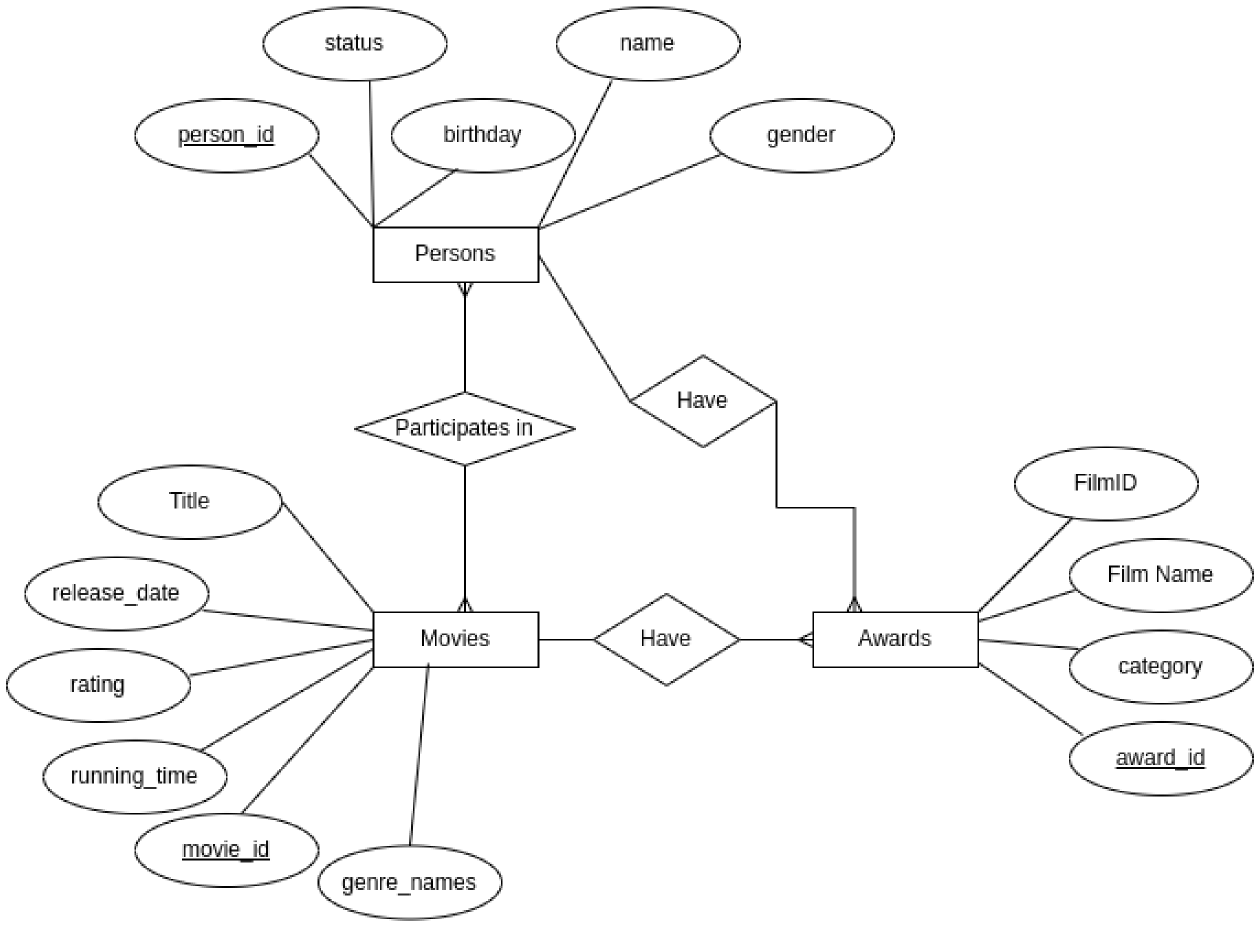
Overview

We were asked to create a database for movies that included information about actors, their awards, filmographies, biographical details, and movies themselves, including details such as titles, release dates, genres, directors, casts, plots, ratings, and awards. The project involved several steps: making an ER diagram, designing a relational schema, initializing and populating the database with Java, and querying the database for specific information.

2. Design & Implementation

2.1 Database

My database consists of 3 main entities Persons, Movies and Awards, and 1 table to establish relationships between Movies and Persons called Movie_Presons.



2.1.1 Movie ER Diagram

Persons Entity: This entity represents individuals with attributes such as **person_id** (a unique identifier for each person), **name**, **birthday**, **gender**, and status. The **Persons** entity could represent actors, directors, or other individuals associated with the film industry.

Movies Entity: This entity includes details about movies, with attributes like **movie_id** (a unique identifier for each movie), **Title**, **release_date**, **rating**, **running_time**, and **genre_names**. The Movies entity stores information related to each movie, including what type of movie it is, how long it runs, and how it is rated.

Awards Entity: This entity contains information about various awards, indicated by attributes such as **award_id** (a unique identifier for each award), **FilmID**, **Film Name**, and **category**. The Awards entity might be tracking awards given to films in different categories, such as 'Best Picture', 'Best Director', etc.

Relationships:

- **Participates in**: This is a many-to-many relationship between **Persons** and **Movies**, indicating that a person can participate in multiple movies, and a movie can involve multiple people.

Have: There are two 'Have' relationships in the diagram:

- One linking **Movies** to **Awards**, suggesting that a movie can have multiple awards, and an award can be associated with multiple movies.
- The other 'Have' relationship links **Persons** to **Awards**, suggesting that a person can have multiple awards, and an award can be associated with multiple people.

2.2 Initialise database

Here's how each part of the class works:

Constructor InitialiseDB: When an instance of **InitialiseDB** is created, it takes a file path for the database (**dbFilePath**). If a database file at this path already exists, it is deleted, ensuring that a fresh database is created each time. A connection to the SQLite database is then established.

Method createTablesFromDDL: This method reads an SQL file from the given DDL file path (**ddlFilePath**). It processes the file line by line, removing SQL comments and accumulating SQL commands. When a semicolon is encountered, it indicates the end of an SQL statement. The method then executes the statement using the Statement object to create tables in the SQLite database.

Method closeConnection: This method closes the database connection. It is a good practice to close the connection to release database resources.

Main Method: The main method is the entry point of the program. It creates an instance of InitialiseDB with the specified database file path. It then calls createTablesFromDDL to read the DDL script and create the tables. Finally, it closes the connection to the database.

Error Handling: The program includes try-catch blocks to handle SQLException and IOException. If an error occurs during the process, an error message is printed to the console.

Logging: The program prints out messages to the console to indicate the status of the connection and the outcome of the table creation process.

2.3 Populating database

For populating database I wrote **PopulateDB** class. The class **PopulateDB** has static String variables pointing to CSV files that store information about films, actors, and directors. These CSV files are in a **data** directory.

Main Method: The main method is the entry point of the program. It establishes a connection to the database and calls various methods to insert data from the CSV files into the database.

Insertion Methods:

- **insertMovies**: Reads the films CSV file and inserts film data into the Movies table in the database.
- **insertPeople**: Reads the actors and directors CSV files and inserts person data into the **Persons** table in the database.
- **insertPeopleMovies**: Inserts relations between people and movies into a **Movies_Persons** table, indicating which actors and directors participated in which movies.
- **insertAward**: Inserts award data into the Awards table. It matches movies from a CSV file with their titles, categories, and whether they won an award.

Utility Methods:

- **getMovieIdByTitle**: Retrieves a movie's ID from the Movies table based on its title.
- **readCSV**: Reads a CSV file and returns a list of string arrays, each representing a row of the CSV file.
- **parseCSVLine**: Parses a line from a CSV, taking into account potential comma delimiters within quoted strings.

Data Insertion Process:

- It inserts movie records and people (actors and directors) records separately.

- It then establishes a relationship between movies and people by inserting records into a linking table.
- Finally, it reads the Oscars data and updates the Awards table, ensuring that awards are matched with the correct movies.

Error Handling: The program includes try-catch blocks for SQL exceptions, IO exceptions, and parse exceptions, to handle any errors that may occur during the database operations or file reading process.

Logging: The system prints out error messages if any exceptions are caught, and a success message once the awards table population is complete.

2.4 Database queries

The Java program in the **queryDB** class is designed to execute predefined SQL queries on an SQLite database. Here's a detailed description of the components and processes within the **queryDB** class:

Database URL: A constant string **URL** is defined to hold the JDBC URL for the SQLite database. This URL is used to connect to the database located at **schemas/schema.db**.

Main Method: The main method establishes a connection to the SQLite database using the URL. It then calls the **executeQuery** method with a predefined **queryNumber** to execute a specific query.

executeQuery Method: This method takes the database connection and a query number as arguments. Based on the query number, it calls one of the predefined query methods (**query1**, **query2**, **query3**, or **query4**) to execute a particular SQL query.

Predefined Query Methods:

- **query1**: Retrieves and prints the titles of all movies in the **Movies** table.
- **query2**: Retrieves and prints the names of all actors from the **Persons** table who acted in the movie titled "Forever My Girl".
- **query3**: Retrieves and prints the titles of movies where either 'ian wright' is an actor or 'dave stewart' is a film director.
- **query4**: Retrieves and prints the names of all persons from the **Persons** table who have birthdays between the years 1950 and 2000.

SQL Execution: Each query method uses a **Statement** object to execute an SQL query. The results are retrieved through a **ResultSet** object. Each query iterates over the **ResultSet** and prints the relevant information to the console.

Error Handling: The program includes a catch block to handle **SQLException**. If an error occurs during database connection or query execution, it prints the error message.

Logging: System outputs are used to print the results of the queries or any error messages to the console.

2.5 Data preparation

Almost all my data I gathered and cleaned using Python. You can find those scripts in daraPrep file.

3. Testing

What is being	name of testing method	pre con ditions	expected outcome	actual outcome	evidence
Database initialisation	db_init() in InitialiseDB	Database is empty	Database should contain tables	Database contain cells	Output that I have been given after writing sqlite3 .open schema.db .tables Showed that I have tables I expected
Getting rid of commentaries in schema	CreateTablesFromDDL() in InitialiseDB	String with schema has comments in it	String should contain schema, but not comments	String contain schema, but not comments	Output from regex101.com showed that my regex works. https://regex101.com/r/Io2foh/1
Close connection after we create db	CloseConnection() in InitialiseDB	Connection is open during the creating process	Connection shell be closed after we create connection	Connection is closed	CloseConnection() outputs Connection to SQLite closed after we execute it, and it doesn't throw and SQLExcepton.

Populating database	PopulateDataBase() in PopulateDB	All tables are initialized, but they are empty	Database would be full	Data base is full	Select * FROM (all tables) Gives us non-null result
Reading csv file	ReadCSV in CSVUtils	We have all csv files in data/ folder	We should get List<String[]> of our CSV	We are getting List<String[]>	If it didn't work, PopulateDB wouldn't work either because it relies on this method
Parsing csv lines	parsCSVLine() in CSVUtils	Start read csv file using readCSV method	We should get exact amount of columns in row including cases when we have commas in titles or names	Works as expected	Title of film with id 232 has comma in it "Roman J. Israel, Esq.". If program didn't work, we would get more columns than expected and skipped over this title as we have this line in insertMovies() method `if (data.size() != 9) continue;`
Query 1	query1() in executeQuery class	Table is populated with all data	We should get an output of all titles in our Movies table	Works as expected	Output from console
Query 2	Query2() in executeQuery class	Table is populated with data	Shold output tom segura	Works as expected	Output from console

Query3	Query3() in executeQuery class	Table is populated with data	Should output Title: Love Songs	Works as expected	Output from console
Query4,	Query4()	Table with data	SQL query retrieves information about people who have participated in movies released since the year 2000 and have a rating greater than 8.	Works as expected	Output from console
Query5	Query5() from QueryDB	Table populated with data	Output of list of people along with the count of movies they've participated	Works as expected	Output from console

4. Evaluation (against requirements)

My project satisfies nearly all specified criteria, except lacking film descriptions in the Movies table. My testing has been thorough, so I'm quite sure that I did well enough.

5. Conclusions (your achievements difficulties and solutions, and what you would have done given more time)

The hardest part of this assignment was to gather data from various sources and then cleaning that afterwards. If I had more time, I would have cleaned up my code, made more functions, added more commentaries etc. I also should have used streams more to enhance my program performance.

Overall, I did my best despite knowing that my submission is far from ideal.

6. Compile instructions

```
 chmod +x run.sh
./run.sh
```
in terminal

7. References

Data:

https://data.world/iliketurtles/movie-dataset

https://api-ninjas.com/api/celebrity

https://www.kaggle.com/datasets/unanimad/the-oscar-award

http://www.omdbapi.com/

https://raw.githubusercontent.com/DLu/oscar_data/main/oscars.csv

ER Diagram:

https://vertabelo.com/blog/chen-erd-notation/

Code:

All my code I've got from lecture slides, example and exercise classes.