CSC 370 – Database Systems

Dr. Sean Chester

Movie Rating System

Project Sprint: 3

June 29th, 2024

Team27J

Alyssa Taylor V00987477

Karan Gosal V00979752

During this Sprint, we have covered up to some parts of **Level-3** for Data Analytics and **Level-1 and Level-2** for Backend Engineering. With Data Analytics we implemented some indexes to the database to many of the tables. For example, the Users table has an index on both the Username and Email attributes, since we expect those to need to be searched often.

With Backend Engineering we initialized Views into the database. These were created with multiple purposes, the main 2 being Data Aggregation and Data Security. For most users, the in-depth data is not only unnecessary, but could be difficult to understand as well. Views will give the data greater readability as it will filter out complex information. As for Data Security, Views will filter out private/protected information that would otherwise be visible to any user with the SELECT privilege, thus helping to ensure the safety of data.

We also looked into ACID properties and how they impact the database in regards to reliability and data integrity. It was determined that the database as a whole does not violate any of the properties, with examples in later sections as proof. In a real life situation however, it would be important to test against these properties in depth for true assurance.

All of these changes are discussed in greater depth in the following sections.

At the end of Sprint 4, we will begin covering **Level-3** of Data Modelling. This will involve using Inheritance Notions. We plan to spend the first week of Sprint 4 discussing how implementation could work, then we will develop inherited properties and weak entity sets into our database design. Additionally, early in Sprint 4 we'll discuss the quality of our ERD compared to some design principles to determine if any changes should be made to the design overall.

We also have determined some **current limitations** with our project, these include:

- The current design has not incorporated the concept of Inheritance and similar properties.
    - Weak Entities have not yet been discussed in depth to determine necessity, which may leave the current data model inaccurately displaying the relationships between system data.
    - Generalization and Inheritance have not yet been incorporated into the system, likely introducing some inefficiency into the database.
- The overall quality of the database has not been evaluated.
    - To ensure quality, we need to compare our database to known design principles. This would help confirm that the database is well designed and usable. So testing for Completeness, Correctness, Minimality, Expressiveness and Readability overall improves the schema.
    - We will also need to look into minimal bases and projecting FD's.

- Currently, there is no logging and recovery mechanism or practice as it is a very important aspect of a system in case of any disaster or malfunction. So, we will look more into that and try to introduce it in our project.

Users

user_id	username	password	email	full_name	is_admin	created_at
1	johndoe	securepassword	johndoe@example.com	John Doe	FALSE	2024-06-01 12:34
2	sidhum	testpassword	sidhumoose@example.com	Sidhu Moose	TRUE	2024-06-01 12:50

Stars_In

cast_id	movie_episode_id	episode_number
4	1	0
5	1	0
1	2	1
2	2	1
3	2	1
1	2	2
3	2	2

Rates

user_id	movie_episode_id	episode_number	rating	review	show_username
1	1	0	5	Great Movie	TRUE
2	2	1	4	Excellent pilot episode	FALSE
2	2	2	5	Enjoyable show	TRUE

Movie_Episodes

movie_episode_id	episode_number	title	release_year	synopsis	has_episodes
1	0	Die Hard	2014	John McClane and a young hacker join forces to take down master cyber-terrorist in Washington D.C.	0
2	0	Big Bang Theory	2005	A woman who moves into an apartment across the hall from two brilliant scientists.	1
2	1	S01: Pilot	2005	Meet all the characters first time.	0
2	2	S02: Leonard meets Penny	2005	In this Leonard meets Penny for the first time.	0

Genres

genre_id	genre_name
0	Scary
1	Comedy
2	Romance
3	Action
4	Thriller

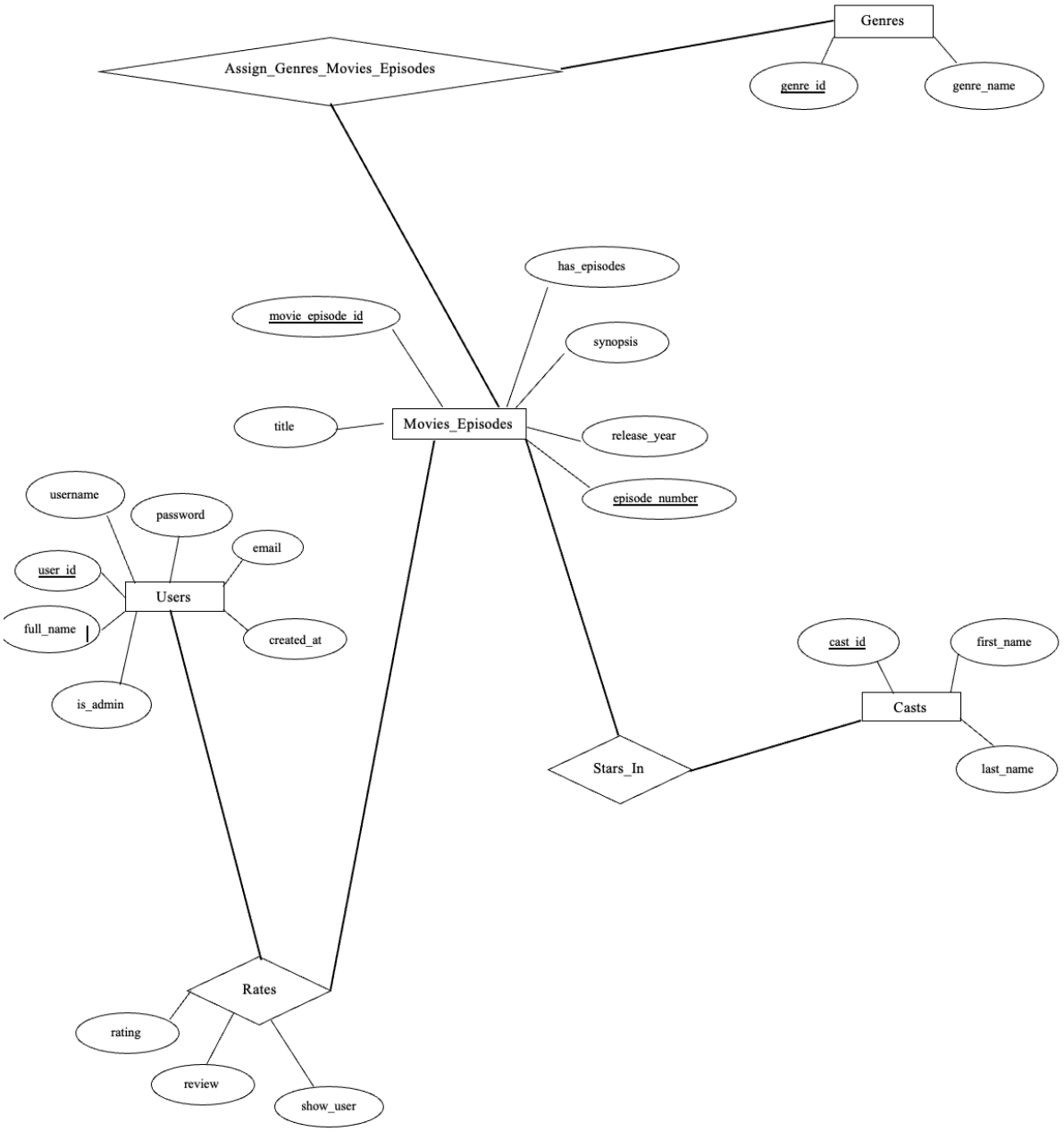
Casts

cast_id	first_name	last_name
1	Johnny	Galecki
2	Jim	Parsons
3	Kaley	Cuoco
4	Bruce	Willis
5	Justin	Long

Assign_Genres_to_Movie_Episodes

genre_id	movie_episode_id	episode_number
3	1	0
4	1	0
1	2	1
1	2	0
2	2	0

All the tables above represent the tables that follow our ERD schema and in the past sprint we have covered the basics for creating tables and creating tuples using the information from the CSV files or manually using INSERT SQL commands. Here is our ERD just in case.



INDEXES:

Indexes can significantly improve the performance of our database queries by allowing faster data retrieval. Keeping in mind that the dataset will grow in size in the future, we have identified these few places where indexing will be helpful and reduce querying time.

- **Users Table**:

  1) Index on *username*: Already exists as they are unique which will be helpful as usernames are often used for login or lookup operations.
  2) Index on *email*: Similar to usernames, emails might be frequently searched as well during login process. As they are also unique, so sql automatically adds index on it.

- **Casts Table**:

  Since, users want to search for casts using their first or last name, index on *first_name* and *last_name* will be helpful.

- **Genres Table**:

  This table will not need additional indexes because it is small and usually used in join operations on *genre_id*.

- **Movie_Episodes Table**:

  1) Index on *title*: To speed up searches by movie or episode title.
  2) Index on *release_year*: User might frequently filter or sort by release year.
  3) Composite index on *movie_episode_id* and *episode_number*: This is already covered by the primary key, so we don't need an additional index.

- **Rates Table**:

  1) Index on *user_id*: This will speed up queries that filter by user, such as finding all ratings by a particular user.
  2) Index on *movie_episode_id* and *episode_number*: To speed up queries filtering by specific movie episodes.

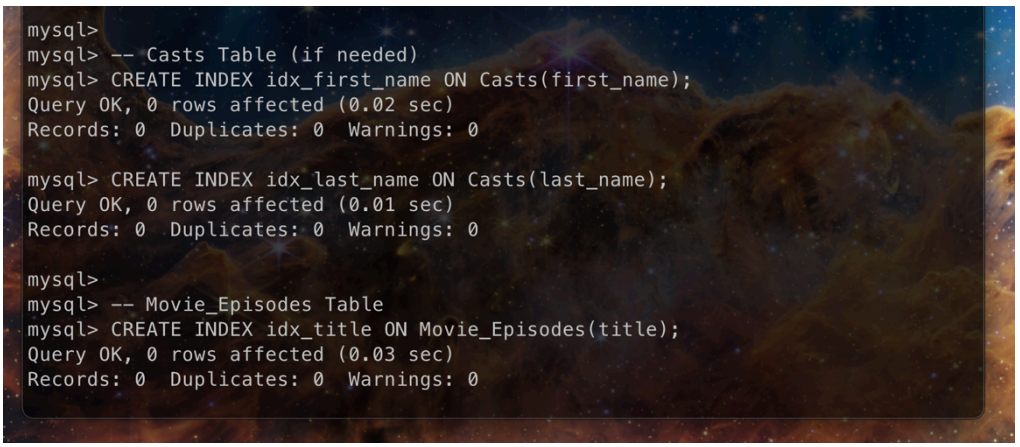
- **Stars_In Table**:

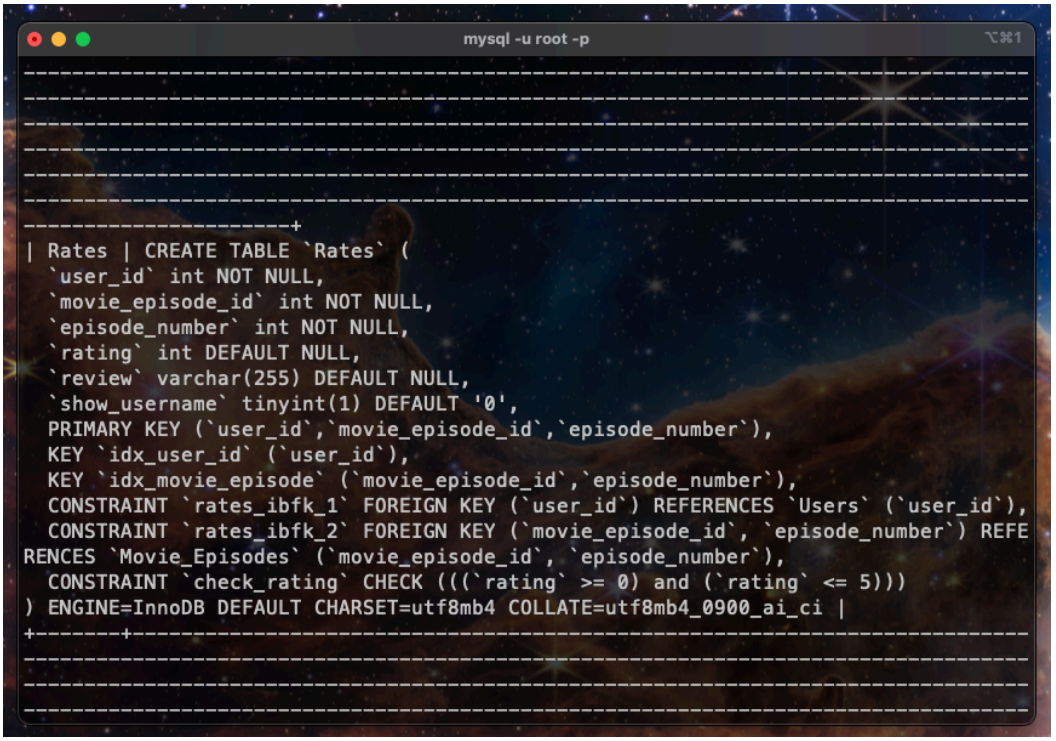
  1) Index on *cast_id*: To speed up queries filtering by cast members.
  2) Composite index on *movie_episode_id* and *episode_number*: This will help with queries that look up specific episodes.

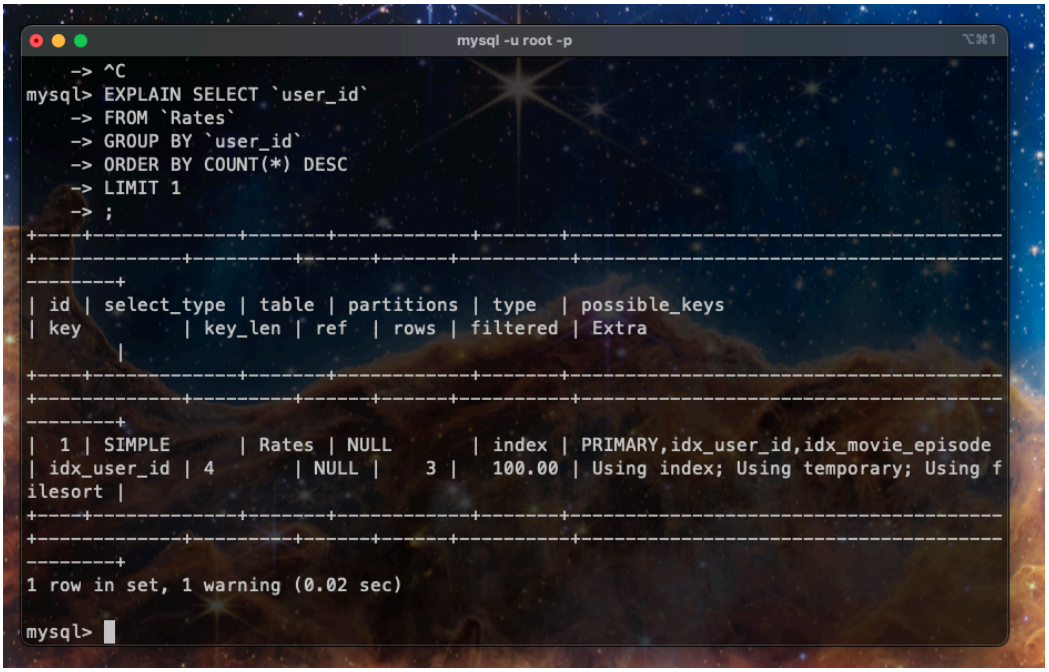
- **Assign_Genres_to_Movie_Episodes Table**:

  1) Index on *genre_id*: To optimize queries filtering by genre.
  2) Composite index on *movie_episode_id* and *episode_number*: To speed up queries associating genres with specific movie episodes.

  Running Example:







All the queries can be found in the separate file "PS3.sql" in the folder including this document.

USER AUTHORIZATION:

User authorization in databases is crucial for ensuring data security, integrity, compliance with regulations, accountability, resource management, and isolation of responsibilities. We can assign role-based authorization for now. So, we create three type of user which are admin, movie_cast_manager and rating_manager.

Roles:

sidhum (Admin): Sidhum serves as the administrator with full privileges across the database.

johndoe (Movie/Cast Manager): Johndoe is responsible for managing movies, casts, and related data within the system.

reviewManager (Reviews Manager): The reviewManager oversees the management of reviews within the system.

Assistants (e.g., movieManagerAssistant, reviewManagerAssistant): These are subordinate roles created by johndoe and reviewManager to assist in their respective tasks. They are granted specific privileges necessary for their roles, ensuring they can perform their duties effectively without needing full access.

The SQL commands added in the "PS3.sql" establish user roles and their associated privileges within the *Movie_Rating_System* database. We create administrative users (*sidhum*), a movie/cast manager (*johndoe*), and a reviews manager (*reviewManager*), each with specific permissions to their roles.

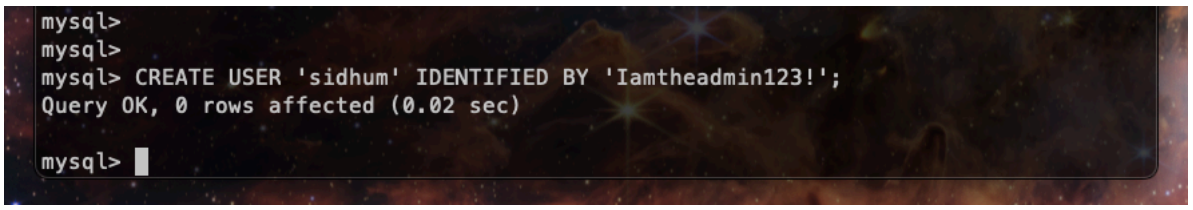
Administrative privileges are granted to *sidhum* for the entire database, allowing user management (CREATE USER) and superuser capabilities (SUPER). For *johndoe*, privileges are granted selectively on tables such as *Casts, Genres, Movie_Episodes, Stars_In*, and *Assign_Genres_to_Movie_Episodes*, enabling actions like select, insert, update, delete, and reference. Similarly, *reviewManager* is configured with permissions to manage review-related data (*Rates* table).

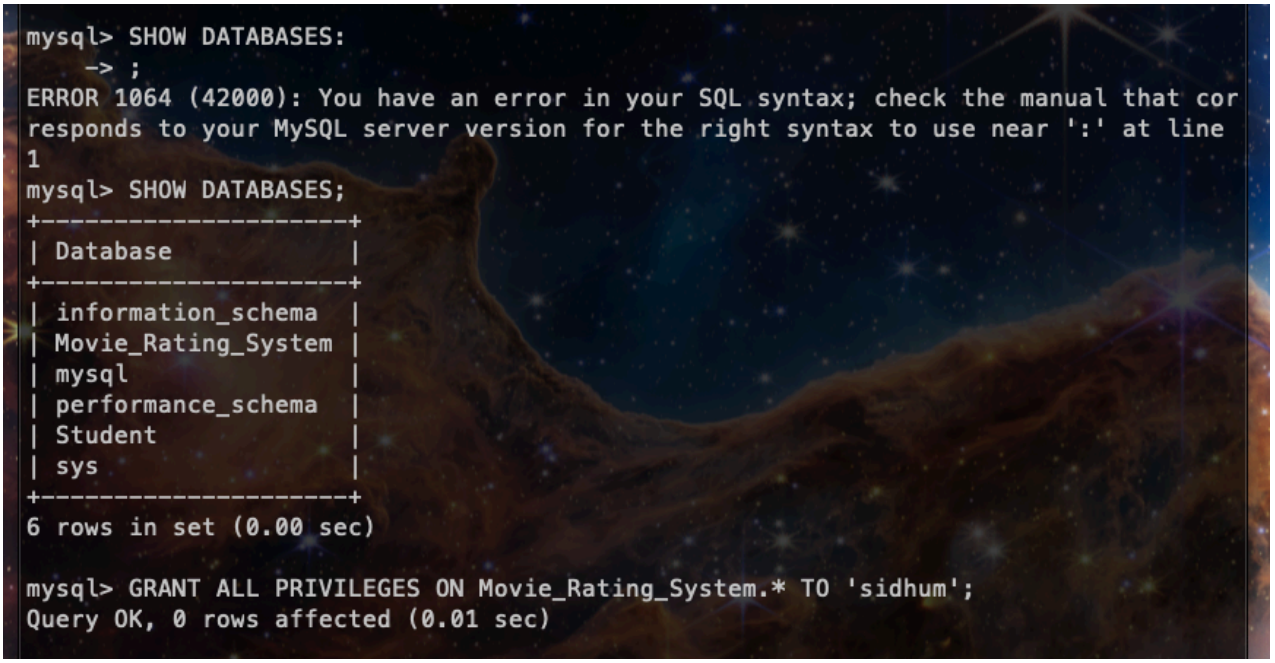
The GRANT OPTION is also given to the users like *johndoe* (movie/cast manager) and *reviewManager* (reviews manager) allowing them to not only have specific privileges themselves but also to grant those same privileges to other users they manage. For instance, *johndoe* can grant select, insert, update, delete, and reference privileges on various tables to *movieManagerAssistant*, enabling them to assist in managing movie-related data without needing higher-level administrative access.

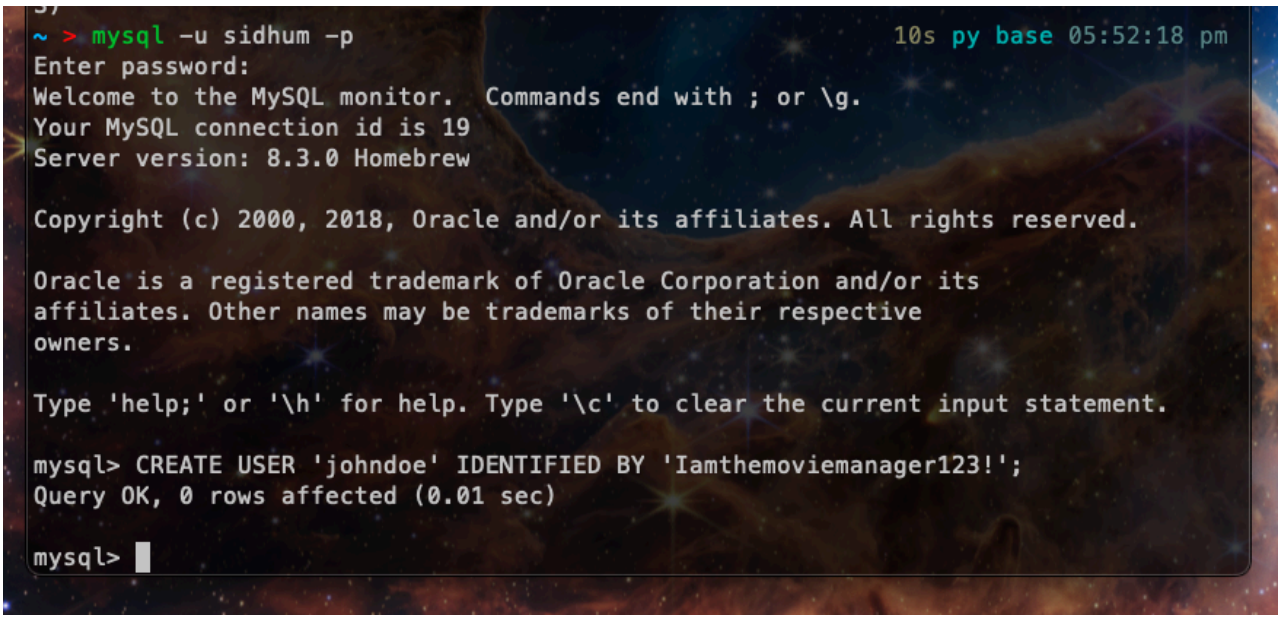
When it comes to revoking privileges, using the REVOKE command allows administrators or users with grant privileges to withdraw specific permissions from other users. This ensures that access to sensitive data or operations can be controlled and adjusted as needed without compromising database security. For example, if *johndoe* no longer requires update privileges on the Casts table, the REVOKE UPDATE ON *movie_rating_system.Casts* FROM *'johndoe'@'%'*; command removes this privilege, restricting *johndoe's* ability to modify cast information.

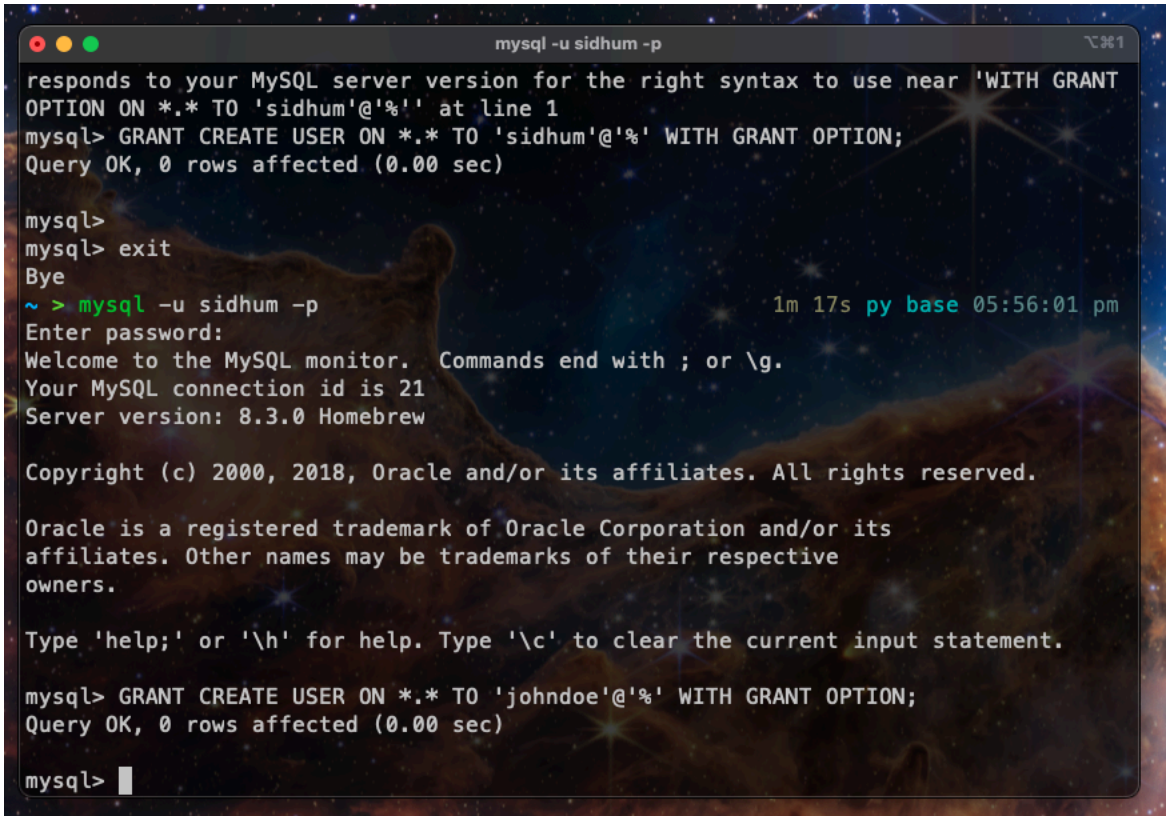
NOTE*: We have tried the REVOKE with CASCADE OR RESTRICT but after reading through the documentation, we found we cannot have the cascade effect here as mySql don't track further relations for the granting. So even if we revoke the permission given to johndoe and johndoe given to their assistant, only johndoe permission is revoked. To revoke the permission for assistant, we need to do that manually.*

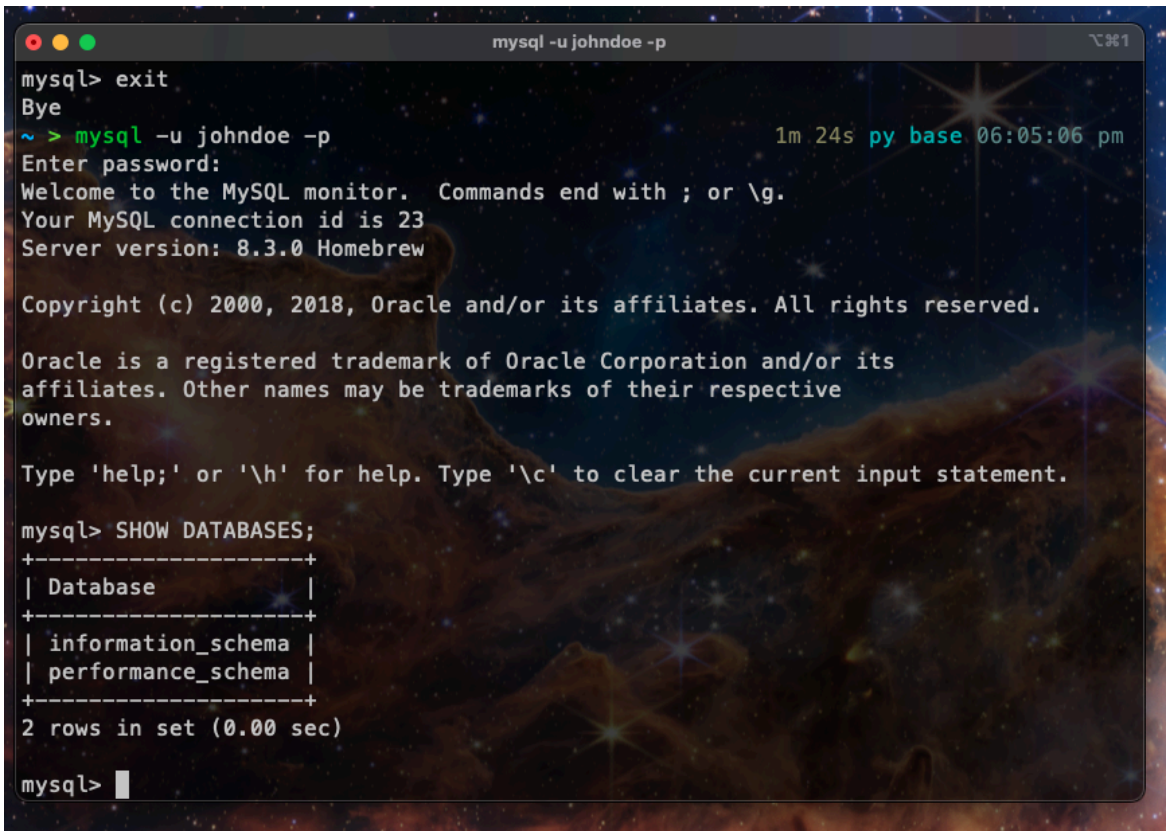
Below are the working screenshots for the commands in the sql file.

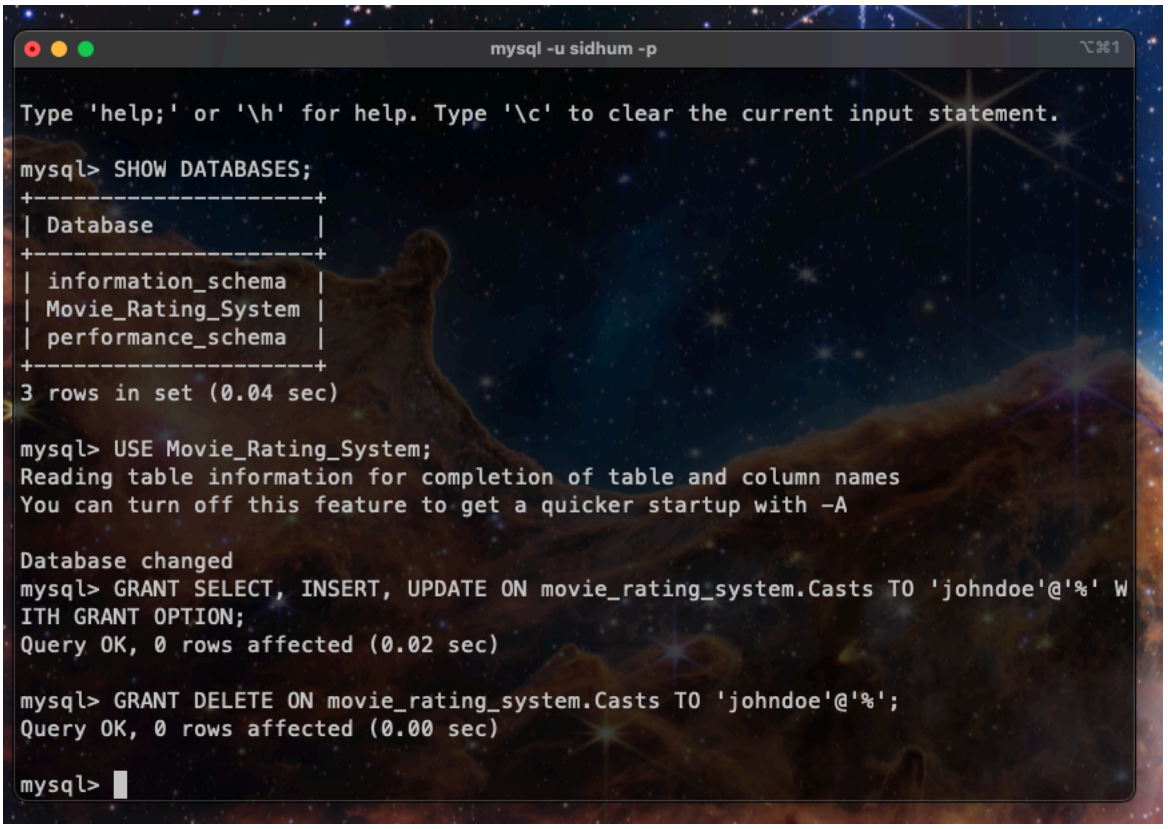

mysql>
mysql>
mysql> CREATE USER 'sidhum' IDENTIFIED BY 'Iamtheadmin123!';
Query OK, 0 rows affected (0.02 sec)
mysql>

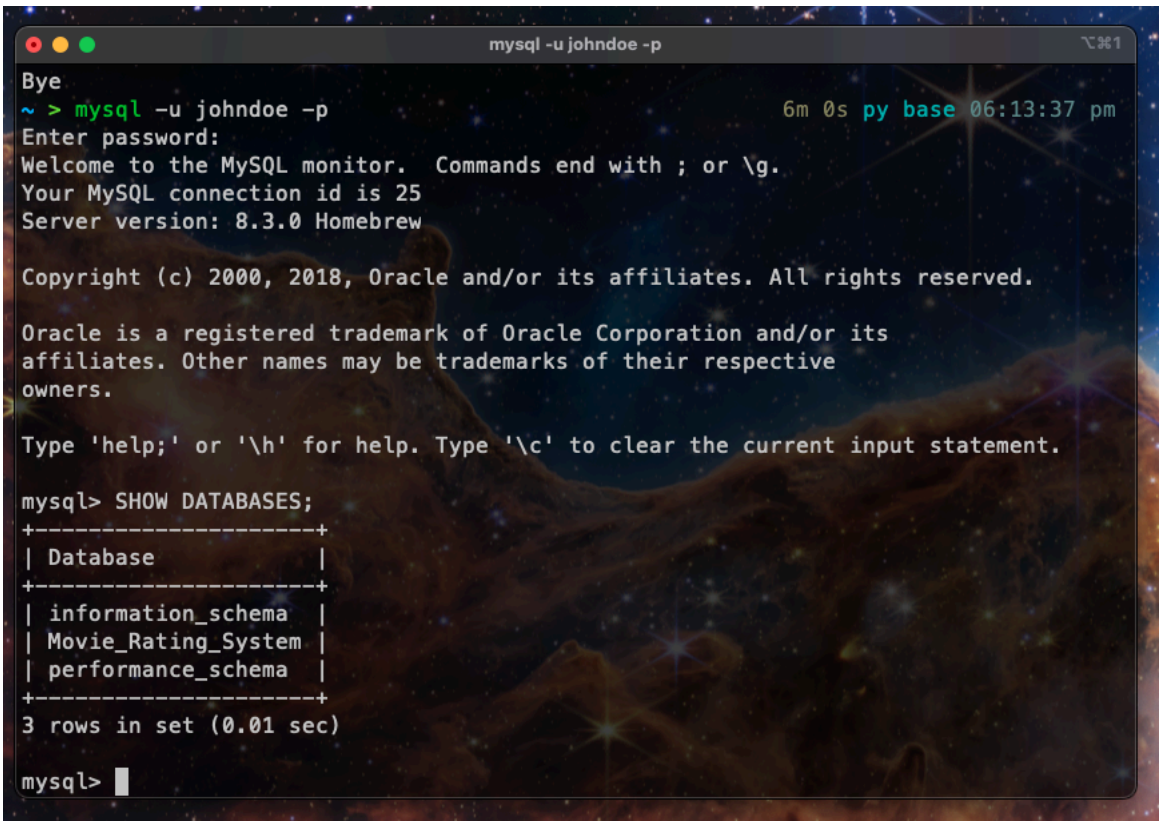

mysql> SHOW DATABASES:
-> ;
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that cor
responds to your MySQL server version for the right syntax to use near ':' at line
1
mysql> SHOW DATABASES;
+--------------------+
| Database |
+--------------------+
| information_schema |
| Movie_Rating_System |
| mysql |
| performance_schema |
| Student |
| sys |
+--------------------+
6 rows in set (0.00 sec)
mysql> GRANT ALL PRIVILEGES ON Movie_Rating_System.* TO 'sidhum';
Query OK, 0 rows affected (0.01 sec)

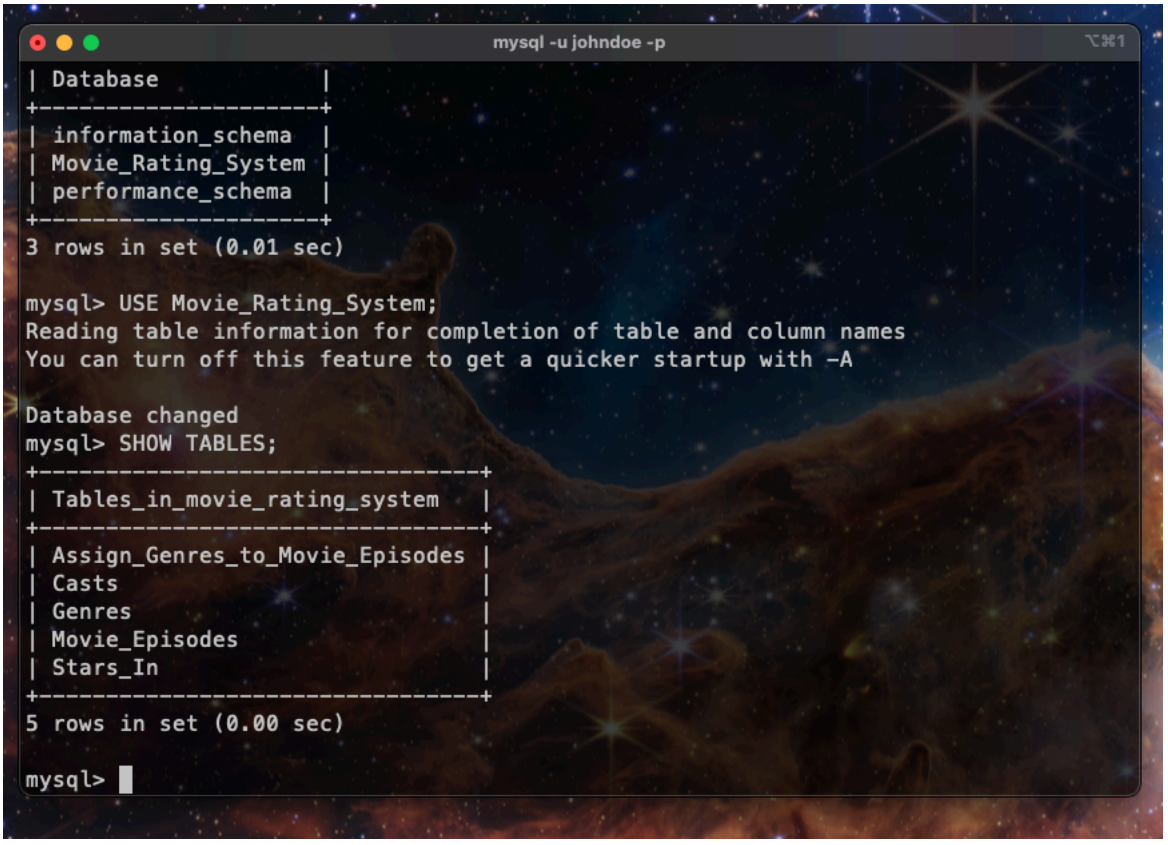

~ > mysql -u sidhum -p 10s py base 05:52:18 pm
Enter password:
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 19
Server version: 8.3.0 Homebrew
Copyright (c) 2000, 2018, Oracle and/or its affiliates. All rights reserved.
Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.
mysql> CREATE USER 'johndoe' IDENTIFIED BY 'Iamthemoviemanager123!';
Query OK, 0 rows affected (0.01 sec)
mysql>

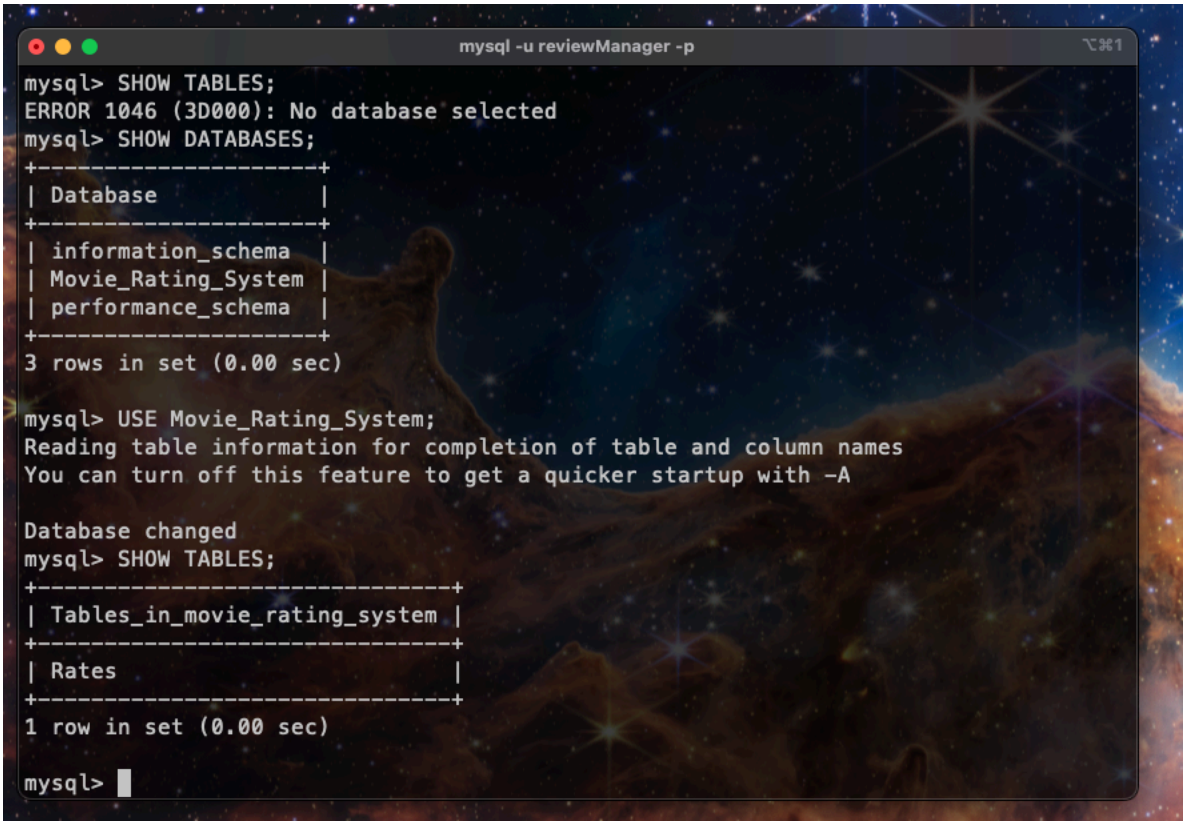

mysql -u sidhum -p
responds to your MySQL server version for the right syntax to use near 'WITH GRANT
OPTION ON *.* TO 'sidhum'@'%'' at line 1
mysql> GRANT CREATE USER ON *.* TO 'sidhum'@'%' WITH GRANT OPTION;
Query OK, 0 rows affected (0.00 sec)
mysql>
mysql> exit
Bye
~ > mysql -u sidhum -p 1m 17s py base 05:56:01 pm
Enter password:
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 21
Server version: 8.3.0 Homebrew
Copyright (c) 2000, 2018, Oracle and/or its affiliates. All rights reserved.
Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.
mysql> GRANT CREATE USER ON *.* TO 'johndoe'@'%' WITH GRANT OPTION;
Query OK, 0 rows affected (0.00 sec)
mysql>

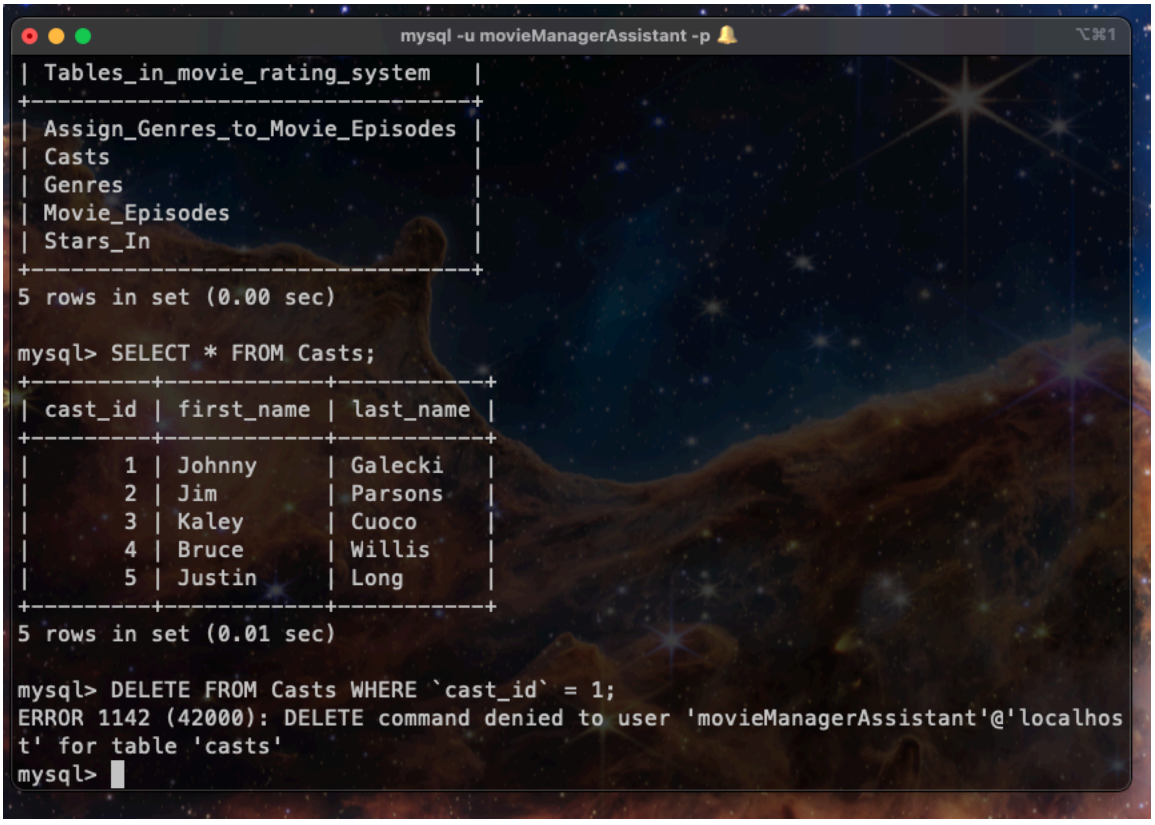

mysql -u johndoe -p
mysql> exit
Bye
~ > mysql -u johndoe -p
1m 24s py base 06:05:06 pm
Enter password:
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 23
Server version: 8.3.0 Homebrew
Copyright (c) 2000, 2018, Oracle and/or its affiliates. All rights reserved.
Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.
mysql> SHOW DATABASES;
| Database |
| information_schema |
| performance_schema |
2 rows in set (0.00 sec)
mysql>


mysql -u sidhum -p
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.
mysql> SHOW DATABASES;
| Database |
| information_schema |
| Movie_Rating_System |
| performance_schema |
3 rows in set (0.04 sec)
mysql> USE Movie_Rating_System;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A
Database changed
mysql> GRANT SELECT, INSERT, UPDATE ON movie_rating_system.Casts TO 'johndoe'@'%' W
ITH GRANT OPTION;
Query OK, 0 rows affected (0.02 sec)
mysql> GRANT DELETE ON movie_rating_system.Casts TO 'johndoe'@'%';
Query OK, 0 rows affected (0.00 sec)
mysql>


mysql -u johndoe -p
Bye
~ > mysql -u johndoe -p 6m 0s py base 06:13:37 pm
Enter password:
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 25
Server version: 8.3.0 Homebrew
Copyright (c) 2000, 2018, Oracle and/or its affiliates. All rights reserved.
Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.
mysql> SHOW DATABASES;
| Database |
| information_schema |
| Movie_Rating_System |
| performance_schema |
3 rows in set (0.01 sec)
mysql>


mysql -u johndoe -p
| Database |
| information_schema |
| Movie_Rating_System |
| performance_schema |
3 rows in set (0.01 sec)
mysql> USE Movie_Rating_System;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A
Database changed
mysql> SHOW TABLES;
| Tables_in_movie_rating_system |
| Assign_Genres_to_Movie_Episodes |
| Casts |
| Genres |
| Movie_Episodes |
| Stars_In |
5 rows in set (0.00 sec)
mysql>


mysql -u reviewManager -p
mysql> SHOW TABLES;
ERROR 1046 (3D000): No database selected
mysql> SHOW DATABASES;
+--------------------+
| Database |
+--------------------+
| information_schema |
| Movie_Rating_System |
| performance_schema |
+--------------------+
3 rows in set (0.00 sec)
mysql> USE Movie_Rating_System;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A
Database changed
mysql> SHOW TABLES;
+--------------------------------+
| Tables_in_movie_rating_system |
+--------------------------------+
| Rates |
+--------------------------------+
1 row in set (0.00 sec)
mysql>


mysql -u movieManagerAssistant -p
| Tables_in_movie_rating_system |
+--------------------------------+
| Assign_Genres_to_Movie_Episodes |
| Casts |
| Genres |
| Movie_Episodes |
| Stars_In |
+--------------------------------+
5 rows in set (0.00 sec)
mysql> SELECT * FROM Casts;
+---------+------------+-----------+
| cast_id | first_name | last_name |
+---------+------------+-----------+
1	Johnny	Galecki
2	Jim	Parsons
3	Kaley	Cuoco
4	Bruce	Willis
5	Justin	Long
+---------+------------+-----------+
5 rows in set (0.01 sec)
mysql> DELETE FROM Casts WHERE `cast_id` = 1;
ERROR 1142 (42000): DELETE command denied to user 'movieManagerAssistant'@'localhost' for table 'casts'
mysql>

```
mysql> REVOKE UPDATE ON movie_rating_system.Casts FROM 'johndoe'@'%';
Query OK, 0 rows affected (0.01 sec)
```

```
mysql -u johndoe -p
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| Movie_Rating_System |
| performance_schema |
+--------------------+
3 rows in set (0.01 sec)

mysql> USE Movie_Rating_System;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> UPDATE `Casts`
    -> SET `last_name` = 'G'
    -> WHERE `cast_id` = 1;
ERROR 1142 (42000): UPDATE command denied to user 'johndoe'@'localhost' for table '
casts'
mysql>
```

Grant Diagram Example for SELECT on Casts and Rates

VIEWS:

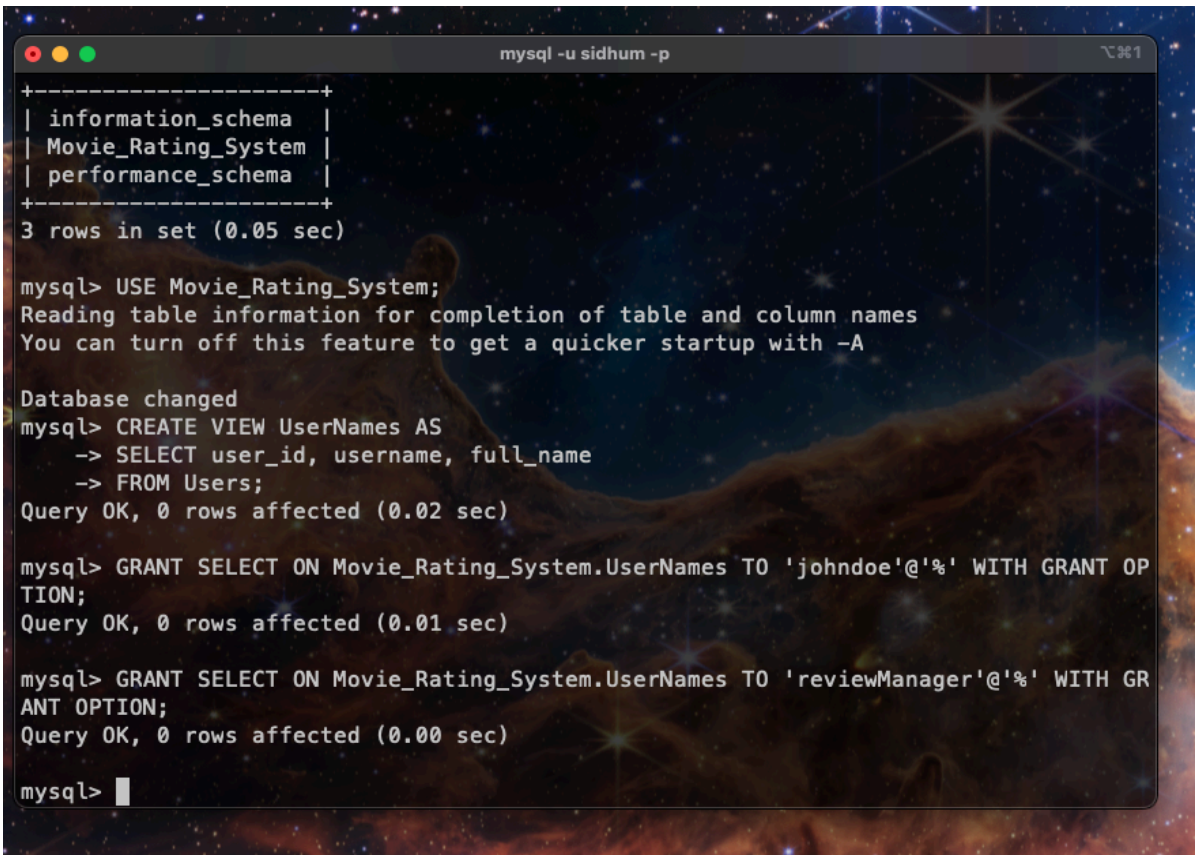
In our Movie Rating System project, views can be particularly useful for simplifying complex queries and enhancing data security and performance.

In our case, we can use views that aggregate data from tables like *Rates* to show average ratings per movie or per genre. For example, a view could calculate average ratings and reviews for each movie episode, making it easier to display summarized information without complex SQL queries every time.

Another use case is to create views that expose only non-sensitive data from tables like *Users* hiding fields like *passwords*. This enhances security by ensuring that only necessary information is accessible to different roles or users.

In the below screenshots, we have created the view and given the select privilege to the *johndoe* user so he can only see the *user_id*, *username* and *full_name* of the users hiding any sensitive information like *password* or e*mails*.

Other examples **for the views are added in the sql file.**


mysql -u sidhum -p
+--------------------+
| information_schema |
| Movie_Rating_System |
| performance_schema |
+--------------------+
3 rows in set (0.05 sec)
mysql> USE Movie_Rating_System;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A
Database changed
mysql> CREATE VIEW UserNames AS
-> SELECT user_id, username, full_name
-> FROM Users;
Query OK, 0 rows affected (0.02 sec)
mysql> GRANT SELECT ON Movie_Rating_System.UserNames TO 'johndoe'@'%' WITH GRANT OPTION;
Query OK, 0 rows affected (0.01 sec)
mysql> GRANT SELECT ON Movie_Rating_System.UserNames TO 'reviewManager'@'%' WITH GRANT OPTION;
Query OK, 0 rows affected (0.00 sec)
mysql>


mysql> SELECT * FROM UserNames;
+---------+----------+-------------+
| user_id | username | full_name |
+---------+----------+-------------+
| 1 | johndoe | John Doe |
| 2 | sidhum | Sidhu Moose |
+---------+----------+-------------+
2 rows in set (0.00 sec)
mysql>

```
mysql> USE Movie_Rating_System;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> DROP VIEW UserNames;
Query OK, 0 rows affected (0.02 sec)

mysql>
```

```
mysql> CREATE VIEW `View_Movie_Episodes` AS
    -> SELECT `movie_episode_id`, `episode_number`, `title`, `release_year`, `syno
psis`, `has_episodes`
    -> FROM `Movie_Episodes`;
Query OK, 0 rows affected (0.07 sec)

mysql> CREATE VIEW `View_Stars_In` AS
    -> SELECT `cast_id`, `movie_episode_id`, `episode_number`
    -> FROM `Stars_In`;
Query OK, 0 rows affected (0.01 sec)

mysql> CREATE VIEW `View_EpisodeCasts` AS
    -> SELECT *
    -> FROM `View_Movie_Episodes` NATURAL JOIN `View_Stars_In`;
Query OK, 0 rows affected (0.02 sec)

mysql> SELECT * FROM `View_EpisodeCasts`;
+------------------+----------------+-------------------------+--------------+---
-------------------------------------------------------------------------------
--------------+--------------+---------+
| movie_episode_id | episode_number | title                   | release_year | sy
nopsis
               | has_episodes | cast_id |
+------------------+----------------+-------------------------+--------------+---
-------------------------------------------------------------------------------
--------------+--------------+---------+
|                1 |              0 | Die Hard                |         2014 | Jo
hn McClane and a young hacker join forces to take down master cyber-terrorist in W
ashington D.C. |            0 |       4 |
|                1 |              0 | Die Hard                |         2014 | Jo
hn McClane and a young hacker join forces to take down master cyber-terrorist in W
ashington D.C. |            0 |       5 |
```

ACID PROPERTIES:

Atomicity: In our Movie Rating System, atomicity ensures that operations like user registration and rating submissions either complete fully or are rolled back entirely in case of failure. For example, when a new user registers, the database ensures that all required data (such as username, password, and email) is inserted completely into the Users table, ensuring no partial data is left in an inconsistent state. Similarly, when a user rates a movie episode, the entire rating transaction—consisting of inserting the rating details into the Rates table—is either committed entirely or aborted if any part of the transaction fails, preserving data integrity. An example for this is not very applicable on our system as most of the things can be done in single queries which are atomic in nature. However, if we think of updating multiple users date created and we need to do it using multiple queries, we want this transaction to be either fully complete or not at all.

Consistency: Consistency guarantees that the database transitions from one valid state to another, enforcing data integrity rules. For instance, when adding a rating, the system verifies that the *user_id* exists in the Users table and that the *movie_episode_id* exists in the *Movie_Episodes* table before allowing the rating insertion. This ensures that only valid data relationships are maintained, preventing orphaned records or inconsistent data states. Constraints such as foreign keys and validation triggers ensure that every rating operation adheres to predefined rules, maintaining overall database consistency.

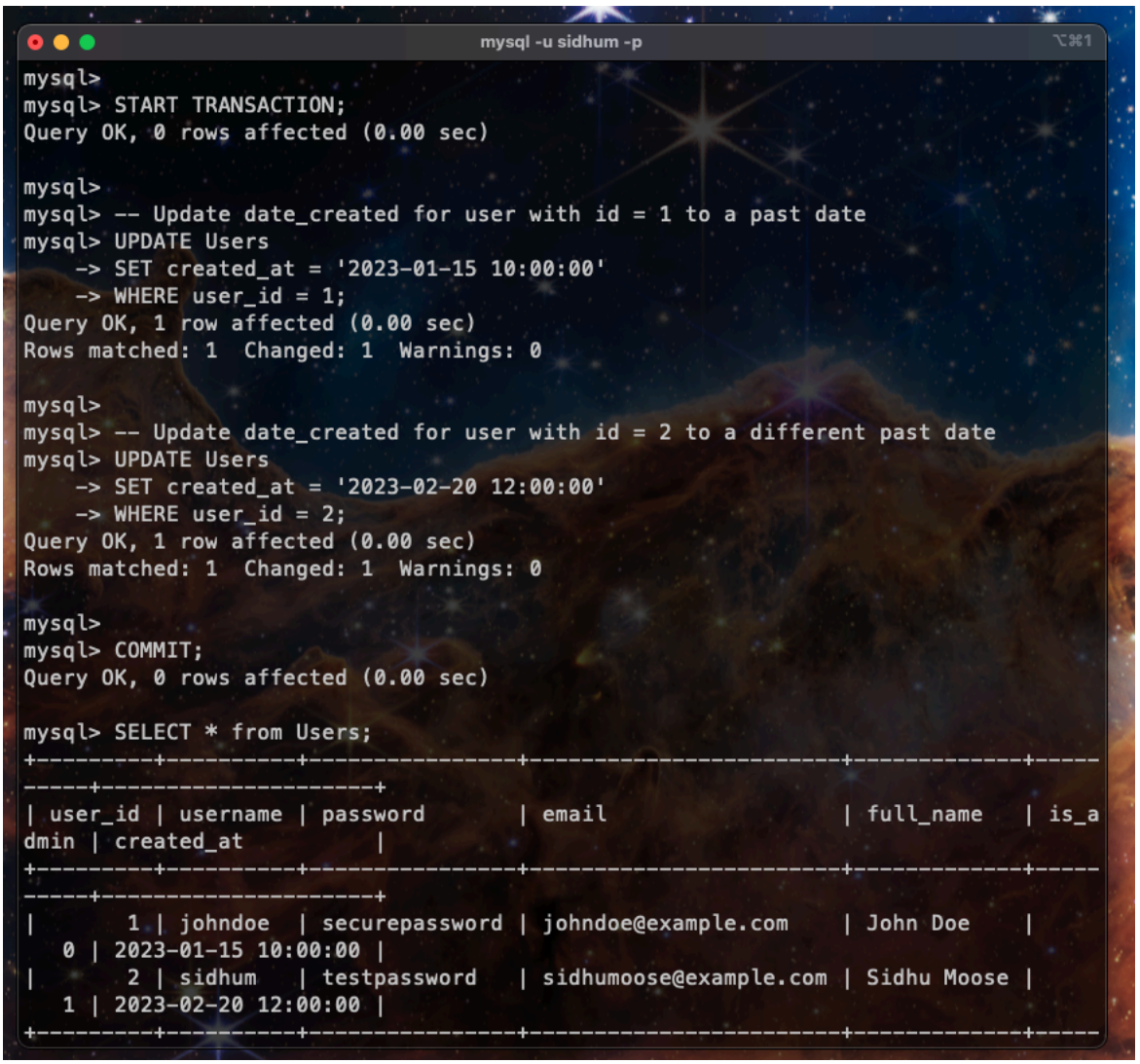
Isolation: Isolation ensures that concurrent transactions execute independently without interference, maintaining data integrity even when multiple users are accessing or modifying data simultaneously. For example, when two users simultaneously rate different movie episodes, the isolation level of transactions prevents them from seeing each other's intermediate states. This prevents issues like lost updates or inconsistent reads, ensuring that each transaction operates on a snapshot of data consistent with its start time, thereby preserving data integrity and avoiding conflicts.

Durability: Durability guarantees that once a transaction is committed, changes persist even in the event of system failures like crashes or power outages. For instance, after a user registers or updates their profile information, the changes are permanently stored in the database, and the system ensures that these modifications are written to non-volatile storage. This ensures that even if the system crashes immediately after a transaction is committed, the database can recover and restore the committed changes, maintaining the integrity and availability of data over time.

TRANSACTION ATOMICITY:

Atomic transactions are essential in database systems to ensure data consistency and integrity. By grouping multiple operations into a single unit of work, atomic transactions guarantee that either all operations succeed and are committed, or none of them are.

Just an example of atomic transaction.



CONCURRENCY AND ISOLATION LEVELS:

In our Movie Rating System project, we would choose different isolation levels based on the requirements of transactions and the trade-offs between concurrency and data consistency. Some scenarios to showcase this:

- **Serializable**

  Use Case: For critical transactions where absolute data consistency is required and concurrent updates could lead to conflicts.

  Example: When updating aggregated statistics across multiple tables or when handling transactions that require precise and consistent views of data.

- **Repeatable Read**

  Use Case: Transactions that require consistent reads of data without phantom reads (where new rows appear after the transaction starts).

  Example: When calculating total ratings for a movie that shouldn't include new ratings added during the calculation period.

- **Read Committed**

  Use Case: Situations where we need to avoid dirty reads (reading uncommitted data) but can tolerate phantom reads.

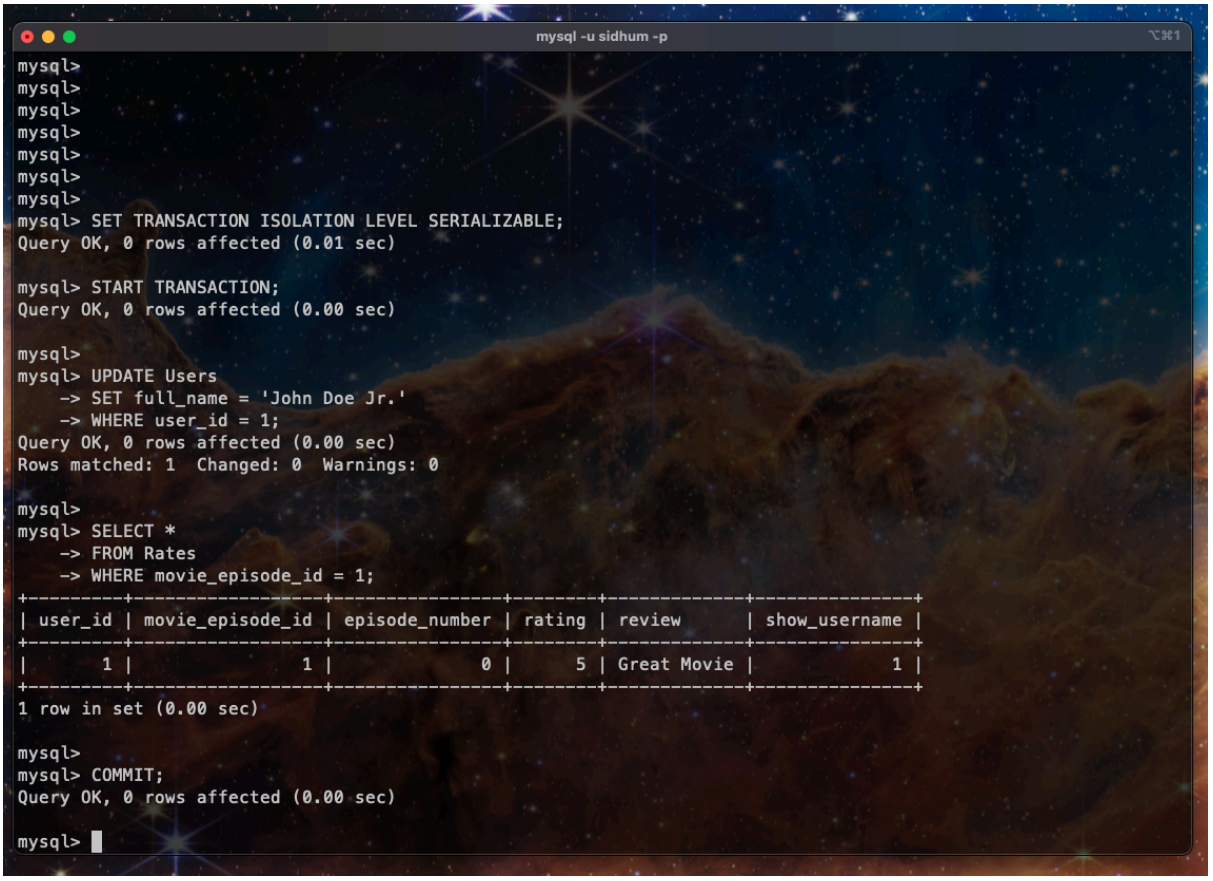
  Example: Displaying the latest ratings or reviews for movies while allowing concurrent updates to the rating data.

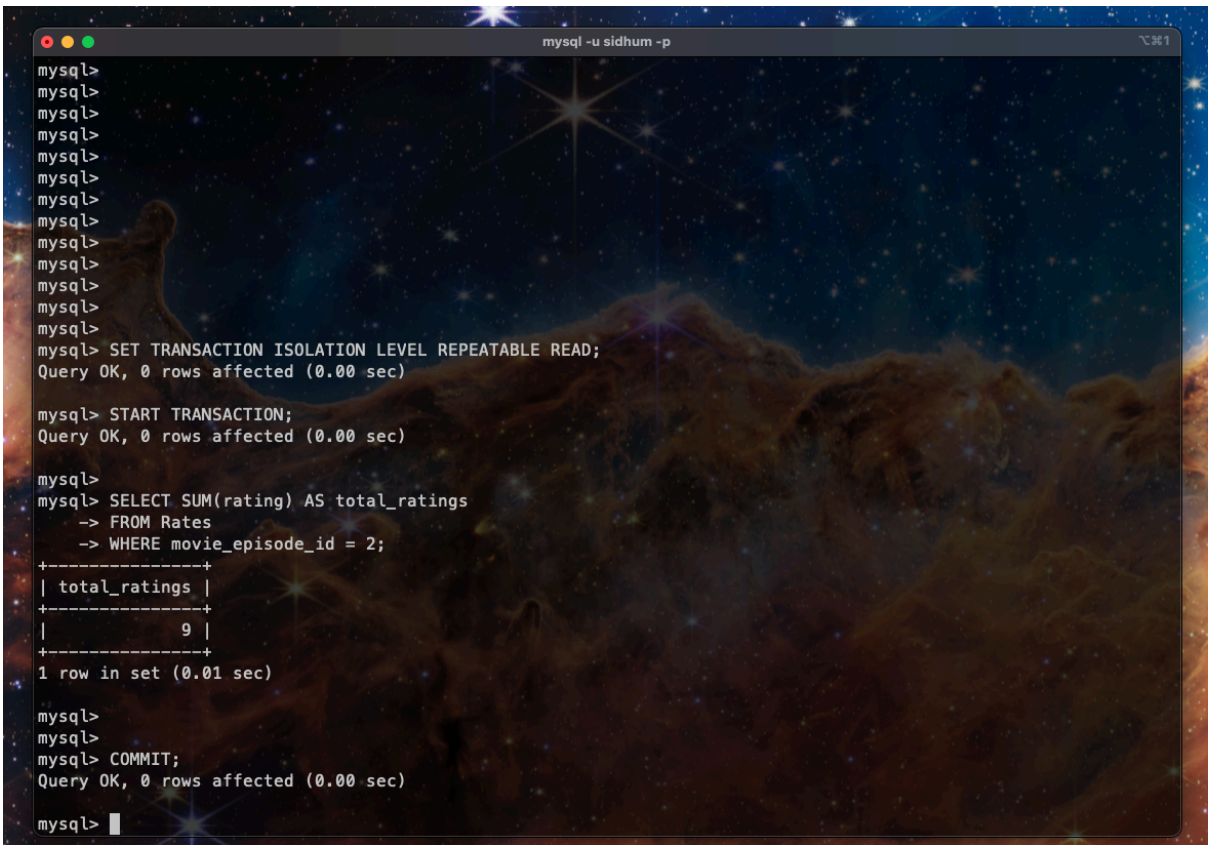
- **Read Uncommitted**

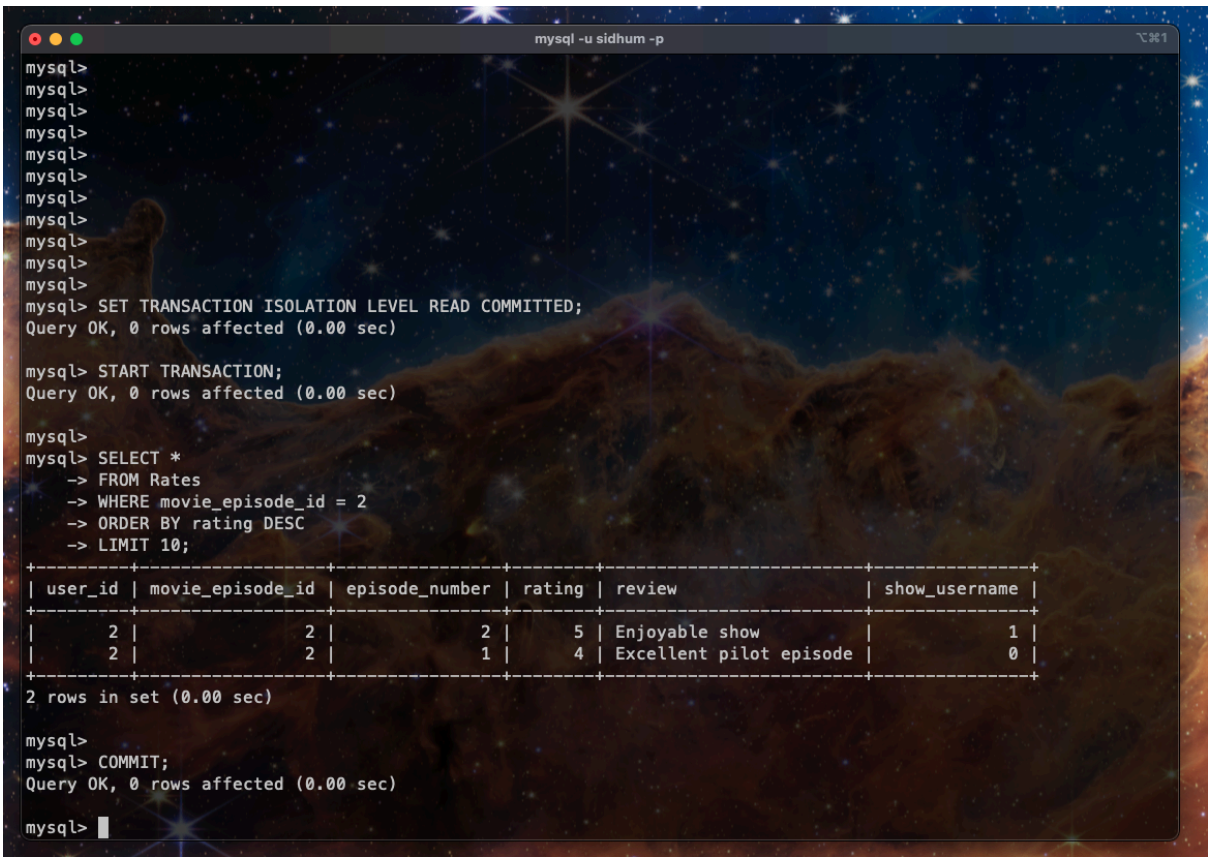
  Use Case: When performance is critical and we can tolerate potentially inconsistent reads (dirty reads).

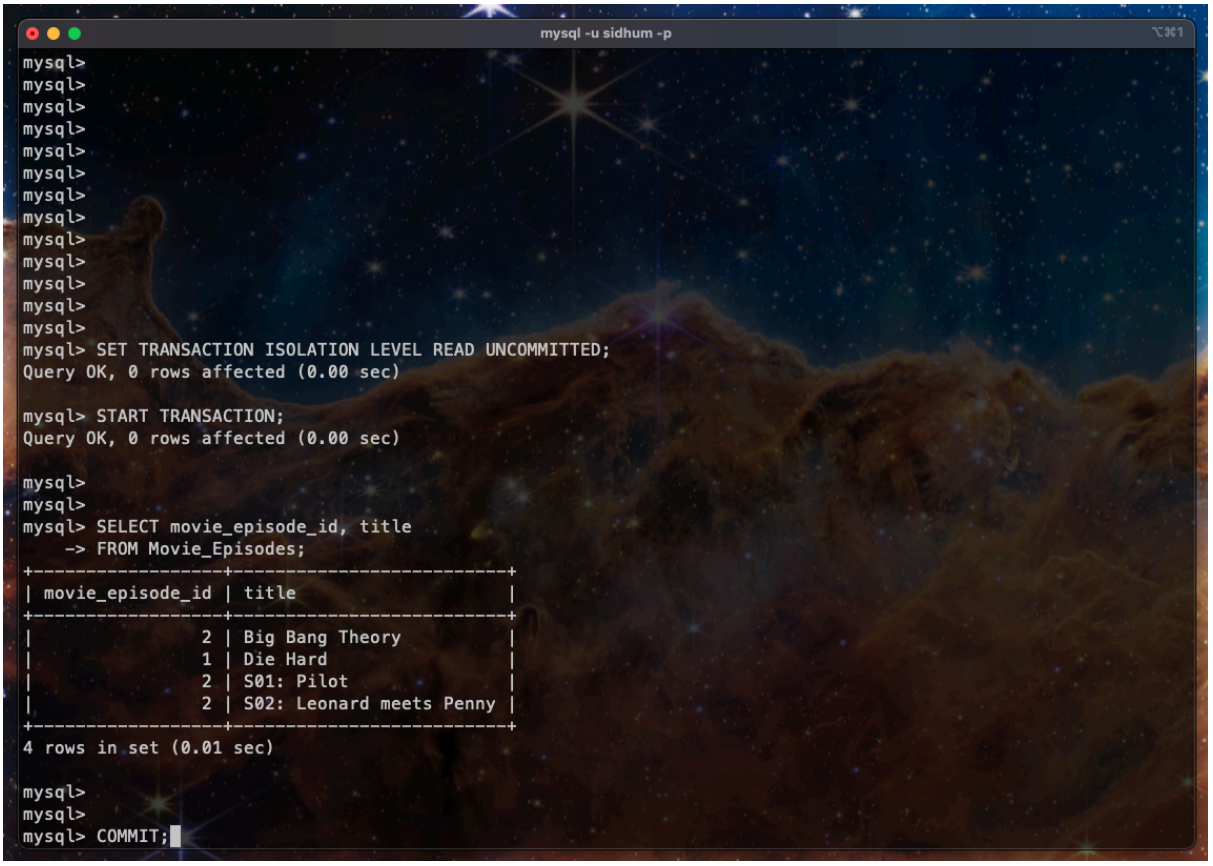
Example: Displaying non-critical information like movie titles or brief summaries where immediate updates aren't crucial.

Below are the screenshots of each type.




mysql -u sidhum -p
mysql>
mysql>
mysql>
mysql>
mysql>
mysql>
mysql>
mysql>
mysql>
mysql>
mysql>
mysql>
mysql>
mysql> SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;
Query OK, 0 rows affected (0.00 sec)
mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)
mysql>
mysql> SELECT SUM(rating) AS total_ratings
-> FROM Rates
-> WHERE movie_episode_id = 2;
| total_ratings |
| 9 |
1 row in set (0.01 sec)
mysql>
mysql>
mysql> COMMIT;
Query OK, 0 rows affected (0.00 sec)
mysql>


mysql -u sidhum -p
mysql>
mysql>
mysql>
mysql>
mysql>
mysql>
mysql>
mysql>
mysql>
mysql>
mysql>
mysql>
mysql> SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
Query OK, 0 rows affected (0.00 sec)
mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)
mysql>
mysql> SELECT *
-> FROM Rates
-> WHERE movie_episode_id = 2
-> ORDER BY rating DESC
-> LIMIT 10;
user_id	movie_episode_id	episode_number	rating	review	show_username
2	2	2	5	Enjoyable show	1
2	2	1	4	Excellent pilot episode	0
2 rows in set (0.00 sec)
mysql>
mysql> COMMIT;
Query OK, 0 rows affected (0.00 sec)
mysql>


mysql -u sidhum -p
mysql>
mysql>
mysql>
mysql>
mysql>
mysql>
mysql>
mysql>
mysql>
mysql>
mysql>
mysql>
mysql>
mysql> SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;
Query OK, 0 rows affected (0.00 sec)
mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)
mysql>
mysql>
mysql> SELECT movie_episode_id, title
-> FROM Movie_Episodes;
+------------------+--------------------------+
| movie_episode_id | title |
+------------------+--------------------------+
2	Big Bang Theory
1	Die Hard
2	S01: Pilot
2	S02: Leonard meets Penny
+------------------+--------------------------+
4 rows in set (0.01 sec)
mysql>
mysql>
mysql> COMMIT;