Introduction

Our team proposes the development of **FilmFinder**, aimed at providing users with an efficient way to search for movies based on various criteria. The platform will utilize the information contained in the TMDB dataset to offer users a customizable movie search experience.

1 Application System

1a System Description:

The platform will serve as a catalog of movies and related information to facilitate users to search for movies using various criteria, including *Genre, Release Date, Language, Popularity***.** A search option would be provided where users can enter their search criteria and receive appropriate movie recommendations. Additionally, users will have the option to view detailed information about each movie, including its title, poster, overview, rating, and other metadata.

Limitations:

The analysis is limited to the data available in the database, which may not cover all the information. The system does not support real time data analysis, as the data set is updated periodically. It supports only basic analysis, it does not support complex analysis.

1b Database Necessity

A database is essential for the Movie Search and Discovery Platform for the following reasons:

- Storage : The application requires an organized repository to efficiently store and manage movie data and its attributes.
- Retrieval : A database facilitates efficient retrieval of information, allowing for quick and accurate search results.
- Data Integrity : A database management system ensures data integrity, uniformity and consistency.
- Scalability : A DBMS provides a decent scalability, to handle a growing volume of movie data and user requests effectively.

1c Justification for Migrating to AWS:

Migrating the database to AWS offers several benefits :

- Cost savings: We can reduce infrastructure costs and pay only for the resources they use, leading to significant cost savings.
- Scalability and flexibility: AWS provides scalable and flexible infrastructure, allowing us to easily adjust resources based on demand.
- Disaster recovery: We can create backup copies of the data across different regions, minimizing the risk of data loss in case of a disaster.
- Accessibility: The global accessibility allows users to access the database from anywhere with an internet connection.
- Agility: AWS enables quick deployment and scaling of new applications.

2 Solution Requirements

2a Requirements of the solution:

- Data Storage and Management: The system should have a very strong data storage to securely store and manage the dataset efficiently.
- Performance: The system should deliver efficient query processing.
- Scalability: The system should be scalable to handle growing datasets and high loads by maintaining the performance.
- Security: Security measures should be implemented to protect the data.
- Highly available and Disaster Recovery: The system should ensure high availability and disaster recovery to reduce the downtime and data loss.
- Data source: The initial source of information for the application is the TMDB dataset.
- Data preprocessing: The raw data needs to be cleaned and normalized in order to ensure query-ability.
- User Interface: The system should provide a user friendly interface to interact with the system and to perform analysis.

2b System's Functionalities and Limitations:

Functionalities:

Users can explore the TMDB dataset by browsing movie metadata, genres, ratings, cast and crew, languages, production companies and keywords. Users can execute custom queries to filter data on a particular criteria. As the application grows, the users can also perform various analytics tasks such as trend analysis, popularity analysis, rating analysis, genre analysis to know more.

Here are a few functionalities and requirements:

Search Functionality: Users should be able to search for movies based on criteria such as genre, release date, language, and popularity.

Filtering Options: Provide users with filtering options to refine their search results, including genre selection, release date range, and language preferences.

Detailed Movie Information: Display detailed information about each movie, including title, poster, overview, rating, and other metadata.
Feedback Mechanism: Provide users with the ability to provide feedback on search results and suggest improvements to the platform.

Limitations:

Data Availability - The service may be limited by the availability and quality of data in the TMDB Analysis database. Inaccurate or incomplete data may lead to poor search results.
Scalability - As the user base / database size increases, we may encounter scalability challenges in terms of handling large numbers of service (data retrieval) requests.
Performance - The performance may be affected by service availability, load, and querying efficiency.

2c User Interaction:

Users will interact with the system through a web-based interface. The interface will feature search and filtering options prominently, allowing anyone, even ***unregistered users***, to have ***read only*** access. Users can access detailed information and ratings on individual movies.

For registered users, the system will provide features such as user accounts and ***search and view history*** to enhance the user experience. Feedback mechanisms, such as rating movies, will be available for registered users.

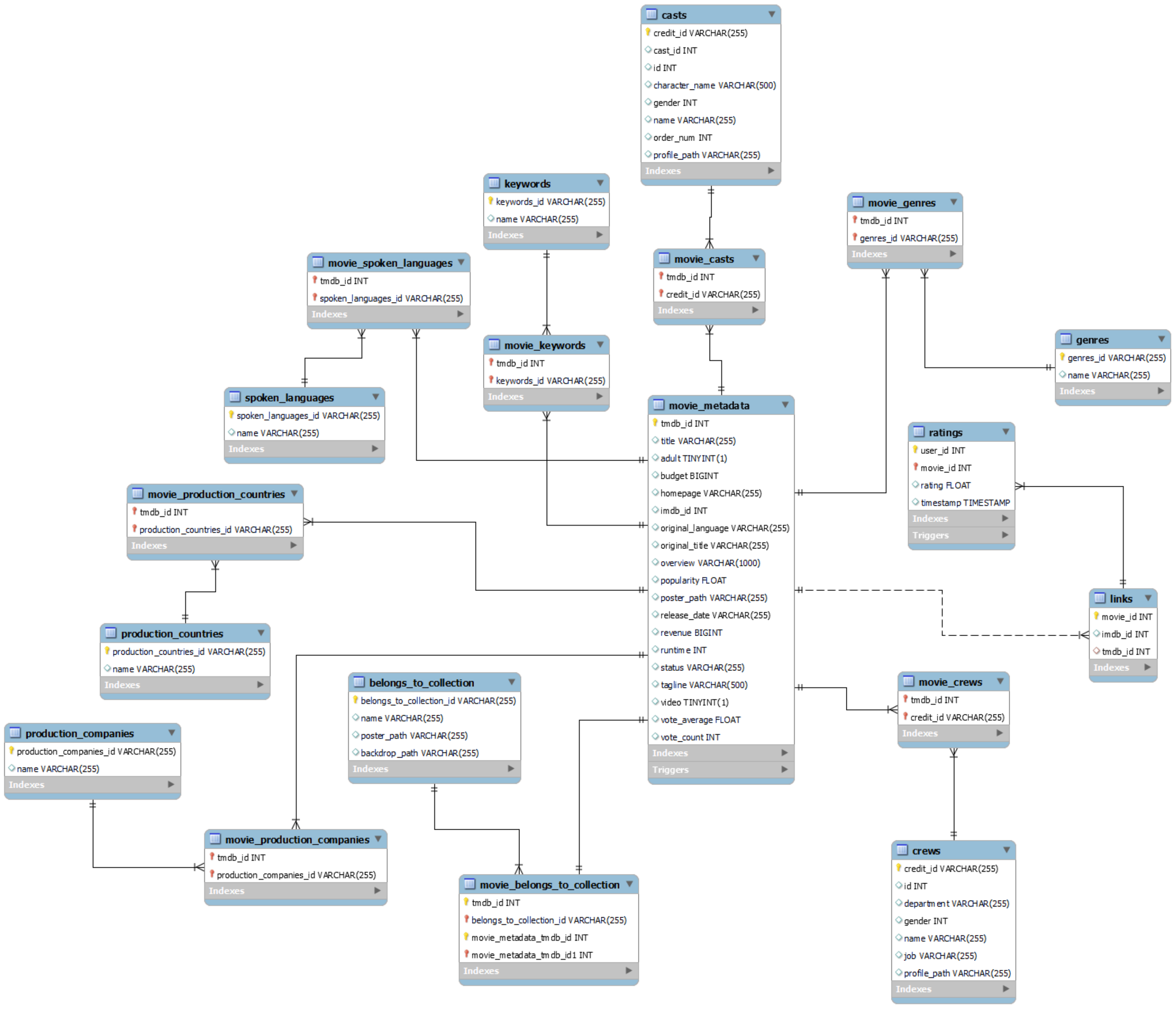
3 Conceptual Database Design

3a Database Requirements

For an application utilizing the TMDB dataset, the database requirements involve:

- **MySQL on RDS:** For scalable and reliable database management.
- **Schema Design:** For efficient storage of all the information on movies, collections, genres, etc., with appropriate keys for data integrity. (Refer ER Diagram in the next section)
- **Views:** To simplify complex queries and to enable restricted access to registered users.
- **Stored Procedures:** For common tasks like fetching recommendations or calculating statistics.
- **Triggers:** For data integrity checks, validation, or automation based on database events.

3b Entity Relationship Diagram

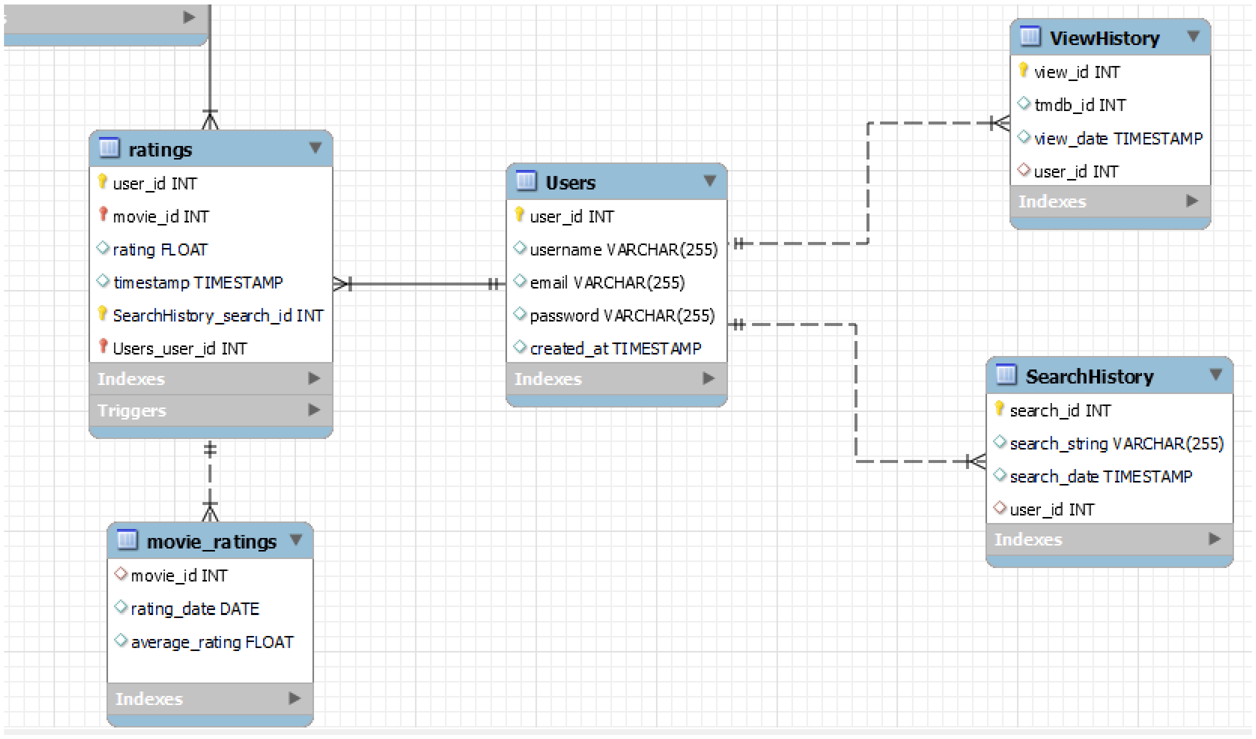


The schema is centered around *movie_metadata* for the most part with tables like *genres* and *spoken_languages* having many-to-many relationships with *movie_metadata.*

Here, *belongs_to_collection* and *movie_metadata* is a one-to-many relationship, but we have a middle table *movie_belongs_to_collection* to avoid too many rows having NULL values in the collection column.

Additional tables:

To enable user-specific functionalities in the application, we have introduced 4 more tables with user data for managing their ratings, average ratings and search / view history.



3c Normalization process:

Normalization is a crucial process in database design which is used to organize data efficiently reducing the data redundancy. Normalization is done by breaking down large tables into smaller, related tables and establishing relationships between them. In the TMDB dataset, normalization can help organize movie-related data into structured tables, eliminating duplication and improving data management.

Normalization in TMDB dataset:

1. First Normal Form (1NF): Each table needs to have atomic values. For instance, in movies_metadata.csv, columns collections, genres, spoken languages, production companies and production countries don't have atomic values. Splitting the one big table of *movie_metadata* into separate tables, *movie_metadata, belongs_to_collection, genres, production_companies, production_countries* and *spoken_languages* with their links preserved in a different set of tables - *movie_genres, movie_belongs_to_collection, movie_production_companies, movie_production_countries* and *movie_spoken_languages* respectively to achieve 1NF atomicity.

2. Second Normal Form (2NF): Non-key attributes are fully functionally dependent on the primary key, that is every attribute depends on the entire primary key, not just part of it. For instance, in the movies_metadata table, attributes such as title, release_date depend on the entire primary key, which is *tmdb_id*.

3. Third Normal Form (3NF): No transitive dependencies between non-key attributes, that is non-key attribute doesn't depend on other non-key attributes. In the TMDB dataset, no non-key attribute depends on a non-key attribute.

Primary keys in the TMDB dataset represent unique identifiers for each movie or related entity. For example, in the *movie_metadata* table, the *tmdb_id* column serves as the primary key, uniquely identifying each Movie record.

Foreign keys are established to represent relationships between different tables. For example, in the *ratings* table, the *movie_id* column may serve as a foreign key referencing the *movie_id* column in the *links* table. This establishes a relationship between the ratings table and the links table, indicating which movie each rating entry is associated with.

In summary, normalization in the TMDB dataset involves structuring data to eliminate redundancy, ensure data integrity, and establish relationships between related entities. This is achieved by breaking down tables, ensuring atomic values, and establishing primary and foreign key relationships between tables.

4 Functional Analysis

4a Functional components of the application:

Data preparation

Data preprocessing and preparation are essential steps in database design to ensure that the data is well-organized, consistent, and optimized for efficient storage and retrieval. In the case of the TMDB dataset, which contains information about movies and related entities, such as collections, genres, production companies, and spoken languages, several preprocessing steps are necessary. Initially, the data is read from CSV files, which contains discrepancies and redundancies. To address this, the data is transformed into a structured format, adhering to the principles of normalization. This involves breaking down large tables into smaller, related tables and establishing relationships between them.

For example, the one big table of movie metadata is split into separate tables, such as movie metadata, belongs_to_collection, genres, production companies, production countries, and spoken languages. Each table is designed to have atomic values and to eliminate transitive dependencies, ensuring that non-key attributes are fully functionally dependent on the primary key. Once the data is in a normalized and structured format, it is connected to a local database, such as MySQL, and populated into the appropriate tables. This preprocessing and preparation process ensures that the data is well-organized, consistent, and ready for efficient storage and retrieval in the database system. All this is accomplished using Python and the corresponding code can be found here.

Before we dive into discussing the functional DB components of the application, let us discuss the access distribution.

One section of the dataset has a variety of attributes centered around the movie_metadata table. The other section of the database consists of user and user related data including ratings and search / view history. Access control is achieved by many ways, one of which is by maintaining different users with different access privileges at the MySQL level.

- **Unregistered users** have access solely to the *movie_details* view, to search for movies based on specific criteria. A separate MySQL user ***viewro*** is created with appropriate read-only access to the *movie_details* view.
- **Registered users** are free to rate movies and have access to recommendations based on their usage history. Queries from registered users are executed from the MySQL user named ***ltdrw*** with limited read-write privileges. This user has read-only access to all the tables in tmdb and read-write access to *ratings, Users, ViewHistory* and *SearchHistory*
- **Administrators** have RW access to all tables and they access the tmdb database via the user ***admin***

The application can be divided into 4 main functional components:

1. **Search and Retrieval:** This component allows users to search for movies based on various criteria. It includes functionalities for search, filtering, and displaying movie details.
2. **Recommendation system based on feedback mechanism:** Allowing for analysis of search patterns, the logs can be used to personalize user experiences by recommending similar or trending movies, also utilizing the user ratings and recommendations.
3. **Admin Ops:** Administrators have access to manage movie data, user ratings, and system settings. They can add, edit, or remove movies, genres, languages, and other metadata.
4. **Logging and Monitoring:** The system logs user interactions, database queries, and system events for monitoring, analysis, and troubleshooting purposes.

4b Database interactions for each Functional components:

1. Search and Retrieval

This component allows users to search for movies based on various criteria. It includes functionalities for search, filtering, and displaying movie details.
This can be split into two sections:

a. *Open to all - Provides access only to* ***movie_details*** *view*
b. *For registered users - Provides restricted access to all the tables*

Table structure:

Users with the **open to all** view have permission only to the movie_details VIEW containing most of the information from movie_metadata.

The **registered user** view would provide querying access to all the tables.

For the full table structure, please refer to section 3b.

Access privileges:

As stated above, we have 3 users at MySQL level - *viewro, ltdrw* and *admin*.

- Users with the **open to all** view would be able to query the database through the ***viewro*** user (MySQL) that has access only to movie_details VIEW.
- The **registered users** with MySQL user **ltdrw** would have read-access to all the tables including write access to a subset of them viz., ratings, Users, ViewHistory and SearchHistory.
- **Administrators** with MySQL user **admin** are the maintainers of the TMDB database, they have write access to all the tables in the schema.

Please refer to the grant access queries here .

SQL code:

1. Find all comedy movies.
2. Get the cast and crew of Ariel movie.
3. Fetch the movie production company and country.
4. Fetch the collection that has more than 10 movies.
5. Top 10 movies based on revenue generated
6. Number of Movies in each genre
7. Top 5 movies based on Popularity score in each genre
8. Top 15 Keywords
9. Weighted Rating based on Number of Votes and Vote Average

Stored Procedures:

1. MovieSearch()
2. CalculateMovieStatistics()
3. CountOfMovies_InYear()
4. display_top_collaborations()
5. filter_movie()

Logging Mechanism:

Logging mechanisms for Administrators and to maintain Search History of registered users are described at the end of this section.

2. Recommendation system based on feedback mechanism:

Allowing for analysis of search patterns, the logs can be used to personalize user experiences by recommending similar or trending movies, also utilizing the user ratings and recommendations.

Table structure:

The tables - *ratings, ViewHistory* and *SearchHistory* serve to provide insights into user behavior on the platform, allowing for analysis of search patterns and the popularity of viewed movies. They can be used to personalize user experiences by recommending similar or trending movies based on search and viewing history.

Access privileges:

*Registered users (MySQL user - **ltdrw**) can rate movies and have access to their search / view history. This requires RW access to user-related tables. The users have to be registered to enjoy maximum benefits of the application and to receive movie recommendations.*

SQL code:

1. Get the average rating of movies and order by highly rated movies.

Triggers:

1. update_average_rating_trigger
2. update_average_rating_log
3. Prevent_negative_rating

Stored Procedures:

1. GenerateMovieRecommendations()
2. GetTopRatedMovies( )

3. Admin Ops -

Administrators have access to manage movie data, user ratings, and system settings. They can add, edit, or remove movies, genres, languages, and other metadata.

Table structure:

All the tables represented in the ER diagram are accessible to admin users. They are the maintainers of the TMDB database who are the only ones to have insert / update / delete permissions to all the Movie tables.

Access privileges:

*All other tables including movie_metada are RW-accessible only to administrators (MySQL user - **admin**).*

SQL code:

1. Introduce new collections
2. Update status after a movie release

Triggers:

1. cascade_delete_movie_records
2. enforce_budget_constraint
3. enforce_unique_title_year
4. check_genre_limit

Stored Procedures:

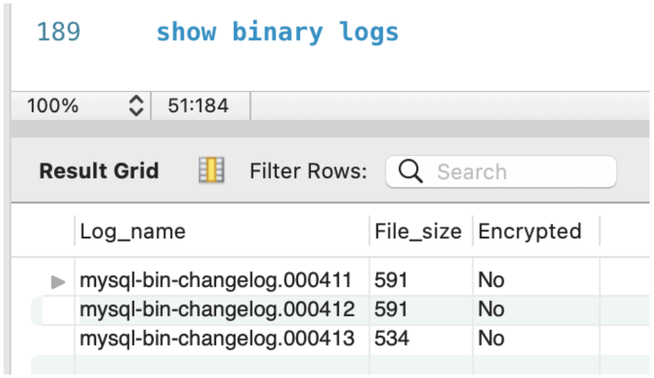
1. check_if_belongs_to_collection()

4. Logging and Monitoring -

The system logs user interactions, database queries, and system events for monitoring, analysis, and troubleshooting purposes.

In-built:

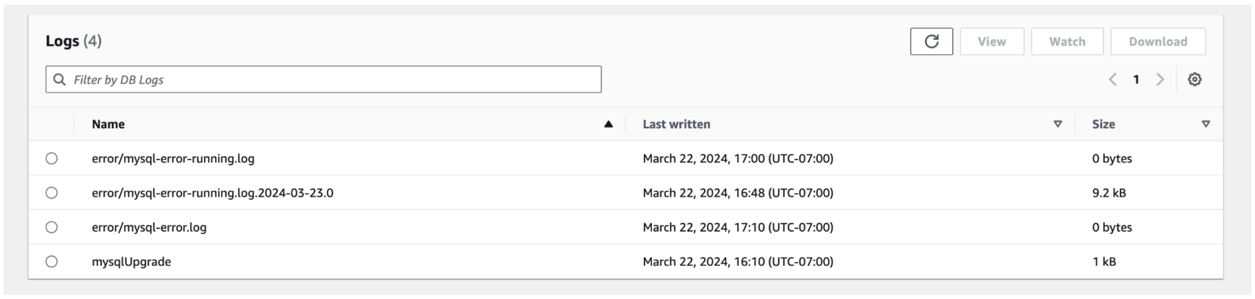
MySQL has in-built logging mechanisms for various purposes, viz., Error Logs, General Query Logs, Binary Logs and Slow Query Logs. To get a log of all queries that manipulated the data, we use the binary logs. These logs are also used to backup and recover the database.



```
show binary logs
```

Log_name	File_size	Encrypted
mysql-bin-changelog.000411	591	No
mysql-bin-changelog.000412	591	No
mysql-bin-changelog.000413	534	No

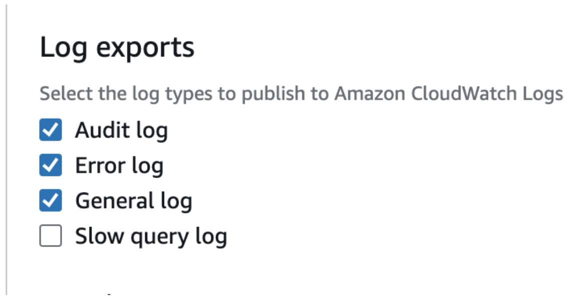
In RDS, there is a Logs and Events tab in which we will be able to access recent events on the DB server and the error logs as well.



Logs (4)

Name	Last written	Size
error/mysql-error-running.log	March 22, 2024, 17:00 (UTC-07:00)	0 bytes
error/mysql-error-running.log.2024-03-23.0	March 22, 2024, 16:48 (UTC-07:00)	9.2 kB
error/mysql-error.log	March 22, 2024, 17:10 (UTC-07:00)	0 bytes
mysqlUpgrade	March 22, 2024, 16:10 (UTC-07:00)	1 kB

We can also use Amazon CloudWatch Logs to monitor, store, and access your log files from RDS sources. CloudWatch Logs enables us to centralize the logs in a single, highly scalable service.



Log exports

Select the log types to publish to Amazon CloudWatch Logs

- [✓] Audit log
- [✓] Error log
- [✓] General log
- [ ] Slow query log

User-defined:

We can also establish a user-defined logging mechanism like how we implemented search and view history. This is more customized as per the requirement, user-friendly and also easily accessible. The "SearchHistory" table maintains the log of search queries, while the "ViewHistory" table records viewed movies corresponding to users. Both tables serve to provide insights into user behavior on the platform, allowing for analysis of search patterns and the popularity of viewed movies, thereby contributing to the recommendation system.

Table structure:

SearchHistory *and* ***ViewHistory*** *maintain a log of movies accessed by registered users. Users and movie_metadata tables also play a part in maintaining the history.*

Access privileges:

Open to registered users with ***ltdrw*** *permission. Only registered users have access to their Search and View history.*

SQL code:

1. Retrieve all search history records for a specific user
2. Retrieve all view history records for a specific user
3. Delete all search history records older than 6 months
4. Retrieve the top 10 users with the highest number of view history records

Triggers:

1. log_popularity_update
2. rating_deletion_audit

Stored Procedures:

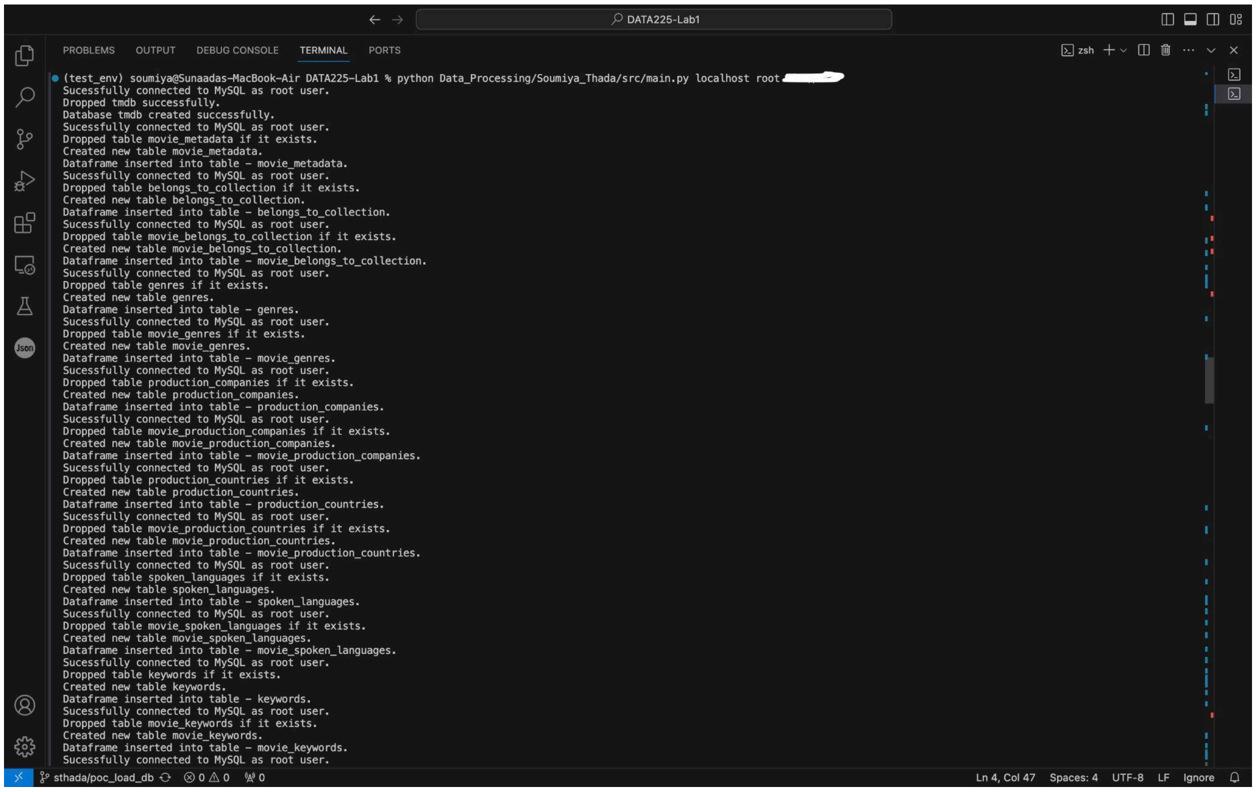
1. LogSearchHistory()
2. LogViewHistory()

5 Database Migration to AWS

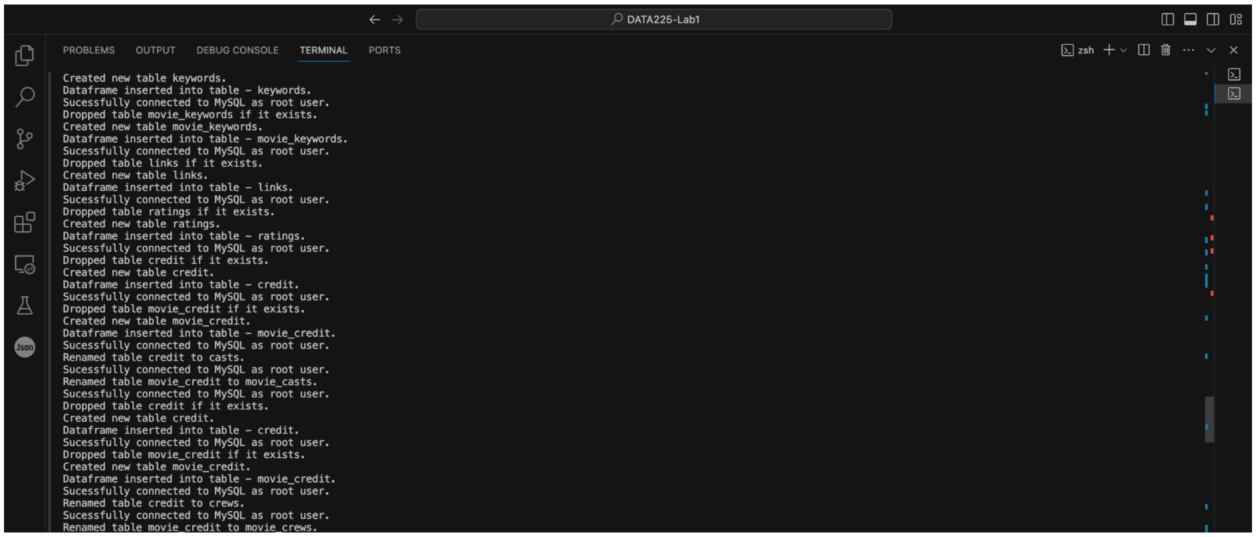
5a Step-by-Step guide:

Source Database Setup:
The dataset imported from multiple csv files was thoroughly examined to identify and rectify any inconsistencies, inaccuracies, or missing values, ensuring data cleanliness and integrity using python scripts.
Subsequently, the dataset was grouped into multiple data frames, each representing a distinct entity or aspect of the data as described in the ER diagram of TMDB. This partitioning facilitated the adherence to the principles of normalization, ensuring that data was organized efficiently and redundancies were minimized. Furthermore, by applying referential integrity principles the relationships between entities were accurately represented and maintained.
Finally, a database was created in the local instance of MySQL and all the preprocessed data was inserted into the tables. The data preprocessing scripts developed in python ensured that the data was structured optimally for seamless integration into MySQL tables, fostering a robust and reliable database foundation for the migration process.

Code Execution Results:



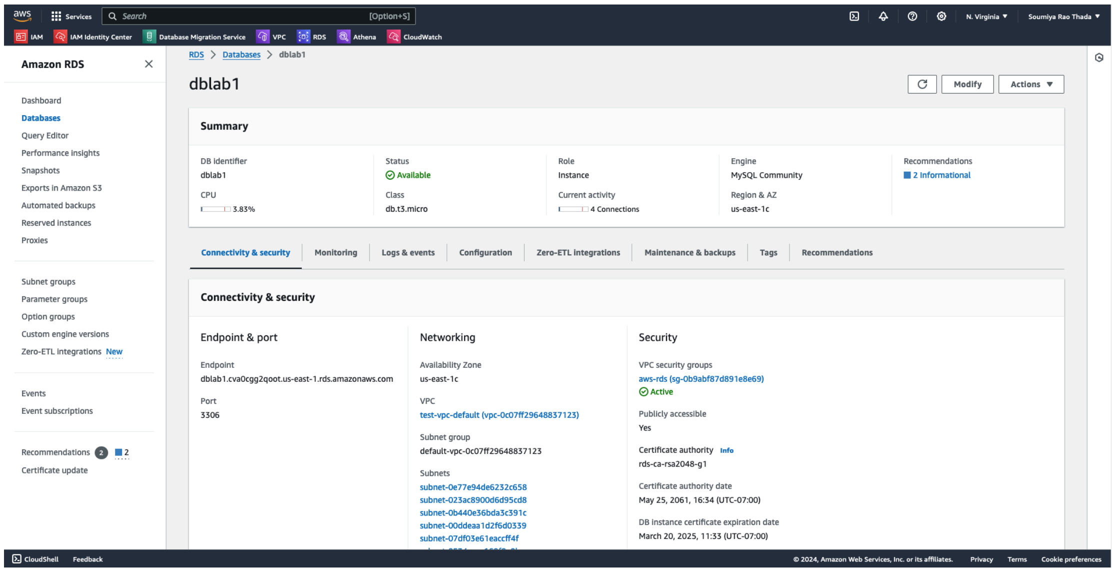
```
(test_env) soumiya@Sunaadas-MacBook-Air DATA225-Lab1 % python Data_Processing/Soumiya_Thada/src/main.py localhost root [redacted]
Sucessfully connected to MySQL as root user.
Dropped tmdb successfully.
Database tmdb created successfully.
Sucessfully connected to MySQL as root user.
Dropped table movie_metadata if it exists.
Created new table movie_metadata.
Dataframe inserted into table - movie_metadata.
Sucessfully connected to MySQL as root user.
Dropped table belongs_to_collection if it exists.
Created new table belongs_to_collection.
Dataframe inserted into table - belongs_to_collection.
Sucessfully connected to MySQL as root user.
Dropped table movie_belongs_to_collection if it exists.
Created new table movie_belongs_to_collection.
Dataframe inserted into table - movie_belongs_to_collection.
Sucessfully connected to MySQL as root user.
Dropped table genres if it exists.
Created new table genres.
Dataframe inserted into table - genres.
Sucessfully connected to MySQL as root user.
Dropped table movie_genres if it exists.
Created new table movie_genres.
Dataframe inserted into table - movie_genres.
Sucessfully connected to MySQL as root user.
Dropped table production_companies if it exists.
Created new table production_companies.
Dataframe inserted into table - production_companies.
Sucessfully connected to MySQL as root user.
Dropped table movie_production_companies if it exists.
Created new table movie_production_companies.
Dataframe inserted into table - movie_production_companies.
Sucessfully connected to MySQL as root user.
Dropped table production_countries if it exists.
Created new table production_countries.
Dataframe inserted into table - production_countries.
Sucessfully connected to MySQL as root user.
Dropped table movie_production_countries if it exists.
Created new table movie_production_countries.
Dataframe inserted into table - movie_production_countries.
Sucessfully connected to MySQL as root user.
Dropped table spoken_languages if it exists.
Created new table spoken_languages.
Dataframe inserted into table - spoken_languages.
Sucessfully connected to MySQL as root user.
Dropped table movie_spoken_languages if it exists.
Created new table movie_spoken_languages.
Dataframe inserted into table - movie_spoken_languages.
Sucessfully connected to MySQL as root user.
Dropped table keywords if it exists.
Created new table keywords.
Dataframe inserted into table - keywords.
Sucessfully connected to MySQL as root user.
Dropped table movie_keywords if it exists.
Created new table movie_keywords.
Dataframe inserted into table - movie_keywords.
Sucessfully connected to MySQL as root user.
```



```
Created new table keywords.
Dataframe inserted into table - keywords.
Sucessfully connected to MySQL as root user.
Dropped table movie_keywords if it exists.
Created new table movie_keywords.
Dataframe inserted into table - movie_keywords.
Sucessfully connected to MySQL as root user.
Dropped table links if it exists.
Created new table links.
Dataframe inserted into table - links.
Sucessfully connected to MySQL as root user.
Dropped table ratings if it exists.
Created new table ratings.
Dataframe inserted into table - ratings.
Sucessfully connected to MySQL as root user.
Dropped table credit if it exists.
Created new table credit.
Dataframe inserted into table - credit.
Sucessfully connected to MySQL as root user.
Dropped table movie_credit if it exists.
Created new table movie_credit.
Dataframe inserted into table - movie_credit.
Sucessfully connected to MySQL as root user.
Renamed table credit to casts.
Sucessfully connected to MySQL as root user.
Renamed table movie_credit to movie_casts.
Sucessfully connected to MySQL as root user.
Dropped table credit if it exists.
Created new table credit.
Dataframe inserted into table - credit.
Sucessfully connected to MySQL as root user.
Dropped table movie_credit if it exists.
Created new table movie_credit.
Dataframe inserted into table - movie_credit.
Sucessfully connected to MySQL as root user.
Renamed table credit to crews.
Sucessfully connected to MySQL as root user.
Renamed table movie_credit to movie_crews.
```

Target Database Setup:

A Relational Database Service (RDS) instance was created to facilitate the storage and management of data. A new user was created in the AWS Management Console with access to RDS service. The necessary parameters for the RDS instance, including the database engine type, instance class, storage type, and allocated storage, were configured based on project requirements. Security settings such as network access permissions, encryption options, and database credentials were established to ensure data security and integrity. Once all configurations were finalized, the RDS instance was provisioned, and the database engine was deployed. Create an empty TMDB database in the new AWS RDS instance.



dblab1

Summary

DB identifier: dblab1
CPU: 3.83%
Status: Available
Class: db.t3.micro
Role: Instance
Current activity: 4 Connections
Engine: MySQL Community
Region & AZ: us-east-1c
Recommendations: 2 Informational

Connectivity & security

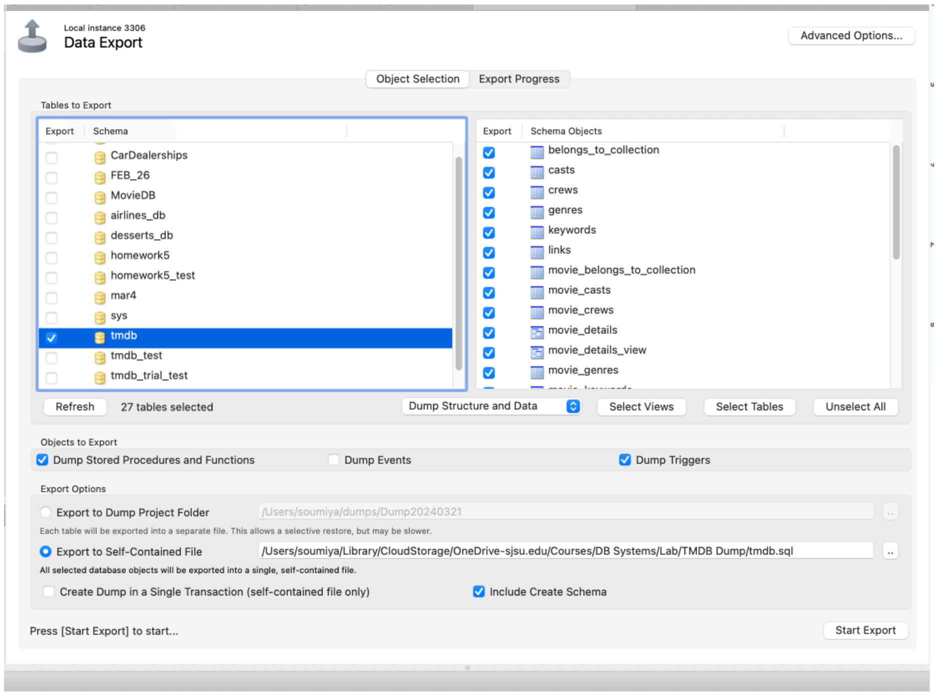
Endpoint & port
Endpoint: dblab1.cva0cgg2qoot.us-east-1.rds.amazonaws.com
Port: 3306

Networking
Availability Zone: us-east-1c
VPC: test-vpc-default (vpc-0c07ff29648837123)
Subnet group: default-vpc-0c07ff29648837123
Subnets:
subnet-0e77e94de6232c658
subnet-023ac8900d6d95cd8
subnet-0b440e36bda3c391c
subnet-00ddeaa1d2f6d0339
subnet-07df03e61eaccff4f

Security
VPC security groups: aws-rds (sg-0b9abf87d891e8e69) Active
Publicly accessible: Yes
Certificate authority: rds-ca-rsa2048-g1
Certificate authority date: May 25, 2061, 16:34 (UTC-07:00)
DB instance certificate expiration date: March 20, 2025, 11:33 (UTC-07:00)

Migration Process Execution:

From the MySQL workbench local instance, export the TMDB Database to a self contained .sql file using Server->Data Export option. While exporting the database select all the tables and views. Also, select the options to dump stored procedures, dump triggers and include create schema.



Local instance 3306
Data Export

Tables to Export

Export	Schema
☐	CarDealerships
☐	FEB_26
☐	MovieDB
☐	airlines_db
☐	desserts_db
☐	homework5
☐	homework5_test
☐	mar4
☐	sys
✓	tmdb
☐	tmdb_test
☐	tmdb_trial_test

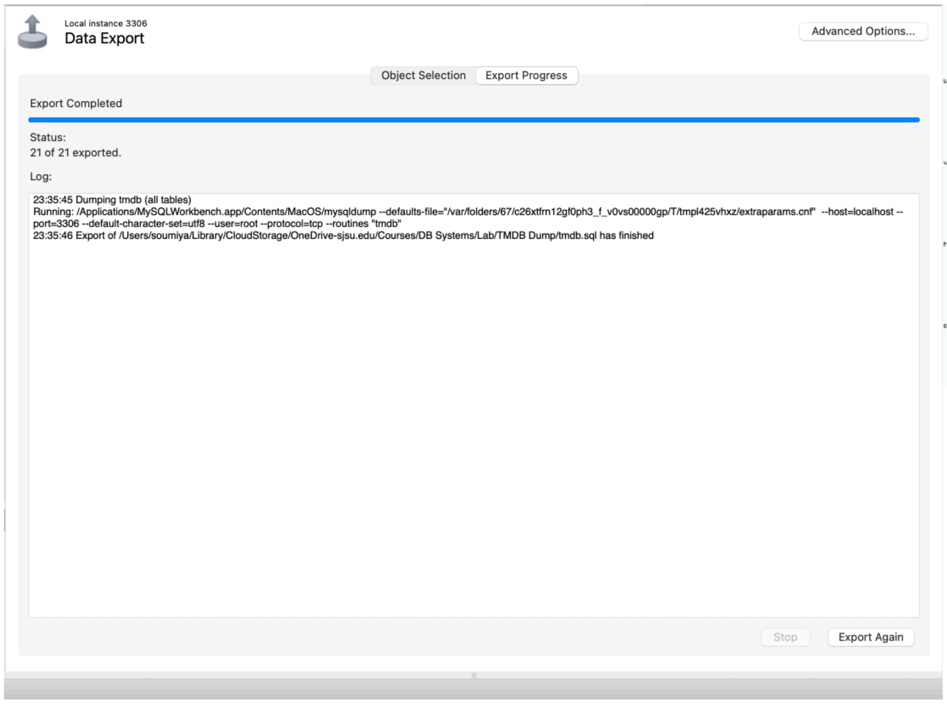
Export	Schema Objects
✓	belongs_to_collection
✓	casts
✓	crews
✓	genres
✓	keywords
✓	links
✓	movie_belongs_to_collection
✓	movie_casts
✓	movie_crews
✓	movie_details
✓	movie_details_view
✓	movie_genres

27 tables selected
Dump Structure and Data

Objects to Export
[✓] Dump Stored Procedures and Functions
[ ] Dump Events
[✓] Dump Triggers

Export Options
( ) Export to Dump Project Folder /Users/soumiya/dumps/Dump20240321
Each table will be exported into a separate file. This allows a selective restore, but may be slower.
(●) Export to Self-Contained File /Users/soumiya/Library/CloudStorage/OneDrive-sjsu.edu/Courses/DB Systems/Lab/TMDB Dump/tmdb.sql
All selected database objects will be exported into a single, self-contained file.
[ ] Create Dump in a Single Transaction (self-contained file only)
[✓] Include Create Schema

Press [Start Export] to start...



Local instance 3306
Data Export

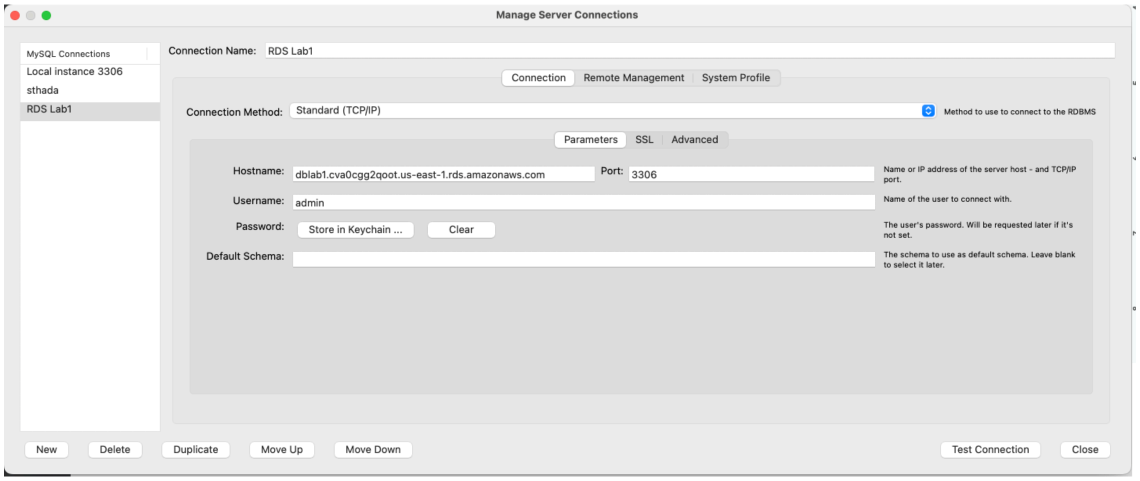
Export Completed

Status:
21 of 21 exported.

Log:

```
23:35:45 Dumping tmdb (all tables)
Running: /Applications/MySQLWorkbench.app/Contents/MacOS/mysqldump --defaults-file="/var/folders/67/c26xtfrn12gf0ph3_f_v0vs00000gp/T/tmpl425vhxz/extraparams.cnf"  --host=localhost --port=3306 --default-character-set=utf8 --user=root --protocol=tcp --routines "tmdb"
23:35:46 Export of /Users/soumiya/Library/CloudStorage/OneDrive-sjsu.edu/Courses/DB Systems/Lab/TMDB Dump/tmdb.sql has finished
```

From MySQL workbench connect to the AWS RDS instance created in the target db setup using the hostname, port, user and password. Verify the connection using the Test Connection option.



Connection Name: RDS Lab1

Connection Method: Standard (TCP/IP) — Method to use to connect to the RDBMS

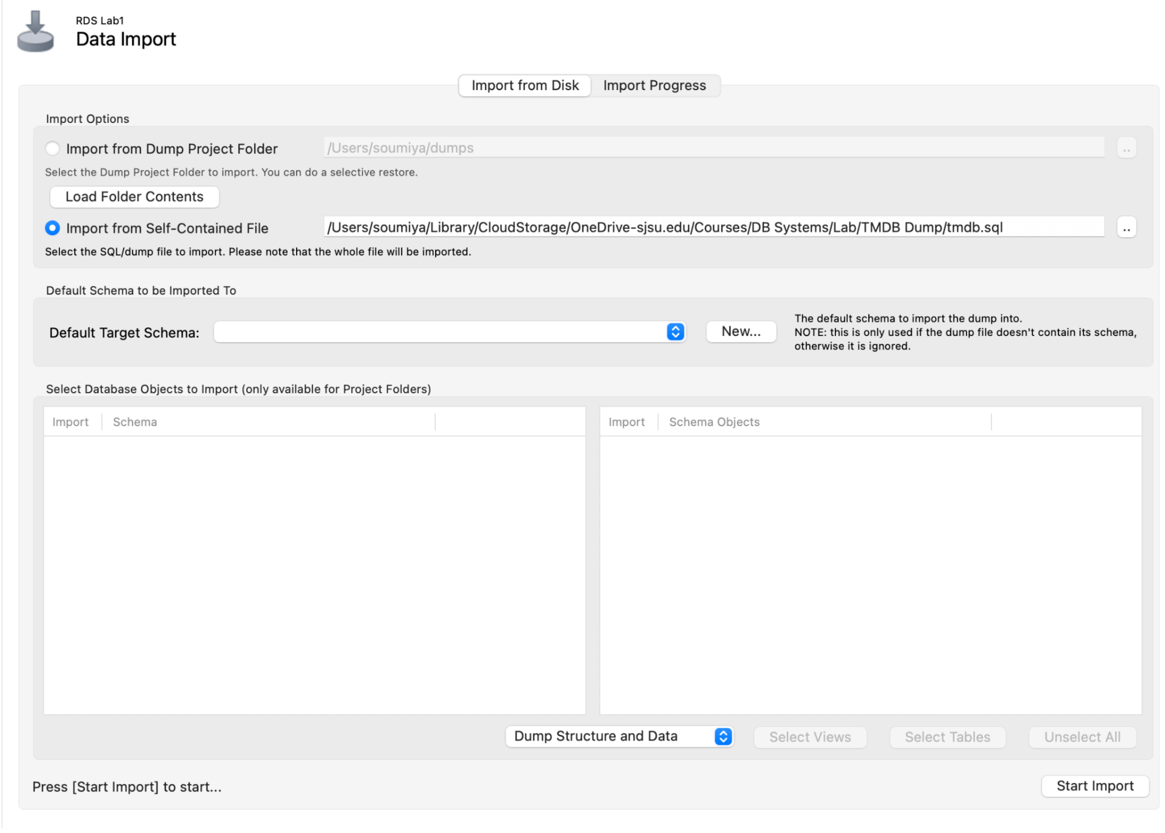
Hostname: dblab1.cva0cgg2qoot.us-east-1.rds.amazonaws.com Port: 3306 — Name or IP address of the server host - and TCP/IP port.

Username: admin — Name of the user to connect with.

Password: — The user's password. Will be requested later if it's not set.

Default Schema: — The schema to use as default schema. Leave blank to select it later.

Using Server->Import option in MySQL RDS instance, import the dump file exported in the previous step.



Import Options

Import from Dump Project Folder /Users/soumiya/dumps

Select the Dump Project Folder to import. You can do a selective restore.

Import from Self-Contained File /Users/soumiya/Library/CloudStorage/OneDrive-sjsu.edu/Courses/DB Systems/Lab/TMDB Dump/tmdb.sql

Select the SQL/dump file to import. Please note that the whole file will be imported.

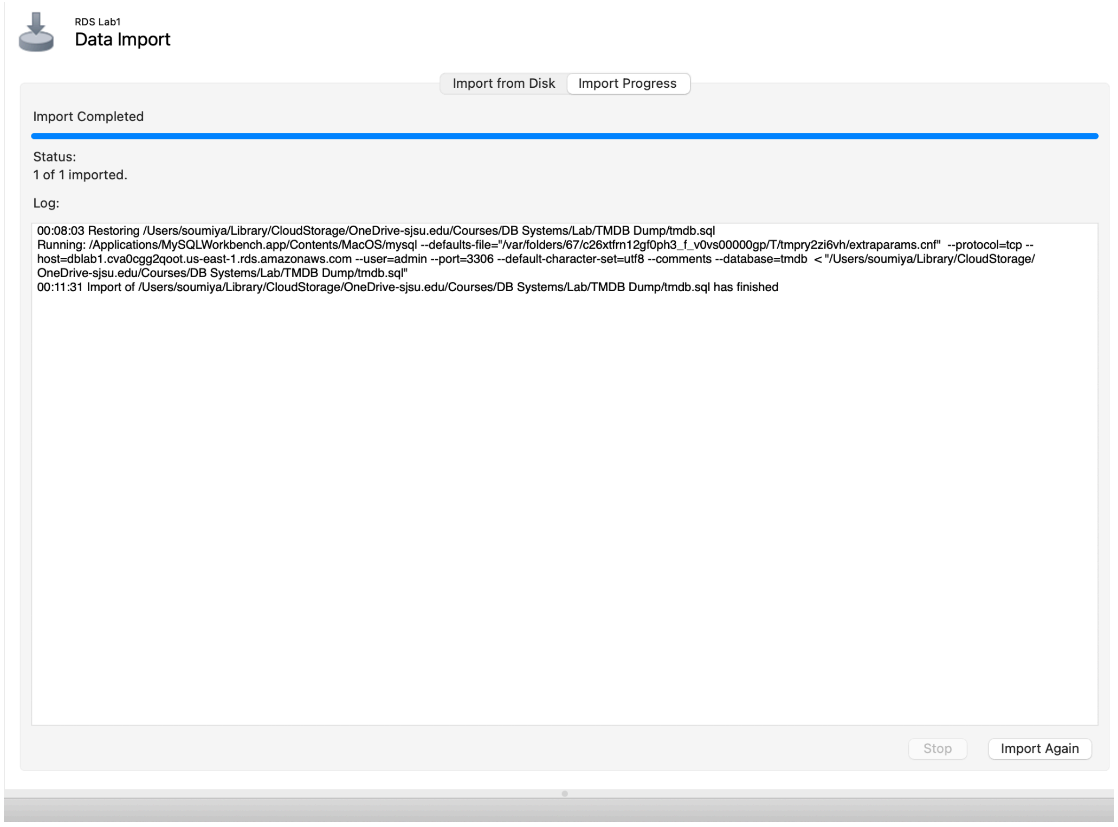
Default Schema to be Imported To

Default Target Schema: — The default schema to import the dump into. NOTE: this is only used if the dump file doesn't contain its schema, otherwise it is ignored.

Select Database Objects to Import (only available for Project Folders)

Dump Structure and Data

Press [Start Import] to start...


RDS Lab1

Data Import

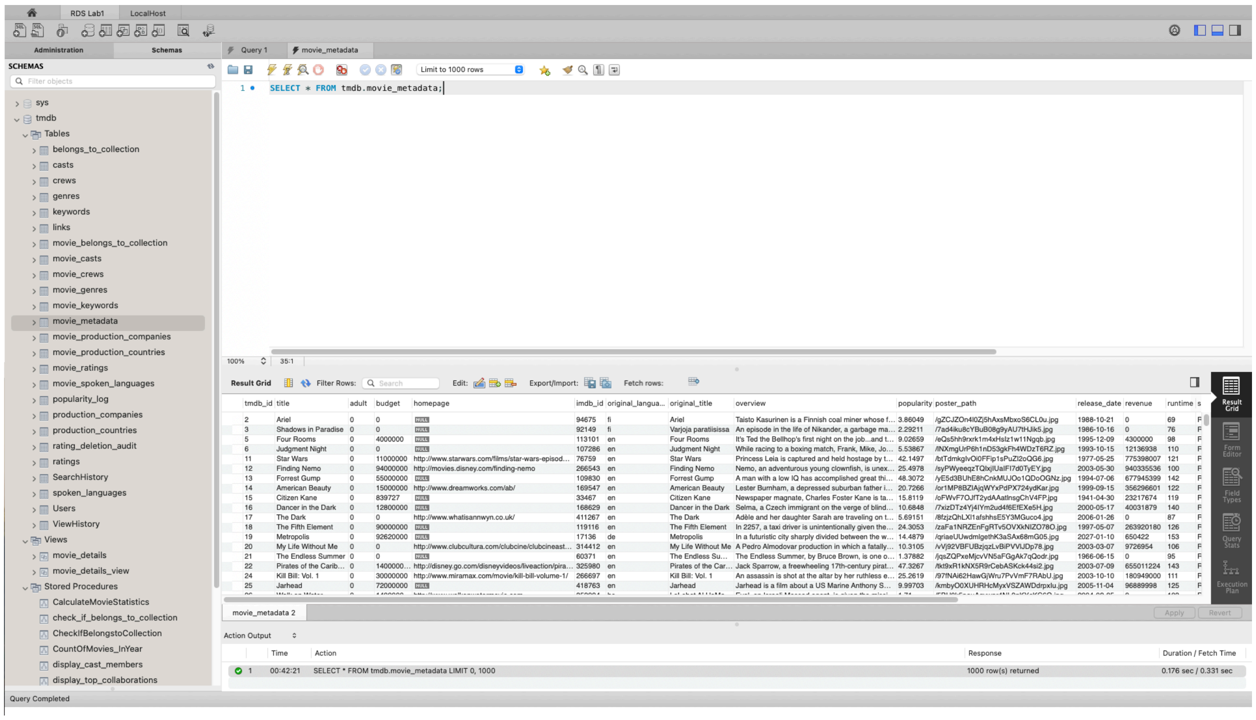
Import Completed

Status:
1 of 1 imported.

Log:

```
00:08:03 Restoring /Users/soumiya/Library/CloudStorage/OneDrive-sjsu.edu/Courses/DB Systems/Lab/TMDB Dump/tmdb.sql
Running: /Applications/MySQLWorkbench.app/Contents/MacOS/mysql --defaults-file="/var/folders/67/c26xtfrn12gf0ph3_f_v0vs00000gp/T/tmpry2zi6vh/extraparams.cnf"  --protocol=tcp --host=dblab1.cva0cgg2qoot.us-east-1.rds.amazonaws.com --user=admin --port=3306 --default-character-set=utf8 --comments --database=tmdb  < "/Users/soumiya/Library/CloudStorage/OneDrive-sjsu.edu/Courses/DB Systems/Lab/TMDB Dump/tmdb.sql"
00:11:31 Import of /Users/soumiya/Library/CloudStorage/OneDrive-sjsu.edu/Courses/DB Systems/Lab/TMDB Dump/tmdb.sql has finished
```

After importing the dump file, refresh the schemas tab and verify that all the tables, stored procedures, triggers and views are available under the TMDB database.



- sys
- tmdb
  - Tables
    - belongs_to_collection
    - casts
    - crews
    - genres
    - keywords
    - links
    - movie_belongs_to_collection
    - movie_casts
    - movie_crews
    - movie_genres
    - movie_keywords
    - movie_metadata
    - movie_production_companies
    - movie_production_countries
    - movie_ratings
    - movie_spoken_languages
    - popularity_log
    - production_companies
    - production_countries
    - rating_deletion_audit
    - ratings
    - SearchHistory
    - spoken_languages
    - Users
    - ViewHistory
  - Views
    - movie_details
    - movie_details_view
  - Stored Procedures
    - CalculateMovieStatistics
    - check_if_belongs_to_collection
    - CheckIfBelongstoCollection
    - CountOfMovies_InYear
    - display_cast_members
    - display_top_collaborations

```
SELECT * FROM tmdb.movie_metadata;
```

100% 35:1

tmdb_id	title	adult	budget	homepage	imdb_id	original_langua...	original_title	overview	popularity	poster_path	release_date	revenue	runtime	s
2	Ariel	0	0	NULL	94675	fi	Ariel	Taisto Kasurinen is a Finnish coal miner whose f...	3.86049	/gZCJZOn4l0Zj5hAxsMbxoS6CL0u.jpg	1988-10-21	0	69	F
3	Shadows in Paradise	0	0	NULL	92149	fi	Varjoja paratiisissa	An episode in the life of Nikander, a garbage ma...	2.29211	/7ad4iku8cYBuB08g9yAU7tHJik5.jpg	1986-10-16	0	76	F
5	Four Rooms	0	4000000	NULL	113101	en	Four Rooms	It's Ted the Bellhop's first night on the job...and t...	9.02659	/eQs5hh9rxrk1m4xHslz1w11Ngqb.jpg	1995-12-09	4300000	98	F
6	Judgment Night	0	0	NULL	107286	en	Judgment Night	While racing to a boxing match, Frank, Mike, Jo...	5.53867	/lNXmgUrP6h1nD53gkFh4WDzT6RZ.jpg	1993-10-15	12136938	110	F
11	Star Wars	0	11000000	http://www.starwars.com/films/star-wars-episod...	76759	en	Star Wars	Princess Leia is captured and held hostage by t...	42.1497	/btTdmkgIvOi0FFip1sPuZI2oQG6.jpg	1977-05-25	775398007	121	F
12	Finding Nemo	0	94000000	http://movies.disney.com/finding-nemo	266543	en	Finding Nemo	Nemo, an adventurous young clownfish, is unex...	25.4978	/syPWyeeqzTQlxjIUalFI7d0TyEY.jpg	2003-05-30	940335536	100	F
13	Forrest Gump	0	55000000	NULL	109830	en	Forrest Gump	A man with a low IQ has accomplished great thi...	48.3072	/yE5d3BUhE8hCnkMUJOo1QDoOGNz.jpg	1994-07-06	677945399	142	F
14	American Beauty	0	15000000	http://www.dreamworks.com/ab/	169547	en	American Beauty	Lester Burnham, a depressed suburban father i...	20.7266	/or1MP8BZIAjqWYxPdPX724ydKar.jpg	1999-09-15	356296601	122	F
15	Citizen Kane	0	839727	NULL	33467	en	Citizen Kane	Newspaper magnate, Charles Foster Kane is ta...	15.8119	/oFWvF7OJfT2ydAAatlnsgChV4FP.jpg	1941-04-30	23217674	119	F
16	Dancer in the Dark	0	12800000	NULL	168629	en	Dancer in the Dark	Selma, a Czech immigrant on the verge of blind...	10.6848	/7xizDTz4Yj4IYm2ud4f6EfEXe5H.jpg	2000-05-17	40031879	140	F
17	The Dark	0	0	http://www.whatisannwyn.co.uk/	411267	en	The Dark	Adèle and her daughter Sarah are traveling on t...	5.69151	/8fzjzQhLXl1afshhsE5Y3MGuco4.jpg	2006-01-26	0	87	F
18	The Fifth Element	0	90000000	NULL	119116	en	The Fifth Element	In 2257, a taxi driver is unintentionally given the...	24.3053	/zaFa1NRZEnFgRTv5OVXkNIZO78O.jpg	1997-05-07	263920180	126	F
19	Metropolis	0	92620000	NULL	17136	de	Metropolis	In a futuristic city sharply divided between the w...	14.4879	/qriaeUUwdmlgethK3aSAx68mG05.jpg	2027-01-10	650422	153	F
20	My Life Without Me	0	0	http://www.clubcultura.com/clubcine/clubcineast...	314412	en	My Life Without Me	A Pedro Almodovar production in which a fatally...	10.3105	/vVj92VBFUBzjqzLvBiPVVlJDp78.jpg	2003-03-07	9726954	106	F
21	The Endless Summer	0	0	NULL	60371	en	The Endless Su...	The Endless Summer, by Bruce Brown, is one o...	1.37882	/jqsZQPxeMjcvVN5aFGgAk7qQodr.jpg	1966-06-15	0	95	F
22	Pirates of the Carib...	0	1400000...	http://disney.go.com/disneyvideos/liveaction/pira...	325980	en	Pirates of the Car...	Jack Sparrow, a freewheeling 17th-century pirat...	47.3267	/tkt9xR1kNX5R9rCebASKck44si2.jpg	2003-07-09	655011224	143	F
24	Kill Bill: Vol. 1	0	30000000	http://www.miramax.com/movie/kill-bill-volume-1/	266697	en	Kill Bill: Vol. 1	An assassin is shot at the altar by her ruthless e...	25.2619	/97fNAi62HawGjWru7PvVmF7RAbU.jpg	2003-10-10	180949000	111	F
25	Jarhead	0	72000000	NULL	418763	en	Jarhead	Jarhead is a film about a US Marine Anthony S...	9.99703	/kmbyO0XUHRHcMyxVSZAWDdrpxlu.jpg	2005-11-04	96889998	125	F

movie_metadata 2

Action Output

	Time	Action	Response	Duration / Fetch Time
1	00:42:21	SELECT * FROM tmdb.movie_metadata LIMIT 0, 1000	1000 row(s) returned	0.176 sec / 0.331 sec

5b Connectivity to AWS using Python:

Python MySQL Connector module was used to establish connectivity with AWS RDS instance by following these steps:

1. **Install MySQL Connector using pip:**

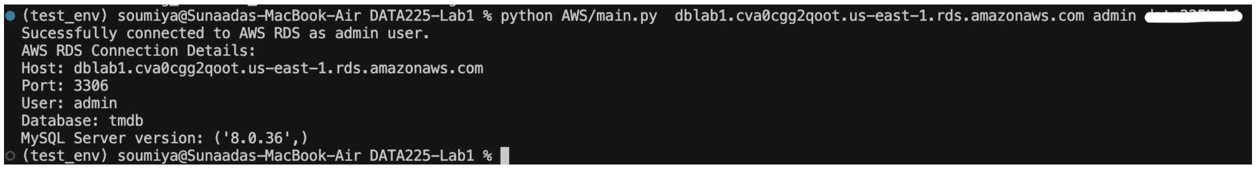
```
pip install mysql-connector-python
```

2. **Get RDS Endpoint and Database Credentials:** Necessary information to connect to AWS RDS instance like endpoint (hostname), port number, database username, and password were gathered from AWS RDS Dashboard.
3. **Connect to RDS Instance:** Using the MySQL Connector a connection was established with the RDS instance. Using the following code snippet:

```
import mysql.connector
from mysql.connector import Error

def get_db_connection(host, user, password, database=None):
        try:
                if (database == None):
                        connection = mysql.connector.connect(host=host, user=user, password=password)
                else:
                        connection = mysql.connector.connect(host=host, user=user, password=password,
database=database)
                print(f"Sucessfully connected to AWS RDS as {user} user.")
                print("AWS RDS Connection Details:")
                print(f"Host: {connection.server_host}")
                print(f"Port: {connection.server_port}")
                print(f"User: {connection.user}")
                print(f"Database: {connection.database}")
                print_executed = True
        except Error as e:
                print(f"Error while connecting to AWS RDS as {user} user.", e)
                quit()
        return connection
```

4. **Verify Connectivity:** To ensure connectivity with AWS RDS instance, details of the connection were printed on the console and a “SELECT VERSION();” query was executed using embedded sql in python



```
(test_env) soumiya@Sunaadas-MacBook-Air DATA225-Lab1 % python AWS/main.py  dblab1.cva0cgg2qoot.us-east-1.rds.amazonaws.com admin [redacted]
Sucessfully connected to AWS RDS as admin user.
AWS RDS Connection Details:
Host: dblab1.cva0cgg2qoot.us-east-1.rds.amazonaws.com
Port: 3306
User: admin
Database: tmdb
MySQL Server version: ('8.0.36',)
(test_env) soumiya@Sunaadas-MacBook-Air DATA225-Lab1 %
```

5c Migration Validation

A SELECT query was executed using embedded sql in python to validate migration. Also, all the stored procedures and triggers in the TMDB database migrated to the AWS RDS instance were listed to ensure that the migration was successful.

Functions to execute the queries and print the results:

```
def execute_query_with_params(host, user, password, database, query, param_values):
        try:
        cnx = get_db_connection(host,user,password,database)
        cursor = cnx.cursor()
        cursor.execute(query, param_values)
        rows = cursor.fetchall()
        column_names = [i[0] for i in cursor.description]
        df = pd.DataFrame(rows, columns=column_names)
        cursor.close()
        cnx.close()
        return df

        except mysql.connector.Error as err:
        print(f"Error: {err}")
        return None

def exexute_and_print_query_results(host,user,password, query, param_values, header):
        result_df = execute_query_with_params(host,user,password,constants.DATABASE_NAME, query,
param_values)

        if result_df is not None:
                print(header)
                print(result_df)
        else:
                print("Query execution failed.")
```

Sample query to fetch top 10 movies by revenue:

```
def fetch_top_movie_by_revenue(host,user,password):
        query = """SELECT
                original_title AS 'Movie Title',
                FORMAT(budget / 1000000, 2) AS 'Budget in Millions',
                FORMAT(revenue / 1000000, 2) AS 'Revenue in Millions',
                FORMAT((revenue - budget) / 1000000, 2) AS 'Profit in Millions'
        FROM
                movie_metadata
        ORDER BY revenue DESC
        LIMIT 10;
        """

        param_values = ()
        header = "Top 10 Movies based on revenue generated"
        exexute_and_print_query_results(host,user,password, query, param_values, header)
```

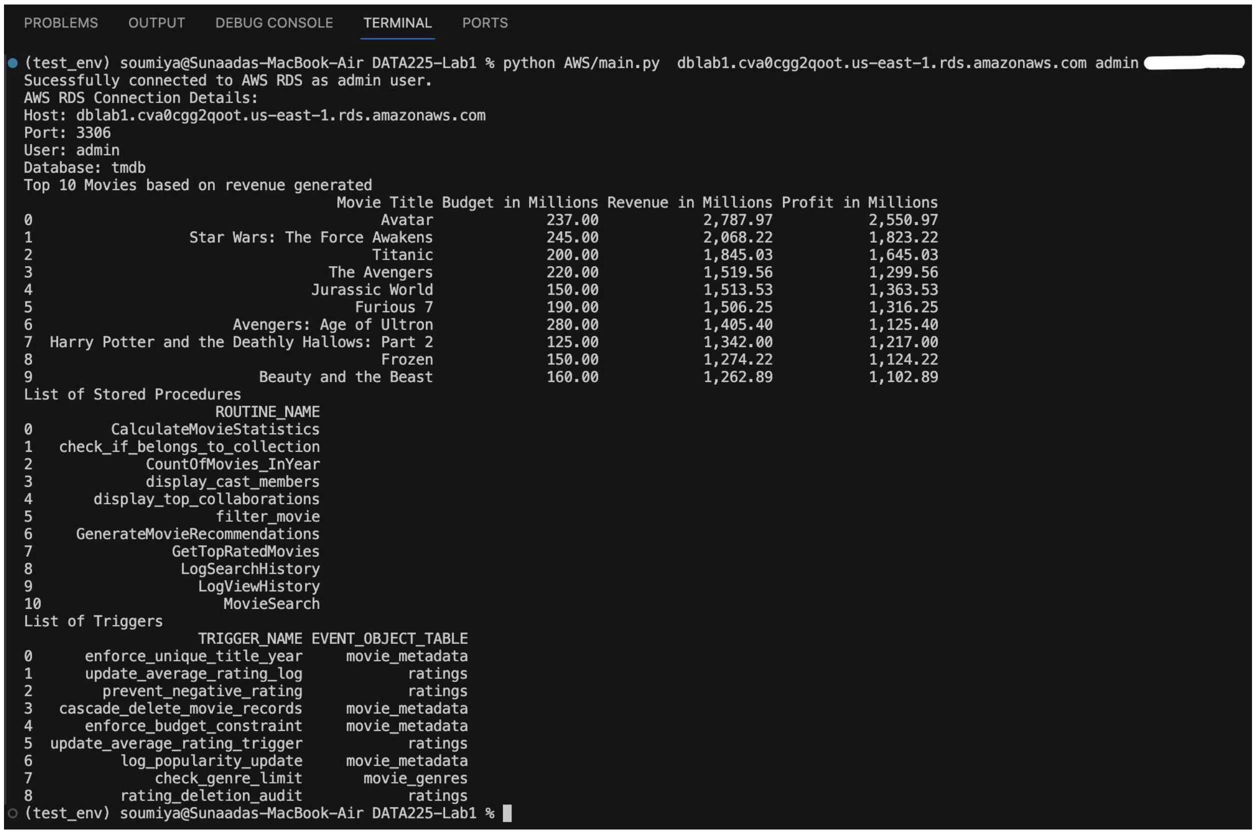
Function to list all the stored procedures in the database:

```
def list_stored_procedures(host,user,password):
        query="SELECT routine_name FROM information_schema.routines WHERE routine_type =
'PROCEDURE' AND routine_schema = %s"
        # Execute the query
        param_values = ((constants.DATABASE_NAME,))
        header = "List of Stored Procedures"
        exexute_and_print_query_results(host,user,password, query, param_values, header)
```

Function to list all the triggers and their associated tables in the database:

```
def list_triggers(host,user,password):
        query="SELECT trigger_name, event_object_table FROM information_schema.triggers WHERE
trigger_schema = %s"
        # Execute the query
        param_values = ((constants.DATABASE_NAME,))
        header = "List of Triggers"
        exexute_and_print_query_results(host,user,password, query, param_values, header)
```

Screenshot of the results:



```
(test_env) soumiya@Sunaadas-MacBook-Air DATA225-Lab1 % python AWS/main.py  dblab1.cva0cgg2qoot.us-east-1.rds.amazonaws.com admin [redacted]
Sucessfully connected to AWS RDS as admin user.
AWS RDS Connection Details:
Host: dblab1.cva0cgg2qoot.us-east-1.rds.amazonaws.com
Port: 3306
User: admin
Database: tmdb
Top 10 Movies based on revenue generated
                                    Movie Title Budget in Millions Revenue in Millions Profit in Millions
0                                        Avatar             237.00            2,787.97           2,550.97
1                  Star Wars: The Force Awakens             245.00            2,068.22           1,823.22
2                                       Titanic             200.00            1,845.03           1,645.03
3                                  The Avengers             220.00            1,519.56           1,299.56
4                                Jurassic World             150.00            1,513.53           1,363.53
5                                     Furious 7             190.00            1,506.25           1,316.25
6                       Avengers: Age of Ultron             280.00            1,405.40           1,125.40
7  Harry Potter and the Deathly Hallows: Part 2             125.00            1,342.00           1,217.00
8                                        Frozen             150.00            1,274.22           1,124.22
9                          Beauty and the Beast             160.00            1,262.89           1,102.89
List of Stored Procedures
                      ROUTINE_NAME
0         CalculateMovieStatistics
1   check_if_belongs_to_collection
2             CountOfMovies_InYear
3             display_cast_members
4       display_top_collaborations
5                     filter_movie
6     GenerateMovieRecommendations
7                GetTopRatedMovies
8                 LogSearchHistory
9                   LogViewHistory
10                     MovieSearch
List of Triggers
                    TRIGGER_NAME EVENT_OBJECT_TABLE
0      enforce_unique_title_year     movie_metadata
1      update_average_rating_log            ratings
2        prevent_negative_rating            ratings
3   cascade_delete_movie_records     movie_metadata
4      enforce_budget_constraint     movie_metadata
5  update_average_rating_trigger            ratings
6          log_popularity_update     movie_metadata
7              check_genre_limit       movie_genres
8          rating_deletion_audit            ratings
(test_env) soumiya@Sunaadas-MacBook-Air DATA225-Lab1 %
```

6 SQL Performance

Screenshot for the commands

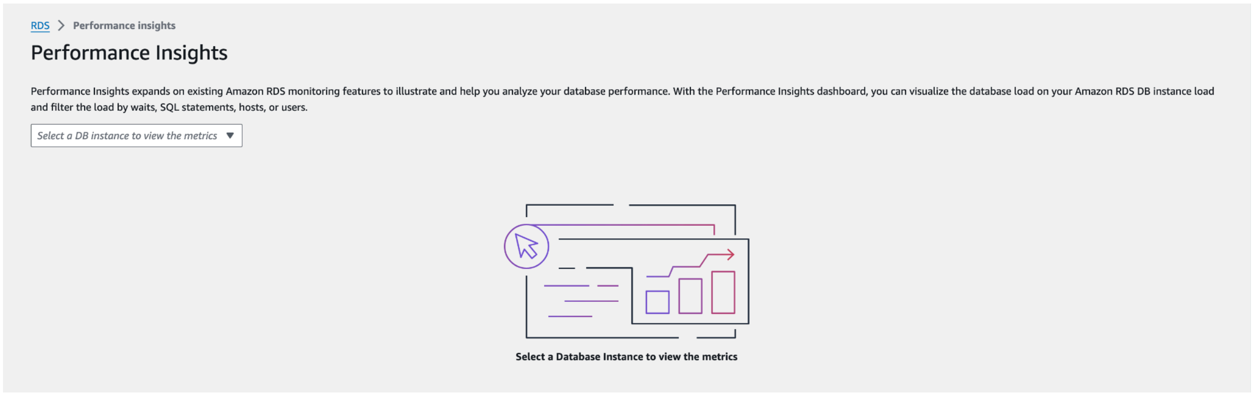
S.NO	Command	DML/DDL	Local Instance- Time Taken in seconds	AWS RDS- Time Taken in seconds
1	CREATE	DDL	0.031	0.156
2	DROP	DDL	0.031	0.125
3	INSERT	DML	0.000	0.078

4	SELECT	DML	0.047	0.109

By analyzing the time taken, we see that the performance in local is faster than RDS for both DDL and DML commands. RDS is a little slow compared to Local MySQL because of the Network Latency. But AWS RDS has other advantages like scalability, manageability, and reliability.

RDS Performance Insights

Performance Insights expands on existing Amazon RDS monitoring features to illustrate and help you analyze your database performance. With the Performance Insights dashboard, you can visualize the database load on your Amazon RDS DB instance load and filter the load by waits, SQL statements, hosts, or users. RDS Performance Insights is not available on the small and micro instance sizes but it's an efficient way to monitor and enhance performance of the application.


RDS > Performance insights
Select a DB instance to view the metrics
Select a Database Instance to view the metrics

Performance Insights

Performance Insights expands on existing Amazon RDS monitoring features to illustrate and help you analyze your database performance. With the Performance Insights dashboard, you can visualize the database load on your Amazon RDS DB instance load and filter the load by waits, SQL statements, hosts, or users.

7 Summary

In conclusion, the migration of the FilmFinder database to AWS RDS offers numerous benefits such as cost savings, scalability, disaster recovery, and accessibility. Despite slight differences in performance compared to a local MySQL instance, AWS RDS provides advantages in terms of manageability and reliability. With the implementation of Performance Insights, the application's performance can be further monitored and optimized. Overall, the migration to AWS RDS ensures a robust and efficient database management solution for the FilmFinder application.

8 References

GitHub URL :
https://github.com/soumiyarao/DATA225-Lab1

Queries and result screenshots used in the application :
https://docs.google.com/document/d/1bF2yBIiPd2gXxN0zh5KZ2pEOvNcZubqzbdRTZV3UfFQ

Performance analysis screenshots :
https://acrobat.adobe.com/link/review?uri=urn%3Aaaid%3Ascds%3AUS%3Acb971d94-4787-3d2f-b987-6b7f5e01006d