Rony Kositsky – 205817893
Iris Taubkin – 208410969

Software Documentation

Data sources

We used two sources for our data:

- We downloaded csv data entries from https://www.kaggle.com. The database is using data from the movies website IMBD.
- We used tmbdv3api as our API source for the movies data.

Database

Our database design is as follows:

- Actors(actor_id, actor_name, bio)
- Movies(movie_id, title, movie_year)
- Movies_budget(movie_id, budget)
- Movies_countries(movie_id, country, language)
- Movies_genres(movie_id, genres)
- Movies_productions(movie_id, director, writer, productions, actors, description)
- Movies_reviews(movie_id, reviews_from_users, reviews_from_critics)
- Movies_votes(movie_id, avg_votes, votes, critics_votes)
- Ratings(movie_id, male_avg, female_avg)

We wanted to create small, dynamic, and readable tables which we could easily read and analyze. Each table catches one aspect of the movie production we wanted to capture, and our analysis will use several joins to answer the questions we want to ask.

A total of several tens of thousands of data entries were entered into the project.

Queries

We designed 8 different queries:

1. Top movies.
    - We get the data from the connection between the movies table and movies_votes table.

2. Top movies with at least X reviews.
    - We get the data from the connection between the movies, movies_votes and movies reviews table.

3. Counts number of movies of each genre (from top movies).
    - We get the data from the connection between the movies, movies_genres and movies_votes tables.

4. Get actor and movies for the actors that are chosen by the user.
    - We get the data from the connection between the movies, actors and movies_productions tables.

5. Number of movies from each country and the average vote of them.
    - We get the data from the connection between the movies_votes and movies_countries tables.

6. Union of all the movies of 3 categories chosen by the user.
    - We conducting several union calls and doing full text search from the movie_productions table.

7. Table with countries total movies budget and avg vote of them.
    - We get the data from the connection between the movies_countries, movies_budget and movies_votes tables.

8. Top rated movies by men union 100 top rated movies by women.
    - We conduct 2 unions from the data we connect between ratings and movies tables.

Optimization

We created several indexing for the full text search we are conducting to optimize our queries.

- In movies_productions table we index the actors and descriptions columns because we will try to match the strings.
- In actors table we index the names and bio because we will be doing string matching with those attributes as well.
- In addition we indexed the votes in movies_votes table. We are using the votes in several queries and it will improve our running time.

Those index selections making all our queries run pretty fast and getting the execution results we aimed for.

Rony Kositsky – 205817893
Iris Taubkin – 208410969

We will go over our code to review its functionality.

Modified Connector

```
class ModifiedConnector:
    """..."""
    def __init__(self):...

    def execute_query(self, cmd):
        self._cursor.execute(cmd)

    def execute_query_with_params(self, cmd, data):
        self._cursor.execute(cmd, data)

    def execute(self, cmd):
        self._cursor.execute(cmd)
        self._connector.commit()

    def execute_with_params(self, cmd, data):
        self._cursor.execute(cmd, data)
        self._connector.commit()

    def close(self):
        self._connector.close()

    def fetch_data(self):
        return self._cursor.fetchall()
```

We use this class to envelope the MySql connector class.

Rony Kositsky – 205817893
Iris Taubkin – 208410969

DB creation

The create table function is

```
def create_table(cmd):
    try:
        connector.execute(cmd)
    except Exception as e:
        print(f"{type(e).__name__} at line {e.__traceback__.tb_lineno} of {__file__}: {e}")
```

To create all the tables we use:

```
def create_db_tables():
    """..."""
    cmds = ['CREATE TABLE movies (movie_id INT, title VARCHAR(1000), movie_year INT, PRIMARY KEY(movie_id));',
            'CREATE TABLE movies_genres (movie_id INT, genres VARCHAR(1000) , PRIMARY KEY(movie_id));',
            'CREATE TABLE movies_countries (movie_id INT, country VARCHAR(1000), language VARCHAR(1000), PRIMARY KEY('
            'movie_id));',
            'CREATE TABLE movies_productions (movie_id INT, director VARCHAR(1000), writer VARCHAR(1000), productions '
            'VARCHAR(1000), actors VARCHAR(1000), description VARCHAR(1000) , PRIMARY KEY(movie_id), FULLTEXT(actors),'
            ' FULLTEXT(description));',
            'CREATE TABLE movies_votes (movie_id INT, avg_votes FLOAT, votes INT, critics_votes INT , PRIMARY KEY('
            'movie_id));',
            'CREATE TABLE movies_budget (movie_id INT, budget FLOAT , PRIMARY KEY(movie_id));',
            'CREATE TABLE actors (actor_id INT, actor_name VARCHAR(1000), bio VARCHAR(10000) , PRIMARY KEY(actor_id), '
            'FULLTEXT(actor_name), FULLTEXT(bio));',
            'CREATE TABLE actors_birth (actor_id INT, place_of_birth VARCHAR(1000) , PRIMARY KEY(actor_id));',
            'CREATE TABLE ratings (movie_id INT, male_avg FLOAT, female_avg FLOAT , PRIMARY KEY(movie_id));',
            'CREATE TABLE movies_reviews (movie_id INT, reviews_from_users INT, reviews_from_critics INT);',
            'CREATE INDEX actor_index ON actors (actor_name);',
            'CREATE INDEX votes_index ON movies_votes (votes);']

    for cmd in cmds:
        create_table(cmd)
```

Rony Kositsky – 205817893
Iris Taubkin – 208410969

Data insertion

```
def insert_all_data():
    insert_movies_data()
    insert_movies_reviews_data()
    insert_movies_genres_data()
    insert_movies_countries_data()
    insert_movies_productions_data()
    insert_movies_votes_data()
    insert_movies_budget_data()
    insert_actors_data()
    insert_ratings_data()


def modifications():
    cmd = '''ALTER TABLE actors MODIFY bio VARCHAR(15000);'''
    connector.execute(cmd)


if __name__ == "__main__":
    insert_all_data()
    # modifications()
    connector.close()
```

We call this function to fill all the tables' data.

Rony Kositsky – 205817893
Iris Taubkin – 208410969

```python
def insert_movies_reviews_data():
    csv_reader = open_csv_reader("IMDb movies.csv")
    cmd = "INSERT INTO movies_reviews (movie_id, reviews_from_users, reviews_from_critics) VALUES (%s, %s, %s)"

    for row in list(csv_reader)[1:]:
        try:
            movie_id = int(row[0][2:])
            reviews_from_users = None
            if row[20] != '':
                reviews_from_users = float(row[20])
            reviews_from_critics = None
            if row[21] != '':
                reviews_from_critics = float(row[21])

            year = int(''.join((ch if ch in '0123456789' else '') for ch in row[3]))
            if year < 1940:
                continue
            data = (movie_id, reviews_from_users, reviews_from_critics)
            connector.execute_with_params(cmd, data)
        except Exception as e:
            print(f"{type(e).__name__} at line {e.__traceback__.tb_lineno} of {__file__}: {e}")
    print("Finished")
```

```python
def insert_movies_data():
    csv_reader = open_csv_reader("IMDb movies.csv")
    cmd = "INSERT INTO movies (movie_id, title, movie_year) VALUES (%s, %s, %s)"

    for row in list(csv_reader)[1:]:
        try:
            movie_id = int(row[0][2:])
            title = row[1]
            year = int(''.join((ch if ch in '0123456789' else '') for ch in row[3]))
            if year < 1940:
                continue
            data = (movie_id, title, year)
            connector.execute_with_params(cmd, data)
        except Exception as e:
            print(f"{type(e).__name__} at line {e.__traceback__.tb_lineno} of {__file__}: {e}")
    print("Finished")
```

Rony Kositsky – 205817893
Iris Taubkin – 208410969

```python
def insert_movies_genres_data():
    csv_reader = open_csv_reader("IMDb movies.csv")
    cmd = "INSERT INTO movies_genres (movie_id, genres) VALUES (%s, %s)"
    for row in list(csv_reader)[1:]:
        try:
            movie_id = int(row[0][2:])
            year = int(''.join((ch if ch in '0123456789' else '') for ch in row[3]))
            if year < 1940:
                continue
            genre = row[5]
            data = (movie_id, genre)
            connector.execute_with_params(cmd, data)
        except Exception as e:
            print(f"{type(e).__name__} at line {e.__traceback__.tb_lineno} of {__file__}: {e}")
    print("Finished")
```

```python
def insert_movies_countries_data():
    csv_reader = open_csv_reader("IMDb movies.csv")
    cmd = "INSERT INTO movies_countries (movie_id, country, language) VALUES (%s, %s, %s)"
    for row in list(csv_reader)[1:]:
        try:
            movie_id = int(row[0][2:])
            country = row[7]
            language = row[8]
            year = int(''.join((ch if ch in '0123456789' else '') for ch in row[3]))
            if year < 1940:
                continue
            if language.strip() == "None" or language.strip() == "":
                language = None
            data = (movie_id, country, language)
            connector.execute_with_params(cmd, data)
        except Exception as e:
            print(f"{type(e).__name__} at line {e.__traceback__.tb_lineno} of {__file__}: {e}")
    print("Finished")
```

Rony Kositsky – 205817893
Iris Taubkin – 208410969

```python
def insert_movies_productions_data():
    csv_reader = open_csv_reader("IMDb movies.csv")
    cmd = "INSERT INTO movies_productions (movie_id, director, writer, productions, actors, description)" \
          " VALUES (%s, %s, %s, %s, %s, %s)"
    for row in list(csv_reader)[1:]:
        try:
            year = int(''.join((ch if ch in '0123456789' else '') for ch in row[3]))
            if year < 1940:
                continue
            movie_id = int(row[0][2:])
            director = row[9]
            writer = row[10]
            productions = row[11]
            actors = row[12]
            description = row[13]
            data = (movie_id, director, writer, productions, actors, description)
            connector.execute_with_params(cmd, data)
        except Exception as e:
            print(f"{type(e).__name__} at line {e.__traceback__.tb_lineno} of {__file__}: {e}")
    print("Finished")
```

```python
def insert_movies_votes_data():
    csv_reader = open_csv_reader("IMDb movies.csv")
    cmd = "INSERT INTO movies_votes (movie_id, avg_votes, votes, critics_votes) VALUES (%s, %s, %s, %s)"

    for row in list(csv_reader)[1:]:
        try:
            year = int(''.join((ch if ch in '0123456789' else '') for ch in row[3]))
            if year < 1940:
                continue
            movie_id = int(row[0][2:])
            avg_votes = float(row[14])
            votes = int(row[15])
            critics_votes = 0
            if row[21] != "":
                critics_votes = float(row[21])
            data = (movie_id, avg_votes, votes, critics_votes)
            connector.execute_with_params(cmd, data)
        except Exception as e:
            print(f"{type(e).__name__} at line {e.__traceback__.tb_lineno} of {__file__}: {e}")
    print("Finished")
```

Rony Kositsky – 205817893
Iris Taubkin – 208410969

```python
def insert_movies_budget_data():
    csv_reader = open_csv_reader("IMDb movies.csv")
    cmd = "INSERT INTO movies_budget (movie_id, budget) VALUES (%s, %s)"
    for row in list(csv_reader)[1:]:
        try:
            year = int(''.join((ch if ch in '0123456789' else '') for ch in row[3]))
            if year < 1940:
                continue
            movie_id = int(row[0][2:])
            budget_str = row[16]
            budget = 0
            if "$" in budget_str:
                budget = float(''.join((ch if ch in '0123456789' else '') for ch in budget_str))
            data = (movie_id, budget)
            connector.execute_with_params(cmd, data)
        except Exception as e:
            print(f"{type(e).__name__} at line {e.__traceback__.tb_lineno} of {__file__}: {e}")
    print("Finished")
```

```python
def insert_actors_data():
    csv_reader = open_csv_reader("IMDb names.csv")
    cmd = "INSERT INTO actors (actor_id, actor_name, bio) VALUES (%s, %s, %s)"
    MAX_SIZE = 15000

    for row in list(csv_reader)[1:]:
        try:
            actor_id = int(row[0][2:])
            name = row[1]
            bio = row[4]
            if len(bio) > MAX_SIZE:
                bio = bio[:MAX_SIZE]
            data = (actor_id, name, bio)
            connector.execute_with_params(cmd, data)
        except Exception as e:
            print(f"{type(e).__name__} at line {e.__traceback__.tb_lineno} of {__file__}: {e}")
    print("Finished")
```

Rony Kositsky – 205817893
Iris Taubkin – 208410969

```
def insert_ratings_data():
    csv_reader = open_csv_reader("IMDb ratings.csv")
    cmd = "INSERT INTO ratings (movie_id, male_avg, female_avg) VALUES (%s, %s, %s)"
    for row in list(csv_reader)[1:]:
        try:
            movie_id = int(row[0][2:])
            male_avg = None
            if row[23] != '':
                male_avg = float(row[23])
            female_avg = None
            if row[33] != '':
                female_avg = float(row[33])
            data = (movie_id, male_avg, female_avg)
            connector.execute_with_params(cmd, data)
        except Exception as e:
            print(f"{type(e).__name__} at line {e.__traceback__.tb_lineno} of {__file__}: {e}")
    print("Finished")
```

“Server”

We didn’t implement a real server in this project, but if we had our query would run with those commands from the back end application and the input we would get from the user (the inputs are explained in the second documentation file).

```
def query_1(num_of_votes, avg_vote):
    cmd = '''select m.title, mmv.votes, mmv.avg_votes
            from movies as m,
            (select mv.votes as votes, mv.movie_id as movie_id, mv.avg_votes as avg_votes
            from movies_votes as mv
            where mv.votes >= %s and mv.avg_votes > %s) as mmv
            where m.movie_id = mmv.movie_id
            order by mmv.avg_votes desc'''

    try:
        cursor.execute_with_params(cmd, (num_of_votes, avg_vote))
    except Exception as e:
        print(f"{type(e).__name__} at line {e.__traceback__.tb_lineno} of {__file__}: {e}")

    ret_data = cursor.fetch_data()
    for (title, votes, avg_votes) in ret_data:
        print("title : {}, votes: {}, avg_votes: {}".format(title, votes, avg_votes))
```

Rony Kositsky – 205817893
Iris Taubkin – 208410969

```
def query_2(num_of_votes, avg_vote, num_of_reviews):
    cmd = '''select tm.title, mr.reviews_from_users
            from movies_reviews as mr,
            (select m.title, mmv.votes, mmv.avg_votes, m.movie_id
            from movies as m,
            (select mv.votes as votes, mv.movie_id as movie_id, mv.avg_votes as avg_votes
            from movies_votes as mv
            where mv.votes >= %s and mv.avg_votes > %s) as mmv
            where m.movie_id = mmv.movie_id
            order by mmv.avg_votes desc ) as tm
            where tm.movie_id = mr.movie_id
            and mr.reviews_from_users > %s'''

    try:
        cursor.execute_with_params(cmd, (num_of_votes, avg_vote, num_of_reviews))
    except Exception as e:
        print(f"{type(e).__name__} at line {e.__traceback__.tb_lineno} of {__file__}: {e}")

    ret_data = cursor.fetch_data()
    for (title, reviews_from_users) in ret_data:
        print("title : {}, reviews_from_users: {}".format(title, reviews_from_users))
```

```
def query_3(avg_vote):
    cmd = '''select mg.genres, count(*)
            from movies_genres mg,
            (select m.movie_id
            from movies_votes m
            where m.avg_votes > (%s)) as topm
            where mg.movie_id = topm.movie_id
            group by mg.genres''' % (avg_vote)

    try:
        # cursor.execute_with_params(cmd, (avg_vote))
        cursor.execute_query(cmd)
    except Exception as e:
        print(f"{type(e).__name__} at line {e.__traceback__.tb_lineno} of {__file__}: {e}")

    ret_data = cursor.fetch_data()
    for (genre, count) in ret_data:
        print("genre : {}, count: {}".format(genre, count))
```

Rony Kositsky – 205817893
Iris Taubkin – 208410969

```
def query_4(list_of_actors_inp):
    list_of_actors = list_of_actors_inp.split(',')
    data = []
    cmd = '''select a.actor_name, a.title
                from
                (select a.actor_name, m.title
                from actors a, movies m, movies_productions mp
                WHERE MATCH(mp.actors) AGAINST (%s)
                and m.movie_id = mp.movie_id
                and  a.actor_name = %s) as a
                union
                select b.actor_name, b.title
                from
                (select a.actor_name, m.title
                from actors a, movies m, movies_productions mp
                WHERE MATCH(mp.actors) AGAINST (%s)
                and m.movie_id = mp.movie_id
                and  a.actor_name = %s) as b
                union
                select c.actor_name, c.title
                from
                (select a.actor_name, m.title
                from actors a, movies m, movies_productions mp
                WHERE MATCH(mp.actors) AGAINST (%s)
                and m.movie_id = mp.movie_id
                and  a.actor_name = %s) as c'''

    for actor in list_of_actors:
        data += [actor.strip(), actor.strip()]

    try:
        print(cmd, str(tuple(data)))
        cursor.execute_with_params(cmd, tuple(data))
```

Rony Kositsky – 205817893
Iris Taubkin – 208410969

```python
def query_5():
    cmd = '''select mc.country, count(mv.movie_id), avg(mv.votes)
            from movies_countries mc, movies_votes mv
            where mc.movie_id = mv.movie_id
            group by mc.country'''

    try:
        cursor.execute_query(cmd)
    except Exception as e:
        print(f"{type(e).__name__} at line {e.__traceback__.tb_lineno} of {__file__}: {e}")

    ret_data = cursor.fetch_data()
    for (country, count, avg) in ret_data:
        print("country : {}, number of movies: {}, average vote: {}".format(country, count, avg))
```

```python
def query_6(list_of_categories_inp):
    list_of_categories = list_of_categories_inp.split(',')
    cmd = '''select mc.title
            from (select m.title
            from movies m, movies_productions mp
            WHERE MATCH(mp.description) AGAINST (%s)) as mc
            union
            select mr.title
            from (select m.title
            from movies m, movies_productions mp
            WHERE MATCH(mp.description) AGAINST (%s)) as mr
            union
            select mk.title
            from (select m.title
            from movies m, movies_productions mp
            WHERE MATCH(mp.description) AGAINST (%s)) as mk'''

    try:
        print(cmd, str(tuple(list_of_categories)))
        cursor.execute_with_params(cmd, tuple(list_of_categories))
    except Exception as e:
        print(f"{type(e).__name__} at line {e.__traceback__.tb_lineno} of {__file__}: {e}")
```

Rony Kositsky – 205817893
Iris Taubkin – 208410969

```
def query_7():
    cmd = '''select  mc.country, sum(mb.budget) as total_budget, avg(mv.avg_votes) as avg_country_votes
            from movies_countries mc, movies_budget mb, movies_votes mv
            where mc.movie_id = mb.movie_id
            and mc.movie_id = mv.movie_id
            group by mc.country
            having sum(mb.budget) > 0
            order by total_budget, avg_country_votes desc'''

    try:
        cursor.execute_query(cmd)
    except Exception as e:
        print(f"{type(e).__name__} at line {e.__traceback__.tb_lineno} of {__file__}: {e}")

    ret_data = cursor.fetch_data()
    for (country, total_budget, avg_country_votes) in ret_data:
        print("country : {}, total_budget: {}, average vote: {}".format(country, total_budget, avg_country_votes))
```

```
def query_8():
    cmd = '''select f.title, f.avg_vote
            from
            (select m.title, r.female_avg as avg_vote
            from ratings r, movies m
            where r.movie_id = m.movie_id
            order by r.female_avg desc
            limit 100) as f
            union
            select m.title, m.avg_vote
            from
            (select m.title, r.male_avg as avg_vote
            from ratings r, movies m
            where r.movie_id = m.movie_id
            order by r.male_avg desc
            limit 100) as m'''

    try:
        cursor.execute_query(cmd)
    except Exception as e:
        print(f"{type(e).__name__} at line {e.__traceback__.tb_lineno} of {__file__}: {e}")

    ret_data = cursor.fetch_data()
    for (title, avg_vote) in ret_data:
        print("title : {}, avg_vote: {}".format(title, avg_vote))
```