Presentación

Nombre:

Abraham Almonte Pérez

Matricula:

100579156

Materia:

Lenguaje de Programación II

Seccion:

Z06

Maestro:

Silverio Del Orbe

Introducción

En el programa se desarrolla un sistema de gestión de películas basado en consola, que permite realizar operaciones CRUD (Crear, Leer, Actualizar y Eliminar) sobre los datos de las películas almacenadas en una base de datos. A través de una interfaz simple y directa, el sistema facilita la interacción con los usuarios para administrar información como el título y la descripción de las películas. El sistema está compuesto por varias capas, incluida una capa de presentación en consola (ConsoleUI), una capa de control (FilmController) que maneja la lógica de negocio, y una capa de acceso a datos que interactúa con la base de datos. Este proyecto está diseñado para demostrar la implementación de buenas prácticas en la gestión de bases de datos y la interacción entre la interfaz de usuario y la lógica de negocio en aplicaciones Java.

Sistema de Gestión de Películas

Descripción General

Este programa implementa un sistema de consulta y generación de reportes basado en la base de datos Sakila. El objetivo principal es obtener información de la tabla film y generar reportes utilizando una interfaz de consola sencilla.

El sistema incluye las siguientes funcionalidades principales:

-Conexión a la base de datos utilizando JDBC.

-Consulta a la tabla film para obtener información sobre películas.

-Generación de reportes con los datos extraídos.

-Una interfaz de consola para ejecutar las operaciones principales.

-Estructura del Programa

El programa está dividido en los siguientes paquetes y clases:

```
Paquete com.sakila.controllers:

Clase eportController:
```

Controlador principal que gestiona la obtención de datos y la generación de reportes.

```
Paquete com.sakila.data:

Clase Film:
```

Representa el modelo de datos para la tabla film.

```
Paquete com.sakila.reports:

Clase ReportGenerator:
```

Genera reportes a partir de los datos proporcionados.

```
Paquete com.sakila.ui:

Clase ConsoleUI:
```

Interfaz de usuario que inicia el programa y gestiona la interacción con el usuario.

Clase `Film`

La clase Film es el modelo de datos que representa los registros de la tabla film en la base de datos. Cada objeto de esta clase corresponde a una película, con tres atributos principales: filmId, title y description.

```
public class Film extends DataContext<Film> {
    private int filmId;
    private String title;
    private String description;
    public Film(int filmId, String title, String description) {
        this.filmId = filmId;
        this.title = title;
        this.description = description;
    }

    public Film() {

    }

    public int getFilmId() { return filmId; }
    public void setFilmId(int filmId) { this.filmId = filmId; }
    public String getTitle() { return title; }
    public void setTitle(String title) { this.title = title; }
    public String getDescription() { return description; }
    public void setDescription(String description) { this.description =
description; }
```

Explicación:

Atributos: Los atributos filmId, title y description corresponden a las columnas de la tabla film.

Constructor: Permite inicializar un objeto Film con valores específicos.

Métodos Getter: Proveen acceso controlado a los atributos de la clase.

Relación con la Base de Datos: Cuando se consulta la tabla film, los datos obtenidos se convierten en instancias de esta clase, facilitando su manejo en Java.

Clase ReportController

Gestiona la conexión con la base de datos, la obtención de datos de la tabla film y la interacción con el generador de reportes.

Método connect() :

```
private Connection connect() {
    try {
```

```
        Class.forName("com.mysql.cj.jdbc.Driver");
        return
DriverManager.getConnection("jdbc:mysql://localhost:3306/sakila", "root",
"password");
    } catch (SQLException | ClassNotFoundException e) {
        e.printStackTrace();
        return null;
    }
}
```

Explicación:

Class.forName(): Carga el controlador JDBC de MySQL.

DriverManager.getConnection(): Establece la conexión con la base de datos Sakila.

"jdbc:mysql://localhost:3306/sakila": URL de conexión.

"root", "password": Credenciales de acceso. (Debes reemplazarlas con las correctas).

Método getFilmsFromDatabase() :

```
    private List<Film> getFilmsFromDatabase() {
        List<Film> films = new ArrayList<>();
        try (Connection connection = connect()) {
            if (connection != null) {
                Statement statement = connection.createStatement();
                ResultSet resultSet = statement.executeQuery("SHOW TABLES
LIKE 'film';");
                if (!resultSet.next()) {
                    System.err.println("La tabla 'film' no existe en la
base de datos.");
                    return films;
                }

                resultSet = statement.executeQuery("SELECT * FROM film");
                while (resultSet.next()) {
                    Film film = new Film(resultSet.getInt("film_id"),
resultSet.getString("title"), resultSet.getString("description"));
                    films.add(film);
                }
            } else {
                System.err.println("Conexion no establecida.");
            }
        } catch (SQLException e) {
            System.err.println("Error al obtener los datos: " +
e.getMessage());
        }
        return films;
    }
```

Explicación:

Statement: Objeto que ejecuta consultas SQL.

ResultSet: Contiene los resultados de la consulta.

Conversión de Datos:

resultSet.getInt("film_id"): Obtiene el ID de la película.

resultSet.getString("title"): Obtiene el título.

resultSet.getString("description"): Obtiene la descripción.

Cada fila del ResultSet se convierte en un objeto Film y se añade a la lista films.

Clase ConsoleUI

Actúa como la interfaz principal del programa, iniciando el flujo de ejecución.

```
package com.sakila.ui;

import com.sakila.controllers.FilmController;
import com.sakila.data.Film;
import java.util.List;
import java.util.Scanner;

public class ConsoleUI {
    private FilmController filmController;

    public ConsoleUI() {
        this.filmController = new FilmController();
    }

    public void displayMenu() {
        Scanner scanner = new Scanner(System.in);
        while (true) {
            System.out.println("Film Management");
            System.out.println("1. Add Film");
            System.out.println("2. Get Film");
            System.out.println("3. Update Film");
            System.out.println("4. Delete Film");
            System.out.println("5. List Films");
            System.out.println("6. Exit");

            int choice = scanner.nextInt();

            switch (choice) {
                case 1:
                    addFilm(scanner);
                    break;
```

```
                case 2:
                    getFilm(scanner);
                    break;
                case 3:
                    updateFilm(scanner);
                    break;
                case 4:
                    deleteFilm(scanner);
                    break;
                case 5:
                    listFilms();
                    break;
                case 6:
                    return;
            }
        }
    }

    public void addFilm(Scanner scanner) {
        System.out.print("Enter film title: ");
        scanner.nextLine(); // Limpiar el buffer del scanner
        String title = scanner.nextLine();
        System.out.print("Enter film description: ");
        String description = scanner.nextLine();

        Film film = new Film(0, title, description);
        filmController.addFilm(film);
    }


    public void getFilm(Scanner scanner) {
        System.out.print("Enter film ID: ");
        int id = scanner.nextInt();

        Film film = filmController.getFilm(id);
        if (film != null) {
            System.out.println(film.getTitle() + ": " +
film.getDescription());
        } else {
            System.out.println("Film not found.");
        }
    }

    public void updateFilm(Scanner scanner) {
        System.out.print("Enter film ID: ");
        int id = scanner.nextInt();
        scanner.nextLine(); // Limpiar el buffer del scanner

        Film film = filmController.getFilm(id);
```

```
        if (film != null) {
            System.out.print("Enter new title: ");
            String title = scanner.nextLine();
            System.out.print("Enter new description: ");
            String description = scanner.nextLine();

            film.setTitle(title);
            film.setDescription(description);

            filmController.updateFilm(film);
        } else {
            System.out.println("Film not found.");
        }
    }


    public void deleteFilm(Scanner scanner) {
        System.out.print("Enter film ID: ");
        int id = scanner.nextInt();
        filmController.deleteFilm(id);
    }

    public void listFilms() {
        List<Film> films = filmController.getAllFilms();
        for (Film film : films) {
            System.out.println(film.getTitle() + ": " +
film.getDescription());
        }
    }

    public static void main(String[] args) {
        ConsoleUI ui = new ConsoleUI();
        ui.displayMenu();
    }
}
```

La clase ConsoleUI actúa como la interfaz de usuario basada en consola para interactuar con el sistema de gestión de películas. Su propósito principal es permitir a los usuarios realizar operaciones CRUD (Crear, Leer, Actualizar y Eliminar) sobre los datos de películas almacenados en la base de datos a través de un menú interactivo.

El atributo filmController conecta la interfaz de usuario con la lógica de negocio, gestionando todas las interacciones con la base de datos. Este se inicializa en el constructor de la clase para garantizar que esté listo para manejar las operaciones.

El método displayMenu() es el núcleo de la interfaz de consola. Muestra un menú interactivo con opciones numeradas para realizar diferentes operaciones. Utiliza un bucle infinito que se rompe cuando el usuario selecciona la opción de salir. El menú emplea un bloque switch para redirigir la ejecución al método correspondiente según la elección del usuario.

Cada operación está implementada en un método específico:

addFilm() solicita al usuario el título y la descripción de una nueva película. Crea un objeto Film con esos datos y lo envía al controlador para que sea añadido a la base de datos.

getFilm() permite al usuario buscar una película ingresando su ID. Si se encuentra la película, se muestra el título y la descripción; si no, se informa que no existe.

updateFilm() solicita el ID de una película, verifica si existe, y luego permite al usuario modificar su título y descripción. Los nuevos datos se actualizan en la base de datos a través del controlador.

deleteFilm() elimina una película de la base de datos usando su ID.

listFilms() obtiene todas las películas almacenadas y las lista en consola, mostrando su título y descripción.

El método main() sirve como punto de entrada al programa. Crea una instancia de ConsoleUI y llama al método displayMenu(), iniciando la interacción con el usuario.

El diseño de esta clase asegura un flujo claro y modular. Cada operación está separada, lo que facilita el mantenimiento y la ampliación del programa. Sin embargo, carece de manejo avanzado de errores para entradas no válidas, como texto cuando se espera un número, lo cual podría mejorarse. Además, gestiona correctamente el uso del Scanner, limpiando el buffer después de leer números para evitar problemas con las entradas subsiguientes.

Dificultades Encontradas y Soluciones

Nombre Incorrecto de Tabla:

Problema: La consulta apuntaba a films en lugar de film.

Solución: Se corrigió el nombre de la tabla en las consultas SQL.

Clase Principal no Encontrada:

Problema: El programa no encontraba la clase ConsoleUI debido a un classpath mal configurado.

Solución: Se incluyeron las dependencias necesarias en el comando de ejecución.

Conexión Nula a la Base de Datos:

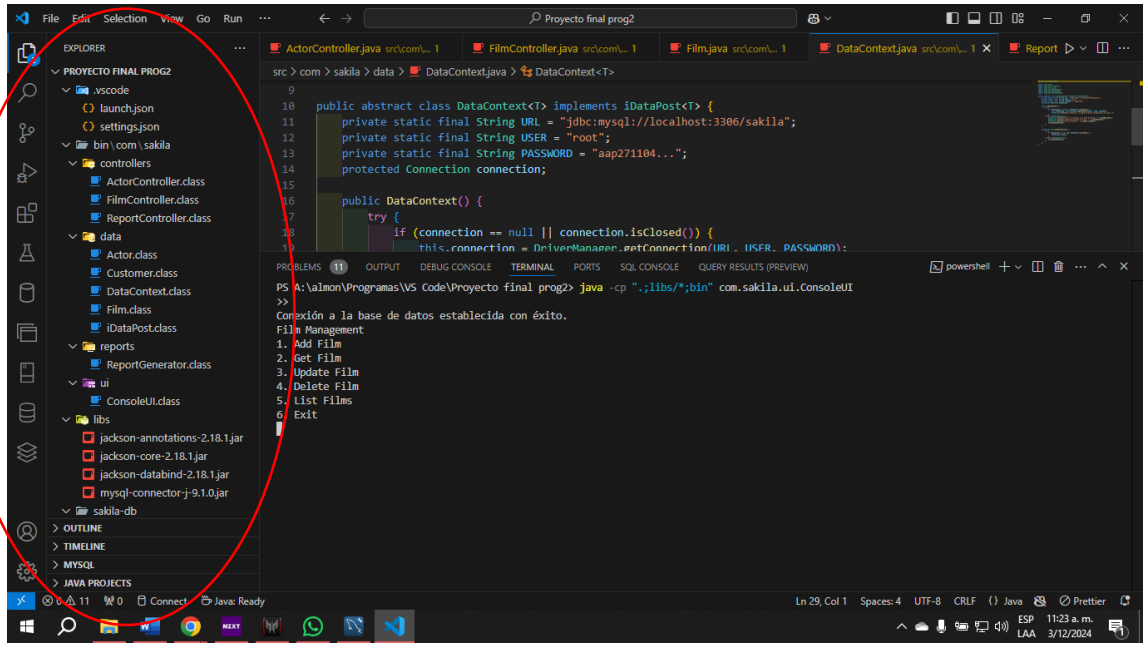
Problema: El método de conexión devolvía null.

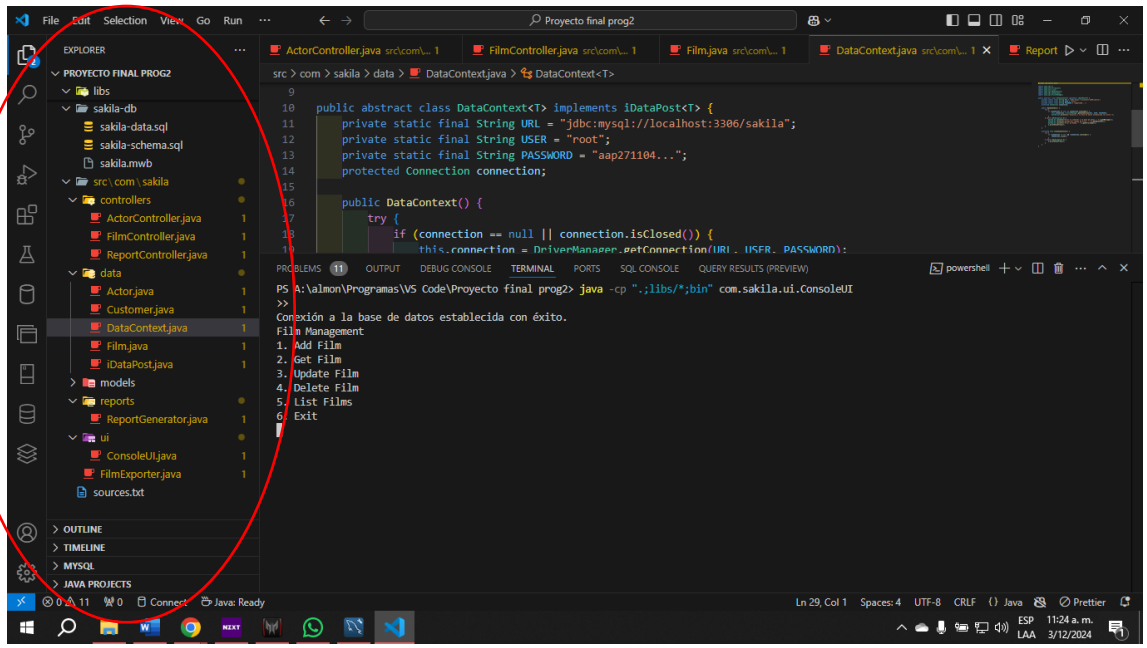
Solución: Se implementó correctamente el método connect() para gestionar la conexión.

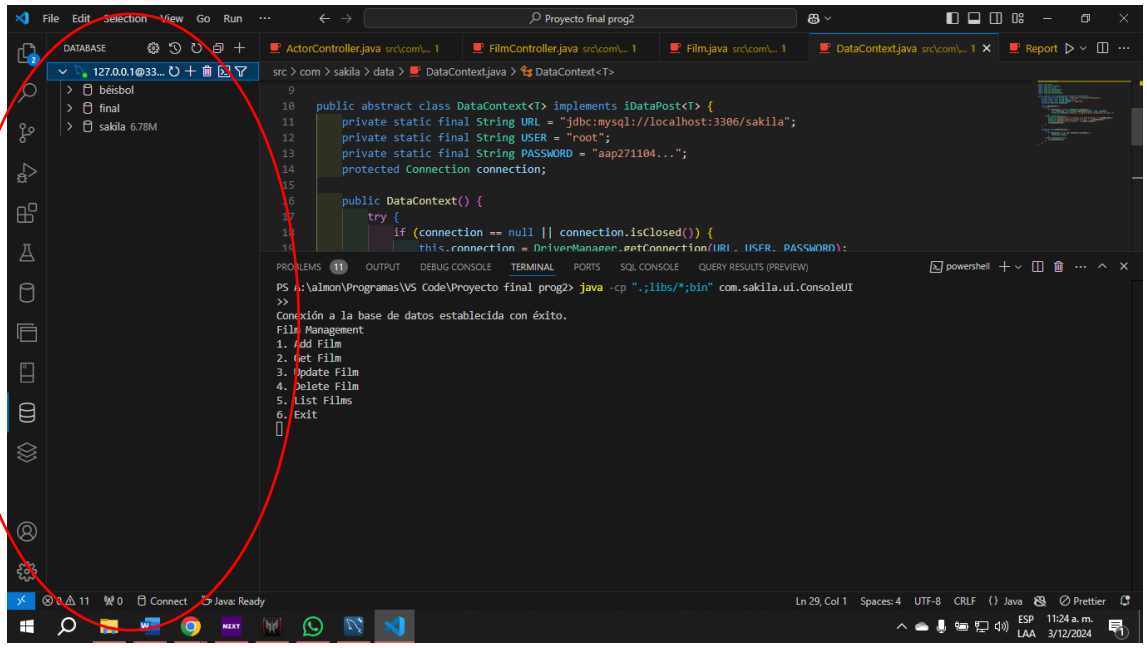
Ejecutar:

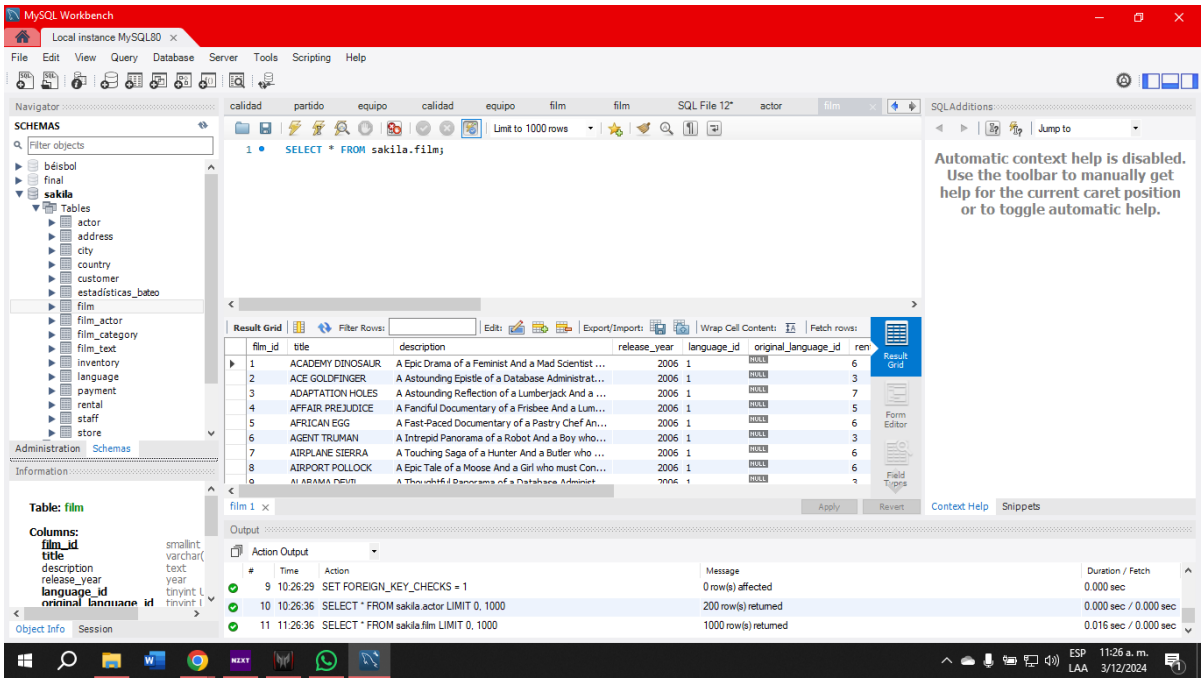
javac -cp "libs/*" -d bin src/com/sakila/**/*.java

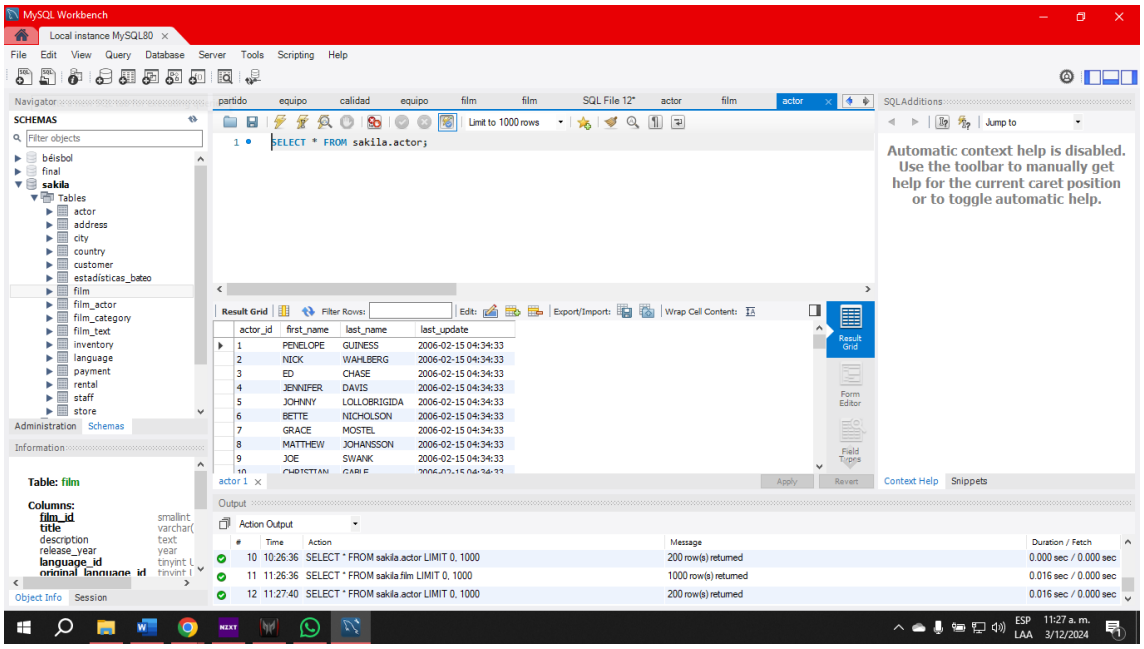
Organization de estructura




File Edit Selection View Go Run
Proyecto final prog2
EXPLORER
PROYECTO FINAL PROG2
libs
sakila-db
sakila-data.sql
sakila-schema.sql
sakila.mwb
src\com\sakila
controllers
ActorController.java
FilmController.java
ReportController.java
data
Actor.java
Customer.java
DataContext.java
Film.java
iDataPost.java
models
reports
ReportGenerator.java
ui
ConsoleUI.java
FilmExporter.java
sources.txt
OUTLINE
TIMELINE
MYSQL
JAVA PROJECTS
ActorController.java src\com\... 1
FilmController.java src\com\... 1
Film.java src\com\... 1
DataContext.java src\com\... 1
Report
src > com > sakila > data > DataContext.java > DataContext<T>
public abstract class DataContext<T> implements iDataPost<T> {
private static final String URL = "jdbc:mysql://localhost:3306/sakila";
private static final String USER = "root";
private static final String PASSWORD = "aap271104...";
protected Connection connection;
public DataContext() {
try {
if (connection == null || connection.isClosed()) {
PROBLEMS 11 OUTPUT DEBUG CONSOLE TERMINAL PORTS SQL CONSOLE QUERY RESULTS (PREVIEW)
powershell
PS A:\almon\Programas\VS Code\Proyecto final prog2> java -cp ".;libs/*;bin" com.sakila.ui.ConsoleUI
>>
Conexión a la base de datos establecida con éxito.
Film Management
1. Add Film
2. Get Film
3. Update Film
4. Delete Film
5. List Films
6. Exit
Ln 29, Col 1 Spaces: 4 UTF-8 CRLF Java Prettier
Connect
Java: Ready
ESP
LAA
11:24 a. m.
3/12/2024


File Edit Selection View Go Run
Proyecto final prog2
DATABASE
127.0.0.1@33...
béisbol
final
sakila 6.78M
ActorController.java src\com\... 1
FilmController.java src\com\... 1
Film.java src\com\... 1
DataContext.java src\com\... 1
Report
src > com > sakila > data > DataContext.java > DataContext<T>
public abstract class DataContext<T> implements iDataPost<T> {
private static final String URL = "jdbc:mysql://localhost:3306/sakila";
private static final String USER = "root";
private static final String PASSWORD = "aap271104...";
protected Connection connection;
public DataContext() {
try {
if (connection == null || connection.isClosed()) {
PROBLEMS 11 OUTPUT DEBUG CONSOLE TERMINAL PORTS SQL CONSOLE QUERY RESULTS (PREVIEW)
powershell
PS A:\almon\Programas\VS Code\Proyecto final prog2> java -cp ".;libs/*;bin" com.sakila.ui.ConsoleUI
>>
Conexión a la base de datos establecida con éxito.
Film Management
1. Add Film
2. Get Film
3. Update Film
4. Delete Film
5. List Films
6. Exit
Ln 29, Col 1 Spaces: 4 UTF-8 CRLF Java Prettier
Connect
Java: Ready
ESP
LAA
11:24 a. m.
3/12/2024


MySQL Workbench
Local instance MySQL80
File Edit View Query Database Server Tools Scripting Help
SCHEMAS
sakila
Tables
actor
address
city
country
customer
estadisticas_bateo
film
film_actor
film_category
film_text
inventory
language
payment
rental
staff
store
1 SELECT * FROM sakila.film;
Result Grid
film_id title description release_year language_id original_language_id
1 ACADEMY DINOSAUR A Epic Drama of a Feminist And a Mad Scientist ... 2006 1 NULL 6
2 ACE GOLDFINGER A Astounding Epistle of a Database Administrat... 2006 1 NULL 3
3 ADAPTATION HOLES A Astounding Reflection of a Lumberjack And a ... 2006 1 NULL 7
4 AFFAIR PREJUDICE A Fanciful Documentary of a Frisbee And a Lum... 2006 1 NULL 5
5 AFRICAN EGG A Fast-Paced Documentary of a Pastry Chef An... 2006 1 NULL 6
6 AGENT TRUMAN A Intrepid Panorama of a Robot And a Boy who... 2006 1 NULL 3
7 AIRPLANE SIERRA A Touching Saga of a Hunter And a Butler who ... 2006 1 NULL 6
8 AIRPORT POLLOCK A Epic Tale of a Moose And a Girl who must Con... 2006 1 NULL 6
Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.
Table: film
Columns:
film_id smallint
title varchar(
description text
release_year year
language_id tinyint
Output
Action Output
Time Action Message Duration / Fetch
9 10:26:29 SET FOREIGN_KEY_CHECKS = 1 0 row(s) affected 0.000 sec
10 10:26:36 SELECT * FROM sakila.actor LIMIT 0, 1000 200 row(s) returned 0.000 sec / 0.000 sec
11 11:26:36 SELECT * FROM sakila.film LIMIT 0, 1000 1000 row(s) returned 0.016 sec / 0.000 sec


MySQL Workbench
Local instance MySQL80
File Edit View Query Database Server Tools Scripting Help
SCHEMAS
sakila
Tables
actor
address
city
country
customer
estadisticas_bateo
film
film_actor
film_category
film_text
inventory
language
payment
rental
staff
store
1 SELECT * FROM sakila.actor;
Result Grid
actor_id first_name last_name last_update
1 PENELOPE GUINESS 2006-02-15 04:34:33
2 NICK WAHLBERG 2006-02-15 04:34:33
3 ED CHASE 2006-02-15 04:34:33
4 JENNIFER DAVIS 2006-02-15 04:34:33
5 JOHNNY LOLLOBRIGIDA 2006-02-15 04:34:33
6 BETTE NICHOLSON 2006-02-15 04:34:33
7 GRACE MOSTEL 2006-02-15 04:34:33
8 MATTHEW JOHANSSON 2006-02-15 04:34:33
9 JOE SWANK 2006-02-15 04:34:33
Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.
Table: film
Columns:
film_id smallint
title varchar(
description text
release_year year
language_id tinyint
Output
Action Output
Time Action Message Duration / Fetch
10 10:26:36 SELECT * FROM sakila.actor LIMIT 0, 1000 200 row(s) returned 0.000 sec / 0.000 sec
11 11:26:36 SELECT * FROM sakila.film LIMIT 0, 1000 1000 row(s) returned 0.016 sec / 0.000 sec
12 11:27:40 SELECT * FROM sakila.actor LIMIT 0, 1000 200 row(s) returned 0.016 sec / 0.000 sec

Conclusion

El desarrollo de este sistema de gestión de películas ha sido una oportunidad para implementar los principios de la arquitectura en capas, facilitando la separación de responsabilidades y mejorando la organización del código. Este proyecto también subraya la importancia de las buenas prácticas de codificación, como el manejo adecuado de entradas del usuario y la gestión de excepciones. Sin embargo, se podrían realizar mejoras en la conexión con la base de datos, en la validación de entradas y en la optimización de la interacción con la base de datos.