

C.3 Laboratorio Semana 3- Angel Gabriel Ortega

Siguiendo el caso de uso de un hospital, crear las tablas necesarias y realizar lo siguiente

Consultas a la Base de Datos con JDBC desde una aplicación Java:

- Establecer una conexión

Solucion:

primero que todo debemos inyectar la dependencia respectiva, esto puede variar si usamos un proyecto con un gestor de dependencias ya sea graddle o maven, en mi caso cree un proyecto maven y mediante la siguiente etiqueta instalamos la dependencia.

```
<dependencies>
  <!-- https://mvnrepository.com/artifact/com.mysql/mysql-connector-j
-->
  <dependency>
    <groupId>com.mysql</groupId>
    <artifactId>mysql-connector-j</artifactId>
    <version>8.4.0</version>
  </dependency>
</dependencies>
```

el conector lo instalamos desde el repositorio de maven oficial
[conector](#)

en cuanto a la conexion a la base de datos con java solo con un try catch podemos solucionar esto:

```
public class conex {
    public static Connection getConnection() {
        Connection conexion = null;
        var baseDatos = "hospitalx";
        var url = "jdbc:mysql://localhost:3306/" + baseDatos;
        var usuario = "root";
        var password = "winchi01";

        //aqui importo la clase del driver de jdbc en la memoria
```

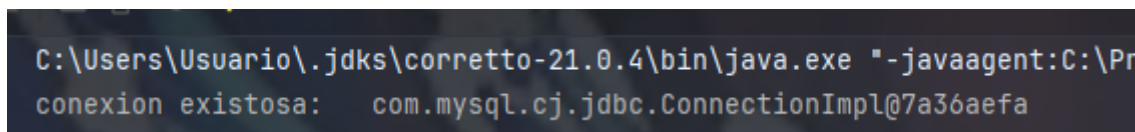
```

    try {
        Class.forName("com.mysql.cj.jdbc.Driver");
        conexion = DriverManager.getConnection(url, usuario, password);
    } catch (ClassNotFoundException | SQLException e){
        System.out.println("paso un error durante la coneccion revisa
bien las credenciales"
        + e.getMessage());
    }
    return conexion;
}

public static void main(String[] args) {
    var conexion = com.project.config.conex.getConnection();
    if (conexion != null){
        System.out.println("conexion existosa:  " + conexion);
    }else {
        System.out.println("error a la hora de conectarse");
    }
}
}

```

dependiendo de los datos la variable conexion se encargara de determinar si la conexion se realizo correctante o fallo algun campo, en mi caso fue el siguiente resultado:



```

C:\Users\Usuario\.jdk\corretto-21.0.4\bin\java.exe -javaagent:C:\Pr...
conexion existosa:  com.mysql.cj.jdbc.ConnectionImpl@7a36aefa

```

- Ejecutar comandos SQL como operaciones CRUD aplicado en la tabla "Cita" o "CitaMedica"

Solucion:

para esto necesitamos una clase para poder reflejar la entidad de la bd en nuestro mapeador que en este caso usaremos dao o data access object.

```

package com.project.domain;

import java.time.LocalDateTime;

public class Appointment {
    private int idAppointment;
    private int idDoctor;

```

```
private int idPatient;
private LocalDateTime appointmentDate;
private String status;
private String reason;

public Appointment() {}

public Appointment(int idAppointment) {
    this.idAppointment = idAppointment;
}

public Appointment(int idDoctor, int idPatient, LocalDateTime
appointmentDate, String status, String reason) {
    this.idDoctor = idDoctor;
    this.idPatient = idPatient;
    this.appointmentDate = appointmentDate;
    this.status = status;
    this.reason = reason;
}

public Appointment(int idAppointment, int idDoctor, int idPatient,
LocalDateTime appointmentDate, String status, String reason) {
    this.idAppointment = idAppointment;
    this.idDoctor = idDoctor;
    this.idPatient = idPatient;
    this.appointmentDate = appointmentDate;
    this.status = status;
    this.reason = reason;
}

public int getIdAppointment() {
    return idAppointment;
}

public void setIdAppointment(int idAppointment) {
    this.idAppointment = idAppointment;
}

public int getIdDoctor() {
    return idDoctor;
}

public void setIdDoctor(int idDoctor) {
```

```
        this.idDoctor = idDoctor;
    }

    public int getIdPatient() {
        return idPatient;
    }

    public void setIdPatient(int idPatient) {
        this.idPatient = idPatient;
    }

    public LocalDateTime getAppointmentDate() {
        return appointmentDate;
    }

    public void setAppointmentDate(LocalDateTime appointmentDate) {
        this.appointmentDate = appointmentDate;
    }

    public String getStatus() {
        return status;
    }

    public void setStatus(String status) {
        this.status = status;
    }

    public String getReason() {
        return reason;
    }

    public void setReason(String reason) {
        this.reason = reason;
    }

    @Override
    public String toString() {
        return "Appointment{" +
            "idAppointment=" + idAppointment +
            ", idDoctor=" + idDoctor +
            ", idPatient=" + idPatient +
            ", appointmentDate=" + appointmentDate +
            ", status='" + status + '\'' +
        }
    }
}
```

```

        ", reason='" + reason + '\\'' +
        '}}';
    }
}

```

- Llamadas a procedimientos almacenados (Cree un procedimiento almacenado que obtenga el numero de recetas de pacientes mayores a 18 años mandando como parametro de entrada en ID del paciente)

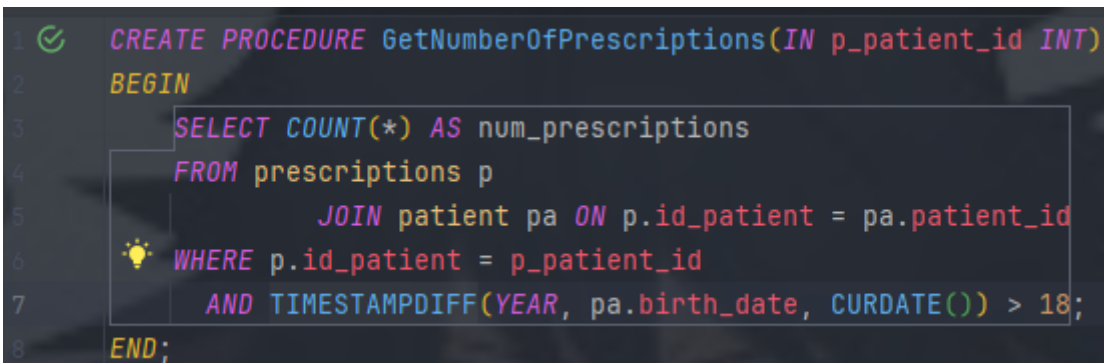
Solucion:

para solucionar esto tenemos que realizar unos cuantos pasos, primero crear el procedimiento almacenado en la base de datos

```

CREATE PROCEDURE GetNumberOfPrescriptions(IN p_patient_id INT)
BEGIN
    SELECT COUNT(*) AS num_prescriptions
    FROM prescriptions p
        JOIN patient pa ON p.id_patient = pa.patient_id
    WHERE p.id_patient = p_patient_id
        AND TIMESTAMPDIFF(YEAR, pa.birth_date, CURDATE()) > 18;
END;

```



```

1 CREATE PROCEDURE GetNumberOfPrescriptions(IN p_patient_id INT)
2 BEGIN
3     SELECT COUNT(*) AS num_prescriptions
4     FROM prescriptions p
5         JOIN patient pa ON p.id_patient = pa.patient_id
6     WHERE p.id_patient = p_patient_id
7         AND TIMESTAMPDIFF(YEAR, pa.birth_date, CURDATE()) > 18;
8 END;

```

seguidamente tenemos que implementar este metodo para poder usarlo dentro de nuestra aplicacion, osea en nuestro dao, de la siguiente manera

```

package com.project.data;

import com.project.domain.Appointment;

import java.sql.*;
import java.time.LocalDate;
import java.util.ArrayList;
import java.util.List;

```

```

import static com.project.config.conex.getConnection;

public class AppointmentDAO {

    private static final String SQL_SELECT = "SELECT * FROM appointments
ORDER BY id_appointment";

    private static final String SQL_SELECT_BY_ID = "SELECT * FROM
appointments WHERE id_appointment = ?";

    private static final String SQL_INSERT = "INSERT INTO appointments
(id_doctor, id_patient, appointment_date, status, reason) VALUES (?, ?, ?,
?, ?)";

    private static final String SQL_UPDATE = "UPDATE appointments SET
id_doctor=?, id_patient=?, appointment_date=?, status=?, reason=? WHERE
id_appointment=?";

    private static final String SQL_DELETE = "DELETE FROM appointments WHERE
id_appointment=?";

    public List<Appointment> listar() {
        List<Appointment> appointmentList = new ArrayList<>();
        Connection con = getConnection();
        try (PreparedStatement ps = con.prepareStatement(SQL_SELECT);
ResultSet rs = ps.executeQuery()) {
            while (rs.next()) {
                Appointment appointment = new Appointment(
                    rs.getInt("id_appointment"),
                    rs.getInt("id_doctor"),
                    rs.getInt("id_patient"),

rs.getTimestamp("appointment_date").toLocalDateTime(),
                    rs.getString("status"),
                    rs.getString("reason")
                );
                appointmentList.add(appointment);
            }
        } catch (SQLException e) {
            System.out.println("Error al listar citas: " + e.getMessage());
        }
        return appointmentList;
    }

    public boolean buscarPorId(Appointment appointment) {
        Connection con = getConnection();
    }

```

```

        try (PreparedStatement ps = con.prepareStatement(SQL_SELECT_BY_ID))
        {
            ps.setInt(1, appointment.getIdAppointment());
            try (ResultSet rs = ps.executeQuery()) {
                if (rs.next()) {
                    appointment.setIdDoctor(rs.getInt("id_doctor"));
                    appointment.setIdPatient(rs.getInt("id_patient"));

                    appointment.setAppointmentDate(rs.getTimestamp("appointment_date").toLocalDate()
                    .toTime());

                    appointment.setStatus(rs.getString("status"));
                    appointment.setReason(rs.getString("reason"));
                    return true;
                }
            }
        } catch (SQLException e) {
            System.out.println("Error al buscar cita por ID: " +
e.getMessage());
        }
        return false;
    }

    public boolean agregar(Appointment appointment) {
        Connection con = getConection();
        try (PreparedStatement ps = con.prepareStatement(SQL_INSERT)) {
            ps.setInt(1, appointment.getIdDoctor());
            ps.setInt(2, appointment.getIdPatient());
            ps.setTimestamp(3,
java.sql.Timestamp.valueOf(appointment.getAppointmentDate()));
            ps.setString(4, appointment.getStatus());
            ps.setString(5, appointment.getReason());
            ps.executeUpdate();
            return true;
        } catch (SQLException e) {
            System.out.println("Error al agregar cita: " + e.getMessage());
        }
        return false;
    }

    public boolean modificar(Appointment appointment) {
        Connection con = getConection();
        try (PreparedStatement ps = con.prepareStatement(SQL_UPDATE)) {
            ps.setInt(1, appointment.getIdDoctor());

```

```

        ps.setInt(2, appointment.getIdPatient());
        ps.setTimestamp(3,
java.sql.Timestamp.valueOf(appointment.getAppointmentDate()));
        ps.setString(4, appointment.getStatus());
        ps.setString(5, appointment.getReason());
        ps.setInt(6, appointment.getIdAppointment());
        ps.executeUpdate();
        return true;
    } catch (SQLException e) {
        System.out.println("Error al modificar cita: " +
e.getMessage());
    }
    return false;
}

public boolean eliminar(Appointment appointment) {
    Connection con = getConnection();
    try (PreparedStatement ps = con.prepareStatement(SQL_DELETE)) {
        ps.setInt(1, appointment.getIdAppointment());
        ps.executeUpdate();
        return true;
    } catch (SQLException e) {
        System.out.println("Error al eliminar cita: " + e.getMessage());
    }
    return false;
}

public static void main(String[] args) {
    var appointmentDao = new AppointmentDAO();

    var nuevaCita = new Appointment(1, 2, LocalDateTime.now(),
"PENDING", "Consulta general");
    var agregado = appointmentDao.agregar(nuevaCita);
    if (agregado)
        System.out.println("Cita agregada: " + nuevaCita);
    else
        System.out.println("No se pudo agregar la cita: " + nuevaCita);

    var citaModificar = new Appointment(1, 1, 2,
LocalDateTime.now().plusDays(1), "CONFIRMED", "Cambio de horario");
    var modificado = appointmentDao.modificar(citaModificar);
    if (modificado)
        System.out.println("Cita modificada: " + citaModificar);
}

```



```

        else
            System.out.println("No se pudo modificar la cita: " +
citaModificar);

        var citaEliminar = new Appointment(1);
        var eliminado = appointmentDao.eliminar(citaEliminar);
        if (eliminado)
            System.out.println("Cita eliminada: " + citaEliminar);
        else
            System.out.println("No se pudo eliminar la cita: " +
citaEliminar);

        System.out.println("Listado de citas:");
        List<Appointment> citas = appointmentDao.listar();
        citas.forEach(System.out::println);

        var citaBuscar = new Appointment(1);
        var encontrado = appointmentDao.buscarPorId(citaBuscar);
        if (encontrado)
            System.out.println("Cita encontrada: " + citaBuscar);
        else
            System.out.println("No se encontró la cita con ID: " +
citaBuscar.getIdAppointment());
    }

    public int getNumberOfPrescriptions(int patientId) {
        int numPrescriptions = 0;
        String sql = "{CALL GetNumberOfPrescriptions(?)}";
        try (Connection con = getConection();
            CallableStatement cs = con.prepareCall(sql)) {
            cs.setInt(1, patientId);
            try (ResultSet rs = cs.executeQuery()) {
                if (rs.next()) {
                    numPrescriptions = rs.getInt("num_prescriptions");
                }
            }
        } catch (SQLException e) {
            System.out.println("Error al obtener el número de recetas: " +
e.getMessage());
        }
        return numPrescriptions;
    }
}

```

```
}
```

luego en la capa o mini UI por consola podemos usar libremente la interaccion que hicimos con estos datos:

```
package com.project.presentation;

import com.project.data.AppointmentDAO;
import com.project.domain.Appointment;

import java.time.LocalDateTime;
import java.util.List;
import java.util.Scanner;

public class AppointmentApp {
    public static void main(String[] args) {
        var salir = false;
        var consola = new Scanner(System.in);
        var appointmentDAO = new AppointmentDAO();

        while (!salir) {
            try {
                mostrarMenu();
                salir = ejecutarOpciones(consola, appointmentDAO);
            } catch (Exception e) {
                System.out.println("Ocurrió un error al ejecutar operación:
" + e.getMessage());
            }
            System.out.println();
        }
        consola.close();
    }

    private static void mostrarMenu() {
        System.out.print("""
            *** Sistema de Gestión de Citas ***
            1. Listar Citas
            2. Registrar Cita
            3. Modificar Cita
            4. Eliminar Cita
            5. Buscar Cita por ID
        """);
    }
}
```

```

        6. buscar recetas de mayores de 18 años
        7. Salir
        Elige una opción:\s
        """);
    }

    private static boolean ejecutarOpciones(Scanner consola, AppointmentDAO
appointmentDAO) {
        var opcion = Integer.parseInt(consola.nextLine());
        var salir = false;

        switch (opcion) {
            case 1 -> {
                System.out.println("Listado de Citas:");
                List<Appointment> citas = appointmentDAO.listar();
                citas.forEach(System.out::println);
            }
            case 2 -> {
                System.out.println("Registrar Cita:");
                System.out.print("ID del Doctor: ");
                var idDoctor = Integer.parseInt(consola.nextLine());
                System.out.print("ID del Paciente: ");
                var idPatient = Integer.parseInt(consola.nextLine());
                System.out.print("Fecha y Hora de la Cita (yyyy-MM-
ddTHH:mm:ss): ");
                var appointmentDate =
LocalDateTime.parse(consola.nextLine());
                System.out.print("Estado (PENDING/CONFIRMED/CANCELLED): ");
                var status = consola.nextLine();
                System.out.print("Motivo: ");
                var reason = consola.nextLine();

                var cita = new Appointment(0, idDoctor, idPatient,
appointmentDate, status, reason);
                var registrado = appointmentDAO.agregar(cita);
                if (registrado)
                    System.out.println("Cita registrada: " + cita);
                else
                    System.out.println("No se pudo registrar la cita.");
            }
            case 3 -> {
                System.out.println("Modificar Cita:");
                System.out.print("ID de la Cita a Modificar: ");

```

```

        var idAppointment = Integer.parseInt(consola.nextLine());
        System.out.print("ID del Doctor: ");
        var idDoctor = Integer.parseInt(consola.nextLine());
        System.out.print("ID del Paciente: ");
        var idPatient = Integer.parseInt(consola.nextLine());
        System.out.print("Fecha y Hora de la Cita (yyyy-MM-
ddTHH:mm:ss): ");

        var appointmentDate =
LocalDateTime.parse(consola.nextLine());
        System.out.print("Estado (PENDING/CONFIRMED/CANCELLED): ");
        var status = consola.nextLine();
        System.out.print("Motivo: ");
        var reason = consola.nextLine();

        var cita = new Appointment(idAppointment, idDoctor,
idPatient, appointmentDate, status, reason);
        var modificado = appointmentDAO.modificar(cita);
        if (modificado)
            System.out.println("Cita modificada: " + cita);
        else
            System.out.println("No se pudo modificar la cita.");
    }
    case 4 -> {
        System.out.println("Eliminar Cita:");
        System.out.print("ID de la Cita a Eliminar: ");
        var idAppointment = Integer.parseInt(consola.nextLine());
        var cita = new Appointment(idAppointment);
        var eliminado = appointmentDAO.eliminar(cita);
        if (eliminado)
            System.out.println("Cita eliminada: " + cita);
        else
            System.out.println("No se pudo eliminar la cita.");
    }
    case 5 -> {
        System.out.println("Buscar Cita por ID:");
        System.out.print("ID de la Cita: ");
        var idAppointment = Integer.parseInt(consola.nextLine());
        var cita = new Appointment(idAppointment);
        var encontrado = appointmentDAO.buscarPorId(cita);
        if (encontrado)
            System.out.println("Cita encontrada: " + cita);
        else
            System.out.println("No se encontró la cita con ID: " +

```

```

idAppointment);
    }
    case 6 -> {
        System.out.println("Buscar recetas de mayores de 18 años:");
        var numPrescriptions =
appointmentDAO.getNumberOfPrescriptions(18);
        System.out.println("Número de recetas: " +
numPrescriptions);
    }
    case 7 -> {
        System.out.println("Hasta luego!");
        salir = true;
    }
    default -> System.out.println("Opción no reconocida");
}
return salir;
}
}
}

```

- Manejo de excepciones (Investigacion)

el manejo de excepciones podría ser cuando hacemos commit o rollback después de un cambio o una compra y resulta que el cliente no tiene el dinero requerido para realizar la compra en este caso como podemos ver en el dao si o si cada operación crud y el procedimiento almacenado están dentro de un try catch para poder siempre capturar los mensajes y errores que obtengamos, el error lo mostramos por consola, un ejemplo sería el método que hace uso del procedimiento almacenado.

```

public int getNumberOfPrescriptions(int patientId) {
    int numPrescriptions = 0;
    String sql = "{CALL GetNumberOfPrescriptions(?)}";
    try (Connection con = getConnection();
        CallableStatement cs = con.prepareCall(sql)) {
        cs.setInt(1, patientId);
        try (ResultSet rs = cs.executeQuery()) {
            if (rs.next()) {
                numPrescriptions = rs.getInt("num_prescriptions");
            }
        }
    } catch (SQLException e) {
        System.out.println("Error al obtener el número de recetas: " +
e.getMessage());
    }
}

```

```
        return numPrescriptions;
    }
```

repositorio fuente para visualizar el proyecto:

[repositorio del proyecto](#)

ORMs para realizar operaciones desde nuestras aplicaciones (Investigacion)

- Copia o captura de algunos ejemplos básicos con algunas tecnologías ORMs (Hibernate, Entity Framework, NHibernate, etc)

Solucion:

como seria bastante largo hacer un proyecto de spring boot desde cero veo mas sencillo tomar un proyecto que desarrolle anteriormente con .net y explicar como se hace uso de entity framework un mapeador por excelencia en c# junto con dapper

```
using Dominio.Entities;
using Dominio.Interfaces;
using Microsoft.AspNetCore.Mvc.Rendering;
using Microsoft.EntityFrameworkCore;
using Persistencia;

namespace Aplicacion.Repository;

    public class
IngredienteRepository:GenericRepository<Ingrediente>, IIngrediente
    {

        private readonly PAppContext _context;

        public IngredienteRepository(PAppContext context) : base(context)
        {
            _context = context;
        }

        public async Task<IEnumerable<Ingrediente>> GetProductosAgotados(int
```

```

cantidad) =>

        await _context.Ingredientes
            .Where(x => x.Stock >= 400)
            .OrderByDescending(p => p.Stock)
            .Take(400)
            .ToListAsync();
    }
}

```

aquí podemos ver que usando entity framework hacemos una consulta asincrónica al contexto de la aplicación `_context` y seguidamente a la tabla donde queremos realizar la consulta, luego pasamos los requerimientos de nuestro día a día como `where` y `orderby` y las condiciones y finalmente convertirlo a una lista para poder devolverlo en un controlador donde queramos usarlo

```

[HttpGet]
[MapToApiVersion("1.1")]
[ProducesResponseType(StatusCodes.Status200OK)]
[ProducesResponseType(StatusCodes.Status400BadRequest)]
public async Task<ActionResult<Pager<IngredienteStock400>>>
Get11([FromQuery] Params productParams)
{
    var result = await _unitOfWork.Ingredientes
                                    .GetAllAsync(productParams.PageIndex,
productParams.PageSize,
                                    productParams.Search);

    var ingredienteslst = _mapper.Map<List<IngredienteStock400>>
(result.registros);
    Response.Headers.Add("X-InlineCount",
result.totalRegistros.ToString());

    return new Pager<IngredienteStock400>(ingredienteslst,
result.totalRegistros,
        productParams.PageIndex, productParams.PageSize,
productParams.Search);
}

```

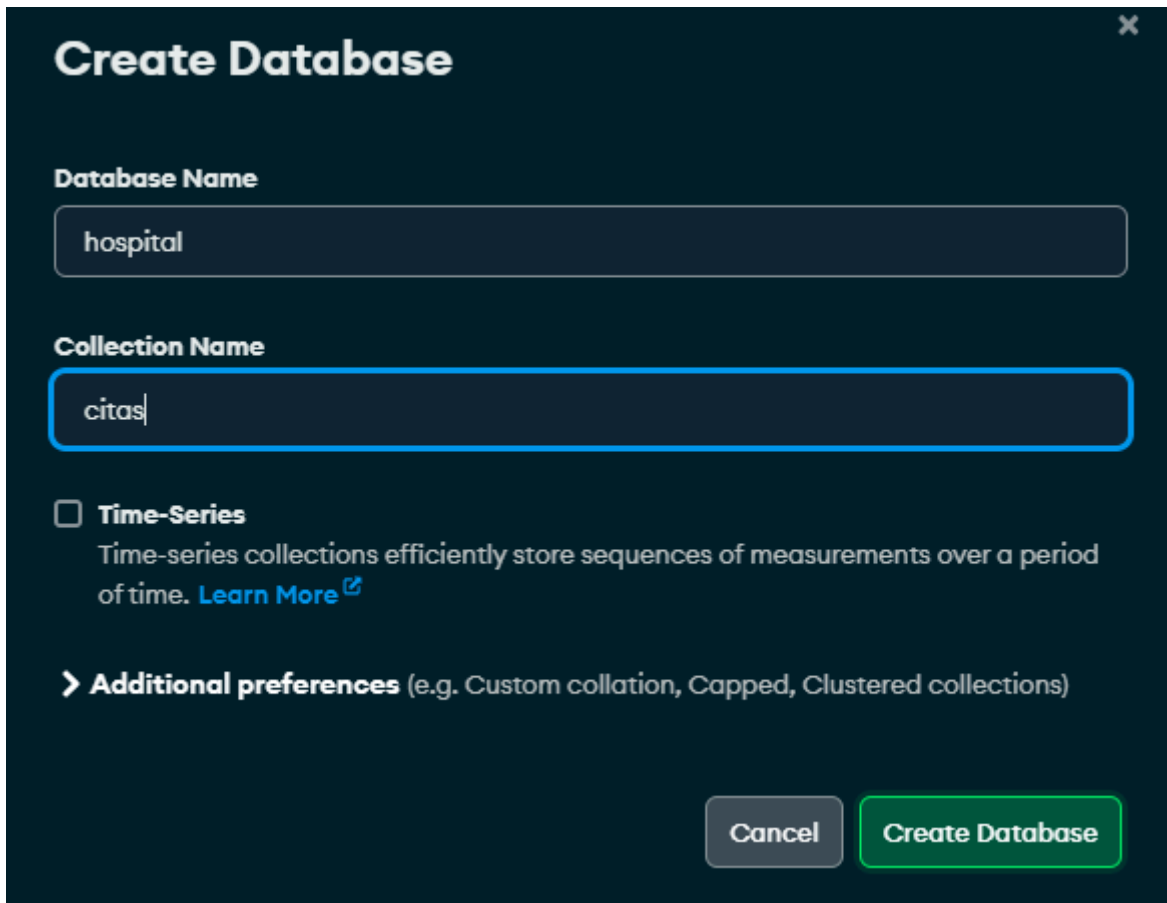
de esta manera usando entity framework podríamos hacer cualquier cosa que nos diera el gusto, desde stored procedures o incluso llamadas a triggers.

Realizar los siguientes pasos para poder tener la instalación de Mongo DB en sus ambientes de trabajo:

UI

- Crear una base de datos

para realizar esta accion es tan sencillo como solo dirigirnos a la ui y realizar lo siguiente, indicar el nombre de la bd y seguidamente, por defecto nos pide añadir una coleccion, en mi caso añadi citas



Create Database

Database Name

hospital

Collection Name

citas

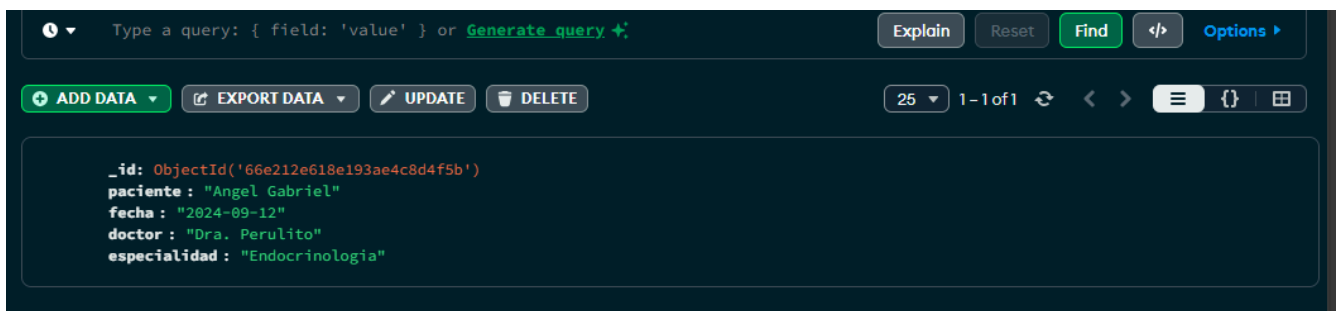
☐ **Time-Series**
Time-series collections efficiently store sequences of measurements over a period of time. [Learn More](#)

> Additional preferences (e.g. Custom collation, Capped, Clustered collections)

Cancel Create Database

- Agregar una colección

por defecto en el paso anterior se nos crea la coleccion base llamada citas. y podemos insertar campos a la coleccion ya mencionada



Type a query: { field: 'value' } or [Generate query](#)

Explain Reset Find </> Options

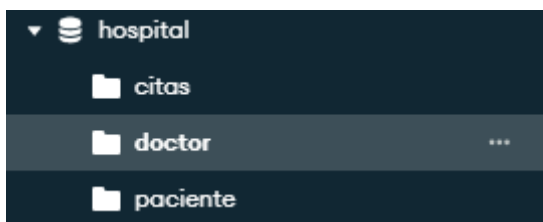
ADD DATA EXPORT DATA UPDATE DELETE

25 1-1 of 1

```
{
  "_id": ObjectId("66e212e618e193ae4c8d4f5b"),
  "paciente": "Angel Gabriel",
  "fecha": "2024-09-12",
  "doctor": "Dra. Perulito",
  "especialidad": "Endocrinologia"
}
```

- Agregar una lista de colecciones

creamos la lista de las colecciones de manera super sencilla



hospital

- citas
- doctor
- paciente

SHELL

- Agregar una colección

mediante el uso de el shell es todavia mucho mas sencillo realizar creacion de colecciones mediante el comando, donde seleccionamos la db, la coleccion y la accion:

```
db.paciente.insertOne({
  "nombre": "Juliana UnU",
  "edad": 19,
  "género": "Femenino",
});
```

si la insercion se realizo correctamente debe devolvernos el id de objeto creado:

```
> db.paciente.insertOne({
  "nombre": "Juliana UnU",
  "edad": 19,
  "género": "Femenino",
});
< {
  acknowledged: true,
  insertedId: ObjectId('66e22039856a8c94e71fbde5')
}
```

- Agregar una lista colección

anteriormente usamos **INSERT ONE** para poder insertar un registro pero si queremos añadir una lista a la coleccion solo debemos usar la notacion **INSERT MANY** la cual pasaremos en formato array debido a que json es basado en javascript

```
db.paciente.insertMany([
  {
    "nombre": "Alana",
    "edad": 25,
    "género": "Femenino",
  },
  {
    "nombre": "Jamilton",
    "edad": 35,
    "género": "Masculino",
  }
]);
```

al igual que en el paso anterior nos debe retornar el object id de que todo salio correctamente:

```

> db.paciente.insertMany([
  {
    "nombre": "Alana",
    "edad": 25,
    "género": "Femenino",
  },
  {
    "nombre": "Jamilton",
    "edad": 35,
    "género": "Masculino",
  }
]);
< {
  acknowledged: true,
  insertedIds: {
    '0': ObjectId('66e22055856a8c94e71fbde6'),
  }
}

```

- Listar una coleccion
esto se puede hacer relativamente sencillo, unicamente con la funcion find

```
db.nombredelaColeccion.find();
```

```

> db.paciente.find();
< {
  _id: ObjectId('66e2192418e193ae4c8d4f5f'),
  nombre: 'Angel Gabriel',
  edad: 18,
  'género': 'Masculino'
}
{
  _id: ObjectId('66e22039856a8c94e71fbde5'),
  nombre: 'Juliana UnU',
  edad: 19,
  'género': 'Femenino'
}
{
  _id: ObjectId('66e22055856a8c94e71fbde6'),
  nombre: 'Alana',
  edad: 25,
  'género': 'Femenino'
}
{
  _id: ObjectId('66e22055856a8c94e71fbde7'),
  nombre: 'Jamilton',
  edad: 35,
  'género': 'Masculino'
}

```

- Editar una coleccion

esto es bastante sencillo si no tenemos registros repetidos pero si queremos uno en especifico debemos hacer uso del object id como criterio de busqueda, por suerte en nuestro caso solo tenemos que pasar el nombre como criterio y todo se hace mas facil

```
db.paciente.updateone();
```

```

> db.paciente.updateOne(
  { "nombre": "Angel Gabriel" },
  {$set: {"edad": 19}}
)
< {
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}

```

- Eliminar una coleccion

de manera muy sencilla tal como lo haríamos en sql podemos hacer esto solo con el comando

DROP para esto me cree una nueva coleccion llamada pruebita y la elimine consecutivamente:

```
db.coleccion.drop();
```

```
> db.pruebita.insertOne({
  "non": "xd",
  "dr": 24
});
< {
  acknowledged: true,
  insertedId: ObjectId('66e223c4856a8c94e71fbde8')
}
> db.pruebita.drop();
< true
hospital> |
```

como podemos ver la coleccion se elimino correctamente

- Eliminar varias colecciones usando un atribut

para lograr esto debemos hacer que tengan un atributo igualado, ejemplo podriamos eliminar todas las pacientes que no sean mujeres.

```
db.paciente.deleteMany({ "género": { $ne: "Femenino" } });
```

Antes:

```
{
  _id: ObjectId('66e2192418e193ae4c8d4f5f'),
  nombre: 'Angel Gabriel',
  edad: 19,
  'género': 'Masculino'
}
{
  _id: ObjectId('66e22039856a8c94e71fbde5'),
  nombre: 'Juliana UnU',
  edad: 19,
  'género': 'Femenino'
}
{
  _id: ObjectId('66e22055856a8c94e71fbde6'),
  nombre: 'Alana',
  edad: 25,
  'género': 'Femenino'
}
{
  _id: ObjectId('66e22055856a8c94e71fbde7'),
  nombre: 'Jamilton',
  edad: 35,
  'género': 'Masculino'
}
```

Despues:

```
> db.paciente.find();
< {
  _id: ObjectId('66e22039856a8c94e71fbde5'),
  nombre: 'Juliana UnU',
  edad: 19,
  'género': 'Femenino'
}
{
  _id: ObjectId('66e22055856a8c94e71fbde6'),
  nombre: 'Alana',
  edad: 25,
  'género': 'Femenino'
}
```

podemos ver que eliminamos los registros que no tenian el genero femenino exitosamente.

Redactado y escrito por:

