**SVKM’s NMIMS**

**School of Technology Management & Engineering, Chandigarh**

A.Y. 2023 - 24

**Course: Database Management Systems**

**Project Report**

|  |  |  |
| --- | --- | --- |
| Program | Btech. Computer Science and Business Systems (CSBS) | |
| Semester | 4 | |
| Name of the Project: | Movie Tracking System using Python and SQL | |
|  | | |
| Details of Project Members |  |  |
| Batch | Roll No. | Name |
| E2 | E044 | Saloni Sivakumar |
| E2 | E053 | Mansa Mohanty |
| E2 | E075 | Prarthana Subudhi |
| Date of Submission: | | |

**Contribution of each project Members:**

|  |  |  |
| --- | --- | --- |
| Roll No. | Name: | Contribution |
| 44 | Saloni Sivakumar | Tables creation,ER diagram,python code,writing report |
| 53 | Mansa Mohanty | SQL queries,python code, writing report |
| 75 | Prarthana Subudhi | SQL queries,python code,writing report |

**Github link :**

**https://github.com/saloniskumar/Movie\_tracker**

**Project Report**

**Movie Tracking System using Python and SQL**

**by**

**Saloni Sivakumar, Roll number:44**

**Mansa Mohanty Roll number: 53**

**Prarthana Subudhi, Roll number: 75**

**Course: DBMS**

**AY: 2023-24**

**Table of Contents**

|  |  |  |
| --- | --- | --- |
| **Sr no.** | **Topic** | **Page no.** |
| **1** | Storyline | 4 |
| **2** | Components of Database Design | 5 |
| **3** | Entity Relationship Diagram | 6 |
| **4** | Relational Model | 6 |
| **5** | Normalization | 7 |
| **6** | SQL Queries | 8 |
| **7** | Learning from the Project | 17 |
| **8** | Project Demonstration | 16 |
| **9** | Self-learning beyond classroom | 17 |
| **10** | Challenges faced | 18 |
| **10** | Conclusion | 18 |

**I. Storyline**

**FUNCTIONAL REQUIREMENTS: -**

1. Movie tracking: As a user, I would like to be able to maintain a record of all my movies in my database and when I have added them.
2. Reviewing: As a user, I would want to give movies a rating of 1 to 10.
3. Classify Movies:-As a user, I would like to be able to sort my movies based on -Currently watching, to watch and already watched.
4. Sort and search movies: As a user,I wish to sort my list based on rating and search movies using movie title, actors, etc
5. View Movie Description: As a user, I wish to automatically see a movie’s description by just searching it’s title
6. Privacy: As a user,I would want my database to be accessible only by me

**NON-FUNCTIONAL REQUIREMENTS:**

**AS A USER: -**

1. Loading time: As a user, I want the application to load as fast as possible
2. Privacy: As a user, I want to control what information about me is accessible to other people..
3. Compatibility: As a user, I would want to run the application in both IOS and Android devices
4. Space: As a user, I would want the application to take less amount of storage on my device

**AS A DEVELOPER: -**

1. Cost and Security: As a developer, I would want every user's data to be safe and secure
2. Performance: As a developer, I want my application to load in less than two seconds and have a quick response time.
3. Scalability: In my capacity as a developer, I want my application to support millions of people worldwide.

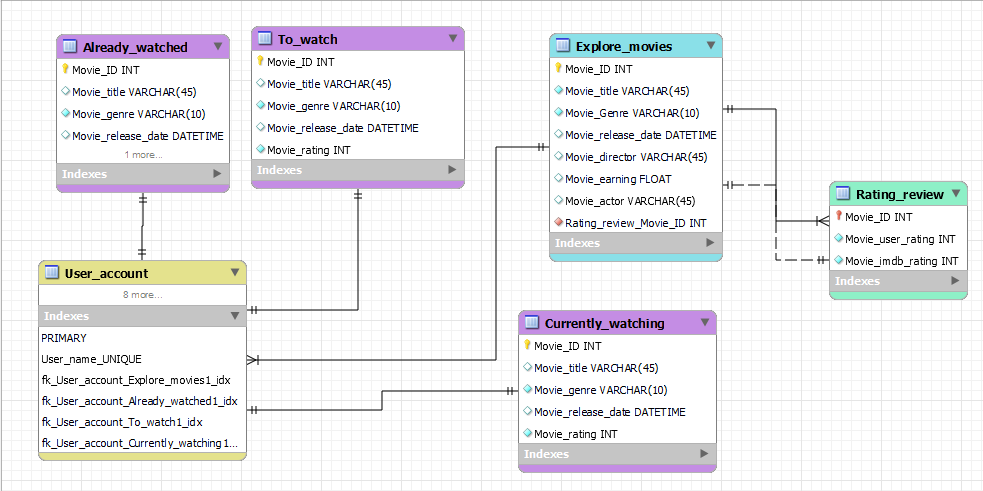
**SYSTEM REQUIREMENTS-**

1. Hardware: 64-bit operating system,8 GB RAM,12th Gen Intel(R) Core(TM) i5-12500T
2. Software: uninterrupted access to IMDB APIs,MySQL workbench(DBMS)

**II. Components of Database Design**

|  |  |  |  |
| --- | --- | --- | --- |
| Name of entity | List of Attributes | Primary Key | Cardinality & Participation |
| User\_account | User\_id,User\_name,User\_password,User\_bio | User\_id | 1-1 with Already\_watched  To\_watch  Currently\_watching  1-M with Explore\_movies,Rating\_review |
| Already\_watched | Movie\_ID,Movie\_title,Movie\_genre,  Movie\_release date, Movie\_rating | Movie\_ID | 1-1 with User\_account |
| To\_watch | Movie\_ID,Movie\_title,Movie\_genre,  Movie\_release date, Movie\_rating | Movie\_ID | 1-1 with User\_account |
| Currently\_watching | Movie\_ID,Movie\_title,Movie\_genre,  Movie\_release date, Movie\_rating | Movie\_ID | 1-1 with User\_account |
| Explore\_movies | Movie\_ID, Movie\_title,Movie\_Genre,  Movie\_release\_date , Movie\_director,  Movie\_earning,Movie\_actor | Movie\_ID | M-1 with User\_account  1-1 with Rating\_review |
| Rating\_review | Movie\_ID,Movie\_user\_rating,  Movie\_imdb\_rating | Movie\_ID | M-1 with User\_account  1-1 with Explore\_movies |

**III. Entity Relationship Diagram**



**IV. Relational Model**

CREATE DATABASE project;

use project;

CREATE TABLE Explore\_movies (

Movie\_ID INT NOT NULL,

Movie\_title VARCHAR(45) NOT NULL,

Movie\_Genre VARCHAR(10) NOT NULL,

Movie\_release\_date DATETIME NOT NULL,

Movie\_director VARCHAR(45) NOT NULL,

Movie\_earning FLOAT NULL,

Movie\_actor VARCHAR(45)NOT NULL,

Rating\_review\_Movie\_ID INT NOT NULL,

PRIMARY KEY ( Movie\_ID )

);

CREATE TABLE Rating\_review (

Movie\_ID INT NOT NULL,

Movie\_user\_rating INT NOT NULL,

Movie\_imdb\_rating INT NOT NULL,

PRIMARY KEY (Movie\_ID),

FOREIGN KEY (Movie\_ID)

REFERENCES Explore\_movies(Movie\_ID)

);

CREATE TABLE Already\_watched (

Movie\_ID INT NOT NULL,

Movie\_title VARCHAR(45) UNIQUE,

Movie\_genre VARCHAR(10) NOT NULL,

Movie\_release\_date DATETIME NOT NULL,

Movie\_rating INT NOT NULL,

PRIMARY KEY ( Movie\_ID )

);

CREATE TABLE To\_watch (

Movie\_ID INT NOT NULL,

Movie\_title VARCHAR(45) UNIQUE,

Movie\_genre VARCHAR(10) NOT NULL,

Movie\_release\_date DATETIME NOT NULL,

Movie\_rating INT NOT NULL,

PRIMARY KEY ( Movie\_ID )

);

CREATE TABLE Currently\_watching (

Movie\_ID INT NOT NULL,

Movie\_title VARCHAR(45) NOT NULL,

Movie\_genre VARCHAR(10) NOT NULL,

Movie\_release\_date DATETIME NOT NULL,

Movie\_rating INT NOT NULL,

PRIMARY KEY ( Movie\_ID )

);

CREATE TABLE User\_account (

User\_id INT NOT NULL,

User\_name VARCHAR(45) UNIQUE,

User\_password VARCHAR(45) NOT NULL,

User\_bio VARCHAR(100) NOT NULL,

PRIMARY KEY ( User\_id)

);

**V. Normalization**

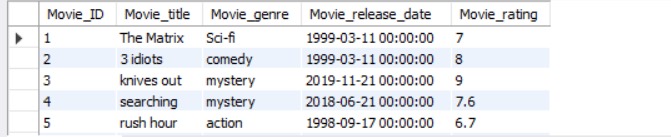
Not applicable as already in 3NF

**VI. SQL Queries**

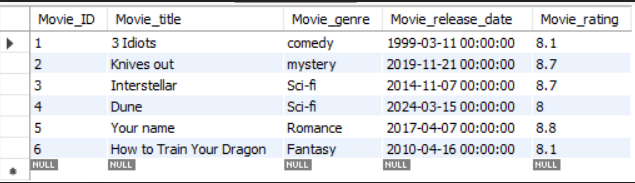
use project;  
insert into to\_watch (Movie\_ID,movie\_title,Movie\_genre,Movie\_release\_date,Movie\_rating) values ('2','3 idiots','comedy','1999-03-11','8.1');  
insert into to\_watch (Movie\_ID,movie\_title,Movie\_genre,Movie\_release\_date,Movie\_rating) values ('3','knives out','mystery','2019-11-21','8.7');  
  
update to\_watch set movie\_rating=7 where movie\_genre='Sci-fi';  
  
delete from to\_watch where movie\_rating='8';  
  
start transaction;  
update to\_watch set Movie\_title='knives out 2' where Movie\_ID='3';  
rollback;  
  
select count(\*)as number\_of\_movies from to\_watch;  
select \* from to\_watch order by Movie\_title desc;  
alter table to\_watch modify column Movie\_rating float(10);  
insert into to\_watch (Movie\_ID,movie\_title,Movie\_genre,Movie\_release\_date,Movie\_rating) values ('4','searching','mystery','2018-06-21','7.6'),  
('5','rush hour','action','1998-09-17','6.7'),  
( '6','hera pheri','comedy','2000-03-31','8.2'),  
('7','interstellar','Sci-fi','2014-11-07','8.7');  
select \* from to\_watch;  
select \* from to\_watch where Movie\_rating <9 and  Movie\_title like "%r";  
select Movie\_genre,count(movie\_title) from to\_watch group by Movie\_genre;  
  
alter table already\_watched modify column Movie\_rating float(10);  
insert into already\_watched (Movie\_ID,movie\_title,Movie\_genre,Movie\_release\_date,Movie\_rating) values ('1','3 Idiots','comedy','1999-03-11','8.1');  
insert into already\_watched (Movie\_ID,movie\_title,Movie\_genre,Movie\_release\_date,Movie\_rating) values ('2','Knives out','mystery','2019-11-21','8.7');  
insert into already\_watched (Movie\_ID,movie\_title,Movie\_genre,Movie\_release\_date,Movie\_rating) values ('3','Interstellar','Sci-fi','2014-11-07','8.7');  
insert into already\_watched (Movie\_ID,movie\_title,Movie\_genre,Movie\_release\_date,Movie\_rating) values ('4','Dune','Sci-fi','2024-03-15','8.0');  
insert into already\_watched (Movie\_ID,movie\_title,Movie\_genre,Movie\_release\_date,Movie\_rating) values ('5','Your name','Romance','2017-04-07','8.8');  
insert into already\_watched (Movie\_ID,movie\_title,Movie\_genre,Movie\_release\_date,Movie\_rating) values ('6','How to Train Your Dragon','Fantasy','2010-04-16','8.1');  
  
select \* from already\_watched;  
update already\_watched set Movie\_rating=8.1 where Movie\_ID=1;  
update already\_watched set Movie\_rating=8.7 where Movie\_ID=2;  
update already\_watched set Movie\_rating=8.7 where Movie\_ID=3;  
update already\_watched set Movie\_rating=8.0 where Movie\_ID=4;  
update already\_watched set Movie\_rating=8.8 where Movie\_ID=5;  
update already\_watched set Movie\_rating=8.1 where Movie\_ID=6;  
select \* from already\_watched;  
  
select \* from already\_watched inner join to\_watch on already\_watched.Movie\_title=to\_watch.Movie\_title;  
select \* from already\_watched left join to\_watch on already\_watched.Movie\_title=to\_watch.Movie\_title;  
select \* from already\_watched right join to\_watch on already\_watched.Movie\_genre=to\_watch.Movie\_genre;  
select \* from already\_watched union select \* from to\_watch;  
select \* from already\_watched intersect select \* from to\_watch;  
select \* from already\_watched except select \* from to\_watch;  
  
alter table currently\_watching modify column Movie\_rating float(10);  
insert into currently\_watching (Movie\_ID,movie\_title,Movie\_genre,Movie\_release\_date,Movie\_rating) values ('1','Bullet train','Action','2022-08-05','7.3');  
insert into currently\_watching (Movie\_ID,movie\_title,Movie\_genre,Movie\_release\_date,Movie\_rating) values ('2','Fight club','Thriller','1999-10-15','8.8');  
insert into currently\_watching (Movie\_ID,movie\_title,Movie\_genre,Movie\_release\_date,Movie\_rating) values ('3','The conjuring','Horror','2013-08-02','7.5');  
insert into currently\_watching (Movie\_ID,movie\_title,Movie\_genre,Movie\_release\_date,Movie\_rating) values ('4','Scary Movie','Comedy','2013-08-02','6.3');  
insert into currently\_watching (Movie\_ID,movie\_title,Movie\_genre,Movie\_release\_date,Movie\_rating) values ('5','The karate kid','Teen drama','2010-07-11','6.2');  
insert into currently\_watching (Movie\_ID,movie\_title,Movie\_genre,Movie\_release\_date,Movie\_rating) values ('6','Kabhi khushi kabhi gham','Musical','2001-12-14','7.4');  
  
select max(Movie\_rating)  
from currently\_watching;  
  
select min(Movie\_rating)  
from currently\_watching;  
  
select avg(Movie\_rating) from currently\_watching;  
  
select \* from to\_watch;  
  
 select mid(Movie\_title,2,3) from currently\_watching;  
 select round(Movie\_rating,2) from to\_watch;  
   
 select Movie\_genre from to\_watch group by Movie\_genre having avg(Movie\_rating)>7.2;  
   
select sum(Movie\_rating) from currently\_watching;  
  
select movie\_id,now() as date\_and\_time from already\_watched;  
  
select Movie\_title,length(Movie\_title) from currently\_watching;

**Outputs:**

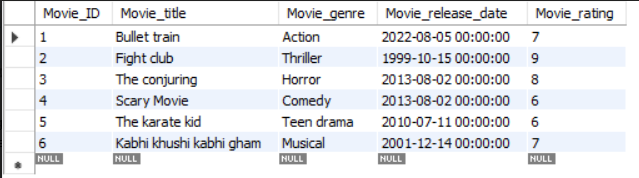
1. **Display to\_watch:**



1. **Display already\_watched:**

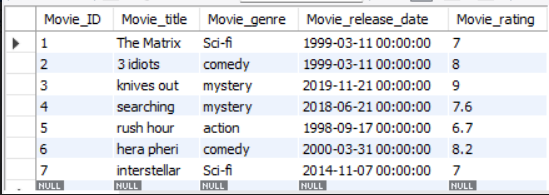
****

1. **Display currently\_watching:**

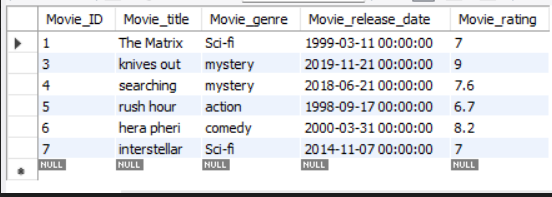


1. **Change movie rating to 7 where genre="Sci-fi":**

update to\_watch set movie\_rating=7 where movie\_genre='Sci-fi';  
**select \* from to\_watch;**

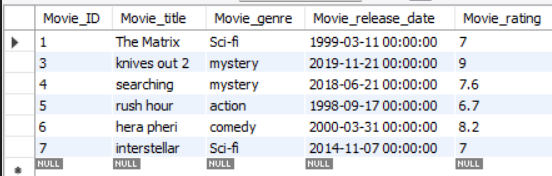
****

1. **Delete movie where movie\_rating="8":** delete from to\_watch where movie\_rating='8';

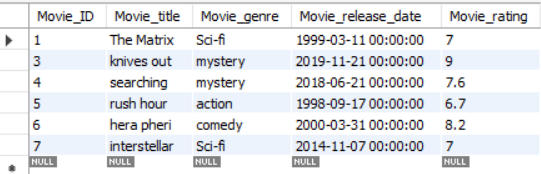


1. **Undo the changes using rollback:**

start transaction;  
update to\_watch set Movie\_title='knives out 2' where Movie\_ID='3';



rollback;  
select \* from to\_watch;

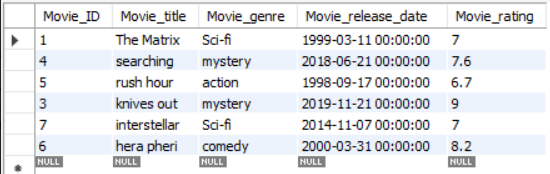


1. **Count the number of movies in to\_watch:**

select count(\*)as number\_of\_movies from to\_watch;

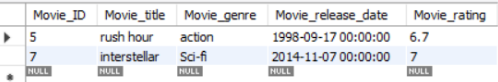


1. **Sort Movie title in descending  order:** select \* from to\_watch order by Movie\_title desc;

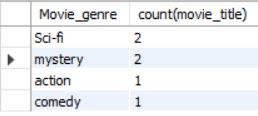


1. **Select movies from to\_watch where movie rating is greater than 9 and movie name ends with "r":**

select \* from to\_watch where Movie\_rating <9 and  Movie\_title like "%r";

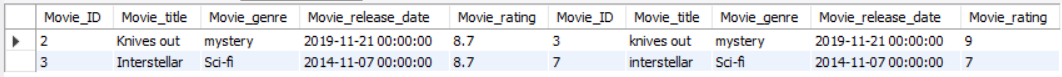


1. **Count movie\_title according to genre:** select Movie\_genre,count(movie\_title) from to\_watch group by Movie\_genre;



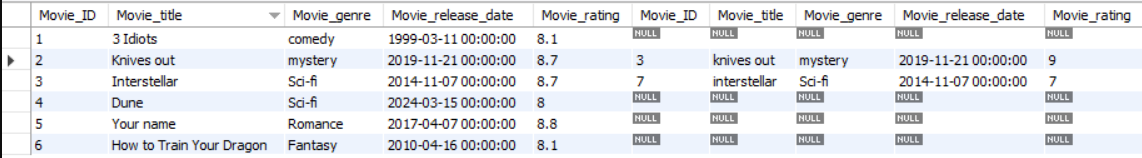
1. **Perform natural join:**

select \* from already\_watched inner join to\_watch on already\_watched.Movie\_title=to\_watch.Movie\_title;



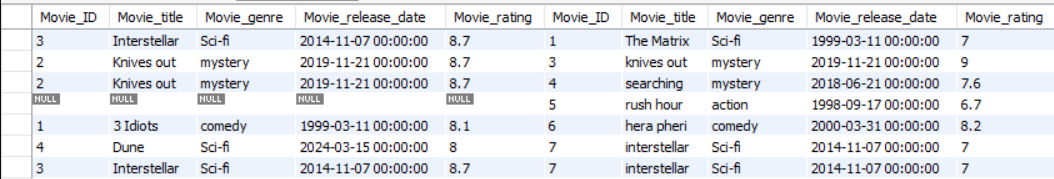
1. **Perform left join:**

select \* from already\_watched left join to\_watch on already\_watched.Movie\_title=to\_watch.Movie\_title;



1. **Perform right join:**

select \* from already\_watched right join to\_watch on already\_watched.Movie\_genre=to\_watch.Movie\_genre;



1. **Perform union operation:**

select \* from already\_watched union select \* from to\_watch;



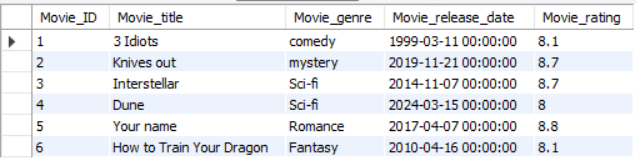
1. **Perform intersection operation:**

select \* from already\_watched intersect select \* from to\_watch;



1. **Perform set difference operation:**

select \* from already\_watched except  select \* from to\_watch;



1. **Select maximum rating from table:**

select max(Movie\_rating)  
from currently\_watching;



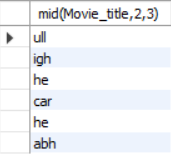
1. **Select minimum rating from table:**

select min(Movie\_rating)  
from currently\_watching;

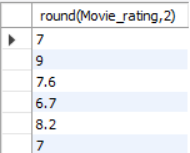


1. **Select a substring starting from 2nd position with a length of 3:**

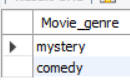
select mid(Movie\_title,2,3) from currently\_watching;



1. **Round Movie\_rating till 2 decimals:**



1. **Select genre from table where average rating is more than 7.2:**

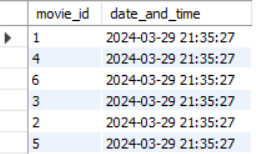


1. **Show the sum of movie\_ratings :**



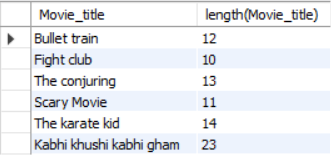
1. **Perform Now() function**

select movie\_id,now() as date\_and\_time from already\_watched;

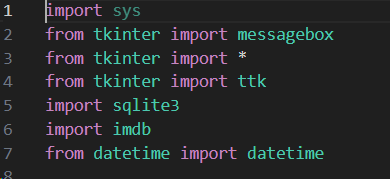
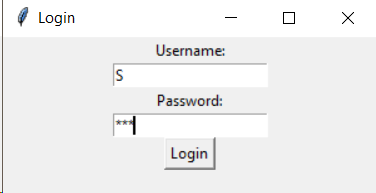


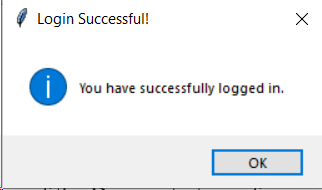
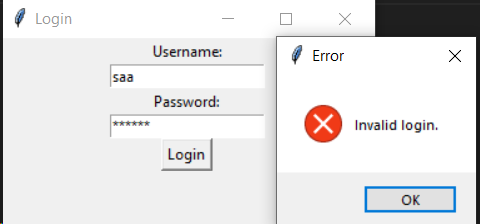
1. **Show length of movie titles:**

select Movie\_title,length(Movie\_title) from currently\_watching;

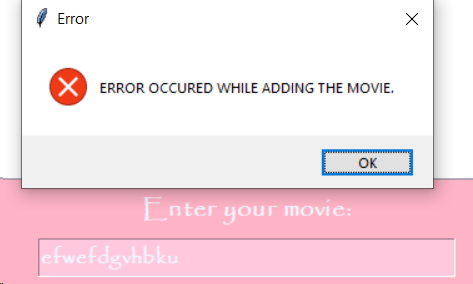


**VI. Project demonstration**





The project was made using Python(tkiner) and SQL.

Only after user’s username and password are correct login is successful and the movie tracker application will open. User can select from 3 lists-To\_watch,Already\_watched and currently\_watching using the drop down list. User can enter the title of a movie of their choice and click ‘Add’. If the movie exists in the IMDB database then it’s successfully added in the respective list along with it’s description. If the movie doesn’t exist in the IMDB database or if the system doesn’t have uninterrupted access to the IMDB database then an error is shown to the user. User can only add a movie once to a list. The user can select any movie from the list and click on the ‘Delete a movie’ button to delete that particular movie.

**VII. Self -Learning beyond classroom**

In working on the DBMS project and engaging with MySQL, I explored several new aspects beyond classroom teachings.

* Real-world Constraints: Working on a project outside of the controlled environment of a classroom exposed me to the complexities of the real-world scenarios.
* Transaction Management: Understanding how to ensure data consistency and integrity through proper transaction management was a valuable skill that I honed through hands-on experience.
* Advanced SQL Queries: While I was familiar with basic SQL queries, I expanded my knowledge by exploring more advanced query functionalities such as subqueries, joins, and aggregate functions,etc.
* Error Handling and Debugging: I learned to identify common errors, diagnose issues effectively which provided valuable insights into troubleshooting techniques.

Overall, working on the movie tracker project allowed me to apply and expand upon the theoretical knowledge gained in the classroom

**VIII. Learning from the Project**

* Working on the movie tracker project in MySQL has been a valuable learning experience that has significantly enhanced my database management skills.
* By creating the database schema, ER diagram, and implementing various SQL queries including, DDL commands, DML commands, transactions, aggregate functions, etc I gained hands-on experience in database design and manipulation.
* Solving real-world queries improved my problem-solving abilities and deepened my understanding of data retrieval and manipulation techniques. I learned to analyze requirements, identify potential issues, and devise effective solutions to ensure the efficiency and integrity of the database.
* Collaborating with team members on the project enhanced my communication and teamwork skills. We discussed database schema designs, resolved conflicts, and coordinated tasks to ensure the successful completion of the project.

**IX. Challenges Faced**

* Database Performance: Ensuring optimal database performance while handling a large volume of movie data, queries, and user interactions to maintain responsiveness and efficiency.
* Database Management: Handling database operations such as storing, retrieving, and updating movie data efficiently in MySQL while ensuring data integrity and performance.
* Safe Mode Limitations: Overcoming challenges related to safe mode restrictions that initially prevented updating or deleting movies, requiring adjustments to database configurations for full functionality.
* Compatibility Issues: Ensuring compatibility between the database management system (MySQL) and the chosen GUI framework or tool (Tkinter) was crucial. Ensuring seamless interaction and data exchange between the database and GUI components required careful consideration of compatibility issues.

**X. Conclusion**

* Practical application of database concepts.
* Enhanced problem-solving skills.
* Understanding of query optimization techniques.
* Improved collaboration and communication abilities.
* Proficiency in error handling and debugging.
* Development of project management skills.
* Emphasis on continuous learning and adaptation.

Overall, the project provided valuable hands-on experience and prepared me for challenges in database management and software engineering.