Database Scheme structure:

When we first got the data, the production companies, languages, genres and movies were all bundled up together. Creating one dense table seemed to break a lot of the principles taught in class so we split them up with connecting tables in between. This way tables haves independent data with interconnecting tables between them. Each query accesses only the relevant data in each table, with little redundancy.

Database optimization performed:

Added indices in all relevant tables. Meant to speed up selection times which are used heavily in our queries.

Description of query:

what does it do, how you optimized the query and how the DB design supports this query

1. **Best production company by genre**
2. **Best production company by language**

What does the query do:

The user inputs genre/language name with the judging criteria: Movie rating ,vote count and difference between the movie revenue and budget. The query returns the best production company satisfying the category (genre, language) and criteria.

How we optimized it:

Use of indices in each of the relevant tables, clustering the data and achieving higher selection times. The data is joined from different tables, the selection is done in each table so that the join is performed on smaller relations.

How the database design support these queries:

Take query #1 (second one is of similar design):

Data is retrieved from genres and movies tables, the selection is done on each of these tables. Afterwords through moviegenres the data is joined from each side via movie\_id and genre\_id (this is where the indices optimize the query).

1. **Browse movies by keyword in description**

This is a full text query.

What does the query do:

Use picks a certain string keyword. The query searches all movies who description has this matching keyword. Some data is return about the movie, such as the director, genres, production companies.

How we optimized it:

Selection done on each relevant database before joins. Use of indices in appropriate tables to speed up finding the relevant relations

How the database design support these queries:

The database design is very modular, each table has particular values, there is little overlap between them.

1. **Find best crew member for a certain production company and role**

Given a production company and a certain crew role. The user inputs criteria parameters, how many votes, revenue and vote average. And the query return the best candidate crew member name to work with the production company.

Again the use of indices helps speed up the selection in each relevant table, the selection is done on each subquery separately so the joins are between smaller relations. We only took the relevant columns to reduce the relation size

API description:

The Movie Database: we iteratively made calls to the API and got data about movies via their unique ID. This data was split into 5 main tables: movies, genres, production\_companies, spoken\_languages and production\_countries. with additional connection data put into movieprodcomp, moviegenres, movielanguages and movieprodcountry.

Using the unique movie\_id we made additional calls to get the actors and crew members or each movie, which were inserted into actors and crew, with the connection to movies table being made via movieactors and moviecrew.

Flow of the application:

Live demonstration in USER MANUAL.