mysql> DROP DATABASE mi_base_de_datos;
Query OK, 0 rows affected (0.01 sec)

mysql>
```

Sección 5: Creación y Gestión de Tablas

Objetivo: Crear, modificar y eliminar tablas en MySQL.

1. Crear una tabla:

- Dentro de la base de datos mi_base_de_datos, el estudiante debe crear una tabla llamada empleados con la siguiente estructura:

```
CREATE TABLE empleados (
    id INT AUTO_INCREMENT PRIMARY KEY,
    nombre VARCHAR(50),
    apellido VARCHAR(50),
    salario DECIMAL(10, 2)
);
```

```
mysql> CREATE TABLE empleados(
    -> id INT AUTO_INCREMENT PRIMARY KEY,
    -> nombre VARCHAR(50) NOT NULL,
    -> apellido VARCHAR(50) NOT NULL,
    -> salario DECIMAL (10,2),
    -> departamento_id INT,
    -> FOREIGN KEY(departamento_id) REFERENCES departamentos(id)
    -> );
Query OK, 0 rows affected (0.17 sec)
```

2. Mostrar tablas existentes:

- Se deben listar todas las tablas en la base de datos actual con el siguiente comando:

```
SHOW TABLES;
```

```
mysql> SHOW TABLES;
+-----+
| Tables_in_mi_base_de_datos |
+-----+
| empleados                   |
+-----+
1 row in set (0.00 sec)

mysql>
```

3. Modificar la estructura de la tabla:

- El estudiante debe agregar una columna fecha_contratacion a la tabla empleados con el siguiente comando:

```
ALTER TABLE empleados ADD fecha_contratacion DATE;
```

```
mysql> ALTER TABLE empleados ADD fecha_contratacion DATE;
Query OK, 0 rows affected (0.02 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

4. Eliminar una tabla:

- Finalmente, se eliminará la tabla empleados con el siguiente comando:

```
DROP TABLE empleados;
```

```
mysql> DROP TABLE empleados;
Query OK, 1 row affected (0.10 sec)
```

Sección 6: Manipulación de Datos

Objetivo: Insertar, actualizar, consultar y eliminar datos en las tablas.

1. Insertar datos en una tabla:

- El estudiante debe insertar registros en la tabla empleados utilizando las siguientes instrucciones SQL:

```
INSERT INTO empleados (nombre, apellido, salario, fecha_contratacion) VALUES ('Juan', 'Pérez', 3000.00, '2023-01-15');
```

```
INSERT INTO empleados (nombre, apellido, salario, fecha_contratacion) VALUES ('Ana', 'Gómez', 3200.00, '2023-03-22');
```

```
INSERT INTO empleados (nombre, apellido, salario, fecha_contratacion) VALUES ('Luis', 'Martínez', 2800.00, '2023-02-10');
```

```
mysql> INSERT INTO empleados (nombre, apellido, salario, fecha_contratacion) VALUES ('juan', 'perez', 3000.00, '2023-01-15');
Query OK, 1 row affected (0.01 sec)

mysql> INSERT INTO empleados (nombre, apellido, salario, fecha_contratacion) VALUES ('ana', 'gomez', 3200.00, '2023-03-22');
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO empleados (nombre, apellido, salario, fecha_contratacion) VALUES ('luis', 'martinez', 2800.00, '2023-02-10');
Query OK, 1 row affected (0.01 sec)

mysql> |
```

- Se deben recuperar todos los registros de la tabla empleados con el siguiente comando:

```
SELECT * FROM empleados;
```

```
mysql> SELECT * FROM empleados;
+----+-----+-----+-----+
| id | nombre | apellido | salario | fecha_contratacion |
+----+-----+-----+-----+
| 1  | juan   | perez   | 3000.00 | 2023-01-15      |
| 2  | ana    | gomez   | 3200.00 | 2023-03-22      |
| 3  | luis   | martinez | 2800.00 | 2023-02-10      |
+----+-----+-----+-----+
3 rows in set (0.00 sec)
```

- El estudiante también debe filtrar los registros donde el salario es mayor a 3000:

```
SELECT * FROM empleados WHERE salario > 3000;
```

```
mysql> SELECT * FROM empleados WHERE salario > 3000;
+----+-----+-----+-----+
| id | nombre | apellido | salario | fecha_contratacion |
+----+-----+-----+-----+
| 2  | ana   | gomez   | 3200.00 | 2023-03-22      |
+----+-----+-----+-----+
1 row in set (0.00 sec)

mysql> |
```

3. Actualizar registros:

- El estudiante debe aumentar el salario de Ana Gómez a 3500 con el siguiente comando:

```
UPDATE empleados SET salario = 3500.00 WHERE nombre = 'Ana' AND apellido = 'Gómez';
```

(volver a ejecutar query SELECT * FROM empleados para ver el registro en la tabla actualizada)

```
mysql> UPDATE empleados SET salario = 3500.00 WHERE nombre = 'ana' AND apellido = 'gomez';
Query OK, 1 row affected (0.01 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> SELECT * FROM empleados;
+----+-----+-----+-----+-----+
| id | nombre | apellido | salario | fecha_contratacion |
+----+-----+-----+-----+-----+
| 1  | juan   | perez   | 3000.00 | 2023-01-15      |
| 2  | ana    | gomez   | 3500.00 | 2023-03-22      |
| 3  | luis   | martinez | 2800.00 | 2023-02-10      |
+----+-----+-----+-----+-----+
3 rows in set (0.00 sec)

mysql> |
```

4. Eliminar registros:

- Finalmente, se eliminará el registro de Luis Martínez con el siguiente comando:

```
DELETE FROM empleados WHERE nombre = 'Luis' AND apellido = 'Martínez';
```

(volver a ejecutar query SELECT * FROM empleados para verificar que el registro fue eliminado de la tabla correctamente)

```
mysql> DELETE FROM empleados WHERE nombre= 'luis' AND apellido= 'martinez';
Query OK, 1 row affected (0.01 sec)

mysql> SELECT * FROM empleados;
+----+-----+-----+-----+-----+
| id | nombre | apellido | salario | fecha_contratacion |
+----+-----+-----+-----+-----+
| 1  | juan   | perez   | 3000.00 | 2023-01-15      |
| 2  | ana    | gomez   | 3500.00 | 2023-03-22      |
+----+-----+-----+-----+-----+
2 rows in set (0.00 sec)

mysql> |
```

CONCLUSION

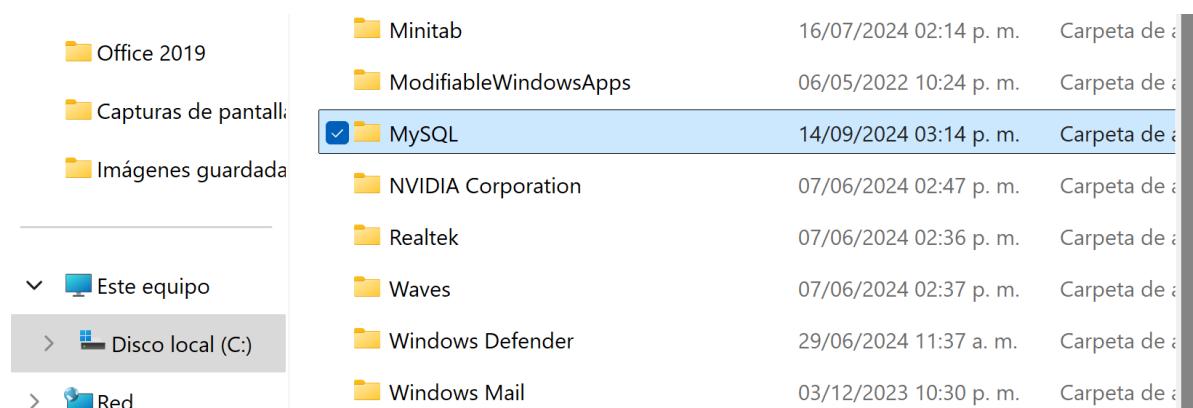
Es interesante empezar a navegar a través de SGBD ya que facilitara a futuro en multiples tareas en las que se requiera tener el control de datos o conjuntos.

El creo que será empezar a familiarizarse con los query o comandos que utilizamos en cada práctica, además de que estamos empezando de lo más básico a un nivel más avanza, por eso es importante no atrasarse y evitar quedarse con alguna duda, solo es cuestión de comprensión, práctica y dedicación.

Un desafío, o mas un problema que se presento en algunas computadoras fue que al momento de ejecutar MySQL desde la terminal cmd ya que se tenia que agregar al pad que posteriormente lo reconociera.

Se soluciono de la siguiente manera:

Abrir el explorador de archivos, dirigirse al disco local (C) y abrir la carpeta de I programa MySQL.



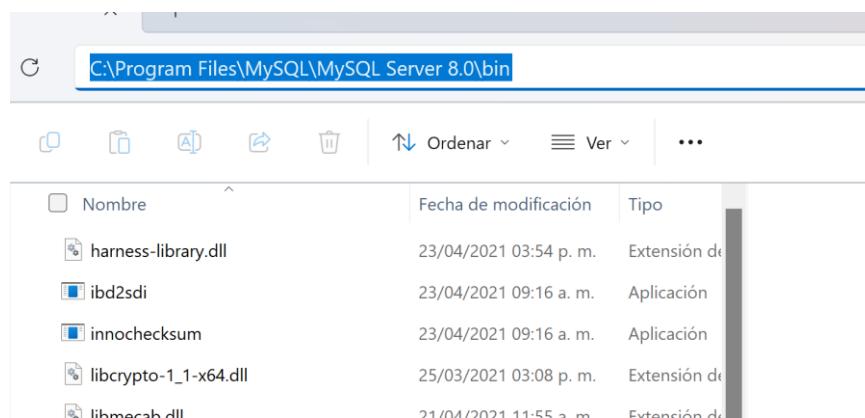
Se observa lo instalado dentro de MySQL y abrimos la carpeta MySQL Server

Nombre	Fecha de modificación	Tipo
MySQL Server 8.0	14/09/2024 03:13 p. m.	Carpeta de archivo
MySQL Shell 8.0	14/09/2024 03:14 p. m.	Carpeta de archivo

Seleccionamos la carpeta de binarios “bin”

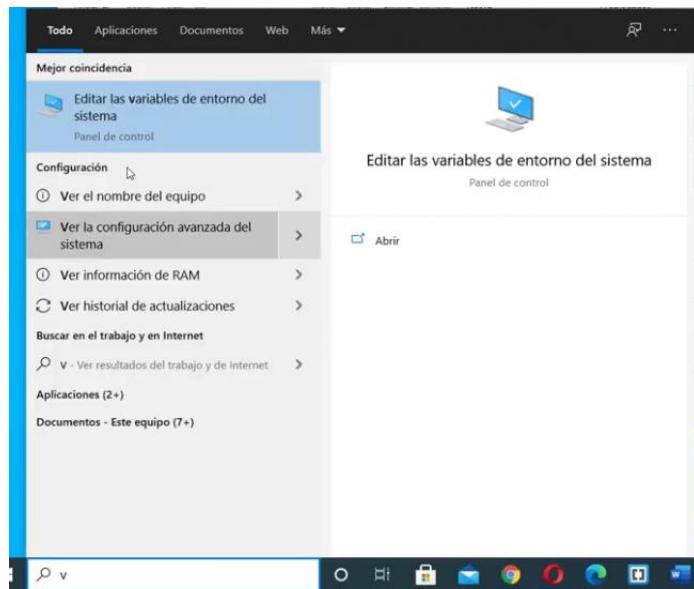
Nombre	Fecha de modificación	Tipo
bin	14/09/2024 03:13 p. m.	Carpeta de arch
dove	14/09/2024 03:12 p. m.	Carpeta de arch

Abrimos la carpeta y bin y seleccionamos y copiamos la ruta que se ha seguido

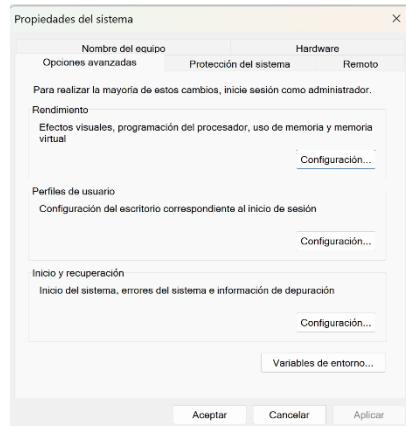


C:\Program Files\MySQL\MySQL Server 8.0\bin

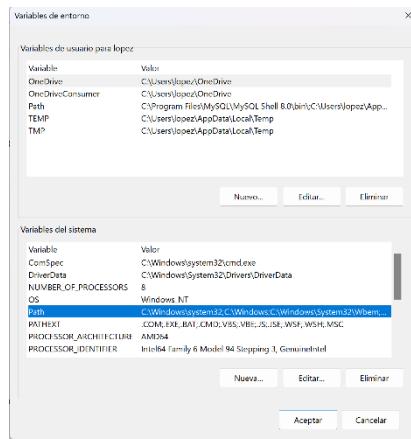
Buscar en la computadora la variable de autentificación, se coloca una V en el buscador y aparece como “Editar las variables de entorno del sistema”, doble clic para ejecutar.



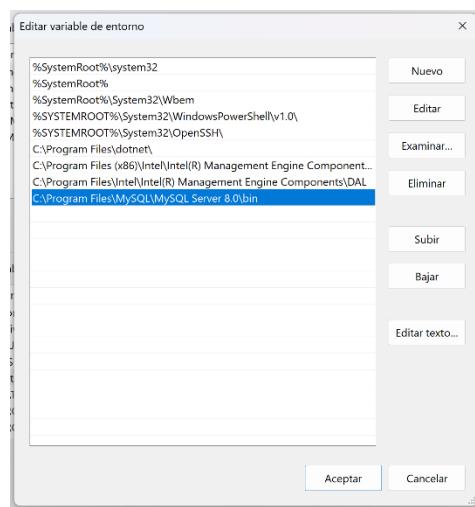
Se ejecuta y aparece una ventana llamada “propiedades del sistema”



Seleccionamos la opción “variables de entorno” que se encuentra en la parte inferior derecha.



Se abrirá otra ventana y seleccionamos la opción de “nuevo” y pegamos la ruta que copiamos anteriormente, una vez pegada pulsa “aceptar” y “aceptar”.



CODIGO COMPLETO

```
C:\Windows\system32\cmd.e: X + ▾

Microsoft Windows [Versión 10.0.22631.4111]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\lopez>mysql -V
mysql Ver 8.0.25 for Win64 on x86_64 (MySQL Community Server - GPL)

C:\Users\lopez>mysql -u root -p
Enter password: *****
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 15
Server version: 8.0.25 MySQL Community Server - GPL

Copyright (c) 2000, 2021, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> CREATE USER 'nuevo_usuario'@'localhost' IDENTIFIED BY 'contraseña_segura';
Query OK, 0 rows affected (0.01 sec)

mysql> GRANT ALL PRIVILEGES ON *.* TO 'nuevo_usuario'@'localhost' WITH GRANT OPTION;
ERROR 1610 (42000): You are not allowed to create a user with GRANT
mysql> GRANT ALL PRIVILEGES ON *.* TO 'nuevo_usuario'@'localhost' WITH GRANT OPTION;
Query OK, 0 rows affected (0.01 sec)

mysql> FLUSH PRIVILEGES;
Query OK, 0 rows affected (0.01 sec)

mysql> CREATE DATABASE;
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near '' at line 1
mysql> CREATE DATABASE mi_base_de_datos;
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near 'DATA DATABASE mi_base_de_datos' at line 1
mysql> CREATE DATABASE mi_base_de_datos;
Query OK, 1 row affected (0.01 sec)

mysql> SHOW DATABASES;
+-----+
| Database |
+-----+
| information_schema |
| mi_base_de_datos |
| mysql |
| performance_schema |
| sakila |
| sys |
| world |
+-----+
7 rows in set (0.00 sec)

mysql> USE mi_base_de_datos;
Database changed
mysql> DROP DATABASE mi_base_de_datos;
Query OK, 0 rows affected (0.01 sec)

mysql> CREATE TABLE empleados (
    >>> id INT AUTO_INCREMENT PRIMARY KEY,
    >>> nombre VARCHAR(50),
    >>> apellido VARCHAR(50),
    >>> salario DECIMAL(10, 2)
    >> );
ERROR 1046 (3D000): No database selected
mysql> CREATE TABLE empleados (
    >>> id INT AUTO_INCREMENT PRIMARY KEY,
    >>> nombre VARCHAR(50),
    >>> apellido VARCHAR(50),
    >>> salario DECIMAL(10, 2)
    >> );
ERROR 1046 (3D000): No database selected
mysql> CREATE DATABASE mi_base_de_datos;
Query OK, 1 row affected (0.01 sec)

mysql> USE mi_base_de_datos;
Database changed
mysql> CREATE TABLE empleados (
    >>> id INT AUTO_INCREMENT PRIMARY KEY,
    >>> nombre VARCHAR(50),
    >>> apellido VARCHAR(50),
    >>> salario DECIMAL(10, 2)
    >> );
Query OK, 0 rows affected (0.03 sec)

mysql> SHOW TABLES;
+-----+
| Tables_in_mi_base_de_datos |
+-----+
| empleados |
+-----+
1 row in set (0.00 sec)

mysql> ALTER TABLE empleados;
Query OK, 0 rows affected (0.01 sec)

mysql> ALTER TABLE empleados ADD fecha_contratacion DATE;
Query OK, 0 rows affected (0.02 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> INSERT INTO empleados (nombre, apellido, salario, fecha_contratacion) VALUES ('juan', 'perez', 3000.00, '2023-01-15');
Query OK, 1 row affected (0.01 sec)

mysql> INSERT INTO empleados (nombre, apellido, salario, fecha_contratacion) VALUES ('ana', 'gomez', 3200.00, '2023-03-22');
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO empleados (nombre, apellido, salario, fecha_contratacion) VALUES ('luis', 'martinez', 2800.00, '2023-02-10');
Query OK, 1 row affected (0.01 sec)

mysql> SELECT * FROM empleados;
+----+-----+-----+-----+
| id | nombre | apellido | salario | fecha_contratacion |
+----+-----+-----+-----+
| 1 | juan | perez | 3000.00 | 2023-01-15 |
| 2 | ana | gomez | 3200.00 | 2023-03-22 |
| 3 | luis | martinez | 2800.00 | 2023-02-10 |
+----+-----+-----+-----+
3 rows in set (0.00 sec)

mysql> SELECT * FROM empleados WHERE salario > 3000;
+----+-----+-----+-----+
| id | nombre | apellido | salario | fecha_contratacion |
+----+-----+-----+-----+
| 2 | ana | gomez | 3200.00 | 2023-03-22 |
+----+-----+-----+-----+
1 row in set (0.00 sec)

mysql> UPDATE empleados SET salario = 3500.00 WHERE nombre = 'ana' AND apellido = 'gomez';
Query OK, 1 row affected (0.01 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> SELECT * FROM empleados;
+----+-----+-----+-----+
| id | nombre | apellido | salario | fecha_contratacion |
+----+-----+-----+-----+
| 1 | juan | perez | 3000.00 | 2023-01-15 |
| 2 | ana | gomez | 3500.00 | 2023-03-22 |
| 3 | luis | martinez | 2800.00 | 2023-02-10 |
+----+-----+-----+-----+
3 rows in set (0.00 sec)

mysql> DELETE FROM empleados WHERE nombre = 'luis' AND apellido = 'martinez';
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near ''martinez'' at line 1
mysql> DELETE FROM empleados WHERE nombre= 'luis' AND apellido= 'martinez';
Query OK, 1 row affected (0.01 sec)

mysql> SELECT * FROM empleados;
+----+-----+-----+-----+
| id | nombre | apellido | salario | fecha_contratacion |
+----+-----+-----+-----+
| 1 | juan | perez | 3000.00 | 2023-01-15 |
| 2 | ana | gomez | 3500.00 | 2023-03-22 |
+----+-----+-----+-----+
2 rows in set (0.00 sec)

mysql>
```



INSTITUTO TECNOLÓGICO DE TIJUANA



TECNOLOGIAS DE BASE DE DATOS

Unidad 1
Sistemas Gestores de Base de Datos

Practica 2

Gestión Avanzada de Tablas y Consultas en MySQL

PRESENTA:
Camacho Lopez Valeria

C22211475

DOCENTE:
Fortunato Ramírez Arzate

TIJUANA, B. C, A 24 DE SEPTIEMBRE DEL 2024

INTRODUCCIÓN

El objetivo de esta práctica es que el estudiante adquiera una comprensión más profunda de la creación y gestión de múltiples tablas en MySQL, así como de la realización de consultas avanzadas utilizando funciones y operaciones SQL.

Sección 1: Creación de Tablas con Relaciones

Objetivo de la Sección:

El estudiante aprenderá a crear tablas que contengan relaciones mediante el uso de claves foráneas, y a insertar datos en dichas tablas.

1. Crear una nueva base de datos:

- Crear y seleccionar la base de datos en la que trabajará el estudiante.

```
CREATE DATABASE empresa;
USE empresa;
```

```
MySQL 8.0 Command Line Client - Unicode
Enter password: *****
welcome to the MySQL monitor. Commands end with ; or \g.
your MySQL connection id is 28
Server version: 8.0.39 MySQL Community Server - GPL
copyright (c) 2000, 2024, oracle and/or its affiliates.
oracle is a registered trademark of oracle corporation and/or its
affiliates. other names may be trademarks of their respective
owners.
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.
mysql> CREATE DATABASE empresa;
Query OK, 1 row affected (0.03 sec)

mysql> SHOW DATABASES;
+-----+
| Database |
+-----+
| empresa |
| information_schema |
| mi_base_de_datos |
| mysql |
| performance_schema |
| sakila |
| sys |
| world |
+-----+
8 rows in set (0.00 sec)

mysql> USE empresa;
Database changed
```

2. Crear una tabla llamada departamentos:

- Diseñar la estructura de una tabla para almacenar los departamentos de una empresa.

```
CREATE TABLE departamentos (
    id INT AUTO_INCREMENT PRIMARY KEY,
    nombre VARCHAR(50) NOT NULL,
    ubicacion VARCHAR(100)
);
```

```
mysql> CREATE TABLE departamentos(
-> id INT AUTO_INCREMENT PRIMARY KEY,
-> nombre VARCHAR(50) NOT NULL,
-> ubicacion VARCHAR(100)
-> );
query ok, 0 rows affected (0.14 sec)
```

3. Crear una tabla llamada empleados:

- Crear una tabla para almacenar los empleados, vinculada a la tabla departamentos mediante una clave foránea.

```
CREATE TABLE empleados (
    id INT AUTO_INCREMENT PRIMARY KEY,
    nombre VARCHAR(50) NOT NULL,
    apellido VARCHAR(50) NOT NULL,
    salario DECIMAL(10,2),
    departamento_id INT,
    FOREIGN KEY (departamento_id) REFERENCES departamentos(id)
);
```

```
mysql> CREATE TABLE empleados(
-> id INT AUTO_INCREMENT PRIMARY KEY,
-> nombre VARCHAR(50) NOT NULL,
-> apellido VARCHAR(50) NOT NULL,
-> salario DECIMAL (10,2),
-> departamento_id INT,
-> FOREIGN KEY(departamento_id) REFERENCES departamentos(id)
-> );
Query OK, 0 rows affected (0.17 sec)
```

4. Insertar datos en ambas tablas:

- Insertar registros en las tablas departamentos y empleados.

```
INSERT INTO departamentos (nombre, ubicacion) VALUES ('Recursos Humanos', 'Planta 1');

INSERT INTO departamentos (nombre, ubicacion) VALUES ('Ventas', 'Planta 2');

INSERT INTO departamentos (nombre, ubicacion) VALUES ('IT', 'Planta 3');

INSERT INTO empleados (nombre, apellido, salario, departamento_id) VALUES ('Ana', 'López', 45000, 1);

INSERT INTO empleados (nombre, apellido, salario, departamento_id) VALUES ('Juan', 'Pérez', 50000, 2);

INSERT INTO empleados (nombre, apellido, salario, departamento_id) VALUES ('Carlos', 'García', 60000, 3);
```

```
mysql> INSERT INTO departamentos (nombre, ubicacion) VALUES('recursos humanos','planta 1');
Query OK, 1 row affected (0.11 sec)

mysql> INSERT INTO departamentos (nombre, ubicacion) VALUES('ventas','planta 2');
Query OK, 1 row affected (0.10 sec)

mysql> INSERT INTO departamentos (nombre, ubicacion) VALUES('IT','planta 3');
Query OK, 1 row affected (0.01 sec)

mysql> INSERT INTO empleados(nombre,apellido,salario,departamento_id)VALUES ('ana','lopez',45000, 1);
Query OK, 1 row affected (0.09 sec)

mysql> INSERT INTO empleados(nombre,apellido,salario,departamento_id)VALUES ('juan','perez',50000, 2);
Query OK, 1 row affected (0.09 sec)

mysql> INSERT INTO empleados(nombre,apellido,salario,departamento_id)VALUES ('carlos','garcia',60000, 3);
Query OK, 1 row affected (0.03 sec)
```

Sección 2: Consultas con Múltiples Tablas (JOINS)

Objetivo de la Sección:

El estudiante aprenderá a realizar consultas que combinen datos de múltiples tablas mediante el uso de JOIN.

1. Mostrar los empleados junto con el nombre de su departamento:

- Realizar una consulta que relacione las tablas empleados y departamentos usando JOIN.

```
SELECT      empleados.nombre,      empleados.apellido,      departamentos.nombre      AS  
departamento FROM empleados JOIN departamentos ON empleados.departamento_id =  
departamentos.id;
```

nombre	apellido	departamento
ana	lopez	recursos humanos
juan	perez	ventas
carlos	garcia	IT

3 rows in set (0.00 sec)

2. Consultar empleados de un departamento específico:

- Consultar solo los empleados que pertenecen al departamento de IT.

```
SELECT empleados.nombre, empleados.apellido  
FROM empleados  
JOIN departamentos ON empleados.departamento_id = departamentos.id  
WHERE departamentos.nombre = 'IT';
```

(se realizó la consulta por separado para visualizar los empleados que corresponden a cada departamento)

nombre	apellido	departamento
diego	perez	ventas
valeria	camacho	ventas

2 rows in set (0.00 sec)

nombre	apellido	departamento
ana	lopez	recursos humanos
emilia	silva	recursos humanos
rafael	herrera	recursos humanos

3 rows in set (0.00 sec)

nombre	apellido	departamento
carlos	garcia	IT
cristina	lopez	IT

2 rows in set (0.00 sec)

Sección 3: Uso de Funciones de Agregación

Objetivo de la Sección:

El estudiante utilizará funciones de agregación como COUNT, AVG, MAX, y MIN para generar estadísticas sobre los datos almacenados.

1. Contar cuántos empleados hay en cada departamento:

- Usar COUNT para contar registros agrupados por departamento.

```
SELECT departamentos.nombre, COUNT(empleados.id) AS num_empleados
FROM empleados
JOIN departamentos ON empleados.departamento_id = departamentos.id
GROUP BY departamentos.nombre;
```

nombre	num_empleados
recursos humanos	3
ventas	2
IT	2

2. Calcular el salario promedio por departamento:

- Usar AVG para calcular el salario promedio por departamento.

```
SELECT departamentos.nombre, AVG(empleados.salario) AS salario_promedio
FROM empleados
JOIN departamentos ON empleados.departamento_id = departamentos.id
GROUP BY departamentos.nombre;
```

nombre	salario_promedio
recursos humanos	55000.000000
ventas	60000.000000
IT	62500.000000

3. Obtener el salario máximo y mínimo de los empleados :

- Usar MAX y MIN para determinar el salario más alto y más bajo.

```
SELECT MAX(salario) AS salario_maximo, MIN(salario) AS salario_minimo
FROM empleados;
```

salario_maximo	salario_minimo
70000.00	45000.00

Sección 4: Modificación de Tablas

Objetivo de la sección:

El estudiante aprenderá a modificar la estructura de las tablas agregando nuevas columnas y actualizando los datos existentes.

1. Agregar una columna llamada `fecha_contratacion` a la tabla `empleados`:

- Agregue una nueva columna para almacenar la fecha de contratación de cada empleado.

```
ALTER TABLE empleados ADD COLUMN fecha_contratacion DATE;
```

```
mysql> ALTER TABLE empleados ADD COLUMN fecha_contratacion DATE;
Query OK, 0 rows affected (0.14 sec)
```

2. Actualizar datos en la nueva columna `fecha_contratacion`:

- Incluir la fecha de contratación para algunos empleados.

```
UPDATE empleados SET fecha_contratacion = '2020-05-10' WHERE nombre = 'Ana';
UPDATE empleados SET fecha_contratacion = '2019-07-22' WHERE nombre = 'Juan';
UPDATE empleados SET fecha_contratacion = '2021-01-15' WHERE nombre = 'Carlos';
```

```
mysql> UPDATE empleados SET fecha_contratacion = '2020-05-10' WHERE nombre = 'Ana';
Query OK, 0 rows affected (0.00 sec)
Rows matched: 1  Changed: 0  Warnings: 0

mysql> UPDATE empleados SET fecha_contratacion = '2019-07-22' WHERE nombre = 'Juan';
Query OK, 0 rows affected (0.00 sec)
Rows matched: 1  Changed: 0  Warnings: 0
```

```
mysql> UPDATE empleados SET fecha_contratacion = '2021-01-15' WHERE nombre = 'Carlos';
Query OK, 0 rows affected (0.00 sec)
Rows matched: 1  Changed: 0  Warnings: 0
```

```
mysql> SELECT * FROM empleados;
```

id	nombre	apellido	salario	departamento_id	fecha_contratacion
2	ana	lopez	45000.00	1	2020-05-10
3	juan	perez	50000.00	1	2019-07-22
4	carlos	garcia	60000.00	1	2021-01-15
5	valeria	camacho	65000.00	2	2020-09-16
6	rafael	herrera	55000.00	2	2023-07-14
7	emilia	silva	57000.00	2	2021-03-18
8	ximena	solis	40000.00	3	2018-11-25
9	anette	vidal	48000.00	3	2019-05-23
10	fernanda	almazan	54000.00	3	2024-10-07
12	Pedro	Sánchez	65000.00	2	NULL

```
10 rows in set (0.00 sec)
```

Sección 5: Consultas Condicionales y Ordenación de Datos

Objetivo de la Sección:

El estudiante realizará consultas condicionales utilizando WHERE y aprenderá a ordenar los resultados con ORDER BY.

1. Consultar empleados contratados después del 1 de enero de 2020:

- Utilizar la cláusula WHERE para obtener resultados basados en una condición.

```
SELECT nombre, apellido, fecha_contratacion
FROM empleados
WHERE fecha_contratacion > '2020-01-01';
```

```
mysql> SELECT nombre, apellido, fecha_contratacion FROM empleados WHERE fecha_contratacion > '2020-01-01';
+-----+-----+-----+
| nombre | apellido | fecha_contratacion |
+-----+-----+-----+
| ana   | lopez   | 2020-05-10
| carlos | garcia  | 2021-01-15
| valeria | camacho | 2020-09-16
| rafael | herrera | 2023-07-14
| emilia | silva   | 2021-03-18
| fernanda | almazan | 2024-10-07
+-----+-----+-----+
6 rows in set (0.05 sec)
```

2. Ordenar los empleados por salario en orden descendente:

- Utilizar ORDER BY para ordenar los resultados de mayor a menor.

```
SELECT nombre, apellido, salario
FROM empleados
ORDER BY salario DESC;
```

```
mysql> SELECT nombre, apellido, salario FROM empleados ORDER BY salario DESC;
+-----+-----+-----+
| nombre | apellido | salario |
+-----+-----+-----+
| valeria | camacho | 65000.00
| Pedro | Sánchez | 65000.00
| carlos | garcia | 60000.00
| emilia | silva | 57000.00
| rafael | herrera | 55000.00
| fernanda | almazan | 54000.00
| juan | perez | 50000.00
| anette | vidal | 48000.00
| ana | lopez | 45000.00
| ximena | solis | 40000.00
+-----+-----+-----+
10 rows in set (0.00 sec)
```

Sección 6: Eliminar Datos y Tablas

Objetivo de la Sección:

El estudiante aprenderá a eliminar registros específicos y tablas completas en MySQL.

1. Eliminar un empleado de la tabla empleados:

- o Borrar un registro específico de la tabla.

```
DELETE FROM empleados WHERE nombre = 'Carlos';
```

```
mysql> DELETE FROM empleados WHERE nombre = 'Carlos';
Query OK, 1 row affected (0.09 sec)
```

```
mysql> SELECT * FROM empleados;
```

id	nombre	apellido	salario	departamento_id	fecha_contratacion
2	ana	lopez	45000.00	1	2020-05-10
3	juan	perez	50000.00	1	2019-07-22
5	valeria	camacho	65000.00	2	2020-09-16
6	rafael	herrera	55000.00	2	2023-07-14
7	emilia	silva	57000.00	2	2021-03-18
8	ximena	solis	40000.00	3	2018-11-25
9	anette	vidal	48000.00	3	2019-05-23
10	fernanda	almazan	54000.00	3	2024-10-07
12	Pedro	Sánchez	65000.00	2	NULL

```
9 rows in set (0.00 sec)
```

2. Eliminar la tabla empleados:

- o Eliminar toda la tabla empleados.

```
DROP TABLE empleados;
```

```
mysql> DROP TABLE empleados;
Query OK, 0 rows affected (0.14 sec)

mysql> SELECT * FROM empleados;
ERROR 1146 (42S02): Table 'empresa.empleados' doesn't exist
mysql> -
```

Se eliminó la tabla de empleados y posteriormente verificamos que haya sido eliminada correctamente ejecutando el query `SELECT * FROM empleados;` arrojando un error ya que dicha tabla ya no existe dentro de la base de datos creada.

CONCLUSION

Se realizó con éxito la práctica de gestión avanzada de tablas y consultas en MySQL con ayuda de nuevos comandos que complementaron los query básicos vistos anteriormente, fue interesante ya que nos demostró con la práctica constante que se pueden realizar diversas combinaciones personalizadas de acuerdo a lo que el usuario necesita para realizar alguna tarea o consulta más elaborada.



INSTITUTO TECNOLÓGICO DE TIJUANA



TECNOLOGIAS DE BASE DE DATOS

Unidad 1
Sistemas Gestores de Base de Datos

Practica 3

Consultas Avanzadas y Optimización en MySQL

PRESENTA:
Camacho Lopez Valeria

C22211475

DOCENTE:
Fortunato Ramírez Arzate

TIJUANA, B. C, A 4 DE OCTUBRE DEL 2024

INTRODUCCIÓN

Conceptos generales

- Índices:** Un índice en MySQL es una estructura de datos que mejora la velocidad de las operaciones de selección en una tabla a costos de una mayor utilización de espacio y tiempo durante las operaciones de inserción y actualización. Los índices funcionan como un índice de un libro, facilitando la búsqueda rápida de registros en una tabla.

Diagrama de Concepto de Índices:

Tabla sin índice:			Tabla con índice:		
ID	Nombre	Apellido	ID	Nombre	Apellido
1	Juan	Pérez	1	Juan	Pérez
2	Ana	Gómez	2	Ana	Gómez
3	Luis	Martínez	3	Luis	Martínez

Los índices permiten encontrar rápidamente valores como Pérez al organizar los datos internamente.

- Subconsultas:** Las subconsultas son consultas SQL anidadas dentro de otras consultas. Se utilizan para obtener resultados intermedios que luego se utilizan en la consulta externa.

Diagrama de Concepto de Subconsultas:

Consulta externa:
SELECT * FROM empleados WHERE salario > (...);

Subconsulta:
(SELECT AVG(salario) FROM empleados)

La subconsulta calcula un valor (el salario promedio en este caso) que luego es utilizado por la consulta externa para obtener los empleados cuyo salario es mayor que el promedio.

- Vistas:** Una vista en MySQL es una tabla virtual que resulta de una consulta. No almacena datos básicamente, pero facilita la reutilización de consultas complejas y permite organizar mejor las consultas.

Diagrama de Concepto de Vistas:

```
Vista creada:  
CREATE VIEW empleados_vista  
departamentos.nombre  
AS SELECT ...  
....;  
  
Consulta a la vista:  
SELECT * FROM empleados_vista;
```

Consulta original:

```
SELECT empleados.nombre,  
FROM empleados JOIN departamentos ON
```

La vista permite simplificar consultas repetitivas.

4. **Transacciones:** Una transacción en MySQL es un conjunto de operaciones SQL que se ejecutan como una unidad. Si alguna de las operaciones falla, todas las demás se revierten, garantizando que la base de datos quede en un estado coherente.

Diagrama de Concepto de Transacciones:

```
START TRANSACTION;          -- Inicia la transacción  
INSERT INTO empleados ...;  -- Operación 1  
INSERT INTO empleados ...;  -- Operación 2  
  
Si todo es correcto:  
COMMIT;                   -- Confirma los cambios  
  
Si hay un error:  
ROLLBACK;                 -- Revierte todos los cambios
```

Las transacciones permiten garantizar la integridad de los datos.

5. **Optimización de Consultas (EXPLAIN):** La instrucción EXPLAIN en MySQL se utiliza para analizar cómo se ejecuta una consulta y sugerir mejoras, como el uso de índices.

Diagrama de Concepto de EXPLAIN:

Consulta: SELECT * FROM empleados WHERE salario > 50000;

Salida de EXPLAIN:

id	select_type	table	type	possible_keys	key	rows
1	SIMPLE	empleados	ALL	NULL	NULL	10000

EXPLAIN muestra cómo MySQL procesa una consulta y ayuda a encontrar áreas de mejora, como la adición de índices.

6. **Manejo de errores:** Los errores en MySQL pueden ocurrir por sintaxis incorrecta o por intentos de ejecutar operaciones no permitidas. Al aprender a gestionar errores, los estudiantes pueden depurar sus consultas y mejorar su código SQL.

Diagrama de errores comunes:

Consulta con error:

```
SELECT * FORM empleados;
```

Error generado:

```
Error: You have an error in your SQL syntax...
```

Consulta corregida:

```
SELECT * FROM empleados;
```

Aprender a identificar y corregir errores en consultas es fundamental para la depuración.

Objetivo General de la Práctica 3:

El objetivo de esta práctica es que el estudiante explore técnicas avanzadas de MySQL, desde el uso de índices y subconsultas hasta la optimización de consultas y el manejo de transacciones, aplicando estos conceptos para gestionar bases de datos de manera eficiente.

Sección 1: Uso de índices

Objetivo: Mejorar el rendimiento de las consultas utilizando índices en MySQL.

1. Crear un índice en la tabla `empleados`:

- El estudiante debe crear un índice en la columna `apellido` para acelerar las consultas que la involucren:

```
CREATE INDEX idx_apellido ON empleados(apellido);
```

```
mysql> CREATE INDEX idx_apellido ON empleados(apellido);
Query OK, 0 rows affected (0.22 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

2. Consultar utilizando el índice:

- Realizar una consulta que aproveche el índice creado para encontrar empleados por su apellido:

```
SELECT * FROM empleados WHERE apellido = 'Pérez';
```

```
mysql> SELECT * FROM empleados WHERE apellido = 'perez';
+----+-----+-----+-----+-----+
| id | nombre | apellido | salario | departamento_id | fecha_contratacion |
+----+-----+-----+-----+-----+
| 3 | juan | perez | 50000.00 | 1 | 2019-07-22 |
+----+-----+-----+-----+-----+
1 row in set (0.04 sec)
```

3. Mostrar los índices de una tabla:

- El estudiante debe listar los índices disponibles en la tabla empleados:

```
SHOW INDEX FROM empleados;
```

```
mysql> SHOW INDEX FROM empleados;
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| Table | Non_unique | Key_name | Seq_in_index | Column_name | Collation | Cardinality | Sub_part | Packed | Null | Index_type | Comment |
|       |           |          |             |             |           |           |           |           |           |           |           |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| empleados | 0 | PRIMARY | 1 | id | A | 9 | NULL | NULL | BTREE | | |
| empleados | 1 | departamento_id | 1 | departamento_id | A | 3 | NULL | NULL | YES | BTREE | |
| empleados | 1 | idx_apellido | 1 | apellido | A | 9 | NULL | NULL | BTREE | |
|       | YES | NULL |           |           |           |           |           |           |           |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
3 rows in set (0.12 sec)
```

Sección 2: Subconsultas

Objetivo: Aprender a utilizar subconsultas para realizar consultas más complejas y obtener resultados más específicos.

1. Subconsulta simple:

- El estudiante debe encontrar los empleados cuyo salario sea mayor al promedio de todos los empleados:

```
SELECT nombre, apellido, salario FROM empleados
WHERE salario > (SELECT AVG(salario) FROM empleados);
```

```
mysql> SELECT nombre, apellido, salario FROM empleados WHERE salario > (SELECT AVG(salario) FROM empleados);
+-----+-----+-----+
| nombre | apellido | salario |
+-----+-----+-----+
| carlos | garcia | 60000.00 |
| valeria | camacho | 65000.00 |
| rafael | herrera | 55000.00 |
| emilia | silva | 57000.00 |
| fernanda | almanza | 54000.00 |
+-----+-----+-----+
5 rows in set (0.05 sec)
```

2. Subconsulta con IN:

- El estudiante debe listar todos los empleados que pertenecen a los departamentos que están en la planta 2:

```
SELECT nombre, apellido FROM empleados WHERE departamento_id IN  
(SELECT id FROM departamentos WHERE ubicacion = 'Planta 2');
```

nombre	apellido
valeria	camacho
rafael	herrera
emilia	silva

Sección 3: Vistas

Objetivo: Facilitar consultas repetitivas y mejorar la organización de consultas complejas mediante la creación de vistas.

1. Crear una vista para mostrar empleados y sus departamentos:

- El estudiante debe crear una vista que combine la información de las tablas empleados y departamentos:

```
CREATE VIEW vista_empleados_departamentos AS SELECT  
empleados.nombre, empleados.apellido, departamentos.nombre AS departamento  
FROM empleados  
JOIN departamentos ON empleados.departamento_id = departamentos.id;
```

nombre	apellido	departamento
ana	lopez	recursos humanos
juan	perez	recursos humanos
carlos	garcia	recursos humanos
valeria	camacho	ventas
rafael	herrera	ventas
emilia	silva	ventas
ximena	solis	it
anette	vidal	it
fernanda	almazan	it

2. Consultar la vista creada:

- El estudiante debe consultar la vista para obtener los datos combinados:

```
SELECT * FROM vista_empleados_departamentos;
```

nombre	apellido	departamento
ana	lopez	recursos humanos
juan	perez	recursos humanos
carlos	garcia	recursos humanos
valeria	camacho	ventas
rafael	herrera	ventas
emilia	silva	ventas
ximena	solis	it
anette	vidal	it
fernanda	almazan	it

1. Modificar una vista:

- El estudiante debe actualizar la vista para incluir el salario de los empleados:

```
CREATE OR REPLACE VIEW vista_empleados_departamentos AS
SELECT empleados.nombre, empleados.apellido, empleados.salario,
departamentos.nombre AS departamento FROM empleados
JOIN departamentos ON empleados.departamento_id =
departamentos.id;
```

```
mysql> CREATE OR REPLACE VIEW vista_empleados_departamentos AS
-> SELECT empleados.nombre, empleados.apellido, empleados.salario, departamentos.nombre AS departamento
-> FROM empleados
-> JOIN departamentos ON empleados.departamento_id = departamentos.id;
Query OK, 0 rows affected (0.01 sec)
```

Sección 4: Transacciones y Control de Integridad

Objetivo: Comprender el uso de transacciones en MySQL para asegurar la consistencia y la integridad de los datos.

1. Iniciar una transacción:

- El estudiante debe iniciar una transacción, realizar una inserción en la tabla `empleados` y luego revertirla:

```
START TRANSACTION;
INSERT INTO empleados (nombre, apellido, salario,
departamento_id) VALUES ('María', 'Fernández', 70000, 1);
ROLLBACK;
```

```
mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO empleados (nombre, apellido, salario, departamento_id) VALUES ('María', 'Fernández', 70000, 1);
Query OK, 1 row affected (0.05 sec)

mysql> ROLLBACK;
Query OK, 0 rows affected (0.09 sec)
```

2. Confirmar cambios usando `COMMIT`:

- El estudiante debe realizar una inserción y confirmar los cambios con `COMMIT`:

```
START TRANSACTION;
INSERT INTO empleados (nombre, apellido, salario,
departamento_id) VALUES ('Pedro', 'Sánchez', 65000, 2);
COMMIT;
```

```
mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO empleados (nombre, apellido, salario, departamento_id) VALUES ('Pedro', 'Sánchez', 65000, 2);
Query OK, 1 row affected (0.00 sec)

mysql> COMMIT;
Query OK, 0 rows affected (0.01 sec)
```

Sección 5: Optimización de Consultas

Objetivo: Mejorar la eficiencia de las consultas mediante la utilización de buenas prácticas y herramientas de MySQL.

1. Uso de EXPLAIN:

- El estudiante debe analizar el plan de ejecución de una consulta para optimizarla:

```
EXPLAIN SELECT * FROM empleados WHERE salario > 50000;
```

```
mysql> EXPLAIN SELECT * FROM empleados WHERE salario > 3100;
+-----+-----+-----+-----+-----+-----+-----+
| id | select_type | table | partitions | type | possible_keys | key | key_len | ref | rows | filtered | Extra |
+-----+-----+-----+-----+-----+-----+-----+
| 1 | SIMPLE      | empleados | NULL      | ALL   | NULL          | NULL | NULL    | NULL | 3    | 33.33  | Using where |
+-----+-----+-----+-----+-----+-----+-----+
1 row in set, 1 warning (0.00 sec)

mysql>
```

El query EXPLAIN indica que se requiere proporcionar detalles y optimiza la información del índice, por eso al momento de ejecutarlo se muestra una tabla con varias columnas que detallan como MySQL ejecutaría la consulta, en este caso solo muestra el proceso que MySQL haría, no muestra datos reales.

2. Optimización de una consulta con índices:

- Cree un índice en la columna `salario` y ejecute de nuevo la consulta para verificar la mejora:

```
CREATE INDEX idx_salario ON empleados(salario);
SELECT * FROM empleados WHERE salario > 50000;
```

```
mysql> CREATE INDEX idx_salario ON empleados(salario);
Query OK, 0 rows affected (0.07 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> SELECT * FROM empleados WHERE salario > 3100;
+-----+-----+-----+-----+-----+
| id | nombre | apellido | salario | fecha_contratacion | rfc |
+-----+-----+-----+-----+-----+
| 2 | emilia | SILVA    | 3200.00 | 2022-03-17        | SILE84537 |
| 4 | emilia | silva    | 3200.00 | 2023-07-25        | NULL       |
| 3 | rafael | herrera  | 3500.00 | 2022-03-17        | RAFHER6443 |
+-----+-----+-----+-----+-----+
3 rows in set (0.01 sec)

mysql>
```

Sección 6: Manejo de Errores y Depuración

Objetivo: Comprender cómo gestionar errores en MySQL y depurar consultas SQL.

1. Captura de errores:

- El estudiante debe ejecutar una consulta incorrecta y observar cómo se maneja el error:

```
SELECT * FORM empleados; -- Error en la palabra FORM
```

```
mysql> SELECT * FORM empleados;
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near 'FORM empleados' at line 1
```

2. Revisión y corrección de errores comunes:

- El estudiante debe identificar el error en la consulta anterior y corregirlo:

```
SELECT * FROM empleados;
```

```
mysql> SELECT * FROM empleados;
+----+-----+-----+-----+-----+
| id | nombre | apellido | salario | fecha_contratacion | rfc      |
+----+-----+-----+-----+-----+
| 1  | valeria | camacho | 3000.00 | 2023-01-15          | CALV040916 |
| 2  | emilia   | SILVA    | 3200.00 | 2022-03-17          | SILE84537  |
| 3  | rafael   | herrera  | 3500.00 | 2022-03-17          | RAFHER6443 |
| 4  | emilia   | silva    | 3200.00 | 2023-07-25          | NULL       |
+----+-----+-----+-----+-----+
4 rows in set (0.00 sec)

mysql> -
```

Es un error común ya que es gramatical y se da en ocasiones que se está escribiendo rápido y no se tiene el cuidado al ejecutar el query, la solución es volver a escribirlo correctamente para que se ejecute adecuadamente y muestre lo que se solicita.

Conclusión

Se realizó una práctica muy didáctica implementando de cierta manera material que no habíamos realizado anteriormente; el uso de los índices nos resultó útil para realizar consultas más rápidas y eficientes ya que nos muestra solo lo que necesitamos dependiendo la tarea o la necesidad, al igual se implementaron algunos query con los que ya estamos familiarizados, pero en esta ocasión para reforzar y analizar el uso de cada uno de ellos.

INSTITUTO TECNOLÓGICO DE TIJUANA



TECNOLOGIAS DE BASE DE DATOS

Unidad 2
Estándares y normas para registros electrónicos en la salud

Practica 4

Interfaz Web y Conexión con MySQL para Gestión de Pacientes

PRESENTA:
Camacho Lopez Valeria

C22211475

DOCENTE:
Fortunato Ramírez Arzate

TIJUANA, B. C, A 8 DE OCTUBRE DEL 2024

Objetivo:

Desarrollar una aplicación web básica con Node.js y MySQL que permita a los estudiantes conectarse a una base de datos, realizar consultas y almacenar información de pacientes desde una interfaz web.

Introducción:

En esta práctica aprenderás a conectar una página web a una base de datos MySQL a través de un servidor Node.js utilizando el framework Express. Vamos a construir una aplicación paso a paso que te permitirá ingresar y consultar información de pacientes y sus signos vitales.

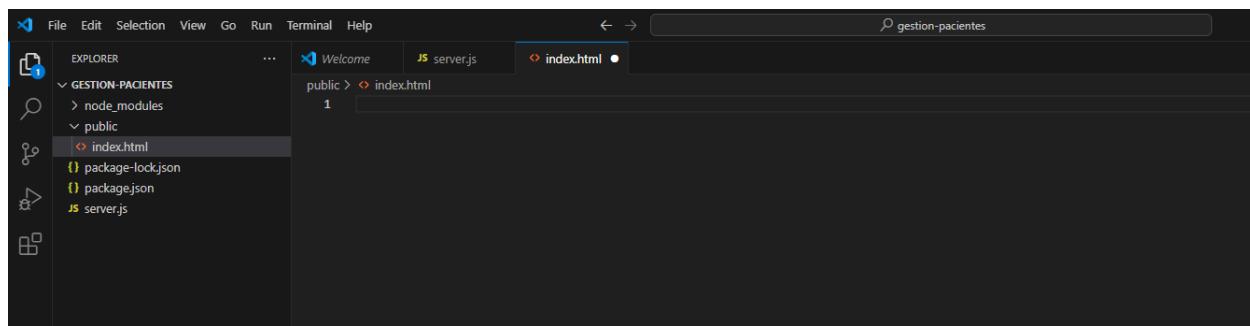
Requisitos:

- Tener instalado Node.js y MySQL.
- Tener un editor de código (VSCode recomendado).
- Instalar las dependencias necesarias (`express`, `mysql2`, `body-parser`).

Paso 1: Crear el servidor y la página web básica

1. Crea la estructura de archivos:

- Crea una carpeta para tu proyecto llamada `gestion-pacientes`.
- Dentro de esta carpeta, crea un archivo llamado `server.js`.
- Crea otra carpeta llamada `public` y dentro de ella, un archivo llamado `index.html`



2. Instala Express y otras dependencias: Abre una terminal en la carpeta del proyecto y ejecuta:

```
npm init -y  
npm install express mysql2 body-parser
```

```

C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.19045.4894]
(c) Microsoft Corporation. All rights reserved.

C:\Users\lopez\Desktop\gestion-pacientes>npm init -y
wrote to C:\Users\lopez\Desktop\gestion-pacientes\package.json:

{
  "name": "gestion-pacientes",
  "version": "1.0.0",
  "main": "server.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "start": "node server.js"
  },
  "keywords": [],
  "author": "Lopez",
  "license": "ISC",
  "description": ""
}

C:\Users\lopez\Desktop\gestion-pacientes>npm install express mysql2 body-parser
added 77 packages, and audited 78 packages in 4s
14 packages are looking for funding
  run `npm fund` for details
2 low severity vulnerabilities
To address all issues, run:
  npm audit fix
Run 'npm audit' for details.

C:\Users\lopez\Desktop\gestion-pacientes>

```

3. Escribe el código básico del servidor: En `server.js`, agrega el siguiente código para crear el servidor Express y servir la página HTML básica:

```

JS serverjs > ...
1 const express = require('express');
2 const app = express();
3 const path = require('path');
4 const bodyParser = require('body-parser');
5
6 app.use(bodyParser.urlencoded({ extended: true }));
7
8 // Servir archivos estáticos (HTML)
9 app.use(express.static(path.join(__dirname, 'public')));
10
11 // Ruta para la página principal
12 app.get('/', (req, res) => {
13   | res.sendFile(path.join(__dirname, 'public', 'index.html'));
14 });
15
16 // Iniciar el servidor
17 app.listen(3000, () => {
18   | console.log(`Servidor corriendo en http://localhost:3000`);
19 });
20

```

4. Escribe el HTML básico: En `public/index.html`, agrega lo siguiente:

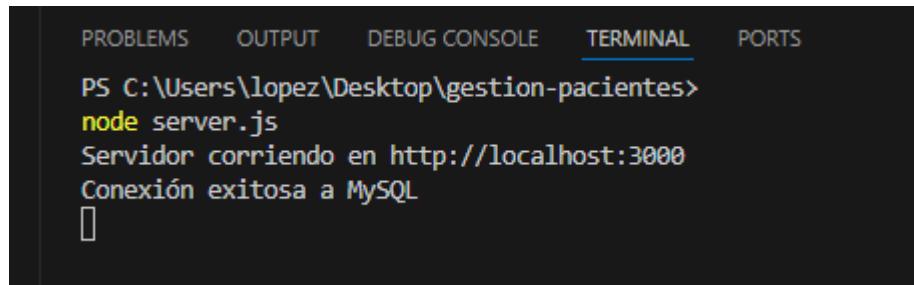
```

public > < index.html > ⚡ html > ⚡ body > ⚡ body > ⚡ form > ⚡ label
1  <!DOCTYPE html>
2  <html lang="es">
3  <head>
4  |  <meta charset="UTF-8">
5  |  <meta name="viewport" content="width=device-width, initial-scale=1.0">
6  |  <link rel="stylesheet" href="styles.css">
7
8  <title>Gestión de Pacientes</title>
9  </head>
10 <body>
11 <h1>Bienvenido a la Gestión de Pacientes</h1>
12 <p>Esta es una página básica para administrar pacientes y sus signos vitales.</p>
13 </body>
14 </body>

```

5. Ejecuta el servidor y prueba:

- En la terminal, ejecuta: node server.js
- Abre un navegador y ve a <http://localhost:3000>. Deberías ver la página con el mensaje de bienvenida.



```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS
PS C:\Users\lopez\Desktop\gestion-pacientes>
node server.js
Servidor corriendo en http://localhost:3000
Conexión exitosa a MySQL
[]
```



Paso 2: Agregar un formulario a la página y procesarlo en el servidor

1. Modificar el HTML para incluir un formulario:

En `public/index.html`, agrega un formulario debajo del texto de bienvenida para registrar a un paciente:

```
14  <form action="/submit-data" method="POST">
15  <label for="name">Nombre del paciente:</label>
16  <input type="text" id="name" name="name">
17
18  <label for="age">Edad:</label>
19  <input type="number" id="age" name="age">
20
21  <label for="heart-rate">Frecuencia Cardiaca (bpm):</label>
22  <input type="number" id="heart-rate" name="heart_rate">
23
24  <label for="weight">Peso (kg):</label>
25  <input type="number" id="weight" name="weight">
26
27  <label for="height">Estatura (cm):</label>
28  <input type="number" id="height" name="height">
29
30  <button type="submit">Guardar</button>
31
32  </form>
```

*Se agregaron más campos de las solicitados para tener un formulario más completo y profesional (nombre del paciente, edad, frecuencia cardiaca, peso y estatura).

2. Agregar la ruta en el servidor para procesar el formulario:

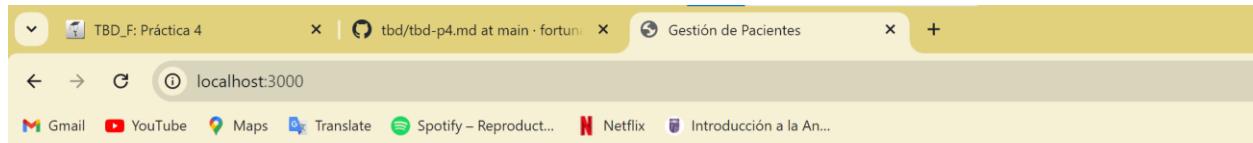
En `server.js`, agrega la ruta para recibir los datos del formulario:

```
// Ruta para procesar el formulario
app.post('/submit-data', (req, res) => {
  const { name, age, heart_rate } = req.body;
  res.send(`Paciente recibido: ${name}, Edad: ${age}, Frecuencia
Cardiaca: ${heart_rate}`);
});

34 // Ruta para procesar el formulario
35 app.post('/submit-data', (req, res) => {
36   const { name, age, heart_rate } = req.body;
37   res.send(`Paciente recibido: ${name}, Edad: ${age}, Frecuencia Cardiaca: ${heart_rate}`);
38 });
```

3. Prueba el formulario:

- Recarga la página en el navegador.
- Introduce los datos de un paciente y envíalos. Deberías ver un mensaje que muestre los datos ingresados.



Bienvenido a la Gestión de Pacientes

Esta es una página básica para administrar pacientes y sus signos vitales.

3

Nombre del paciente: Edad: Frecuencia Cardiaca (bpm): Guardar



Paciente recibido: valeria, Edad: 20, Frecuencia Cardiaca: 93

Paso 3: Conectar el servidor a MySQL

1. Crear una base de datos y tabla en MySQL:

Abre MySQL desde la terminal y ejecuta los siguientes comandos para crear la base de datos y la tabla de pacientes:

```
CREATE DATABASE gestion_pacientes;
USE gestion_pacientes;
CREATE TABLE pacientes (
    id INT AUTO_INCREMENT PRIMARY KEY,
    nombre VARCHAR(100),
    edad INT,
    frecuencia_cardiaca INT
);
```

```
mysql> CREATE DATABASE gestion_pacientes;
Query OK, 1 row affected (0.02 sec)

mysql> USE gestion_pacientes;
Database changed
mysql> CREATE TABLE pacientes (
    ->     id INT AUTO_INCREMENT PRIMARY KEY,
    ->     nombre VARCHAR(100),
    ->     edad INT,
    ->     frecuencia_cardiaca INT
    -> );
Query OK, 0 rows affected (0.05 sec)

mysql>
```

2. Agregar la conexión a MySQL en el servidor:

Modifica `server.js` para conectarse a MySQL y agregar los datos del paciente a la base de datos:

```
const mysql = require('mysql2');

// Configurar conexión a MySQL
const connection = mysql.createConnection({
    host: 'localhost',
    user: 'root',
    password: 'password', // Cambia por tu contraseña
    database: 'gestion_pacientes'
});

connection.connect(err => {
    if (err) {
        console.error('Error conectando a MySQL:', err);
        return;
    }
    console.log('Conexión exitosa a MySQL');
```

```

});;

// Ruta para guardar datos en la base de datos
app.post('/submit-data', (req, res) => {
  const { name, age, heart_rate } = req.body;

  const query = 'INSERT INTO pacientes (nombre, edad,
frecuencia_cardiaca) VALUES (?, ?, ?)';
  connection.query(query, [name, age, heart_rate], (err, result) => {
    if (err) {
      return res.send('Error al guardar los datos en la base de
datos.');
    }
    res.send(`Paciente ${name} guardado en la base de datos.`);
  });
});

```

```

EXPLORER           JS server.js x index.html
GESTION-PACIENTES
  > node_modules
  > public
    > index.html
  package-lock.json
  package.json
  server.js

server.js
  ...
  23 app.post('/submit-data', (req, res) => {
  24   const mysql = require('mysql2');
  25
  26   // Configurar conexión a MySQL
  27   const connection = mysql.createConnection({
  28     host: 'localhost',
  29     user: 'root',
  30     password: 'CALV040916', // Cambia por tu contraseña
  31     database: 'gestion_pacientes'
  32   });
  33
  34   connection.connect(err => {
  35     if (err) {
  36       console.error('Error conectando a MySQL:', err);
  37       return;
  38     }
  39     console.log('Conexión exitosa a MySQL');
  40   });
  41
  42   // Ruta para guardar datos en la base de datos
  43   app.post('/submit-data', (req, res) => {
  44     const { name, age, heart_rate } = req.body;
  45
  46     const query = 'INSERT INTO pacientes (nombre, edad, frecuencia_cardiaca) VALUES (?, ?, ?)';
  47     connection.query(query, [name, age, heart_rate], (err, result) => {
  48       if (err) {
  49         return res.send('Error al guardar los datos en la base de datos.');
  50       }
  51       res.send(`Paciente ${name} guardado en la base de datos.`);
  52     });
  53   });
  54 });

```

3. Prueba la conexión:

- Vuelve a iniciar el servidor con `node server.js`.
- Completa el formulario en la página y verifica que los datos del paciente se guarden en la base de datos.



Paciente alicia guardado en la base de datos.

Paso 4: Consultar los datos desde la base de datos

1. Agregar una nueva ruta para consultar los datos:

En `server.js`, agrega la siguiente ruta para obtener la lista de pacientes guardados:

```
// Ruta para mostrar los datos de la base de datos
app.get('/pacientes', (req, res) => {
  connection.query('SELECT * FROM pacientes', (err, results) => {
    if (err) {
      return res.send('Error al obtener los datos.');
    }
    res.send(results);
  });
});
```

```
34 // Ruta para guardar datos en la base de datos
35 app.post('/submit-data', (req, res) => {
36   const { name, age, heart_rate, weight, height } = req.body;
37
38   const query = 'INSERT INTO pacientes (nombre, edad, frecuencia_cardiaca, peso, estatura) VALUES (?, ?, ?, ?, ?)';
39   connection.query(query, [name, age, heart_rate, weight, height], (err, result) => {
40     if (err) {
41       return res.send('Error al guardar los datos en la base de datos.');
42     }
43     res.send(`Paciente ${name} guardado en la base de datos.`);
44   });
45 });
46 });
47 }
```

2. Prueba la consulta:

- Abre tu navegador y ve a `http://localhost:3000/pacientes`. Deberías ver una lista de los pacientes guardados con su nombre, edad y frecuencia cardíaca.



Paso 5: Guardar y consultar datos desde la página

1. Agregar un botón en el HTML para ver los datos guardados:

En public/index.html, agrega este botón al final:

```
<button onclick="window.location.href='/pacientes'">Ver Pacientes  
Guardados</button>
```

```
<button onclick="window.location.href='/pacientes'">Ver Pacientes Guardados</button>
```

2. **Prueba la funcionalidad completa:** Ahora deberías poder:

- Enviar datos de pacientes desde el formulario y guardarlos en la base de datos.
- Consultar los pacientes guardados usando el botón "Ver Pacientes Guardados".

INSTITUTO TECNOLÓGICO DE TIJUANA



TECNOLOGIAS DE BASE DE DATOS

Unidad 2
Estándares y normas para registros electrónicos en la salud

Practica 5

Estilización con CSS y Visualización de Datos en una Tabla HTML

PRESENTA:
Camacho Lopez Valeria

C22211475

DOCENTE:
Fortunato Ramírez Arzate

TIJUANA, B. C, A 15 DE OCTUBRE DEL 2024

Objetivo:

Aprender a aplicar estilos básicos a una página web usando CSS y visualizar los datos de una base de datos en una tabla HTML estilizada.

Introducción:

En esta práctica, aprenderás a mejorar la apariencia de tu página web utilizando hojas de estilo en cascada (CSS) y a mostrar los datos consultados de la base de datos en una tabla HTML con un diseño más atractivo. Se te guiará paso a paso para aplicar estilos y crear una tabla que muestre la información de los pacientes de manera clara y visualmente agradable.

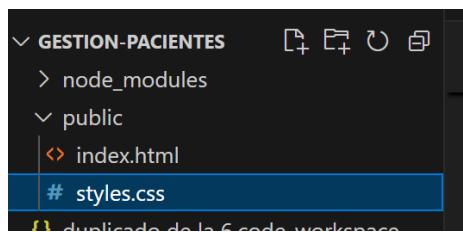
Requisitos:

- Práctica 4 completada.
- Archivos y servidor creados en la carpeta `gestion-pacientes`.
- Tener instalados `Node.js`, `MySQL` y las dependencias (`express`, `mysql2`, `body-parser`).

Paso 1: Crear un archivo CSS para la página web

1. Crea una hoja de estilos CSS:

- Dentro de la carpeta `public`, crea un nuevo archivo llamado `styles.css`.
- Este archivo será el que usemos para estilizar los elementos de la página HTML.



2. Enlazar el archivo CSS a tu página HTML:

Modifica `public/index.html` para que enlace el archivo `styles.css`. Agrega la siguiente línea dentro de la etiqueta `<head>`:

```
<link rel="stylesheet" href="styles.css">
```

```
public > index.html > head
1   <!DOCTYPE html>
2   <html lang="es">
3   <head>
4     <meta charset="UTF-8">
5     <meta name="viewport" content="width=device-width, initial-scale=1.0">
6     <link rel="stylesheet" href="styles.css">
```

3. Prueba la conexión con el archivo CSS:

Agrega algo de estilo básico en `styles.css` para asegurarte de que esté correctamente enlazado:

```
body {  
    font-family: Arial, sans-serif;  
    background-color: #f0f0f0;  
}  
  
h1 {  
    color: #333;  
    text-align: center;  
}
```

Paso 2: Aplicar estilos al formulario de registro de pacientes

1. Estiliza el formulario:

Agrega los siguientes estilos en `styles.css` para hacer que el formulario luzca más atractivo:

```
form {  
    width: 300px;  
    margin: 0 auto;  
    padding: 20px;  
    background-color: #fff;  
    border-radius: 8px;  
    box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);  
}  
  
label {  
    display: block;  
    margin: 10px 0 5px;  
    font-weight: bold;  
}  
  
input {  
    width: 100%;  
    padding: 8px;  
    margin-bottom: 10px;  
    border: 1px solid #ccc;  
    border-radius: 4px;  
}  
  
button {  
    width: 100%;  
    padding: 10px;  
    background-color: #28a745;  
    color: white;  
    border: none;  
    border-radius: 4px;  
    cursor: pointer;
```

```

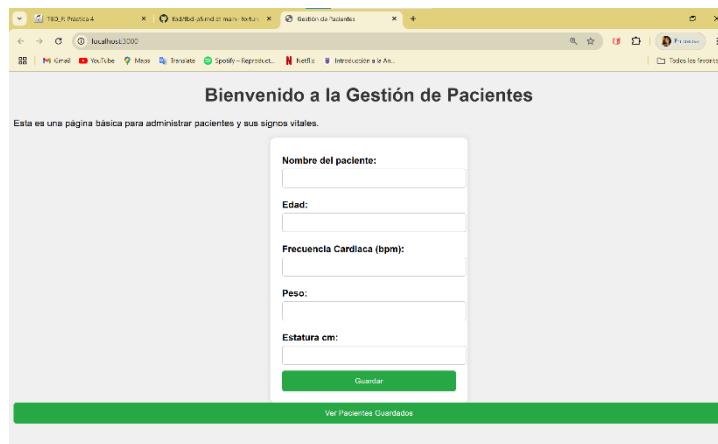
    }

button:hover {
    background-color: #218838;
}

```

2. Recarga la página:

Al recargar la página, el formulario para registrar pacientes debería verse más estilizado con un diseño centrado y limpio.



Paso 3: Crear una tabla estilizada para visualizar los pacientes

1. Modificar la ruta para obtener los datos en formato HTML:

En `server.js`, modifica la ruta `/pacientes` para que muestre los datos dentro de una tabla HTML en lugar de solo texto:

```

// Ruta para mostrar los datos de la base de datos en formato HTML
app.get('/pacientes', (req, res) => {
  connection.query('SELECT * FROM pacientes', (err, results) => {
    if (err) {
      return res.send('Error al obtener los datos.');
    }

    let html = `
      <html>
        <head>
          <link rel="stylesheet" href="/styles.css">
          <title>Pacientes</title>
        </head>
        <body>
          <h1>Pacientes Registrados</h1>
          <table>
            <thead>
              <tr>

```

```

        <th>Nombre</th>
        <th>Edad</th>
        <th>Frecuencia Cardiaca (bpm)</th>
    </tr>
</thead>
<tbody>
`;

results.forEach(paciente => {
    html += `
        <tr>
            <td>${paciente.nombre}</td>
            <td>${paciente.edad}</td>
            <td>${paciente.frecuencia_cardiaca}</td>
        </tr>
`;
});
html += `
    </tbody>
</table>
<button onclick="window.location.href='/'>Volver</button>
</body>
</html>
`;
res.send(html);
}) ;
}) ;

```

2. **Estilizar la tabla en CSS:** Agrega los siguientes estilos en `styles.css` para hacer que la tabla se vea más agradable:

```

table {
    width: 80%;
    margin: 20px auto;
    border-collapse: collapse;
}

th, td {
    padding: 12px;
    text-align: left;
    border-bottom: 1px solid #ddd;
}

th {
    background-color: #4CAF50;
    color: white;
}

tr:hover {
    background-color: #f5f5f5;
}

button {

```

```

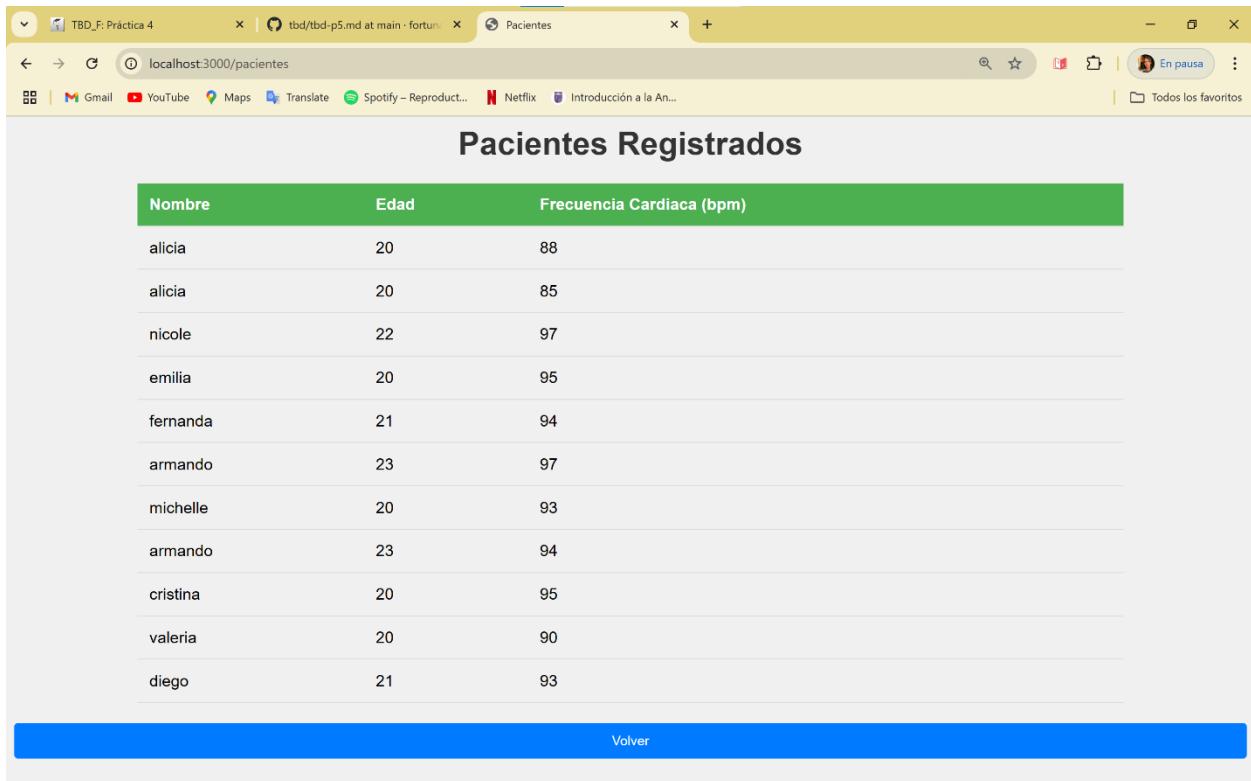
display: block;
margin: 20px auto;
padding: 10px;
background-color: #007bff;
color: white;
border: none;
border-radius: 4px;
cursor: pointer;
}

button:hover {
    background-color: #0056b3;
}

```

3. Prueba la visualización de la tabla:

- Guarda los cambios y reinicia el servidor si es necesario.
- Ve a <http://localhost:3000/pacientes>. Deberías ver los datos de los pacientes dentro de una tabla bien estilizada con encabezados y colores que hacen que la tabla se vea profesional.



The screenshot shows a web browser window with the title bar "TBD_F: Práctica 4" and the address bar "localhost:3000/pacientes". The main content is a table titled "Pacientes Registrados" with the following data:

Nombre	Edad	Frecuencia Cardiaca (bpm)
alicia	20	88
alicia	20	85
nicole	22	97
emilia	20	95
fernanda	21	94
armando	23	97
michelle	20	93
armando	23	94
cristina	20	95
valeria	20	90
diego	21	93

A blue button at the bottom right of the table says "Volver".

Paso 4: Personalización adicional

1. Mejora el diseño de la página:

Revisa el archivo de styles.css, ¿Existen estilos para elementos repetidos? ¿Qué otra adecuación le podrías hacer al sistema para que la apariencia en general sea más agradable? Modifica los estilos, personaliza los colores o el diseño usando más propiedades CSS. Podrían agregar más estilos como:

- **Colores personalizados.**
- **Fuente diferente.**



A screenshot of a web browser window titled "TBD_F: Práctica 4". The address bar shows "localhost:3000/pacientes". The main content is a table titled "Pacientes Registrados" with the following data:

Nombre	Edad	Frecuencia Cardiaca (bpm)	Peso	Estatura
alicia	20	88	null	null
alicia	20	85	null	null
nicole	22	97	null	null
emilia	20	95	null	null
fernanda	21	94	null	null
armando	23	97	null	null
michelle	20	93	null	null
armando	23	94	75	181
cristina	20	95	56	168
valeria	20	90	55	169
diego	21	93	60	165
Karla	20	88	55	167

At the bottom of the table, there is a dark button with the word "Volver".

INSTITUTO TECNOLÓGICO DE TIJUANA



TECNOLOGIAS DE BASE DE DATOS

Unidad 2
Estándares y normas para registros electrónicos en la salud

Practica 6

Mejoras en la Interfaz Web y Consultas SQL Avanzadas

PRESENTA:
Camacho Lopez Valeria

C22211475

DOCENTE:
Fortunato Ramírez Arzate

TIJUANA, B. C, A 20 DE OCTUBRE DEL 2024

Objetivo:

Desarrollar una interfaz web más interactiva para la gestión de pacientes, integrando consultas SQL más avanzadas y mostrando los resultados en una tabla HTML estilizada. Los estudiantes podrán realizar operaciones de consulta específicas y manipular la forma en que los datos se muestran.

Introducción:

En esta práctica, continuarás mejorando la aplicación web desarrollada en las prácticas anteriores. Ahora aprenderás a realizar consultas más complejas sobre los datos almacenados en la base de datos, además de estilizar los resultados de manera personalizada usando solo CSS. Se espera que los estudiantes dominen el uso de `SELECT` con filtros (`WHERE`), ordenación de resultados (`ORDER BY`), y más consultas SQL avanzadas vistas en las prácticas anteriores.

Requisitos:

- Haber completado la Práctica 5.
- Tener el entorno de Node.js y MySQL listo y funcionando con los archivos creados previamente en la carpeta `gestion-pacientes`.
- Haber instalado las dependencias necesarias (`express`, `mysql2`, `body-parser`).

Paso 1: Modificar el formulario para incluir un filtro de consulta

1. Agregar campos de búsqueda en el HTML:

Vamos a añadir un pequeño formulario en `public/index.html` para que los usuarios puedan buscar pacientes por nombre y edad. Modifica el archivo para incluir los nuevos campos justo debajo del formulario actual:

```
<h2>Buscar Pacientes</h2>
<form action="/buscar-pacientes" method="GET">
  <label for="name-search">Nombre del paciente:</label>
  <input type="text" id="name-search" name="name_search">

  <label for="age-search">Edad del paciente:</label>
  <input type="number" id="age-search" name="age_search">

  <button type="submit">Buscar</button>
</form>
```

```
32
33  <h2>Buscar Pacientes</h2>
34  <form action="/buscar-pacientes" method="GET">
35    <label for="name-search">Nombre del paciente:</label>
36    <input type="text" id="name-search" name="name_search">
37
38    <label for="age-search">Edad del paciente:</label>
39    <input type="number" id="age-search" name="age_search">
40
41    <button type="submit">Buscar</button>
42  </form>
```

Paso 2: Implementar la lógica de búsqueda en el servidor

1. Agregar una nueva ruta para manejar las búsquedas:

En `server.js`, agrega una ruta que procesará la búsqueda en la base de datos según los filtros de nombre y edad introducidos. Usa el siguiente código para manejar la consulta:

```
// Ruta para buscar pacientes según filtros
app.get('/buscar-pacientes', (req, res) => {
  const { name_search, age_search } = req.query;
  let query = 'SELECT * FROM pacientes WHERE 1=1';

  if (name_search) {
    query += ` AND nombre LIKE '%${name_search}%'`;
  }
  if (age_search) {
    query += ` AND edad = ${age_search}`;
  }

  connection.query(query, (err, results) => {
    if (err) {
      return res.send('Error al obtener los datos.');
    }

    let html = `
      <html>
        <head>
          <link rel="stylesheet" href="/styles.css">
          <title>Resultados de Búsqueda</title>
        </head>
        <body>
          <h1>Resultados de Búsqueda</h1>
          <table>
            <thead>
              <tr>
                <th>Nombre</th>
                <th>Edad</th>
                <th>Frecuencia Cardiaca (bpm)</th>
              </tr>
            </thead>
            <tbody>
    `;

    results.forEach(paciente => {
      html += `
        <tr>
          <td>${paciente.nombre}</td>
          <td>${paciente.edad}</td>
          <td>${paciente.frecuencia_cardiaca}</td>
        </tr>
    `;
    });
  });
});
```

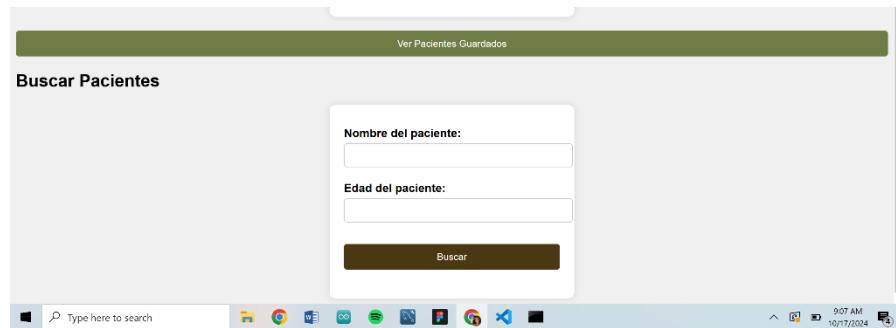
```

        html += `
            </tbody>
        </table>
        <button onclick="window.location.href=' / '">Volver</button>
    </body>
</html>
`;

res.send(html);
}) ;
});

```

Con esta lógica, los usuarios pueden buscar pacientes por nombre o edad (o ambos). Si no se ingresa ningún filtro, se mostrarán todos los pacientes.



Paso 3: Añadir consultas avanzadas y mostrar los resultados

1. Agregar un botón para ordenar por frecuencia cardíaca:

Vamos a añadir un botón en el HTML para que los usuarios puedan ordenar la lista de pacientes por su frecuencia cardíaca. Modifica el archivo `public/index.html` para incluir este botón:

```
<button onclick="window.location.href='/ordenar-pacientes'">Ordenar por  
Frecuencia Cardiaca</button>
```

```
42  </form>
43  <button onclick="window.location.href='/pacientes'">Ver Pacientes Guardados</button>
44  <button onclick="window.location.href='/ordenar-pacientes'">Ordenar por Frecuencia Cardiaca</button>
45  <button onclick="window.location.href='/edadProm-pacientes'">Promedio de edades de todos los pacientes</button>
```

2. Implementar la ruta para ordenar por frecuencia cardíaca:

En `server.js`, añade la lógica para ordenar los resultados de pacientes por su frecuencia cardíaca:

```

// Ruta para ordenar pacientes por frecuencia cardiaca
app.get('/ordenar-pacientes', (req, res) => {
  const query = 'SELECT * FROM pacientes ORDER BY frecuencia_cardiaca DESC';

  connection.query(query, (err, results) => {
    if (err) {
      return res.send('Error al obtener los datos.');
    }

    let html = `
      <html>
        <head>
          <link rel="stylesheet" href="/styles.css">
          <title>Pacientes Ordenados</title>
        </head>
        <body>
          <h1>Pacientes Ordenados por Frecuencia Cardiaca</h1>
          <table>
            <thead>
              <tr>
                <th>Nombre</th>
                <th>Edad</th>
                <th>Frecuencia Cardiaca (bpm)</th>
              </tr>
            </thead>
            <tbody>
              `;

    results.forEach(paciente => {
      html += `
        <tr>
          <td>${paciente.nombre}</td>
          <td>${paciente.edad}</td>
          <td>${paciente.frecuencia_cardiaca}</td>
        </tr>
      `;
    });

    html += `
      </tbody>
      </table>
      <button onclick="window.location.href='/'>Volver</button>
    </body>
  </html>
  `;

  res.send(html);
  }));
});

```

Buscar Pacientes

Nombre del paciente:

Edad del paciente:

[Ver Pacientes Guardados](#)

[Ordenar por Frecuencia Cardiaca](#)

Paso 4: Insertar médicos en la base de datos

En esta práctica, también integraremos la inserción de **médicos** en la base de datos.

1. Crear una nueva tabla de médicos en MySQL:

En la terminal de MySQL, crea la tabla `medicos`:

```
CREATE TABLE medicos (
    id INT AUTO_INCREMENT PRIMARY KEY,
    nombre VARCHAR(100),
    especialidad VARCHAR(100)
);
```

```
mysql> CREATE TABLE medicos (
->     id INT AUTO_INCREMENT PRIMARY KEY,
->     nombre VARCHAR(100),
->     especialidad VARCHAR(100)
-> );
```

```
mysql> SHOW TABLES;
+-----+
| Tables_in_gestion_pacientes |
+-----+
| medicos
| pacientes
+-----+
2 rows in set (0.02 sec)
```

Se observa que la tabla se creó correctamente

2. Modificar el HTML para permitir la inserción de médicos:

En `public/index.html`, agrega un formulario para insertar médicos:

```
<h2>Registrar Médico</h2>
<form action="/insertar-medico" method="POST">
    <label for="medico-name">Nombre del médico:</label>
    <input type="text" id="medico-name" name="medico_name">
```

```
47  <h2>Registrar Médico</h2>
48  <form action="/insertar-medico" method="POST">
49      <label for="medico-name">Nombre del médico:</label>
50      <input type="text" id="medico-name" name="medico_name">
51
52      <label for="especialidad">Especialidad:</label>
53      <input type="text" id="especialidad" name="especialidad">
54
55      <button type="submit">Guardar Médico</button>
56  </form>
57
```

3. Agregar la lógica en el servidor para insertar médicos:

En `server.js`, añade la lógica para insertar médicos en la base de datos:

```
// Ruta para insertar un nuevo médico
app.post('/insertar-medico', (req, res) => {
  const { medico_name, especialidad } = req.body;
  const query = 'INSERT INTO medicos (nombre, especialidad) VALUES (?, ?)';
  connection.query(query, [medico_name, especialidad], (err, result) =>
  {
    if (err) {
      return res.send('Error al insertar el médico.');
    }
    res.send(`Médico ${medico_name} guardado exitosamente.`);
  });
});

261 // Ruta para insertar un nuevo médico
262 app.post('/insertar-medico', (req, res) => {
263   const { medico_name, especialidad } = req.body;
264   const query = 'INSERT INTO medicos (nombre, especialidad) VALUES (?, ?)';
265
266   connection.query(query, [medico_name, especialidad], (err, result) => {
267     if (err) {
268       return res.send('Error al insertar el médico.');
269     }
270     res.send(`Médico ${medico_name} guardado exitosamente.`);
271   });
272 });
273});
```

Paso 5: Estilizar los resultados con CSS

Como en la Práctica 5, asegúrate de que los resultados mostrados en tablas estén bien estilizados. Puedes reutilizar o mejorar los estilos que ya tienes en `styles.css`.

Paso 6: Probar y finalizar

1. **Ejecuta el servidor** con `node server.js`.
2. **Prueba** las nuevas funcionalidades:
 - o Busca pacientes por nombre y edad.

The screenshot shows a search interface titled "Buscar Pacientes". It contains two input fields: "Nombre del paciente:" with the value "karla" and "Edad del paciente:" with the value "20". Below the inputs is a large "Buscar" button.

Resultados de Búsqueda

id	Nombre	Edad	Frecuencia Cardiaca (bpm)	Peso (kg)	Estatura (cm)
12	Karla	20	88	55	167

[Volver](#)

- Ordena los pacientes por su frecuencia cardíaca.

Pacientes Ordenados por Frecuencia Cardiaca

id	Nombre	Edad	Frecuencia Cardiaca (bpm)	Peso (kg)	Estatura (cm)
3	nico	22	97	null	null
6	armando	23	97	null	null
4	emilia	20	95	null	null
9	cristina	20	95	56	168
5	fernanda	21	94	null	null
8	armando	23	94	75	181
7	michelle	20	93	null	null
11	diego	21	93	60	165
10	valeria	20	90	55	169
1	alicia	20	88	null	null
12	Karla	20	88	55	167
2	alicia	20	85	null	null

[Volver](#)

- Inserta médicos en la base de datos y verifica que se guarden correctamente.

Registrar Médico

Nombre del médico:

Especialidad:

[Guardar Médico](#)

Médicos Registrados

id	Nombre	Especialidad
1	fernando avila	neurologia
2	rafael sanchez	dermatologo
3	david aceves	medico general

[Volver](#)

```
mysql> SELECT * FROM medicos;
+---+-----+-----+
| id | nombre      | especialidad   |
+---+-----+-----+
| 1 | fernando avila | neurologia    |
| 2 | rafael sanchez | dermatologo   |
| 3 | david aceves  | medico general|
+---+-----+-----+
3 rows in set (0.00 sec)
```

INSTITUTO TECNOLÓGICO DE TIJUANA



TECNOLOGIAS DE BASE DE DATOS

Unidad 3
Manejo de Base de datos

Practica 7

Autenticación de Usuarios y Protección de Rutas

PRESENTA:
Camacho Lopez Valeria

C22211475

DOCENTE:
Fortunato Ramírez Arzate

TIJUANA, B. C, A 20 DE NOVIEMBRE DEL 2024

Objetivo:

Implementar un sistema de autenticación básico en la aplicación de gestión de pacientes y médicos, con re direccionamientos entre páginas de inicio de sesión, registro y la página principal. Los usuarios deberán iniciar sesión para acceder a la aplicación. Las rutas estarán protegidas mediante sesiones, y las contraseñas se guardarán en la base de datos de forma segura con hashing.

Requisitos:

- Haber completado la Práctica 6.
- Tener instalados los módulos express, mysql2, body-parser, bcrypt, y express-session.
- Una estructura de archivos HTML separados para cada página: index.html (principal), login.html (iniciar sesión), y registro.html (registro de usuario).
- La base de datos debe tener una tabla usuarios para manejar el acceso.

Paso 1: Crear la tabla usuarios para almacenar credenciales

En tu base de datos MySQL, crea una tabla usuarios para almacenar la información de autenticación de los usuarios. Incluye las siguientes columnas:

```
CREATE TABLE usuarios (
    id INT AUTO_INCREMENT PRIMARY KEY,
    nombre_usuario VARCHAR(50) UNIQUE,
    password_hash VARCHAR(255)
);
```

```
mysql> CREATE TABLE usuarios (
->     id INT AUTO_INCREMENT PRIMARY KEY,
->     nombre VARCHAR(50) UNIQUE,
->     correo VARCHAR(50) UNIQUE,
->     contrasena_hash VARCHAR(50) UNIQUE,
->     tipo_usuario VARCHAR(50) UNIQUE
-> );
Query OK, 0 rows affected (0.05 sec)

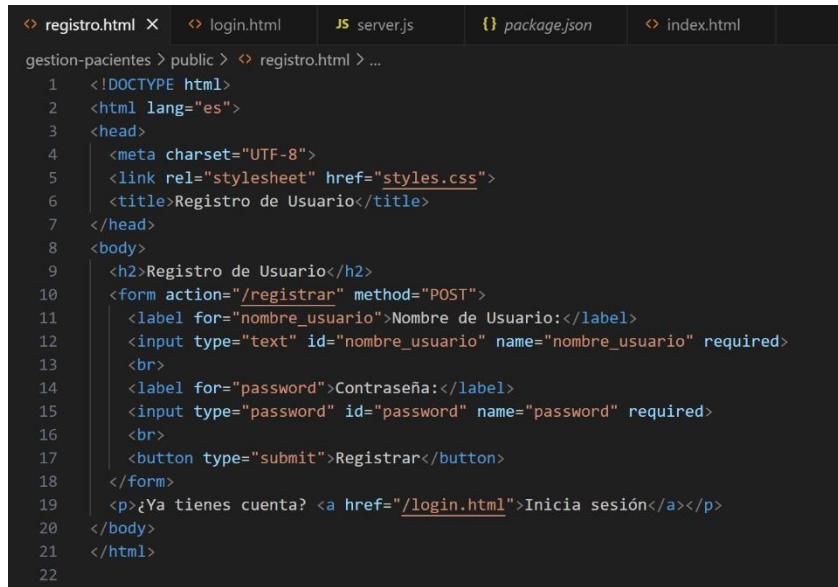
mysql>
```

Se tiene como llave primaria y en autoincremento la columna de ID para los usuarios que se vayan a registrar en la base de datos, al igual que el nombre y la contraseña en hash se asignan a que tengan valores únicos e irrepetibles, en el caso de la contraseña en hash es para que se registre la contraseña, pero no sea visible, solo se verá una cadena de caracteres que "sustituyen" la contraseña original para que nadie más tenga acceso a ella.

Paso 2: Crear la página de Registro (registro.html)

En la carpeta public, crea un archivo llamado registro.html. Esta página permitirá que los nuevos usuarios se registren en el sistema. Agrega el siguiente código HTML:

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <title>Registro de Usuario</title>
</head>
<body>
    <h2>Registro de Usuario</h2>
    <form action="/registrar" method="POST">
        <label for="nombre_usuario">Nombre de Usuario:</label>
        <input type="text" id="nombre_usuario" name="nombre_usuario" required>
        <br>
        <label for="password">Contraseña:</label>
        <input type="password" id="password" name="password" required>
        <br>
        <button type="submit">Registrar</button>
    </form>
    <p>¿Ya tienes cuenta? <a href="/login">Inicia sesión</a></p>
</body>
</html>
```



The screenshot shows a terminal window with several tabs open. The active tab is 'registro.html' which contains the code provided above. Other tabs include 'login.html', 'server.js', 'package.json', and 'index.html'. The terminal also displays the directory path: 'gestion-pacientes > public > registro.html > ...'. The code is numbered from 1 to 22.

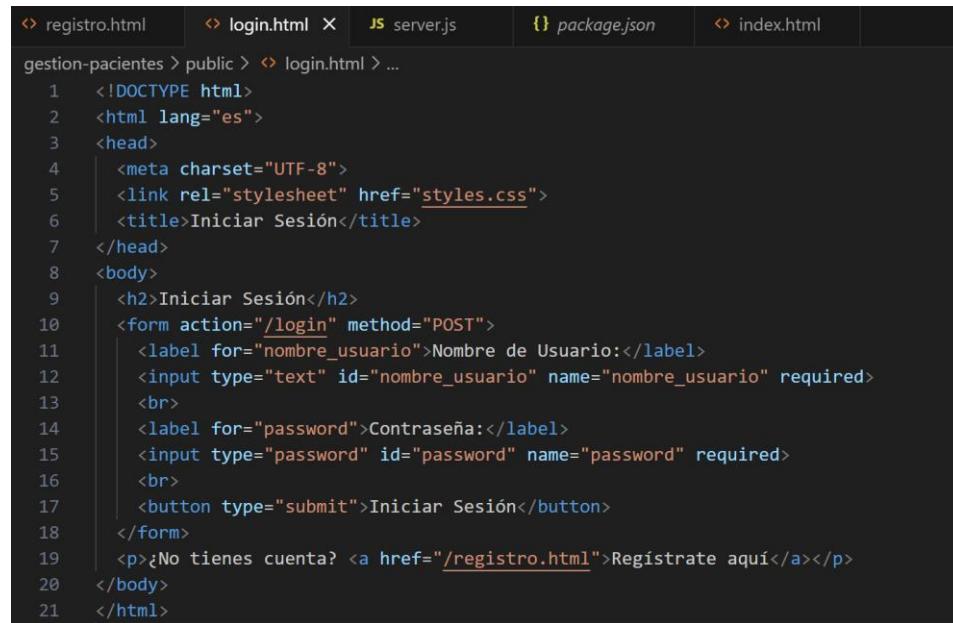
```
gestion-pacientes > public > registro.html > ...
1   <!DOCTYPE html>
2   <html lang="es">
3   <head>
4       <meta charset="UTF-8">
5       <link rel="stylesheet" href="styles.css">
6       <title>Registro de Usuario</title>
7   </head>
8   <body>
9       <h2>Registro de Usuario</h2>
10      <form action="/registrar" method="POST">
11          <label for="nombre_usuario">Nombre de Usuario:</label>
12          <input type="text" id="nombre_usuario" name="nombre_usuario" required>
13          <br>
14          <label for="password">Contraseña:</label>
15          <input type="password" id="password" name="password" required>
16          <br>
17          <button type="submit">Registrar</button>
18      </form>
19      <p>¿Ya tienes cuenta? <a href="/login.html">Inicia sesión</a></p>
20  </body>
21 </html>
22
```

Se tiene que verificar que las variables coincidan con las ya establecidas en la base de datos, al igual es necesario colocar .html a la ruta de iniciar sesión; también es necesario agregarle el apartado de styless para que aparezca el cuestionario de iniciar sesión de una manera mas presentable.

Paso 3: Crear la página de Inicio de Sesión (login.html)

En la carpeta public, crea un archivo llamado login.html. Esta página será para que los usuarios existentes inicien sesión.

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <title>Iniciar Sesión</title>
</head>
<body>
    <h2>Iniciar Sesión</h2>
    <form action="/login" method="POST">
        <label for="nombre_usuario">Nombre de Usuario:</label>
        <input type="text" id="nombre_usuario" name="nombre_usuario" required>
        <br>
        <label for="password">Contraseña:</label>
        <input type="password" id="password" name="password" required>
        <br>
        <button type="submit">Iniciar Sesión</button>
    </form>
    <p>¿No tienes cuenta? <a href="/registro">Regístrate aquí</a></p>
</body>
</html>
```



```
gestion-pacientes > public > login.html > ...
1  <!DOCTYPE html>
2  <html lang="es">
3  <head>
4      <meta charset="UTF-8">
5      <link rel="stylesheet" href="styles.css">
6      <title>Iniciar Sesión</title>
7  </head>
8  <body>
9      <h2>Iniciar Sesión</h2>
10     <form action="/login" method="POST">
11         <label for="nombre_usuario">Nombre de Usuario:</label>
12         <input type="text" id="nombre_usuario" name="nombre_usuario" required>
13         <br>
14         <label for="password">Contraseña:</label>
15         <input type="password" id="password" name="password" required>
16         <br>
17         <button type="submit">Iniciar Sesión</button>
18     </form>
19     <p>¿No tienes cuenta? <a href="/registro.html">Regístrate aquí</a></p>
20 </body>
21 </html>
```

También es necesario agregar la línea de styles para que el usuario pueda observar una interfaz más amigable visualmente, además no hay que olvidar color .html en la última línea de "regístrate aquí".

Paso 4: Configurar la lógica del servidor en server.js

1. Instalar y configurar las dependencias necesarias:

Ejecuta el siguiente comando para instalar express-session y bcrypt:

```
npm install express-session bcrypt
```

```
C:\Users\lopez\Desktop\gestion-pacientes>npm install express-session bcrypt
up to date, audited 143 packages in 2s

17 packages are looking for funding
  run `npm fund` for details

2 low severity vulnerabilities

To address all issues, run:
  npm audit fix

Run `npm audit` for details.
```

Estas nuevas librerías pueden cargarse directamente desde una terminal en visual studio code o abrir una terminal con cmd desde la ruta de la carpeta de la base de datos creada.

2. Agregar configuración de sesión y rutas de autenticación:

En server.js, configura las sesiones y crea las rutas para registrar usuarios, iniciar sesión y cerrar sesión.

```
const express = require('express');
const session = require('express-session');
const bcrypt = require('bcrypt');
const mysql = require('mysql2');
const app = express();

// Configuración de la sesión
app.use(session({
  secret: 'secretKey',
  resave: false,
  saveUninitialized: false,
}));

app.use(express.urlencoded({ extended: true }));

// Conexión a MySQL
```

```

const connection = mysql.createConnection({
  host: 'localhost',
  user: 'root',
  password: '',
  database: 'gestion_pacientes'
});

connection.connect();

// Registro de usuario
app.post('/registrar', async (req, res) => {
  const { nombre_usuario, password } = req.body;
  const passwordHash = await bcrypt.hash(password, 10);

  connection.query('INSERT INTO usuarios (nombre_usuario, password_hash)
VALUES (?, ?)', [nombre_usuario, passwordHash], (err) => {
    if (err) {
      return res.send('Error al registrar el usuario.');
    }
    res.redirect('/login');
  });
});

// Iniciar sesión
app.post('/login', (req, res) => {
  const { nombre_usuario, password } = req.body;

  connection.query('SELECT * FROM usuarios WHERE nombre_usuario = ?',
  [nombre_usuario], async (err, results) => {
    if (err || results.length === 0) {
      return res.send('Usuario no encontrado.');
    }

    const user = results[0];
    const match = await bcrypt.compare(password, user.password_hash);
    if (match) {
      req.session.userId = user.id;
      res.redirect('/');
    } else {
      res.send('Contraseña incorrecta.');
    }
  });
});

// Cerrar sesión
app.get('/logout', (req, res) => {
  req.session.destroy();
  res.redirect('/login');
});

```

```

gestion-pacientes > JS server.js > ...
1  const express = require('express');
2  const app = express();
3  const path = require('path');
4  const bodyParser = require('body-parser');
5  const mysql = require('mysql2');
6  const session = require('express-session');
7  const bcrypt = require('bcrypt');
8

```

Es importante colocar todas las variables hasta el encabezado del server, a excepción de las variables específicas de alguna función.

```
9  // Configurar conexión a MySQL
10 const connection = mysql.createConnection({
11   host: 'localhost',
12   user: 'root',
13   password: 'CALV040916', // cambia por tu contraseña
14   database: 'gestion_pacientes'
15 });
16
17 connection.connect(err => {
18   if (err) {
19     console.error('Error conectando a MySQL:', err);
20     return;
21   }
22   console.log('Conexión exitosa a MySQL');
23 });
```

En la configuración a MySQL se refiere a la sección del código en la que vamos a establecer una conexión entre nuestro servidor y la base de datos, es importante definir el host y poner la misma contraseña con la que iniciamos sesión al abrir MySQL desde una terminal cmd.



```
registro.html  login.html  server.js  package.json  index.html
gestion-pacientes > server.js > ...
63
64 // Iniciar sesión
65 app.post('/login', (req, res) => {
66   const { nombre_usuario, password } = req.body;
67
68   connection.query('SELECT * FROM usuarios WHERE nombre_usuario = ?',
69   [nombre_usuario], async (err, results) => {
70     if (err || results.length === 0) {
71       return res.send('Usuario no encontrado.');
72     }
73
74     const user = results[0];
75     const match = await bcrypt.compare(password, user.password_hash);
76     if (match) {
77       req.session.userId = user.id;
78       res.redirect('/');
79     } else {
80       res.send('Contraseña incorrecta.');
81     }
82   });
83 });
84
85 // Registro de usuario
86 app.post('/registrar', async (req, res) => {
87   const { nombre_usuario, password } = req.body;
88   const passwordHash = await bcrypt.hash(password, 10);
89
90   connection.query('INSERT INTO usuarios (nombre_usuario, password_hash) VALUES (?, ?)',
91   [nombre_usuario, passwordHash], (err) => {
92     if (err) {
93       return res.send('Error al registrar el usuario.');
94     }
95     res.redirect('/login.html');
96   });
97 });
98
99 // Cerrar sesión
100 app.get('/logout', requireLogin,(req, res) => {
101   req.session.destroy();
102   res.redirect('/login.html');
103});
```

Una vez que todo está en orden, primero podemos verificar el query directamente en la base de datos para confirmar que dicha instrucción se está ejecutando adecuadamente de acuerdo a la sintaxis del programa.

A continuación, empieza la sección en la que definimos rutas y algunas tablas correspondientes a formularios.

Paso 5: Proteger las rutas con autenticación de sesión

1. Agrega un middleware en server.js para proteger las rutas. Sólo los usuarios autenticados podrán acceder a ellas.

```
function requireLogin(req, res, next) {
  if (!req.session.userId) {
    return res.redirect('/login');
  }
  next();
}

// Ruta protegida (Página principal después de iniciar sesión)
app.get('/', requireLogin, (req, res) => {
  res.sendFile(__dirname + '/public/index.html');
});

// Otras rutas protegidas
app.get('/buscar-pacientes', requireLogin, (req, res) => {
  // lógica de búsqueda de pacientes
});

app.get('/ordenar-pacientes', requireLogin, (req, res) => {
  // lógica de ordenamiento de pacientes
});

app.post('/insertar-medico', requireLogin, (req, res) => {
  // lógica de inserción de médicos
});
```

```
24
25  // Configuración de la sesión
26  app.use(session({
27    secret: 'secretKey',
28    resave: false,
29    saveUninitialized: false,
30  }));
31
32  app.use(express.urlencoded({ extended: true }));
33
34  function requireLogin(req, res, next) {
35    if (!req.session.userId) {
36      return res.redirect('/login.html');
37    }
38    next();
39  }
40
```

Tenemos la primera función la cual nos va a ayudar en el momento que tenemos más de un tipo de usuario, el cual se requiere que obligatoriamente inicie sesión para tener acceso a cierta cantidad de información dependiendo de su rol.

Al igual, se asigna la ruta que queremos que sea la principal, en este caso por defecto se nombre como "/", señalando que es el index.

```
41 // Ruta para la página principal
42 app.get('/',requireLogin ,(req, res) => {
43   res.sendFile(path.join(__dirname, 'public', 'index.html'));
44 });
45
```

Paso 6: Modificar la página principal (index.html)

1. Actualiza index.html en la carpeta public para mostrar una opción de cerrar sesión y recibir al usuario.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <title>Gestión de Pacientes y Médicos</title>
  <link rel="stylesheet" href="/styles.css">
</head>
<body>
  <h2>Bienvenido a la Gestión de Pacientes</h2>
  <a href="/logout">Cerrar Sesión</a>
  <!-- Formularios y botones de insertar/consultar médicos y pacientes -->
</body>
</html>
```

```
registro.html login.html server.js package.json index.html
gestion-pacientes > public > index.html > html > body > form > input#weight
1  <!DOCTYPE html>
2  <html lang="es">
3  <head>
4    <meta charset="UTF-8">
5    <meta name="viewport" content="width=device-width, initial-scale=1.0">
6    <link rel="stylesheet" href="styles.css">
7    <title>Gestión de Pacientes</title>
8  </head>
9  <body>
10    <h1>Bienvenido a la Gestión de Pacientes</h1>
11    <p>Esta es una página básica para administrar pacientes y sus signos vitales.</p>
12    <a href="/logout">Cerrar Sesión</a>
13
14    <form action="/submit-data" method="POST">
15      <label for="name">Nombre del paciente:</label>
16      <input type="text" id="name" name="name">
```

Se actualizó la página principal ya que se ha agregado la función de cerrar sesión, incluido un botón que facilite la ejecución de la nueva acción.

Paso 7: Prueba de funcionamiento

1. Prueba el registro de usuario:

- Accede a /registro y crea una cuenta.

The image contains two screenshots of a web browser window. The top screenshot shows the 'Iniciar Sesión' (Login) page at localhost:3000/login.html. It has fields for 'Nombre de Usuario:' and 'Contraseña:', and a 'Iniciar Sesión' button. Below the form is a link '¿No tienes cuenta? Regístrate aquí'. The bottom screenshot shows the 'Registro de Usuario' (User Registration) page at localhost:3000/registro.html. It has fields for 'Nombre de Usuario:', 'Contraseña:', and 'Código de acceso:', and a 'Registrar' button. Below the form is a link '¿Ya tienes cuenta? Inicia sesión'.

Se muestra primero la página con el formulario para iniciar sesión, primero debemos crear un nuevo registro, podemos comprobar que el registro fue exitoso desde la base de datos o igual iniciando sesión sin errores.

```
mysql> SELECT * FROM usuarios;
+----+-----+-----+
| id | nombre_usuario | password_hash |
+----+-----+-----+
| 1  | valeria      | $2b$10$JRSX8KVPmcOTSXHB8Q7V9uYmaFB68eW.ZRC3dM/HtqP58mZkFgUeG |
| 6  | kevin        | $2b$10$cP5ABxQQFUpyeFXNKty290iFb2aQ3fc7r1BKfugY1VHM/5BamgTBS |
| 7  | vianney       | $2b$10$4Fxyq1GDjHyZJD2cIBxRnO6/mUs9ONVIkoNqVYczZNi.iUH9LpHjy |
| 8  | david         | $2b$10$wkHtH5ByqBEIlaEud7tRmuIqlTEv9ycx1HZ5mUnCggxE5B1cWV4gu |
| 13 | alejandra     | $2b$10$FjQVPj/rIpwTJT35iXCmYu6S3vmfl5HE0/5FQyoi9NqpjMfjXtx16 |
+----+-----+-----+
5 rows in set (0.00 sec)
```

- Luego, inicia sesión con las credenciales creadas.

The screenshot shows a web browser window with the URL `localhost:3000/registro.html`. The title bar says "Registro de Usuario". The form contains fields for "Nombre de Usuario" (alejandra) and "Contraseña" (empty). A "Registrar" button is at the bottom. Below the form, a link says "¿Ya tienes cuenta? [Inicia sesión](#)".

The screenshot shows a web browser window with the URL `localhost:3000`. The title bar says "Bienvenido a la Gestión de Pacientes". The page displays a message: "Esta es una página básica para administrar pacientes y sus signos vitales." and a "Cerrar Sesión" link. It features a form for entering patient information: "Nombre del paciente", "Edad", "Frecuencia Cardíaca (bpm)", "Peso (kg)", and "Estatura (cm)". A "Guardar" button is at the bottom. Below the form, there's a "Buscar Pacientes" section with a search input field labeled "Nombre del paciente:".

Una vez que iniciamos sesión exitosamente, se muestra página principal

2. Prueba de protección de rutas:

- Asegúrate de que solo los usuarios registrados puedan acceder a las rutas de `/`, `/buscar-pacientes`, `/ordenar-pacientes`, y `/insertar-medico`.

The screenshot shows a web browser window with the URL `localhost:3000/pacientes`. The title bar says "Pacientes Registrados". The table has columns: Id, Nombre, Edad, Frecuencia Cardíaca (bpm), Peso (kg), and Estatura (cm). The data is as follows:

Id	Nombre	Edad	Frecuencia Cardíaca (bpm)	Peso (kg)	Estatura (cm)
1	alicia	20	88	null	null
2	alicia	20	85	null	null
3	nicole	22	97	null	null
4	emilia	20	95	null	null
5	fernanda	21	94	null	null
6	armando	23	97	null	null
7	michelle	20	93	null	null
8	armando	23	94	75	181
9	cristina	20	95	56	168
10	valeria	20	90	55	169
11	diego	21	93	60	165
12	Karla	20	88	55	167
13	kevin	24	85	70	178
14	vianney	19	80	55	166

A "Volver" button is at the bottom of the table.

localhost:3000

Gmail YouTube Maps Translate Spotify - Reproduct... Netflix Introducción a la An...

Todos los favoritos

Buscar Pacientes

Nombre del paciente:

Edad del paciente:

Buscar

Ver Pacientes Guardados

Ordenar por Frecuencia Cardíaca

Promedio de edades de todos los pacientes

Registrar Médico

Nombre del médico:

Especialidad:

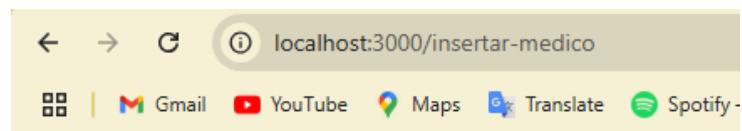
Guardar Médico

Registrar Médico

Nombre del médico:
alan cervantes

Especialidad:
pediatría

Guardar Médico



localhost:3000/medicos

Gmail YouTube Maps Translate Spotify - Reproduct... Netflix Introducción a la An...

Todos los favoritos

Médicos Registrados

ID	Nombre	Especialidad
1	fernando avila	neurología
2	rafael sanchez	dermatólogo
3	david aceves	medico general
4	alan cervantes	pediatría

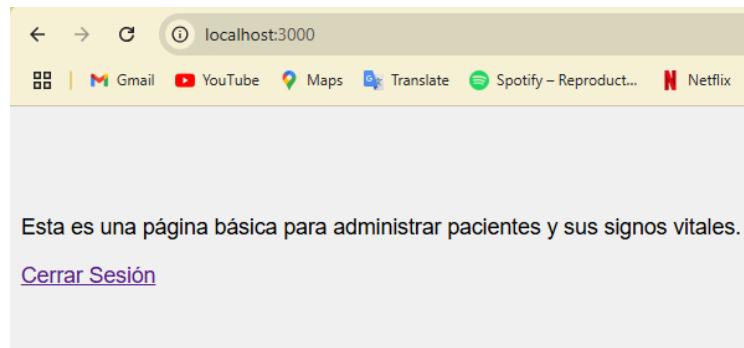
Volver

Pacientes Ordenados por Frecuencia Cardíaca					
ID	Nombre	Edad	Frecuencia Cardíaca (bpm)	Peso (kg)	Estatura (cm)
3	nicole	22	97	null	null
6	armando	23	97	null	null
4	emilia	20	95	null	null
9	cristina	20	95	56	168
5	fernanda	21	94	null	null
8	armando	23	94	76	181
7	michelle	20	93	null	null
11	diego	21	93	60	165
10	valeria	20	90	55	169
1	alicia	20	88	null	null
12	Karla	20	88	55	167
2	alicia	20	85	null	null
13	kevin	24	85	70	178
14	vianney	19	80	55	166

Se observa que se tiene acceso a todas las rutas definidas, así como el correcto funcionamiento de los formularios para registrar nuevos médicos y pacientes.

3. Prueba de cierre de sesión:

- Usa el enlace de cierre de sesión para terminar la sesión y verificar que las rutas protegidas ya no sean accesibles.



Se obtiene un cierre de sesión exitoso, al momento de ejecutar la acción nos redirige a la página de iniciar sesión.

INSTITUTO TECNOLÓGICO DE TIJUANA



TECNOLOGIAS DE BASE DE DATOS

Unidad 3
Manejo de Base de datos

Practica 8

Gestión de Roles de Usuario, Control de Acceso y Navegación

PRESENTA:
Camacho Lopez Valeria

C22211475

DOCENTE:
Fortunato Ramírez Arzate

TIJUANA, B. C, A 22 DE NOVIEMBRE DEL 2024

Objetivo:

Implementar:

1. Tipos de usuario con distintos permisos y accesos.
2. Un sistema de registro protegido por un código especial de acceso proporcionado por el administrador.
3. Una barra de navegación para facilitar la navegación según el tipo de usuario.

Requisitos

- Haber completado la Práctica 7.

Estructura de Archivos

La estructura del proyecto será la siguiente:

proyecto/

```
└── server.js
└── public/
    ├── index.html
    ├── login.html
    ├── registro.html
    ├── styles.css
    └── navbar.html // Nueva página para la barra de navegación
└── package.json
└── node_modules/
```

Paso 1: Modificar la Base de Datos para los Roles de Usuario

1. Tabla de Usuarios: Agrega una columna tipo_usuario en la tabla usuarios, que defina el rol. Ejemplo:
 - admin: Acceso completo (puede ver, editar, eliminar, y gestionar otros usuarios).
 - medico: Acceso para consultar y actualizar registros de pacientes.
 - paciente: Acceso limitado para ver su propia información.
2. Tabla de Códigos de Registro: Crea una tabla para almacenar códigos de acceso con sus permisos:

```
CREATE TABLE codigos_acceso (
  codigo VARCHAR(10) PRIMARY KEY,
  tipo_usuario VARCHAR(20) NOT NULL
);
```

```
mysql> CREATE TABLE codigos_acceso (
    ->     codigo VARCHAR(10) PRIMARY KEY,
    ->     tipo_usuario VARCHAR(20) NOT NULL
    -> );
```

- Los códigos pueden ser únicos para cada tipo de usuario y se requerirán en el formulario de registro.

Paso 2: Configurar Rutas con Control de Acceso en server.js

1. Agregar Middleware para Autorización de Roles: Define una función requireRole para validar que el usuario tiene el rol necesario para acceder a ciertas rutas.

```

function requireRole(role) {
    return (req, res, next) => {
        if (req.session.user && req.session.user.tipo_usuario === role) {
            next();
        } else {
            res.status(403).send('Acceso denegado');
        }
    };
}

31  function requireLogin(req, res, next) {
32      if (!req.session.user) {
33          return res.redirect('/login.html');
34      }
35      next();
36  }
37
38  function requireRole(role,role) {
39      return (req, res, next) => {
40          if (req.session.user && req.session.user.tipo_usuario === role,role) {
41              next();
42          } else {
43              res.status(403).send('Acceso denegado');
44          }
45      };
46  }
47
48  app.use(bodyParser.urlencoded({ extended: true }));
49

```

Se agregó una nueva función para que ahora dependiendo del role asignado al usuario, pueda ver determinada información y otra este restringida.

2. Rutas Protegidas por Rol: Configura rutas específicas para cada tipo de usuario. Por ejemplo:

```

// Ruta para que solo admin pueda ver todos los usuarios
app.get('/ver-usuarios', requireLogin, requireRole('admin'), (req, res) =>
{
    const query = 'SELECT * FROM usuarios';
    connection.query(query, (err, results) => {
        if (err) return res.send('Error al obtener usuarios');
        res.send(results);
    });
});

// Ruta para que los médicos puedan ver pacientes
app.get('/ver-pacientes', requireLogin, requireRole('medico'), (req, res) => {
    // Lógica de consulta para ver pacientes
});

```

```

125
126 // Ruta para que solo admin pueda ver todos los usuarios
127 app.get('/ver-usuarios', requireLogin, requireRole('admin','medico'), (req, res) => {
128   const query = 'SELECT * FROM usuarios';
129   connection.query(query, (err, results) => {
130     if (err) {
131       return res.send('Error al obtener usuarios');
132     }
133
134     let html = `
135       <html>
136         <head>
137           <link rel="stylesheet" href="/styles.css">
138           <title>Usuarios Registrados</title>
139         </head>
140         <body>
141           <h1>Usuarios Registrados</h1>
142           <table>
143             <thead>
144               <tr>
145                 <th>Id</th>
146                 <th>Nombre</th>
147                 <th>Tipo de usuario</th>
148               </tr>
149             </thead>
150             <tbody>
151           `;
152   });
153
154   results.forEach(usuario => {
155     html += `
156       <tr>
157         <td>${usuario.id}</td>
158         <td>${usuario.nombre_usuario}</td>
159         <td>${usuario.tipo_usuario}</td>
160       </tr>
161     `;
162   });
163
164   html += `
165     </tbody>
166   </table>
167   <button onclick="window.location.href='/'>Volver</button>
168 </body>
169 </html>
170 `;
171
172   res.send(html);
173 });
174 });
175

```

Es necesario agregar el formato de una table html para que la ruta pueda mostrar los datos obtenidos de la base de datos, es decir, la lógica de consulta.

En este caso, esta ruta se hizo para que tanto los administradores como el medico puedan consultar los usuarios registrados en la base de datos.

Paso 3: Configurar Registro con Código de Acceso

- Modificación en registro.html: Añade un campo para ingresar el código de acceso en el formulario de registro.

```

<form action="/registro" method="POST">
  <label for="username">Nombre de usuario:</label>
  <input type="text" id="username" name="username" required>
  <label for="password">Contraseña:</label>
  <input type="password" id="password" name="password" required>
  <label for="codigo_acceso">Código de acceso:</label>
  <input type="text" id="codigo_acceso" name="codigo_acceso" required>
  <button type="submit">Registrarse</button>
</form>

```

```

public > registro.html > HTML
1  <!DOCTYPE html>
2  <html lang="es">
3    <head>
4      <meta charset="UTF-8">
5      <link rel="stylesheet" href="styles.css">
6      <title>Registro de Usuario</title>
7    </head>
8    <body>
9      <h2>Registro de Usuario</h2>
10     <form action="/registro" method="POST">
11       <label for="nombre_usuario">Nombre de Usuario:</label>
12       <input type="text" id="nombre_usuario" name="nombre_usuario" required>
13       <br>
14       <label for="password">Contraseña:</label>
15       <input type="password" id="password" name="password" required>
16       <br>
17       <label for="codigo_acceso">Código de acceso:</label>
18       <input type="text" id="codigo_acceso" name="codigo_acceso" required>
19       <button type="submit">Registrarse</button>
20     </form>
21     <p>¿Ya tienes cuenta? <a href="/login.html">Inicia sesión</a></p>
22   </body>
23 </html>

```

Se realizan cambios en el archive de registro, además se agrega la línea de código de styles para que esa interfaz también tenga una apariencia más amigable con el usuario. Es importante colocar .html a login para que redirija a esa pagina.

Validación de Código en el Registro: En server.js, agrega la lógica para verificar el código de acceso.

```
app.post('/registro', (req, res) => {
    const { username, password, codigo_acceso } = req.body;

    const query = 'SELECT tipo_usuario FROM codigos_acceso WHERE codigo = ?';
    connection.query(query, [codigo_acceso], (err, results) => {
        if (err || results.length === 0) {
            return res.send('Código de acceso inválido');
        }

        const tipo_usuario = results[0].tipo_usuario;
        const hashedPassword = bcrypt.hashSync(password, 10);

        const insertUser = 'INSERT INTO usuarios (username, password, tipo_usuario) VALUES (?, ?, ?)';
        connection.query(insertUser, [username, hashedPassword, tipo_usuario], (err) => {
            if (err) return res.send('Error al registrar usuario');
            res.redirect('/login.html');
        });
    });
});
```

```
// Registro de usuario
app.post('/registrar', (req, res) => {
  const { nombre_usuario, password, codigo_acceso } = req.body;
  const query = 'SELECT tipo_usuario FROM codigos_acceso WHERE codigo = ?';
  connection.query(query, [codigo_acceso], (err, results) => {
    if (err || results.length === 0) {
      return res.send('Código de acceso inválido');
    }

    const tipo_usuario = results[0].tipo_usuario;
    const hashedPassword = bcrypt.hashSync(password, 10);

    const insertUser = 'INSERT INTO usuarios (nombre_usuario, password_hash, tipo_usuario) VALUES (?, ?, ?)';
    connection.query(insertUser, [nombre_usuario, hashedPassword, tipo_usuario], (err) => {
      if (err) return res.send('Error al registrar usuario');
      res.redirect('/login.html');
    });
  });
});
```

Es importante señalar que las rutas que empiezan con **app.post** son aquellas las que nos van a redirigir a una página en la que solo se van a mostrar datos previamente registrados, mientras que las rutas que inicien con **app.get** serán aquellas en forma de formulario en las que nos permitirán registrar datos para su posterior consulta.

Paso 4: Modificación en el Código de Inicio de Sesión

En el código de la ruta de inicio de sesión (/login), después de validar que la contraseña es correcta, agrega el tipo_usuario a req.session.user. Así, cada vez que un usuario inicie sesión, la información de su tipo de usuario se guardará en la sesión y estará disponible para todas las rutas protegidas.

Por ejemplo:

```
app.post('/login', (req, res) => {
  const { username, password } = req.body;

  // Consulta para obtener el usuario y su tipo
  const query = 'SELECT * FROM usuarios WHERE username = ?';
  connection.query(query, [username], (err, results) => {
    if (err) {
      return res.send('Error al obtener el usuario');
    }

    if (results.length === 0) {
      return res.send('Usuario no encontrado');
    }

    const user = results[0];

    // Verificar la contraseña
    const isPasswordValid = bcrypt.compareSync(password, user.password);
    if (!isPasswordValid) {
      return res.send('Contraseña incorrecta');
    }

    // Almacenar la información del usuario en la sesión
    req.session.user = {
      id: user.id,
      username: user.username,
      tipo_usuario: user.tipo_usuario // Aquí se establece el tipo de
    usuario en la sesión
    };

    // Redirigir al usuario a la página principal
    res.redirect('/');
  });
});
```

```

59
60 // Iniciar sesión
61 app.post('/login', (req, res) => {
62   const { nombre_usuario, password } = req.body;
63
64   // Consulta para obtener el usuario y su tipo
65   const query = 'SELECT * FROM usuarios WHERE nombre_usuario = ?';
66   connection.query(query, [nombre_usuario], (err, results) => {
67     if (err) {
68       return res.send('Error al obtener el usuario');
69     }
70
71     if (err || results.length === 0) {
72       return res.send('Usuario no encontrado.');
73     }
74
75     const usuarios= results[0];
76
77     // Verificar la contraseña
78     const isPasswordValid = bcrypt.compareSync(password, usuarios.password_hash);
79     if (!isPasswordValid) {
80       return res.send('Contraseña incorrecta');
81     }
82
83     // Almacenar la información del usuario en la sesión
84     req.session.user = {
85       id: usuarios.id,
86       username: usuarios.nombre_usuario,
87       tipo_usuario: usuarios.tipo_usuario // Aquí se establece el tipo de usuario en la sesión
88     };
89     //Redirigir al usuario a la pagina principal
90     res.redirect('/');
91   });
92 });

```

Paso 5: Crear y Configurar una Barra de Navegación Dinámica (navbar.html) con Petición AJAX desde el Navegador

En este paso, vamos a crear una barra de navegación dinámica que se adapte al tipo de usuario. La barra de navegación se construirá en un archivo aparte llamado navbar.html, y el contenido cambiará según los permisos del usuario (por ejemplo, admin, médico, paciente). Para obtener el tipo de usuario, haremos una solicitud AJAX desde el navegador hacia el servidor, de forma que podamos controlar y mostrar las opciones de navegación apropiadas en la página principal.

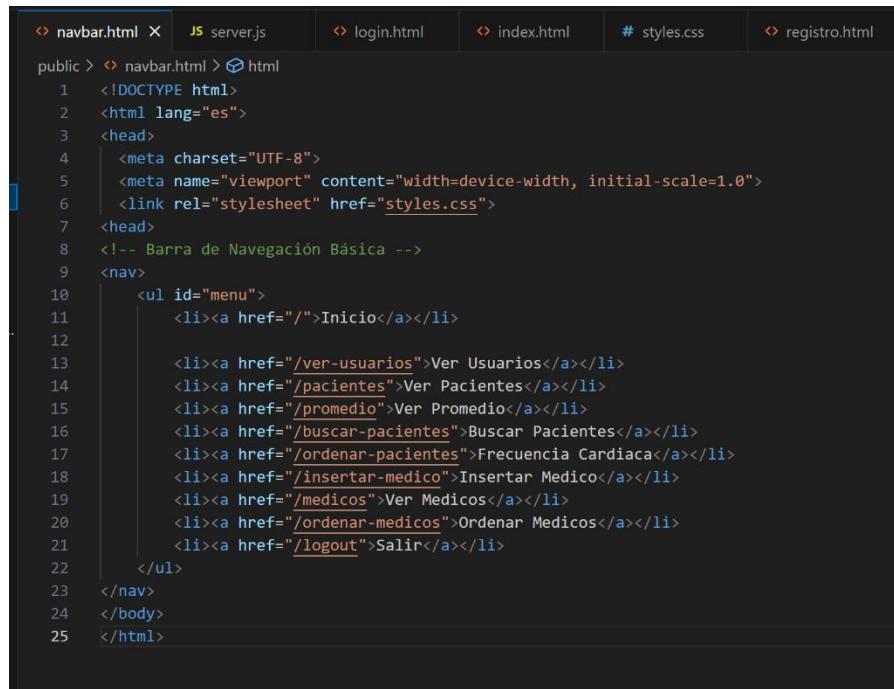
1. Crear el archivo de la barra de navegación navbar.html

En la carpeta public, crea un archivo llamado navbar.html. Este archivo contendrá los enlaces de navegación básicos y será el mismo para todos los usuarios. Los enlaces

adicionales se añadirán dinámicamente según el tipo de usuario, que será obtenido mediante AJAX desde el servidor.

Contenido de navbar.html:

```
<!-- Barra de Navegación Básica -->
<nav>
    <ul id="menu">
        <li><a href="/">Inicio</a></li>
    </ul>
</nav>
```



```
public > navbar.html > JS server.js > login.html > index.html > styles.css > registro.html
1  <!DOCTYPE html>
2  <html lang="es">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <link rel="stylesheet" href="styles.css">
7  </head>
8  <!-- Barra de Navegación Básica -->
9  <nav>
10     <ul id="menu">
11         <li><a href="/">Inicio</a></li>
12
13         <li><a href="/ver-usuarios">Ver Usuarios</a></li>
14         <li><a href="/pacientes">Ver Pacientes</a></li>
15         <li><a href="/promedio">Ver Promedio</a></li>
16         <li><a href="/buscar-pacientes">Buscar Pacientes</a></li>
17         <li><a href="/ordenar-pacientes">Frecuencia Cardíaca</a></li>
18         <li><a href="/insertar-medico">Insertar Médico</a></li>
19         <li><a href="/medicos">Ver Médicos</a></li>
20         <li><a href="/ordenar-medicos">Ordenar Médicos</a></li>
21         <li><a href="/logout">Salir</a></li>
22     </ul>
23 </nav>
24 </body>
25 </html>
```

Es necesario agregar cada botón que se desea mostrar en la barra didáctica de navegación en la página principal, acompañado del nombre asignado en el server, así como la línea de styles.

2. Configurar una ruta en el servidor para enviar el tipo de usuario

En server.js, crea una nueva ruta que envíe el tipo de usuario (admin, médico, o paciente) al cliente en formato JSON. Esta ruta será solicitada desde el navegador para determinar qué opciones de navegación mostrar.

```
// Ruta para obtener el tipo de usuario actual
app.get('/tipo-usuario', requireLogin, (req, res) => {
    res.json({ tipo_usuario: req.session.user.tipo_usuario });
});
```

```
120
121 // Ruta para obtener el tipo de usuario actual
122 app.get('/tipo-usuario', requireLogin, (req, res) => {
123   | res.json({ tipo_usuario: req.session.user.tipo_usuario });
124 });
125
```

Esta ruta se protegerá con requireLogin para asegurar que solo los usuarios autenticados puedan acceder a la información del tipo de usuario.

3. Incorporar navbar.html en index.html

Ahora, en index.html, incluimos navbar.html utilizando JavaScript. Esto permite cargar el menú básico y posteriormente hacer una solicitud AJAX para actualizar el menú según el tipo de usuario.

Contenido de index.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Página Principal</title>
  <link rel="stylesheet" href="styles.css">
</head>
<body>
  <!-- Incluir barra de navegación desde navbar.html -->
  <div id="navbar"></div>

  <script>
    // Insertar el contenido de navbar.html en el elemento con id "navbar"
    fetch('/navbar.html')
      .then(response => response.text())
      .then(data => {
        document.getElementById('navbar').innerHTML = data;
      })
      .catch(error => console.error('Error cargando el navbar:', error));
  </script>
</body>
</html>
```

```

public > <html lang="es">
9   <body>
64
65
66   <script>
67     // Insertar el contenido de navbar.html en el elemento con id "navbar"
68     fetch('/navbar.html')
69       .then(response => response.text())
70       .then(data => {
71         document.getElementById('navbar').innerHTML = data;
72       })
73       .catch(error => console.error('Error cargando el navbar:', error));
74
75     // Solicitar el tipo de usuario y ajustar el menú en función de este
76     fetch('/tipo-usuario')
77       .then(response => response.json())
78       .then(data => {
79         const menu = document.getElementById('menu');
80         const tipoUsuario = data.tipo_usuario;
81
82         // Agregar opciones de menú según el tipo de usuario
83         if (tipoUsuario === 'admin') {
84           menu.innerHTML += '<li><a href="/ver-usuarios">Ver Usuarios</a></li>';
85           menu.innerHTML += '<li><a href="/gestionar-registros">Gestionar Registros</a></li>';
86         } else if (tipoUsuario === 'medico') {
87           menu.innerHTML += '<li><a href="/ver-pacientes">Ver Pacientes</a></li>';
88           menu.innerHTML += '<li><a href="/editar-pacientes">Editar Pacientes</a></li>';
89         } else if (tipoUsuario === 'paciente') {
90           menu.innerHTML += '<li><a href="/ver-mis-datos">Mis Datos</a></li>';
91         }
92
93         // Opción de cerrar sesión para todos los tipos de usuario
94         menu.innerHTML += '<li><a href="/logout">Cerrar Sesión</a></li>';
95       })
96       .catch(error => console.error('Error obteniendo el tipo de usuario:', error));
97     </script>
98
99   </body>
100 </html>

```

En este paso, estamos solicitando navbar.html y añadiéndolo al elemento div con id="navbar".

4. Hacer una Solicitud AJAX para Obtener el Tipo de Usuario y Actualizar el Menú de Navegación

Después de cargar navbar.html, se necesita realizar una segunda solicitud para obtener el tipo de usuario desde el servidor y actualizar el menú de navegación con las opciones apropiadas.

En el mismo script de index.html, añade el siguiente código:

```

<script>
  // Solicitar el tipo de usuario y ajustar el menú en función de este
  fetch('/tipo-usuario')
    .then(response => response.json())
    .then(data => {
      const menu = document.getElementById('menu');
      const tipoUsuario = data.tipo_usuario;

      // Agregar opciones de menú según el tipo de usuario
      if (tipoUsuario === 'admin') {
        menu.innerHTML += '<li><a href="/ver-usuarios">Ver Usuarios</a></li>';
      }
    })
    .catch(error => console.error('Error obteniendo el tipo de usuario:', error));
</script>

```

```

        menu.innerHTML += '<li><a href="/gestionar-registros">Gestionar Registros</a></li>';
    } else if (tipoUsuario === 'medico') {
        menu.innerHTML += '<li><a href="/ver-pacientes">Ver Pacientes</a></li>';
        menu.innerHTML += '<li><a href="/editar-pacientes">Editar Pacientes</a></li>';
    } else if (tipoUsuario === 'paciente') {
        menu.innerHTML += '<li><a href="/ver-mis-datos">Mis Datos</a></li>';
    }

    // Opción de cerrar sesión para todos los tipos de usuario
    menu.innerHTML += '<li><a href="/logout">Cerrar Sesión</a></li>';
}
.catch(error => console.error('Error obteniendo el tipo de usuario:', error));
</script>

```

Con este código, se realiza una solicitud fetch a la ruta /tipo-usuario para obtener el tipo de usuario y, según su valor, se añaden las opciones correspondientes al menú.

5. Estilizar el Menú en styles.css

Puedes añadir estilos a la barra de navegación en styles.css para darle un diseño uniforme y atractivo. A continuación, se presenta un ejemplo básico:

```

/* Estilo para el contenedor de la barra de navegación */
nav {
    background-color: #333;
    color: white;
    padding: 10px;
}

/* Estilo para los enlaces del menú */
nav ul {
    list-style-type: none;
    margin: 0;
    padding: 0;
    display: flex;
}

nav ul li {
    margin-right: 20px;
}

nav ul li a {
    color: white;
    text-decoration: none;
    padding: 5px 10px;
}

nav ul li a:hover {

```

```

background-color: #555;
border-radius: 4px;
}

```

```

82    /* Estilo para el contenedor de la barra de navegación */
83    nav {
84        background-color: #333;
85        color: white;
86        padding: 10px;
87    }
88
89    /* Estilo para los enlaces del menú */
90    nav ul {
91        list-style-type: none;
92        margin: 0;
93        padding: 0;
94        display: flex;
95    }
96
97    nav ul li {
98        margin-right: 20px;
99    }
100
101   nav ul li a {
102       color: white;
103       text-decoration: none;
104       padding: 5px 10px;
105   }
106
107
108  nav ul li a:hover {
109      background-color: #555;
110      border-radius: 4px;
111  }

```

Se agregan nuevos apartados para que la barra de navegación sea amigable visualmente y este acorde al tema ya establecido en prácticas anteriores, se añade un fondo oscuro a la barra y se resalta el botón cuando el cursor se desplace sobre ellas.

Paso 6: Prueba de Funcionalidad y Control de Acceso

1. Registro de Usuarios con Código de Acceso: Registra usuarios utilizando diferentes códigos de acceso para cada tipo de usuario.

MEDICO:

The screenshot shows a web browser window with the URL localhost:3000/registro.html. The page title is "Registro de Usuario". There are three input fields: "Nombre de Usuario" containing "sergio jaramillo", "Contraseña" containing three dots (...), and "Código de acceso" containing "ABCD". A large green "Registrar" button is at the bottom. The browser's address bar and various icons are visible at the top.

Registro de Usuario

Nombre de Usuario:
angelica lopez

Contraseña:
...

Código de acceso:
EDFG

Registrar

¿Ya tienes cuenta? [Inicia sesión](#)

Registro de Usuario

Nombre de Usuario:
gabriela

Contraseña:
...

Código de acceso:
HJK

Registrar

¿Ya tienes cuenta? [Inicia sesión](#)

2. Prueba de Control de Acceso:

- Intenta acceder a rutas protegidas con diferentes usuarios y roles.
- Verifica que el acceso es permitido o denegado correctamente según el tipo de usuario.

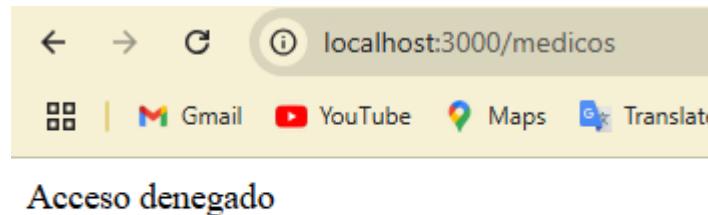
MEDICO: tiene acceso a

- ver los pacientes registrados

ID	Nombre	Edad	Frecuencia Cardíaca (bpm)	Peso (kg)	Estatura (cm)
1	alicia	20	88	null	null
2	alicia	20	85	null	null
3	nicole	22	97	null	null
4	emilia	20	95	null	null
5	fernanda	21	94	null	null
6	armando	23	97	null	null
7	michelle	20	93	null	null
8	armando	23	94	75	181
9	cristina	20	95	56	168
10	valeria	20	90	55	169
11	diego	21	93	60	165
12	Karla	20	88	55	167
13	kevin	24	85	70	178
14	vianney	19	80	55	166

Volver

Médico NO puede consultar a los médicos registrados



- Agrega las rutas y la funcionalidad de cada ruta de acuerdo a lo que consideres necesario además de la que se tienen de la práctica anterior: ver, ordenar, eliminar, etc., de acuerdo a cada usuario.

3. Barra de Navegación Condicional: Comprueba que solo se muestran los enlaces correspondientes al tipo de usuario que ha iniciado sesión.



CONCLUSION:

En esta práctica en específico si se presentaron más problemas de los esperados, principalmente por el análisis de los códigos al complementarlos con prácticas anteriores complementar las líneas de código que hacían falta; también se presentaron problemas al momento de asignar variables ya que no estaban escritas adecuadamente y por lo tanto no coincidía con la base de datos y no podía hacerse la consulta exitosamente.

Fue interesante conocer nuevas maneras de hacer interfaces conectadas a un servidor y una base de datos externa implementando roles y códigos de acceso determinados, al igual en cuenta a las contraseñas con hash como método de seguridad.