Databases project

Danila Pechenev, Gwenn Garrigues

Contents

1 Data of the project

1.1 Initial data: Oscars Dataset

The first step of the project was to choose the data.

For this purpose, we selected a Kaggle dataset concerning the various Oscar nominations throughout the years: Oscar dataset.

This dataset traces the history of the Oscars across 97 editions. It allows us to view the films that were nominated, whether they won or not, in the various categories present at the Academy Awards.

It contains various pieces of information for each nomination, including the edition and the individuals involved (director or actor).

The dataset is divided into several columns:

- **year**: the year of the ceremony.
- **edition**: the numbered edition of the Oscars (closely related to the year).
- **award**: the award category (e.g., "Best Actor in a Leading Role", "Best Actress in a Supporting Role", "Best Animated Movie", etc.).
- **film title**: the title of the nominated film.
- **nomination actor**: in the case where the Oscar describes an actor or an artist (or a group of artists).
- **nomination producers**: the producer(s) associated with the nominated film.
- **nomination country**: the country associated with the film (in the case of a country-related nomination).
- **nomination character name**: the name of the character portrayed in the context of a role-related nomination.
- **nomination description**: description of the nomination (mostly null).
- **is winner**: defines whether the nomination is a victory or not.
- **acceptance speech text**: the nominee's speech in the event of a win.
- **acceptance speech url**: link to the speech.

We decided to choose this dataset because it possesses a large number of columns, with some columns allowing for interesting connections and the potential inclusion of other datasets.

1.2 New data: Movies Dataset

Following the inclusion of this dataset, we simulate the scenario in which someone approaches us and requests the addition of their movie dataset to link it with our `Oscars` dataset. To do this, we are using a Kaggle dataset: Movies Dataset.

This dataset, containing over 1 million movies, holds extensive information on various films, encompassing all types of movies, such as TV movies, adult films (which will be subsequently removed), and other film types.

The different characteristics are:

- **id**: providing a unique identifier for each movie.

- **title**: the title of the movie.
- **vote average**: the average rating of the movie.
- **vote count**: the number of votes.
- **status**: the release status of the film (e.g., if released or announced).
- **release date**: the release date of the film.
- **revenue**: the film's revenue (in euros).
- **budget**: the budget for the production of the film.
- **runtime**: the duration of the film.
- **adult**: whether the film is an adult film.
- **original language**: the original language of the movie.
- **original title**: the title in the original language of the movie.
- **information about the production**: such as the companies and the country.
- **genres**: all the genres that describe the film.

2 Oscars dataset

2.1 Database Modeling

2.1.1 CDM

Once the initial dataset was selected, the first step was to decide which columns to keep. We subsequently chose to retain only certain columns from the `Oscars` dataset, deeming some to be uninteresting or unusable.

We decided not to keep the `nomination_country` column, as it was mostly null (99%), along with the columns related to the acceptance speech (`acceptance_speech_text`, `acceptance_speech_url`) and the nomination description (`nomination_description`).

We then decided to split our dataset into several entities to allow for better representation of the data.

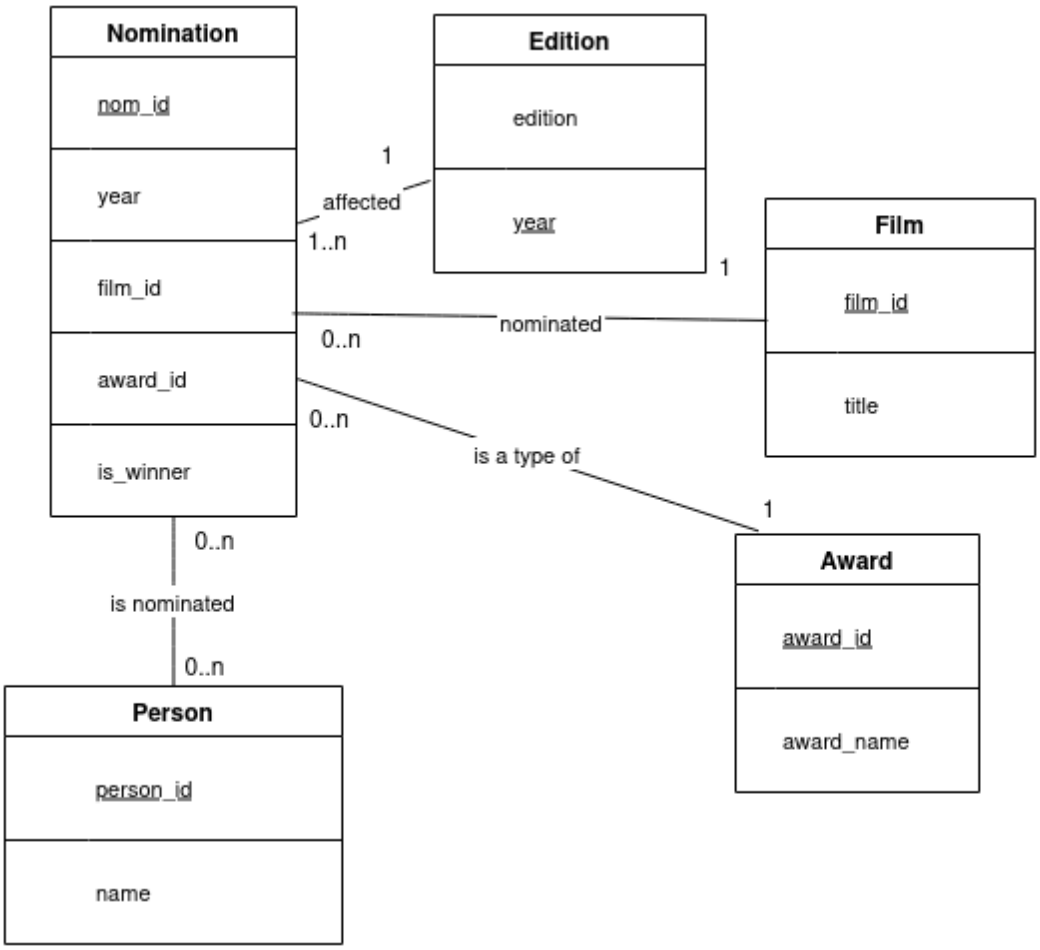

Figure 1: Entity-Relationship Diagram for the Oscars Dataset

The `Film` table represents the movies with their title and an ID to identify them, helping to manage films with potentially identical titles. Furthermore, this implementation will allow us in the future to add more detailed data linked to the movies.

The `Person` table operates similarly, allowing us to reference every individual appearing in the `nomination actor` or `nomination producers` columns. This enables us to link them as needed to the various nominations involving one or more people. This `Person` table can also be used in the future to link the `Oscars` data to more precise information about the individuals involved in the nominations.

We also decided to separate the awards into a dedicated `Award` table to avoid repetition and to store the ID of an award for each nomination, allowing us to subsequently find the award type.

Although the edition and the year are closely linked, we decided to keep both and place them in an `Edition` table to save the data and ensure easy access.

Finally, the core table, the `Nomination` table, groups the information related to the years of Oscar, the film IDs, the award IDs, and whether the person/film is the winner or not (`is_winner`). Most elements in the `Nomination` table will be foreign keys, linking this central table to the other tables in our schema.

2.1.2 LDM

Once the entity-relationship schema was defined, the next step was to translate this conceptual model into a logical data model (LDM). This implementation involved translating the entities into tables, defining the columns, specifying the primary keys and foreign keys, and establishing the constraints derived from the initial conceptual model.

The nomination relationship is many-to-many, so we have introduced a table to store the pairs nomination-person and incorporate the actors roles (character).

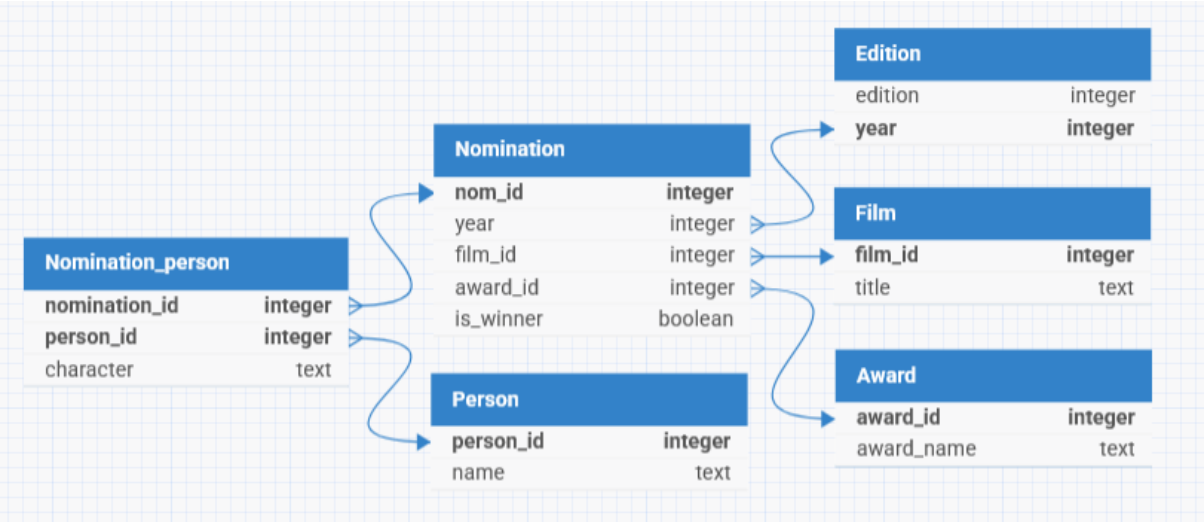

Figure 2: LDM diagram for the Oscars Dataset

LDM schema:
Edition : Edition(**year**, edition),
Film : Film(**film_id**, title),
Award : Award(**award_id**, award_name),
Person : Person(**person_id**, name),
Nomination : Nomination(**nom_id**, #year, #film_id, #award_id, is_winner),
Nomination_person(**#nomination_id**, **#person_id**, character)

2.1.3 Database constraints

We have determined the folowwing database constraints:

- The IDs used as primary keys for all tables are **auto-generated integers**.
- The primary key for the **Nomination_Person** table is formed by the tuple (#nomination_id, #person_id), which is are the foreign key too.
- Except the attribute **character** in the Nomination_Person table, all other attributes are defined as **not null**, as the tables would otherwise lose critical information.
- The **character** attribute might be **null**, as a person may be nominated without having a specific character role in the film (e.g., a director or a makeup artist).

2.2 Data preprocessing

Before creating the relational tables, we performed several preprocessing steps on the raw `oscars.csv` dataset in order to clean the data, remove inconsistencies, and prepare the attributes for normalization.

1. **Loading the dataset.** We imported the dataset obtained from Kaggle and inspected its dimensionality and the proportion of missing values in each column.

2. **Removal of nominations without an associated film.** Around 11% of rows had `film_title = NaN`, corresponding to honorary prizes or awards not linked to a specific movie. Since such entries cannot be associated with the `Film` table, all rows with missing `film_title` were removed.

3. **Dropping irrelevant or unusable text columns.** The dataset contains several long textual fields that are not needed for the database design (e.g., citations, award descriptions, or speech transcripts). The following columns were removed:

   - `nomination_citation`
   - `nomination_description`
   - `acceptance_speech_text`
   - `nomination_country`
   - `acceptance_speech_url`

4. **Unifying nominee information.** The dataset distinguishes between actors (`nomination_actor`) and producers (`nomination_producers`), but only one of these fields is non-null for each row. To consolidate the schema, we merged both into a single field:

$$\texttt{nomination\_people} = \begin{cases} \texttt{nomination\_actor} & \text{if available,} \\ \texttt{nomination\_producers} & \text{otherwise.} \end{cases}$$

   The two original columns were then dropped.

5. **Parsing and normalizing the list of people.** The `nomination_people` column contains highly inconsistent free-form text, including multiple nominees in a single string, connectors (*and*, `&`), group names, company names, and other patterns not corresponding to individuals. To extract valid person names, we defined a custom parsing function that:

   (a) normalizes connectors (replacing "&" and "and" with commas),

   (b) splits each entry into candidate segments,

   (c) removes entries corresponding to groups, organizations, or single-word entities,

   (d) reconstructs incomplete name patterns (e.g. "John and Mary Smith" → "John Smith", "Mary Smith"),

   (e) keeps only segments that contain at least two words (one word defines a country).

   The result is stored as a new list-valued column:

```
oscars['people_list'] = oscars.nomination_people.apply(parse_people)
```

6. **Removing the intermediate column.** After extracting the structured list of names, the original `nomination_people` text column was removed.

This preprocessing ensured that the dataset was clean, consistent, and fully compatible with the relational schema we created before.

2.3 Database creation

After preprocessing the data, we needed to create the database itself. We use the PostgreSQL DBMS and its corresponding syntax.

1. **Schema creation.** We create a new schema called films.

2. **Creating tables.** We create the tables in accordance with the LDM defined above. To create primary keys, we use the PostgreSQL syntax INTEGER GENERATED ALWAYS AS IDENTITY PRIMARY KEY. We define constraints for primary keys, foreign keys, unique constraints, and NOT NULL constraints. All table columns, except the character column, cannot be NULL, as this would violate the logic of the database. In addition, when defining a FOREIGN KEY, we specify ON DELETE CASCADE to ensure that the database remains consistent when rows in related tables are deleted.

2.4 Table population

After we created all the necessary tables, we wanted to fill them with the data existing in the Oscar dataset. For this, at first, we attempted to populate the tables by issuing individual `INSERT INTO` statements for each row.

While being functionally correct, this method worked very slowly. Executing many individual SQL statements over a database connection is slow because each statement must be parsed, planned, executed, and committed separately.

After that we tried to find an appropriate solution on the Internet. We decided to use bulk-insert. Instead of sending thousands of individual SQL commands to the database server, a bulk insert groups many rows together and sends them as a single, more efficient request.

In other words, instead of adding rows to the tables one by one by extracting data from the pandas DataFrame, we will now create a temporary table `tmp_oscars` directly in the database, perform a single bulk insert to send all the data there, and then distribute the data from this table into the database tables within the SQL server itself. To do this, we perform the following main steps:

1. **Dataset preparation.** We convert the `people_list` column to a string because a column in SQL cannot have a list type. Then we write the contents of the table to a buffer `csv_buffer` (an in-memory CSV) so that we can later simulate reading the table from STDIN.

2. **Creating the temporary table.** We create a temporary table `tmp_oscars` that mirrors the columns of the original `oscars` table. All the data will then be inserted into it.

3. **Bulk insert.** We use the `copy_expert` function of psycopg2 to run a COPY FROM STDIN. It performs a one-time bulk insert of all the rows from the `oscars` table into `tmp_oscars`.

4. **Inserting into Edition.** We find all unique pairs of edition and year in `tmp_oscars` and insert them into the `Edition` table.

5. **Inserting into Award.** We find all unique values of the `award` column in `tmp_oscars` and insert them into the `Award` table.

6. **Inserting into Film.** We find all unique values of the `film_title` column in `tmp_oscars` and insert them into the `Film` table.

7. **Inserting into Nomination.** We populate the `Nomination` table by selecting data from `tmp_oscars` and linking it with the corresponding entries in `Film` and `Award`. For each

row in `tmp_oscars`, we match the film title to its `film_id` and the award name to its `award_id`, then insert the combination of `year`, `film_id`, `award_id`, and `is_winner` into the `Nomination` table.

8. **Inserting into Person.** We extract unique individual person names associated with nominations. First, we expand the comma-separated `people_list` from `tmp_oscars` into separate rows using `unnest`, while also linking each entry to its corresponding nomination record. From this expanded dataset, we select all distinct, trimmed person names and insert them into the `Person` table.

9. **Inserting into Nomination_person.** We link each nomination to the people associated with it. Using the expanded dataset where the `people_list` has already been split into individual names we match each person to their corresponding record in the `Person` table. For every combination of nomination and person, we insert a row into `Nomination_person`, including the `character` name when applicable. If a given (`nomination_id`, `person_id`) pair already exists, the insertion is skipped (ON CONFLICT DO NOTHING).

2.5 Queries

Once all the tables were set up, we were able to perform a number of queries in order to verify the database functionality. These queries allow us to both verify the operation of the database and simulate the different information that a user might request.

Example queries :
1. Count the number of times a film was nominated for Oscar for each film and order them by a number of nominations.

```
SELECT film.title, COUNT(Nomination.nom_id) AS nom_total
FROM film
JOIN nomination ON film.film_id = nomination.film_id
GROUP BY film.title
ORDER BY nom_total desc;
```

Table 1: Result for query 1

title	**nom_total**
A Star Is Born	25
West Side Story	18
Titanic	16
Moulin Rouge	15
Little Women	14

2. List all categories in which Titanic was nominated.

```
SELECT award.award_name
FROM award
JOIN nomination ON nomination.award_id = award.award_id
JOIN film ON nomination.film_id = film.film_id
WHERE film.title = 'Titanic'
```

3. Get the names of people who played Joker in a movie.

Table 2: Result for query 2

award_name
Actress In A Leading Role
Actress In A Supporting Role
Art Direction
Best Picture
Cinematography
Costume Design
Directing
Film Editing
Makeup
Music (Original Dramatic Score)
Music (Original Song)
Sound
Sound Effects Editing
Visual Effects
Art Direction (Black-And-White)
Writing (Story And Screenplay)

```
SELECT person.name
FROM person
JOIN nomination_person ON person.person_id = nomination_person.person_id
WHERE nomination_person.character = 'Joker'
```

Table 3: Result for query 3

name
Heath Ledger

4. Get the film with the highest win-to-nomination ratio.

```
WITH film_stats AS (
SELECT film.title, n.year, COUNT(n.nom_id) AS total_noms,
SUM(n.is_winner::INT) AS total_wins
FROM film
JOIN nomination AS n ON film.film_id = n.film_id
GROUP BY film.title, n.year
HAVING COUNT(n.nom_id) >= 5
)
SELECT title, year, total_wins, total_noms, total_wins / total_noms AS conversion_rate
FROM film_stats
ORDER BY conversion_rate DESC, total_noms DESC LIMIT 1
```

Table 4: Result for query 4

title	year	total_wins	total_noms	conversion_rate
The Lord of the Rings: The Return of the King	2003	11	11	1

These queries demonstrate that the table schema allows for the execution of queries to efficiently extract information regarding the Oscars, films, and actors, as well as their roles.

3 New data: movies dataset

3.1 Database Modeling

3.1.1 CDM

After the initial implementation of the database and its populating someone comes to us and asks us if it is possible to add their film dataset to our current database in order to be able to make links between film-related information and the awards or nominations attributed to them.

First, we got rid of irrelevant columns (for example, `backdrop_path` giving the link to an image for the film, or `adult` which we used at the beginning to clean the data by removing adult films). We kept the column `release_date` allowing us to date the film releases and make links with the Oscars dataset; the columns `vote_average` and `vote_count` giving information on the film's popularity; a number of columns giving numerical information such as `revenue`, `runtime` and `budget`.

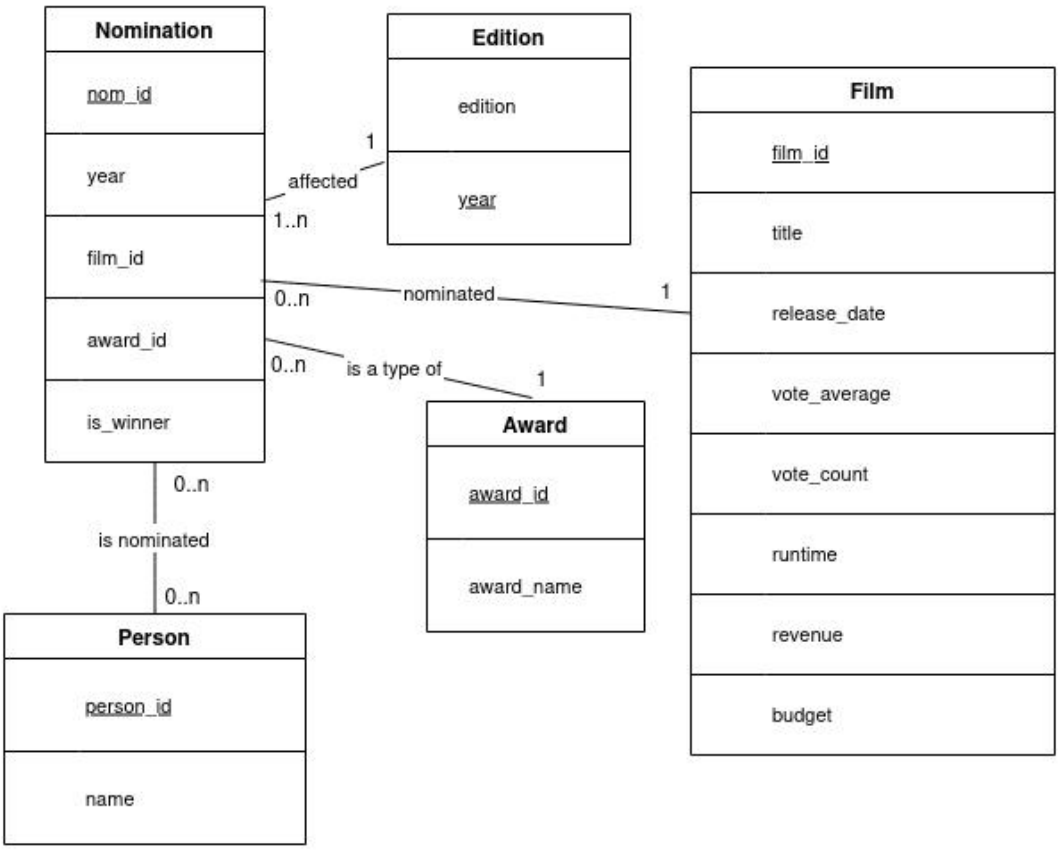

Figure 3: Entity-Relationship Diagram for the Films and Oscars datasets

3.1.2 LDM

Finally, the addition of this dataset adds attributes to our table but does not change its primary key and preserves the `title` attribute.

Given that some films present in the `sscars` dataset might not be present in a new dataset, the attributes added to our film entity may be null.

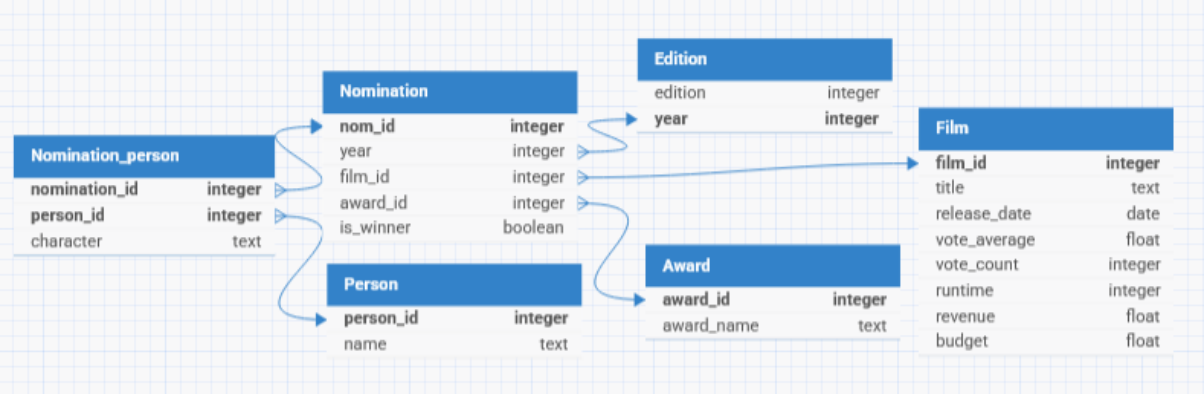

Figure 4: LDM diagram for the Films and Oscars datasets

3.2 Preprocessing

Once the data was imported, a number of modifications were made in order to prepare and clean the data.

Deleting useless rows. One of the first steps was to delete rows considered as useless. Such as the adult movie, the movies without any name or the movies not even released (which can't appear in the `Oscars` dataset).

After that, we decided to **delete duplicate rows**. However, to define what constituted a duplicate, we had to define conditions. Two films with the same name are considered duplicates if and only if they were released in the same year. Given that films can share the same name, this was necessary to remove double entries without deleting actual films (Example: A star is born 2018 and A star is born 1976).

Cleaning data. The values in the `release_date` column were defined as datetime to process them accordingly. When the attributes `budget`, `revenue`, and `runtime` were equal to 0, we decided to consider them as NaN, as this data makes no sense and could subsequently distort our queries. Similarly, when the attribute `vote_count` was null, we decided to modify the attribute `vote_average` to NaN, so as not to distort the queries (0 votes have no average).

Filtering films present in both datasets. In order to add the information present in the second dataset to the existing films in the `Oscars` dataset. We had to filter the films by title correspondence by putting all titles in lower case to avoid case sensitivity.

Once this data was collected, we sorted the films by title and by the film's release date closest to the date of the first Oscar nomination. When the corresponding films were found, 4733 films matched out of 5090 unique films in the `Oscars` dataset.

3.3 Table Modification and Creation

Once the data was prepared, we had to modify the existing film table taking into account the already existing tables and relationships. For this, the columns `release_date`, `vote_average`, `vote_count`, `runtime`, `revenue` and `budget` were added. `revenue` and `budget` have NUMERIC type, which is better for large data.

Also, the title uniqueness constraint had to be removed to accommodate the arrival of a large amount of data, leading to duplicate titles, but we added a new uniqueness constraint on the pair (`title`, `release_date`).

3.4 Table population

Once the table was updated, we added the related data from the `Oscars` dataset to the existing films in the database, then took a reasonable number of samples from the second dataset (10,000) making sure not to add duplicates.

3.5 Queries

After everything was set up, we made various queries:
1. Top 10 of the highest rated films (among those with more than 10,000 votes) and their number of victories

```
WITH film_win AS ( select film_id, count(*) AS nb_win
FROM nomination WHERE is_winner=true GROUP BY film_id)
SELECT title, vote_average, nb_win
FROM film F JOIN film_win fw USING (film_id)
WHERE vote_average IS NOT NULL AND vote_count > 10000
ORDER BY vote_average DESC
LIMIT 10;
```

Table 5: Result of query 1

title	**vote_average**	**nb_win**
The Godfather	8.707000	3
The Godfather Part II	8.591000	6
Schindler's List	8.573000	7
Spirited Away	8.539000	1
Parasite	8.515000	4
The Dark Knight	8.512000	2
Pulp Fiction	8.488000	1
Forrest Gump	8.477000	6
The Lord of the Rings: The Return of the King	8.474000	11
Life Is Beautiful	8.455000	3

2. Films with `vote_average` below 6 nominated for Oscar.

```
SELECT DISTINCT f.title, f.vote_average, n.is_winner
FROM film f
JOIN nomination n USING(film_id)
WHERE vote_average < 6 AND vote_count > 1000;
```

3. Average budget of nominated films with Brad Pitt.

```
WITH brad_movie AS (
    SELECT n.film_id
    FROM nomination n
    JOIN nomination_person np ON n.nom_id = np.nomination_id
    JOIN person p USING(person_id)
    WHERE p.name = 'Brad Pitt'
```

Table 6: Result of query 2

title	vote_average	is_winner
102 Dalmatians	5.456000	False
Babe: Pig in the City	5.553000	False
Batman Forever	5.409000	False
Fifty Shades of Grey	5.882000	False
Hail, Caesar!	5.914000	False
Hollow Man	5.918000	False
Into the Woods	5.750000	False
Junior	5.167000	False
Mirror Mirror	5.918000	False
Norbit	5.584000	False
Poseidon	5.841000	False
Snow White and the Huntsman	5.982000	False
Suicide Squad	5.909000	True
Superman Returns	5.738000	False
The Black Dahlia	5.621000	False
The Midnight Sky	5.766000	False
The Nutty Professor	5.625000	True
The Wolfman	5.842000	True

```
    )
SELECT AVG(f.budget)
FROM film f
JOIN brad_movie USING(film_id);
```

Table 7: Result of query 3

avg
59600000.000000

4. Find the film that won an Oscar with the lowest budget.

```
SELECT f.title, f.budget
FROM film f
JOIN nomination n ON f.film_id = n.film_id
JOIN award a ON n.award_id = a.award_id
WHERE n.is_winner AND f.budget != 'NaN'
ORDER BY  f.budget ASC
LIMIT 1;
```

Table 8: Result of query 4

title	budget
Kiss of the Spider Woman	11.000000

Finally, linking these two tables allows us to connect important information and create links between different table attributes and/or different datasets. For example, there is a strong

correlation between public opinion and the number of Oscar wins. We can see that the top 10 films in terms of public opinion are award-winning films.

4 Prospects

We have created a structured and clear database schema in order to simplify the integration of new entries and/or new datasets. Adding a dataset concerning the people involved in the Oscars dataset would be possible by linking the `Person` table.

If, in the future, such table `Person` is added to the database, a number of useful queries can be made, for example, the youngest Oscar nominees or French film directors featured in nominated films.

The project code is hosted on GitHub:
`https://github.com/Danila-Pechenev/databases_project`