1. `#include <iostream>`: Esta biblioteca proporciona las funciones básicas de entrada y salida estándar en C++. Esto incluye `std::cout` para imprimir en la consola y `std::cin` para leer desde la consola.

2. `#include <pqxx/pqxx>`: Esta biblioteca, `pqxx`, es una biblioteca de C++ que proporciona una interfaz para trabajar con PostgreSQL, un sistema de gestión de bases de datos relacionales. Permite a los programas en C++ conectarse a una base de datos PostgreSQL y realizar operaciones como consultas y actualizaciones.

3. `#include <cstdlib>`: Esta biblioteca proporciona varias funciones útiles relacionadas con el sistema y el entorno en tiempo de ejecución. Incluye funciones como `std::exit`, `std::system`, y otras para manipulación de cadenas de caracteres, manejo de memoria dinámica, y control del flujo de ejecución.

4. `#include <string>`: Esta biblioteca proporciona la clase `std::string`, que es una clase de C++ para trabajar con cadenas de caracteres de forma más conveniente y segura que las cadenas de caracteres de estilo C.

5. `#include <vector>`: Esta biblioteca proporciona la clase `std::vector`, que es un contenedor de C++ que almacena elementos en un orden secuencial y proporciona acceso rápido a los elementos individuales. Los vectores son dinámicos, lo que significa que su tamaño puede cambiar durante el tiempo de ejecución.

6. `#include <regex>`: Esta biblioteca proporciona soporte para expresiones regulares en C++. Las expresiones regulares son patrones de búsqueda que se utilizan para encontrar coincidencias dentro de las cadenas de caracteres. La biblioteca `<regex>` permite compilar y ejecutar expresiones regulares en cadenas de caracteres para realizar operaciones como búsqueda, reemplazo y validación.

Aquí tienes una explicación detallada de cada una de las funciones proporcionadas:

1. `void limpiarConsola() { system("clear"); }`: Esta función se encarga de limpiar la consola. Utiliza la función del sistema `clear` para limpiar la pantalla de la consola. Esto es útil para mantener la salida de la consola limpia y organizada.

2. `void limpiarBuffer() { cin.clear(); cin.ignore(numeric\_limits<streamsize>::max(), '\n'); }`: Esta función se encarga de limpiar el búfer de entrada del flujo de entrada estándar (`cin`). Primero, llama a `cin.clear()` para restablecer cualquier error de estado en el flujo de entrada. Luego, utiliza `cin.ignore()` para descartar cualquier carácter restante en el búfer de entrada, hasta encontrar un salto de línea (`'\n'`). Esto es útil para evitar que caracteres adicionales ingresados accidentalmente interfieran con las futuras operaciones de entrada.

3. `void pausa() { cout << "Pulse Enter para continuar..."; cin.get(); }`: Esta función imprime un mensaje en la consola solicitando al usuario que pulse Enter para continuar. Luego, utiliza `cin.get()` para esperar a que el usuario presione la tecla Enter antes de continuar con la ejecución del programa. Es una manera simple de pausar la ejecución y permitir que el usuario vea la salida en la consola.

4. `void ejecutarConsulta(connection &c, const string &query) { /\* ... \*/ }`: Esta función ejecuta una consulta SQL en la base de datos utilizando la conexión `c` proporcionada y la consulta SQL especificada en `query`. Primero, crea un objeto `nontransaction` utilizando la conexión `c`, lo que indica que la consulta no debe realizarse dentro de una transacción. Luego, ejecuta la consulta utilizando `n.exec(query)`. Si la consulta se ejecuta correctamente, la función confirma la transacción utilizando `n.commit()`. Si se produce algún error durante la ejecución de la consulta, captura la excepción y muestra un mensaje de error.

5. `bool validarFormatoFecha(const string &fecha) { /\* ... \*/ }`: Esta función valida si una cadena de caracteres (`fecha`) tiene el formato de una fecha en formato "YYYY-MM-DD". Utiliza una expresión regular para verificar si la cadena de caracteres cumple con el patrón especificado. Retorna `true` si la cadena de caracteres tiene el formato correcto y `false` en caso contrario.

6. `bool validarFormatoHora(const string &hora) { /\* ... \*/ }`: Similar a la función `validarFormatoFecha`, esta función valida si una cadena de caracteres (`hora`) tiene el formato de una hora en formato "HH:MM:SS". Utiliza una expresión regular para verificar si la cadena de caracteres cumple con el patrón especificado. Retorna `true` si la cadena de caracteres tiene el formato correcto y `false` en caso contrario.

7. `bool existeDatoTabla(connection &c, const string &tabla) { /\* ... \*/ }`: Esta función verifica si existen datos en una tabla específica de la base de datos. Ejecuta una consulta SQL que verifica la existencia de al menos un registro en la tabla especificada. Retorna `true` si hay datos en la tabla y `false` si la tabla está vacía. Si se produce algún error durante la ejecución de la consulta, muestra un mensaje de error y lanza una excepción.

La función proporcionada, `conectar()`, se encarga de establecer una conexión con la base de datos PostgreSQL utilizando los parámetros especificados y retorna un objeto `connection` que representa la conexión establecida. Aquí está una explicación detallada de cada parte de la función:

1. `connection conectar() { /\* ... \*/ }`: Esta línea declara la función `conectar()` que devuelve un objeto `connection`.

2. `try { /\* ... \*/ }`: Esta parte del código indica que se va a intentar ejecutar un bloque de código y capturar cualquier excepción que pueda surgir durante la ejecución del bloque.

3. `connection c("dbname = peliculas user = postgres password = 1234admin hostaddr = 127.0.0.1 port = 5432");`: Aquí se crea un objeto `connection` llamado `c` utilizando el constructor de la clase `connection`. Se especifican los parámetros de conexión, como el nombre de la base de datos (`dbname`), el usuario (`user`), la contraseña (`password`), la dirección del host (`hostaddr`), y el puerto (`port`). Esto establece la conexión con la base de datos PostgreSQL utilizando los parámetros proporcionados.

4. `if (c.is\_open()) { /\* ... \*/ }`: Se verifica si la conexión se ha abierto correctamente utilizando el método `is\_open()` del objeto `connection`. Si la conexión está abierta, significa que la conexión fue exitosa y se muestra un mensaje indicando la conexión exitosa junto con el nombre de la base de datos. Luego, se devuelve el objeto `connection` creado.

5. `else { /\* ... \*/ }`: Si la conexión no se abre correctamente, significa que ha ocurrido un error durante la conexión. Se muestra un mensaje indicando que no se pudo establecer la conexión y se solicita al usuario que pulse Enter para salir del programa. Después, se lanza una excepción de tipo `runtime\_error` con un mensaje indicando el error de conexión.

6. `catch (const exception &e) { /\* ... \*/ }`: Si ocurre alguna excepción durante la ejecución del bloque de código dentro del `try`, se captura la excepción y se maneja en este bloque. Se imprime el mensaje de error correspondiente y se relanza la excepción para que pueda ser manejada en un nivel superior.

7. `pausa();`: Finalmente, se llama a la función `pausa()` para pausar la ejecución del programa después de manejar cualquier excepción. Esta función espera a que el usuario pulse Enter antes de continuar.

La función `crearTablas` se encarga de crear las tablas necesarias en la base de datos para almacenar información sobre películas, directores, actores, funciones de cine, promociones, opiniones, entre otros. Aquí está una explicación detallada de cada parte de la función:

1. La función `crearTablas` toma un parámetro `connection &c`, que representa la conexión a la base de datos con la que se realizarán las consultas.

2. Se utiliza un bloque `try-catch` para manejar posibles excepciones que puedan ocurrir durante la ejecución del código dentro del bloque `try`.

3. Se define una serie de cadenas de consulta SQL para crear cada una de las tablas en la base de datos. Cada consulta SQL se almacena en una variable de tipo `string`.

4. Se ejecuta cada consulta SQL utilizando la función `ejecutarConsulta`, pasando como argumento la conexión `c` y la consulta SQL correspondiente.

5. Después de ejecutar todas las consultas SQL para crear las tablas, se muestra un mensaje indicando que las tablas se han creado correctamente.

6. Se utilizan comandos del sistema (`system("sleep 1")` y `system("sleep 2")`) para agregar pausas en la ejecución del programa, proporcionando una breve espera antes de continuar con la ejecución.

7. Si se produce algún error durante la ejecución del bloque `try`, se captura la excepción, se imprime un mensaje de error indicando el problema y se llama a la función `pausa()` para pausar la ejecución y permitir al usuario ver el mensaje de error antes de lanzar la excepción nuevamente.

8. La función finaliza después de manejar cualquier excepción que pueda surgir durante la creación de las tablas.

La función `insertarPelicula` se encarga de permitir al usuario ingresar los datos de una película y luego insertar esos datos en la tabla `Peliculas` de la base de datos. Aquí está una explicación detallada de cada parte de la función:

1. Se limpia la consola y se limpia el búfer de entrada utilizando las funciones `limpiarConsola()` y `limpiarBuffer()` respectivamente.

2. Se declaran variables locales para almacenar los datos de la película, como el título de distribución, título original, género, idioma original, etc.

3. Se solicita al usuario que ingrese cada uno de los datos de la película utilizando `cout` y `cin`. Se realizan algunas validaciones básicas de los datos ingresados, como verificar que no estén vacíos y que no excedan ciertas longitudes.

4. Se utiliza un `vector<string>` para almacenar los países de origen de la película. Los países se ingresan como una cadena separada por comas y se separan utilizando un `stringstream`.

5. Se verifica si existen datos suficientes en las tablas `Directores` y `Actores` utilizando la función `existeDatoTabla`. Si no hay suficientes datos, se muestra un mensaje de error y se devuelve sin insertar la película.

6. Se construye la consulta SQL de inserción utilizando los datos ingresados por el usuario. Se utiliza concatenación de cadenas para formar la consulta SQL, teniendo en cuenta el formato adecuado para ciertos tipos de datos como los arrays y las fechas.

7. Se ejecuta la consulta SQL utilizando la función `ejecutarConsulta` y se muestra un mensaje indicando que los datos de la película han sido insertados exitosamente.

8. Si ocurre algún error durante la ejecución del bloque `try`, se captura la excepción, se imprime un mensaje de error indicando el problema y se lanza la excepción nuevamente.

9. Finalmente, se llama a la función `pausa()` para pausar la ejecución y permitir al usuario ver el mensaje de éxito o de error antes de continuar.

La función `insertarDirector` permite al usuario ingresar los datos de un director y luego insertar esos datos en la tabla `Directores` de la base de datos. Aquí está una explicación detallada de cada parte de la función:

1. Se limpia la consola y se limpia el búfer de entrada utilizando las funciones `limpiarConsola()` y `limpiarBuffer()` respectivamente.

2. Se declaran variables locales para almacenar los datos del director, como el nombre y la nacionalidad.

3. Se solicita al usuario que ingrese el nombre del director utilizando `cout` y `cin`. Se realiza una validación básica para asegurarse de que el nombre no esté vacío y no exceda una longitud máxima de 100 caracteres.

4. Se solicita al usuario que ingrese la nacionalidad del director utilizando `cout` y `cin`. Se realiza una validación básica para asegurarse de que la nacionalidad no esté vacía y no exceda una longitud máxima de 50 caracteres.

5. Se construye la consulta SQL de inserción utilizando los datos ingresados por el usuario. Se utiliza concatenación de cadenas para formar la consulta SQL con los valores del nombre y la nacionalidad del director.

6. Se ejecuta la consulta SQL utilizando la función `ejecutarConsulta` y se muestra un mensaje indicando que los datos del director han sido insertados exitosamente.

7. Si ocurre algún error durante la ejecución del bloque `try`, se captura la excepción, se imprime un mensaje de error indicando el problema y se lanza la excepción nuevamente.

8. Finalmente, se llama a la función `pausa()` para pausar la ejecución y permitir al usuario ver el mensaje de éxito o de error antes de continuar.

La función `insertarActor` permite al usuario ingresar los datos de un actor y luego insertar esos datos en la tabla `Actores` de la base de datos. Aquí está una explicación detallada de cada parte de la función:

1. Se limpia la consola y se limpia el búfer de entrada utilizando las funciones `limpiarConsola()` y `limpiarBuffer()` respectivamente.

2. Se declaran variables locales para almacenar los datos del actor, como el nombre y la nacionalidad.

3. Se solicita al usuario que ingrese el nombre del actor utilizando `cout` y `cin`. Se realiza una validación básica para asegurarse de que el nombre no esté vacío y no exceda una longitud máxima de 100 caracteres.

4. Se solicita al usuario que ingrese la nacionalidad del actor utilizando `cout` y `cin`. Se realiza una validación básica para asegurarse de que la nacionalidad no esté vacía y no exceda una longitud máxima de 50 caracteres.

5. Se construye la consulta SQL de inserción utilizando los datos ingresados por el usuario. Se utiliza concatenación de cadenas para formar la consulta SQL con los valores del nombre y la nacionalidad del actor.

6. Se ejecuta la consulta SQL utilizando la función `ejecutarConsulta` y se muestra un mensaje indicando que los datos del actor han sido insertados exitosamente.

7. Si ocurre algún error durante la ejecución del bloque `try`, se captura la excepción, se imprime un mensaje de error indicando el problema y se lanza la excepción nuevamente.

8. Finalmente, se llama a la función `pausa()` para pausar la ejecución y permitir al usuario ver el mensaje de éxito o de error antes de continuar.

La función `insertarPersonaje` permite al usuario ingresar los datos de un personaje y luego insertar esos datos en la tabla `Personajes` de la base de datos. Aquí está una explicación detallada de cada parte de la función:

1. Se limpia la consola y se limpia el búfer de entrada utilizando las funciones `limpiarConsola()` y `limpiarBuffer()` respectivamente.

2. Se declaran variables locales para almacenar los datos del personaje, como el nombre del personaje y el ID del actor al que pertenece.

3. Se solicita al usuario que ingrese el nombre del personaje utilizando `cout` y `cin`. Se realiza una validación básica para asegurarse de que el nombre no esté vacío y no exceda una longitud máxima de 100 caracteres.

4. Se solicita al usuario que ingrese el ID del actor al que pertenece el personaje utilizando `cout` y `cin`. Se realiza una validación básica para asegurarse de que el ID del actor sea un número positivo.

5. Se construye la consulta SQL de inserción utilizando los datos ingresados por el usuario. Se utiliza concatenación de cadenas para formar la consulta SQL con los valores del nombre del personaje y el ID del actor.

6. Se ejecuta la consulta SQL utilizando la función `ejecutarConsulta` y se muestra un mensaje indicando que los datos del personaje han sido insertados exitosamente.

7. Si ocurre algún error durante la ejecución del bloque `try`, se captura la excepción, se imprime un mensaje de error indicando el problema y se lanza la excepción nuevamente.

8. Finalmente, se llama a la función `pausa()` para pausar la ejecución y permitir al usuario ver el mensaje de éxito o de error antes de continuar.

La función `insertarCine` permite al usuario ingresar los datos de un cine y luego insertar esos datos en la tabla `Cines` de la base de datos. Aquí está una explicación detallada de cada parte de la función:

1. Se limpia la consola y se limpia el búfer de entrada utilizando las funciones `limpiarConsola()` y `limpiarBuffer()` respectivamente.

2. Se declaran variables locales para almacenar los datos del cine, como el nombre, la dirección y el teléfono.

3. Se solicita al usuario que ingrese el nombre del cine utilizando `cout` y `cin`. Se realiza una validación básica para asegurarse de que el nombre no esté vacío y no exceda una longitud máxima de 100 caracteres.

4. Se solicita al usuario que ingrese la dirección del cine utilizando `cout` y `cin`. Se realiza una validación básica para asegurarse de que la dirección no esté vacía y no exceda una longitud máxima de 200 caracteres.

5. Se solicita al usuario que ingrese el teléfono del cine utilizando `cout` y `cin`. Se realiza una validación básica para asegurarse de que el teléfono no esté vacío y no exceda una longitud máxima de 20 caracteres.

6. Se construye la consulta SQL de inserción utilizando los datos ingresados por el usuario. Se utiliza concatenación de cadenas para formar la consulta SQL con los valores del nombre, la dirección y el teléfono del cine.

7. Se ejecuta la consulta SQL utilizando la función `ejecutarConsulta` y se muestra un mensaje indicando que los datos del cine han sido insertados exitosamente.

8. Si ocurre algún error durante la ejecución del bloque `try`, se captura la excepción, se imprime un mensaje de error indicando el problema y se lanza la excepción nuevamente.

9. Finalmente, se llama a la función `pausa()` para pausar la ejecución y permitir al usuario ver el mensaje de éxito o de error antes de continuar.

La función `insertarSala` permite al usuario ingresar los datos de una sala y luego insertar esos datos en la tabla `Salas` de la base de datos. Aquí está una explicación detallada de cada parte de la función:

1. Se limpia la consola y se limpia el búfer de entrada utilizando las funciones `limpiarConsola()` y `limpiarBuffer()` respectivamente.

2. Se declaran variables locales para almacenar los datos de la sala, como el nombre, el número de sala, la cantidad de butacas y el ID del cine al que pertenece la sala.

3. Se solicita al usuario que ingrese el nombre de la sala utilizando `cout` y `cin`. Se realiza una validación básica para asegurarse de que el nombre no esté vacío y no exceda una longitud máxima de 100 caracteres.

4. Se solicita al usuario que ingrese el número de la sala utilizando `cout` y `cin`. Se realiza una validación básica para asegurarse de que el número de sala sea un número positivo.

5. Se solicita al usuario que ingrese la cantidad de butacas de la sala utilizando `cout` y `cin`. Se realiza una validación básica para asegurarse de que la cantidad de butacas sea un número positivo.

6. Se solicita al usuario que ingrese el ID del cine al que pertenece la sala utilizando `cout` y `cin`. Se realiza una validación básica para asegurarse de que el ID del cine sea un número positivo.

7. Se construye la consulta SQL de inserción utilizando los datos ingresados por el usuario. Se utiliza concatenación de cadenas para formar la consulta SQL con los valores del nombre, el número de sala, la cantidad de butacas y el ID del cine.

8. Se ejecuta la consulta SQL utilizando la función `ejecutarConsulta` y se muestra un mensaje indicando que los datos de la sala han sido insertados exitosamente.

9. Si ocurre algún error durante la ejecución del bloque `try`, se captura la excepción, se imprime un mensaje de error indicando el problema y se lanza la excepción nuevamente.

10. Finalmente, se llama a la función `pausa()` para pausar la ejecución y permitir al usuario ver el mensaje de éxito o de error antes de continuar.

La función `insertarFuncion` permite al usuario ingresar los datos de una función y luego insertar esos datos en la tabla `Funciones` de la base de datos. Aquí está una explicación detallada de cada parte de la función:

1. Se limpia la consola y se limpia el búfer de entrada utilizando las funciones `limpiarConsola()` y `limpiarBuffer()` respectivamente.

2. Se declaran variables locales para almacenar los datos de la función, como el día de la semana, la hora de comienzo, el ID de la sala y el ID de la película.

3. Se solicita al usuario que ingrese el día de la semana para la función utilizando `cout` y `cin`. Se realiza una validación básica para asegurarse de que el día de la semana no esté vacío y no exceda una longitud máxima de 20 caracteres.

4. Se solicita al usuario que ingrese la hora de comienzo de la función utilizando `cout` y `cin`. Se realiza una validación utilizando la función `validarFormatoHora()` para asegurarse de que la hora ingresada tenga el formato correcto (HH:MM).

5. Se solicita al usuario que ingrese el ID de la sala para la función utilizando `cout` y `cin`. Se realiza una validación básica para asegurarse de que el ID de la sala sea un número positivo.

6. Se solicita al usuario que ingrese el ID de la película para la función utilizando `cout` y `cin`. Se realiza una validación básica para asegurarse de que el ID de la película sea un número positivo.

7. Se verifica la existencia de datos en las tablas `Salas` y `Peliculas` utilizando la función `existeDatoTabla()` para asegurarse de que haya suficientes datos disponibles para asociar con la función.

8. Se construye la consulta SQL de inserción utilizando los datos ingresados por el usuario. Se utiliza concatenación de cadenas para formar la consulta SQL con los valores del día de la semana, la hora de comienzo, el ID de la sala y el ID de la película.

9. Se ejecuta la consulta SQL utilizando la función `ejecutarConsulta` y se muestra un mensaje indicando que los datos de la función han sido insertados exitosamente.

10. Si ocurre algún error durante la ejecución del bloque `try`, se captura la excepción, se imprime un mensaje de error indicando el problema y se lanza la excepción nuevamente.

11. Finalmente, se llama a la función `pausa()` para pausar la ejecución y permitir al usuario ver el mensaje de éxito o de error antes de continuar.

La función `insertarPromocion` permite al usuario ingresar los datos de una promoción y luego insertar esos datos en la tabla `Promociones` de la base de datos. Aquí está una explicación detallada de cada parte de la función:

1. Se limpia la consola y se limpia el búfer de entrada utilizando las funciones `limpiarConsola()` y `limpiarBuffer()` respectivamente.

2. Se declaran variables locales para almacenar los datos de la promoción, como la descripción y el descuento.

3. Se solicita al usuario que ingrese la descripción de la promoción utilizando `cout` y `cin`. Se realiza una validación básica para asegurarse de que la descripción no esté vacía.

4. Se solicita al usuario que ingrese el descuento de la promoción utilizando `cout` y `cin`. Se realiza una validación básica para asegurarse de que el descuento sea un número no negativo.

5. Se construye la consulta SQL de inserción utilizando los datos ingresados por el usuario. Se utiliza concatenación de cadenas para formar la consulta SQL con los valores de la descripción y el descuento.

6. Se ejecuta la consulta SQL utilizando la función `ejecutarConsulta` y se muestra un mensaje indicando que los datos de la promoción han sido insertados exitosamente.

7. Si ocurre algún error durante la ejecución del bloque `try`, se captura la excepción, se imprime un mensaje de error indicando el problema y se lanza la excepción nuevamente.

8. Finalmente, se llama a la función `pausa()` para pausar la ejecución y permitir al usuario ver el mensaje de éxito o de error antes de continuar.

La función `insertarOpinion` permite al usuario ingresar los datos de una opinión sobre una película y luego insertar esos datos en la tabla `Opiniones` de la base de datos. A continuación, una explicación detallada de cada parte de la función:

1. Se limpia la consola y se limpia el búfer de entrada utilizando las funciones `limpiarConsola()` y `limpiarBuffer()` respectivamente.

2. Se declaran variables locales para almacenar los datos de la opinión, como el ID de la película, el nombre de la persona, la edad, la fecha de registro, la calificación y el comentario.

3. Se solicita al usuario que ingrese el ID de la película utilizando `cout` y `cin`. Se realiza una validación básica para asegurarse de que el ID de la película sea un número positivo.

4. Se solicita al usuario que ingrese el nombre de la persona utilizando `cout` y `getline(cin, nombre\_persona)`. Se realiza una validación básica para asegurarse de que el nombre de la persona no esté vacío y no exceda los 100 caracteres.

5. Se solicita al usuario que ingrese la edad utilizando `cout` y `cin`. Se realiza una validación básica para asegurarse de que la edad sea un número positivo.

6. Se solicita al usuario que ingrese la fecha de registro utilizando `cout` y `getline(cin, fecha\_registro)`. Se realiza una validación básica para asegurarse de que la fecha de registro tenga el formato adecuado (AAAA-MM-DD).

7. Se solicita al usuario que ingrese la calificación utilizando `cout` y `getline(cin, calificacion)`. Se realiza una validación básica para asegurarse de que la calificación no esté vacía.

8. Se solicita al usuario que ingrese el comentario utilizando `cout` y `getline(cin, comentario)`. Se realiza una validación básica para asegurarse de que el comentario no esté vacío.

9. Se construye la consulta SQL de inserción utilizando los datos ingresados por el usuario. Se utiliza concatenación de cadenas para formar la consulta SQL con los valores de la película, el nombre de la persona, la edad, la fecha de registro, la calificación y el comentario.

10. Se ejecuta la consulta SQL utilizando la función `ejecutarConsulta` y se muestra un mensaje indicando que los datos de la opinión han sido insertados exitosamente.

11. Si ocurre algún error durante la ejecución del bloque `try`, se captura la excepción, se imprime un mensaje de error indicando el problema y se lanza la excepción nuevamente.

12. Finalmente, se llama a la función `pausa()` para pausar la ejecución y permitir al usuario ver el mensaje de éxito o de error antes de continuar.

Explicación detallada de la sintaxis de las consultas SQL utilizadas en las funciones de mostrar datos:

1. \*\*mostrarPeliculas\*\*:

- La consulta SQL utiliza la cláusula `SELECT` para seleccionar las columnas que se mostrarán en la salida.

- Selecciona las columnas de la tabla `Peliculas` (`id\_pelicula`, `titulo\_distribucion`, `genero`, etc.).

- Utiliza una unión izquierda (`LEFT JOIN`) con la tabla `Peliculas\_Directores` (`pd`) para conectar las películas con sus respectivos directores.

- También se une (`LEFT JOIN`) con la tabla `Directores` (`d`) para obtener el nombre del director.

- Se renombran las columnas de la tabla `Directores` con el alias `d` para poder acceder a ellas de forma calificada (`d.nombre`).

2. \*\*mostrarPersonajes\*\*:

- Similar a `mostrarPeliculas`, esta consulta selecciona columnas de la tabla `Personajes` (`id\_personaje`, `nombre\_personaje`).

- Utiliza una unión izquierda con la tabla `Actores` (`a`) para conectar los personajes con sus respectivos actores.

- Se renombra la columna `nombre` de la tabla `Actores` con el alias `actor` para poder acceder a ella de forma calificada (`a.nombre AS actor`).

3. \*\*mostrarSalas\*\*:

- Esta consulta selecciona columnas de la tabla `Salas` (`id\_sala`, `nombre`, `numero`, etc.).

- Utiliza una unión izquierda con la tabla `Cines` (`c`) para conectar las salas con los cines a los que pertenecen.

- Se renombra la columna `nombre` de la tabla `Cines` con el alias `cine` para poder acceder a ella de forma calificada (`c.nombre AS cine`).

4. \*\*mostrarFunciones\*\*:

- La consulta selecciona columnas de la tabla `Funciones` (`id\_funcion`, `dia\_semana`, `hora\_comienzo`).

- Utiliza uniones izquierdas con las tablas `Salas` y `Peliculas` para obtener el nombre de la sala y el título de la película respectivamente.

- Se renombran las columnas `nombre` de la tabla `Salas` y `Peliculas` con los alias `sala` y `pelicula` respectivamente.

5. \*\*mostrarOpiniones\*\*:

- Esta consulta selecciona todas las columnas de la tabla `Opiniones`.

- No se utilizan uniones en esta consulta, simplemente selecciona los datos de la tabla `Opiniones`.

En cada consulta, se utiliza una combinación de `SELECT`, `FROM`, `LEFT JOIN` y `AS` para obtener los datos necesarios de las tablas involucradas en la base de datos. Además, se utilizan alias para renombrar las columnas y facilitar su referencia en el código.

### Funciones de Menú

- \*\*menuInsertar\*\*: Permite al usuario seleccionar qué tipo de datos desea insertar en la base de datos. Llama a las funciones correspondientes para insertar películas, directores, actores, salas, funciones, promociones, opiniones, etc.

- \*\*menuMostrar\*\*: Similar a `menuInsertar`, pero permite al usuario seleccionar qué tipo de datos desea mostrar. Llama a las funciones correspondientes para mostrar películas, directores, actores, salas, funciones, promociones, opiniones, etc.

- \*\*menuPrincipal\*\*: Es el punto de entrada principal del programa. Presenta al usuario un menú principal donde puede elegir entre agregar datos, mostrar datos o salir del programa. Llama a `menuInsertar` o `menuMostrar` según la opción seleccionada por el usuario.

### Función Principal (main)

- \*\*main\*\*: Conecta el programa a la base de datos, crea las tablas necesarias si aún no existen, y luego presenta el menú principal al usuario. Después de que el usuario decide salir del programa, cierra la conexión con la base de datos y termina la ejecución.

### Explicación Adicional

- Se utilizan pausas (`system("sleep 1")`) para proporcionar un tiempo de espera visual entre las diferentes acciones del usuario.

- Se manejan errores mediante excepciones, mostrando un mensaje de error al usuario en caso de que ocurra algún problema durante la ejecución.

- Se utilizan limpiadores de consola y buffers para mejorar la experiencia del usuario y evitar errores en la entrada de datos.

Este programa proporciona una interfaz básica para interactuar con una base de datos de gestión de películas, permitiendo al usuario agregar y visualizar información de manera intuitiva.