Movie Inventory System

**Team Members: Aayush Gupta (2017125), Aarush Talwar (2017213), Drishti Jain (2017148), Karamveer Singh (2017156), Sachin Nandal (2017185)**

Through this project, we aim to implement a DVD rental system modeled on a schema that consists of more than 10 tables- including but not limited to the movie inventory, the associated cast details, the employees, admin, investor and customers' checkouts. The relational database allows for easy rentals and employee management. Apart from that, it leverages payment handling with features like late fines, etc. It incorporates Machine Learning algorithms to recommend movies to the users based on previous rentals; all encompassed in a user-friendly ensemble that enables easy acquisitions and returns. To further enhance this, each employee of the store has limited access to the database which is superseded by the admin. We query the database to analyze how different customer bases employ the application and its impact on the store earnings and movie patterns.

**Stakeholders:**

A **user/buyer** can

1. Search for a movie by
   1. Title,
   2. release date,
   3. actors name,
   4. Genre
   5. Rating, etc
2. Rent a movie for a user-specified period and return it
3. Pay for the rental
4. Change his/her details like name, address, etc.

The **Owner/super admin** can view

1. Find the total gross
2. Can remove user/employee
3. Can remove a movie from database
4. Find movies which have been rented the most
5. Add new employees to the database
6. Change employee details from id to password

An **employee/admin** can

1. Remove a user
2. Search the movie (all filters)
3. Add a new movie to the system
4. Add a new user
5. View the rental records

A **DVD Distributor (investor)** can:

1. Can access which film category is the most popular
2. Get the total sale/profit.
3. Gauge which actor enjoys the highest viewership.
4. Access the database to find geographical viewership
5. Analyze different customer bases to deploy more resources

Key questions for each stakeholder:

**User:**

1. Which movies were released recently and added to the database?
2. Which movie should I watch next?
3. Which movies have received the highest rating?
4. Which movies have I watched?
5. How many movies have I watched in a certain span of time?
6. Do I have to pay any late fine?

**Owner:**

1. How much money has the store made?
2. How many employees are working at the store?
3. What is the salary of a certain employee?
4. How much profit the store has earned owing to late fine?
5. What kind of movies should be added to the system to make more money?
6. Which movies should be removed from the inventory owing to low values?

**Employee**:

1. Which users have watched the most movies?
2. How many users are registered?
3. How many users are overdue?
4. Which movies are being watched the most?
5. What genre is most popular amongst a certain base?

**Distributor**:

1. Is the owner making any profit?
2. What genre of DVDs should be sold here?
3. Which genre of DVDs should be pulled out?
4. Which dates are the busiest?o
5. Is the service confined locally?

**TABLES**

|  |  |
| --- | --- |
| **Buyer** | **Data type** |
| buyer\_id**(PK)** | int |
| first\_name | varchar(45) |
| last\_name | varchar(45) |
| Email | varchar(45) |
| Addr\_id (**FK)** | int |
| create\_date | DATE |

* Buyer\_id- primary key assigned to each customer
* First\_name-first name of every customer
* Last\_name-last name of the customer
* Email-the email associated with the customer
* Addr\_id- foreign key linked to the address table
* Create\_id- the date when the customer got a unique id in the database

|  |  |
| --- | --- |
| **movie\_category** | **Data type** |
| movie\_id **(FK)** | int |
| category\_id **(FK)** | int |

* Movie\_id- foreign key to link to the movie table
* Category\_id- foreign key to link to the category table

|  |  |
| --- | --- |
| **movie** | **Data Type** |
| movie\_id **(PK)** | int |
| title | varchar(45) |
| release\_year | int |
| rental\_duration | int |
| rental\_rate | int |
| length | int |
| rating | int |
| last\_update | timestamp |

* Movie\_id- Primary key assigned to each unique movie
* Title- the name of the movie
* Release\_year- the year in which the movie was released
* Rental\_duration- user specified days for which the movie is rented
* Rental\_rate- amt of the dvd rental for the above rental\_duration
* Length- the time (in min) for which the movie lasts
* Rating- the rating assigned to movies like PG-13,R,etc.
* Last\_update- the time stamp when a particular movie detail was last changed.

|  |  |
| --- | --- |
| **Category** | **Data type** |
| category\_id **(PK)** | int |
| Name | varchar(45) |

* Category\_id- primary key assigned to each genre
* Name-type of genre

|  |  |
| --- | --- |
| **movies\_actor** | **Data type** |
| actor\_id **(FK)** | int |
| movie\_id **(FK)** | int |

* Actor\_id- Foriegn key to link to the actor table
* Movie\_id- foreign key to link to the movie table

|  |  |
| --- | --- |
| **Rental** | **Data type** |
| rental\_id**(PK)** | int |
| Buyer\_id (**FK**) | int |
| employee\_id**(FK)** | int |
| movie\_id**(FK)** | int |
| rental\_date | date |
| return\_date | date |

* Rental\_id- primary key to identify each rental
* Buyer\_id- foriegn key linking to the buyer table
* Employee\_id-foreign key linking to the employee who is issuing the rental
* Movie\_id- foreign key linking to the movie which is being issued
* Rental\_date-the date on which the movie was issued for rental
* Return\_date- the date on which the movie was returned back

|  |  |
| --- | --- |
| **Employee** | **Data type** |
| employee\_id **(PK)** | int |
| first\_name | varchar(45) |
| last\_name | varchar(45) |
| Addr\_id (**FK**) | int |
| Email | varchar(45) |
| active | boolean |
| username | varchar(45) |
| password | varchar(45) |

* Employee\_id-primary key associated with each employee of the store
* First\_name- The first name of the employee
* last\_name- The last name of the employee.
* Addr\_id- foreign key linked to the address table
* Email-email address of the employee
* Active- boolean value to indicate if the employee has left or not
* Username- username of the employee to access the database
* Password- the password associated with the above username

|  |  |
| --- | --- |
| **Address** | **Data type** |
| addr\_id **(PK)** | int |
| Buyer\_id (**FK)** | int |
| employee\_id**(FK)** | int |
| address | varchar(150) |
| postal\_code | varchar(45) |
| phone | varchar(12) |

* Addr\_id- primary key assigned to each unique address
* Buyer\_id- foreign key to link to buyer
* Employee\_id- foreign key to link to the employee table
* Address-the address of the buyer/employee
* Postal\_code- the zip code of the address
* Phone- the phone no of the addressee

|  |  |
| --- | --- |
| **payments** | **Data type** |
| payment\_id **(PK)** | int |
| buyer\_id**(FK)** | int |
| employee\_id(**FK)** | int |
| rental\_id(**FK)** | int |
| movie\_id(**FK)** | int |
| amt | int |
| payment\_date | date |

* payment\_id- primary key associated with the payment of every rental
* Buyer\_id- foreign key linked to the buyer who is renting the movie as per the buyer table
* Employee\_id- foreign key associated with the employee table to indicate the employee who processed the payment
* Rental\_id-foreign key to identify the rental being made. It allows us to calculate the late fee, if any.
* Movie\_id- foreign key linking to the movie table and the movie being issued for rental/
* Amt- the total amount to be paid.
* payment\_date- the date on which the payment was made.

|  |  |
| --- | --- |
| **actor** | **Data type** |
| actor\_id **(PK)** | int |
| first\_name | varchar(45) |
| last\_name | varchar(45) |

* Actor\_id- Primary key assigned to each unique actor
* First\_name- The first name of each actor
* Last\_name- the last name of each actor

|  |  |
| --- | --- |
| **investor** | **Data type** |
| investor\_id **(PK)** | int |
| first\_name | varchar(45) |
| initial\_investment | int |
| last\_name | varchar(45) |
| email | varchar(50) |
| password | varchar(500) |
| Addr\_id **(FK)** | int |
| create\_date | timestamp |

* Investor\_id: unique id to identify each investor
* first\_name: name of investor
* initial\_investment: initial investment by the investor
* last\_name: name of investor
* email: mail ID of investor
* password: credential details of the investor
* addr\_id: foreign key linking to the address table
* create\_date: date on which investor signed up

|  |  |
| --- | --- |
| **investment** | **Data type** |
| investment\_id **(PK)** | int |
| Investor\_id **(FK)** | int |
| amount | decimal(50,2) |
| payment\_date | timestamp |

* invest\_id: unique id of each investment
* investor\_id: id of investor, foreign key to investor table
* amount: amount invested into the system
* Payment\_date: date on which investment was made

|  |  |
| --- | --- |
| **admin** | **Data type** |
| passwd | varchar(500) |

* passwd: credential details, hashed and stored.

Encompassing the work of week 6, the required indices and triggers have been added.

Views created allow the buyer to see the movie list with appropriate details. The staff has the ability to view the customer list and the sales/profit.

**VIEWS**

* buyer\_list- This provides a list of buyers, with first name and last name concatenated together and address information including the contact details combined into a single view. It incorporates data from the buyer and address tables.
* movie\_list- This contains a formatted view of the movie table, with a list of actors for each movie. It incorporates data from the movie, category, movie\_category, actor, and movie\_actor tables
* Sales\_by\_movie\_category- This provides a list of total sales, broken down by individual movie category. It incorporates data from the payment, rental, movie, movie\_category, and category tables.

**TRIGGERS**

* Buyer\_trigger\_ai- Sets the create\_date column to current time and date as rows are inserted.
* movie\_trigger\_ai- Sets the last\_update column to current time and date as rows are inserted.
* Movie\_trigger\_au- Sets the last\_update column to current time and date as rows are updated.
* payment\_trigger\_ai- Sets the payment\_date column to current time and date as rows are inserted.
* payment\_trigger\_au- Sets the last\_update column to current time and date as rows are updated.

**SQL Queries - Relational and Algebraic**

**Find the names and addresses of all employees**

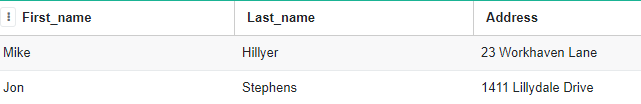
select emp.first\_name, emp.last\_name, addr.address

from employee emp

left join addr

on emp.addr\_id = addr.addr\_id;

employee.first\_name, employee.last\_name, addr.address employee ⟕ employee.addr\_id = addr.addr\_id addr



**Find the sum of transaction amounts of each employee in the year 2020**

select emp.first\_name, emp.last\_name, sum(trans.amount)

from employee emp

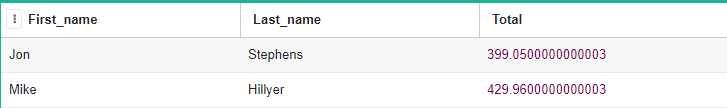
left join payment trans

on emp.employee\_id = trans.employee\_id

WHERE strftime('%Y',trans.payment\_date) == '2020'

group by emp.first\_name, emp.last\_name;

γ employee.first\_name, employee.last\_nameemployee.first\_name, employee.last\_name, sum(amount) σ year(payment.payment\_date) == 2005 (employee ⟕ employee.employee\_id = payment.employee\_id payment)



**Find all movies along with the number of actors listed in that movie, where the number of listed actors is > 7**

select mv.title, count(\*) number\_of\_actors

from movie mv

inner join movie\_actor mv\_act

on mv.movie\_id = mv\_act.movie\_id

group by mv.title having count(\*) > 7

order by number\_of\_actors desc;

τnum\_actors desc (σ num\_actors > 7 (γ movie.title σmovie.title, num\_actors<- count(\*) (movie ⟕movie.movie\_id = movie\_actor.movie\_id movie\_actor)))



**Find the highest paying customers, i.e. customers who have spent more than $125. Display their name along with the total amount paid**

select cust.first\_name, cust.last\_name, sum(trans.amount) 'Total Amount Paid'

from payment trans

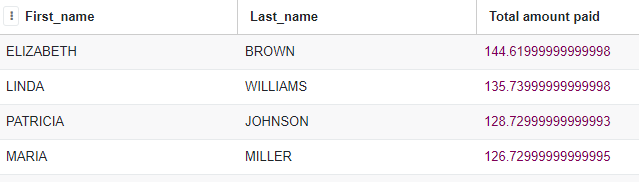
join buyer cust

on trans.buyer\_id = cust.buyer\_id

group by cust.first\_name, cust.last\_name HAVING SUM(trans.amount) > 125

order by sum(trans.amount) DESC;

τ Amount desc (σ Amount > 125 γ buyer.first\_name, buyer.last\_name σbuyer.first\_name, buyer.last\_name, Amount <- sum(payment.amount) (payment ⟕ payment.buyer\_id = buyer.buyer\_id buyer))



**Find all action movies**

select movie\_id, title

from movie

where movie\_id in (

select movie\_id

from movie\_category

where category\_id in (

select category\_id

from category

where name = 'Action'

)

);

ρ M (movie) ; ρ MC (movie\_category) ; ρ A (category)

π M.movie\_id, M.title σ C.name = ‘Action’ ^ C.category\_id = MC.category\_id ^ MC.movie\_id = M.movie\_id ’ (M x MC x C)



**Find First name and last name of the actors in the movie ‘Deer Virginian’**

select first\_name, last\_name

from actor

where actor\_id in (

select actor\_id

from movie\_actor

where movie\_id in (

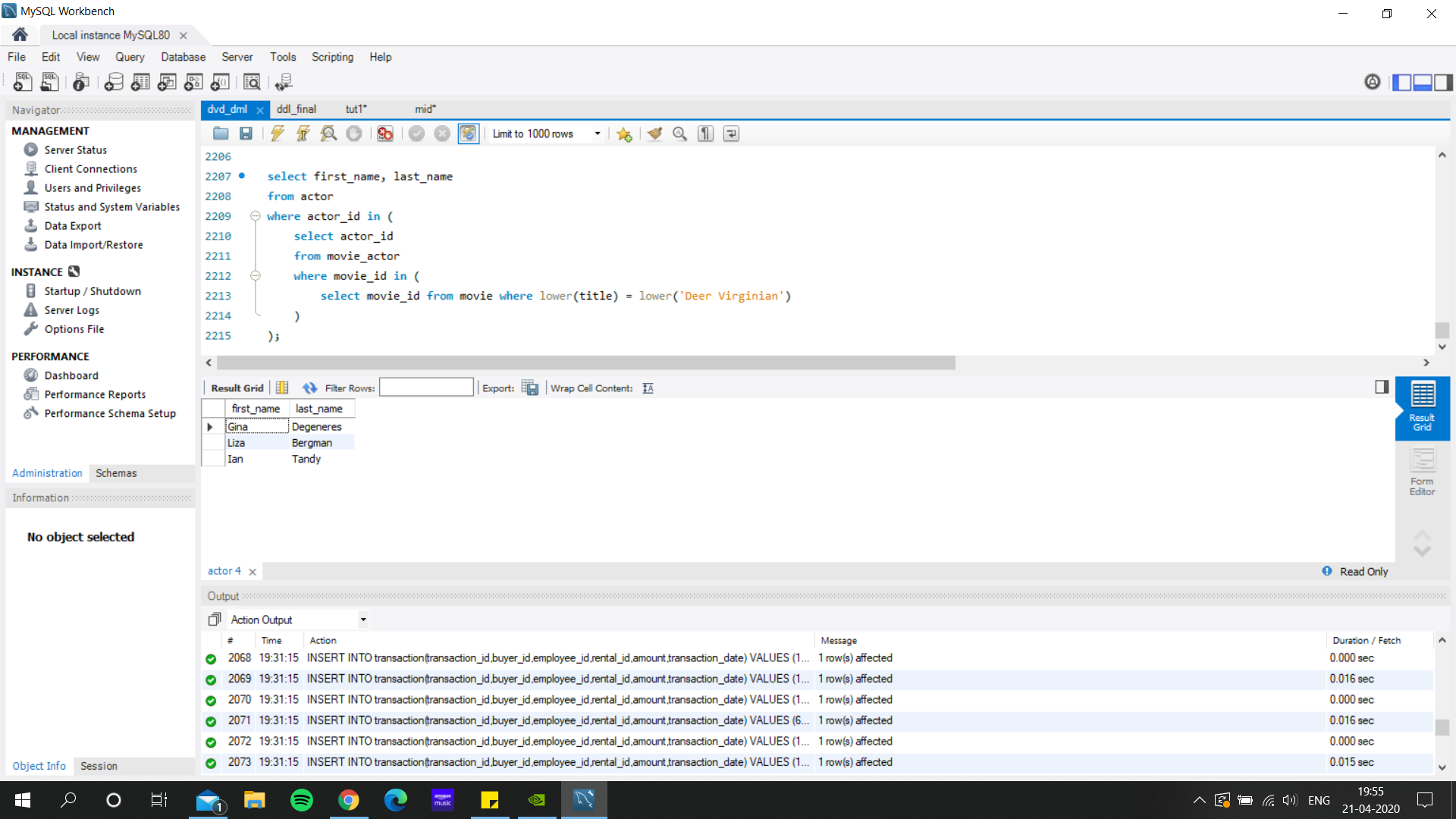
select movie\_id from movie where lower(title) = lower('Deer Virginian')

)

);

ρ M (movie) ; ρ MA (movie\_actor) ; ρ A (actor)

π A.first\_name, A.last\_name σ M.title = ‘Deer Virginian’ ^ M.movie\_id = MA.movie\_id ^ MA.actor\_id = A.actor\_id ’ (M x MA x A)

****

**Find the movie title and its release year having the actor “Alec Wayne”**

select title, release\_year

from movie

where movie\_id in (

select movie\_id

from movie\_actor

where actor\_id in (

select actor\_id

from actor

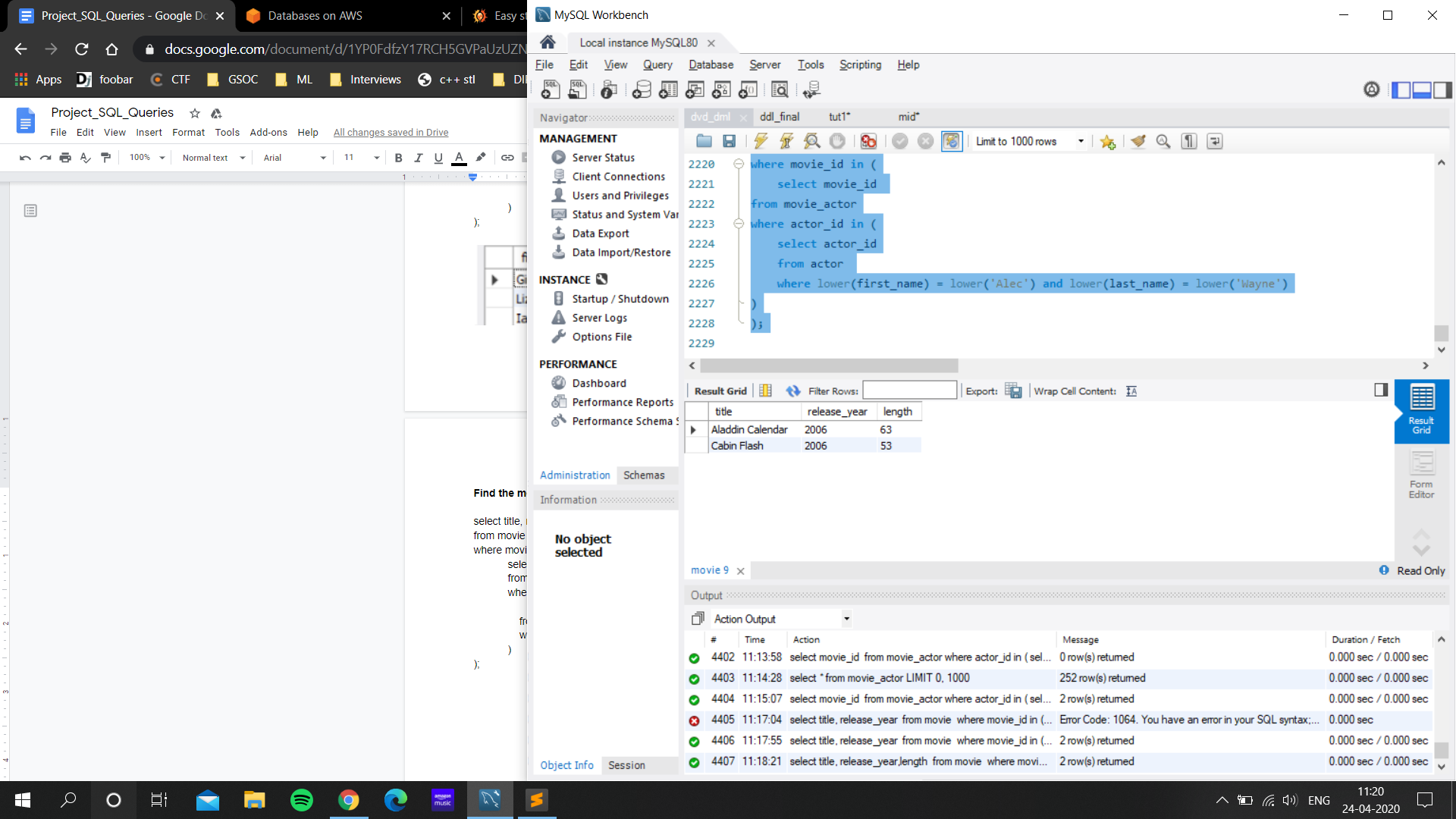
where lower(first\_name) = lower('Alec') and lower(last\_name) = lower('Wayne')

)

);

ρ M (movie) ; ρ MA (movie\_actor)

π M.title, M.release\_year σ M.movie\_id = MA.movie\_id ^ MA.first\_name = ‘Alec’ ^MA.last\_name = ‘Wayne’ (M x MA)



**Find the movie having length lesser than 60 mins i.e. 1 hour and view them in decreasing order.**

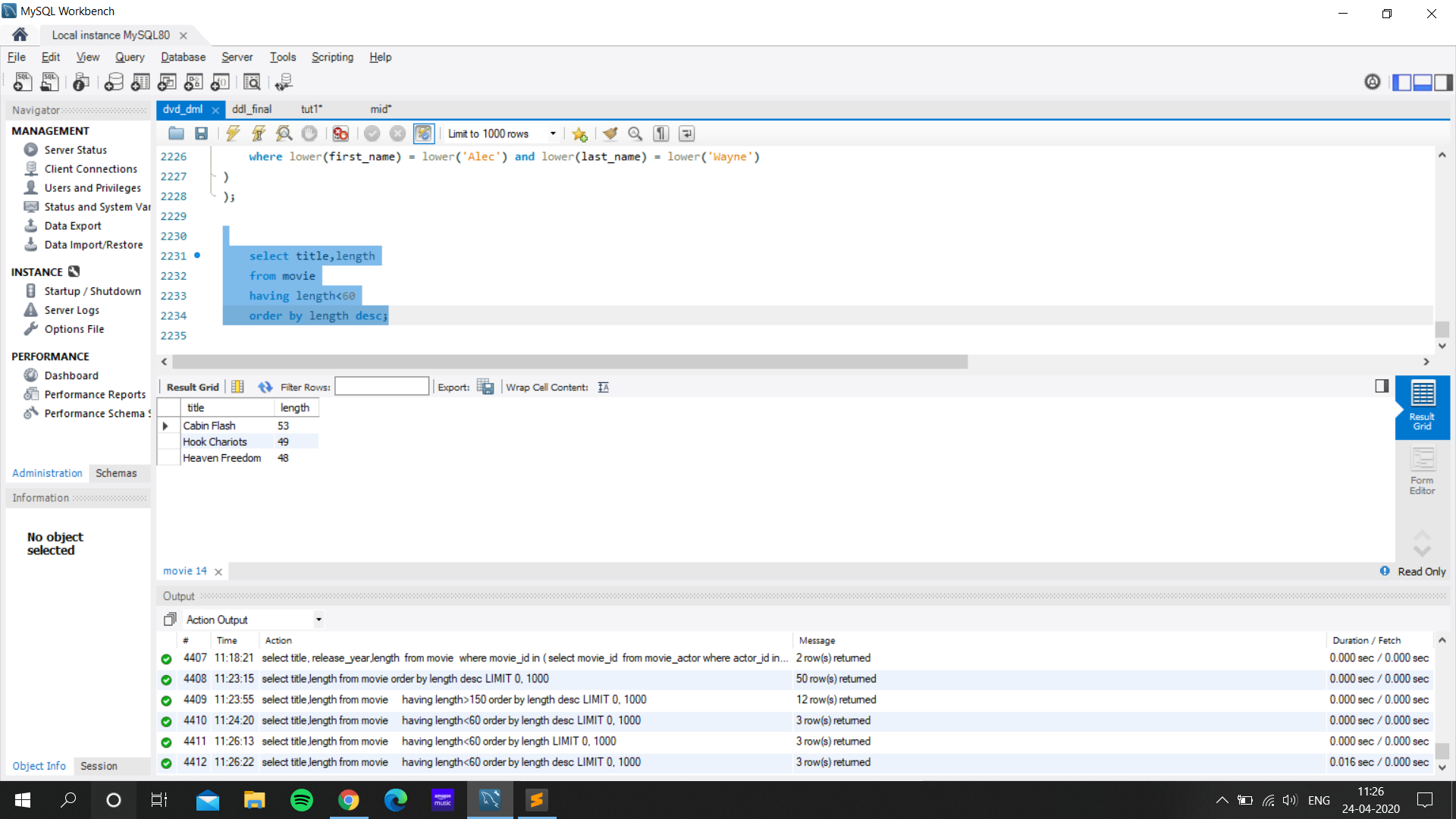
select title,length

from movie

Where length<60

order by length desc;

τ length desc π title, length σ length < 60 movie

****

**Find the first name, last name, and emails of the customers who bought movies on 24-05-2005**

select first\_name,last\_name,email

from buyer

where buyer\_id in (

select buyer\_id

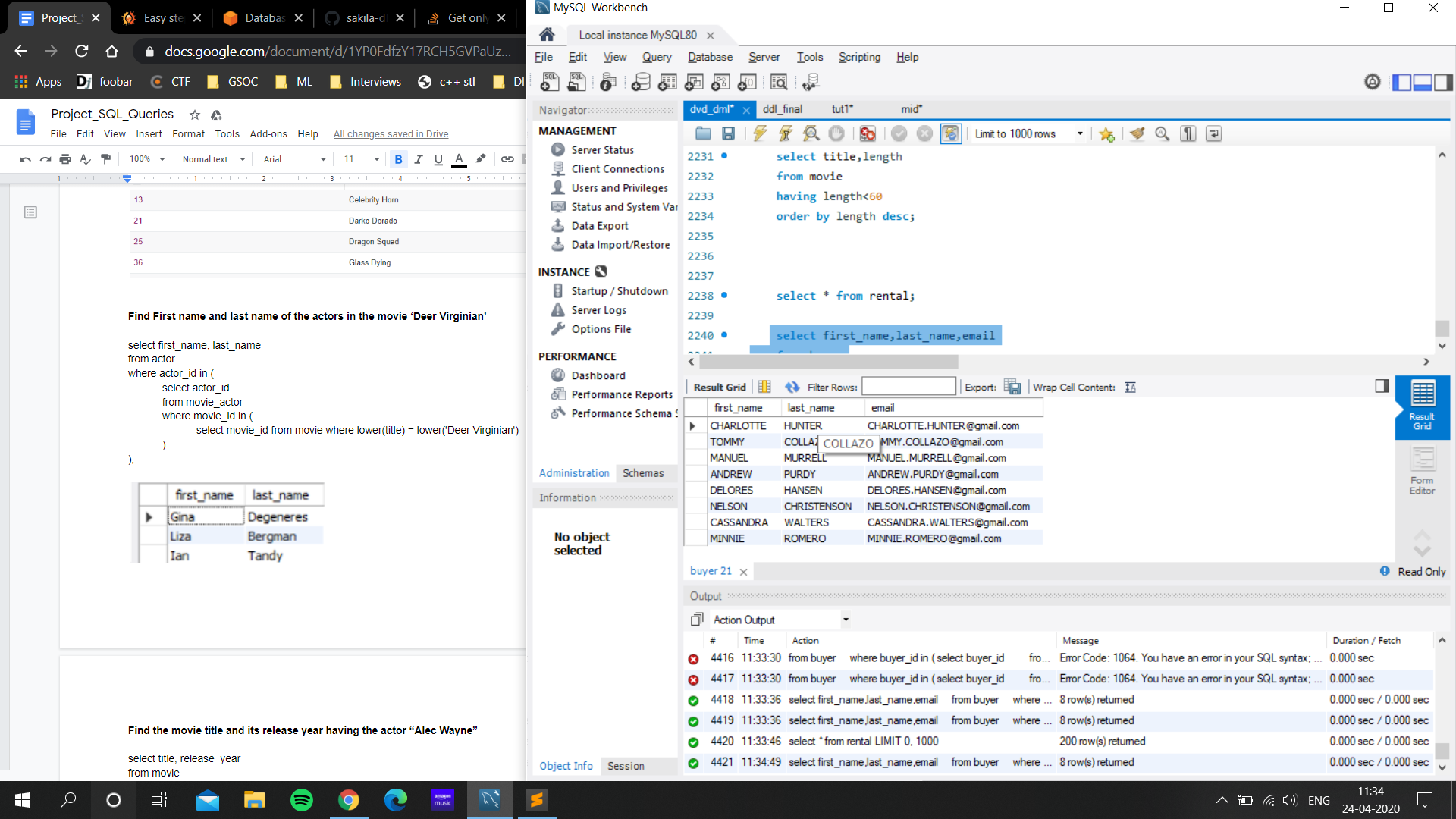
from rental

where date(rental\_date) = "2005-05-24"

);

ρ B1 (buyer) ; ρ R (Rental)

π B1.first\_name, B1.last\_name, B1.email σ R.buyer\_id = B1.buyer\_id ^ date(R.rental\_date) = ‘2005-05-24’ (B1 x R)

****

**Find the name of movies and number of times it has been rented till date**

select mv.title, count(\*) rented\_times

from movie mv

inner join rental r

on mv.movie\_id = r.movie\_id

group by r.movie\_id having count(\*) >0

order by rented\_times desc;

τ rented desc (σ rented > 0 γ movie.titleσ movie.title, rented <- count (\*) (movie ⨝movie.movie\_id = rental.movie\_id rental))

**Find movies with the same category of a person**

select \* from movie where movie\_id in (

select movie\_id from movie\_category where category\_id in (

select category\_id from movie\_category where movie\_id in(

select movie\_id

from rental

where buyer\_id in(

select buyer\_id from buyer where lower(email)= lower('MARIA.MILLER@gmail.com')

))

)

);

ρ B1 (buyer) ; ρ R (Rental) ; ρ M (movie) ; ρ MC (movie\_category)

π M.title (σ B1.email = ‘maria.miller@gmail.com’ ^ B1.buyer\_id = R.buyer\_id ^ R.movie\_id = MC.movie\_id ^ MC.movie\_id = M.movie\_id (B1 x R x M x MC)

**-- a query to find all orders of user id 1**

select title, movie\_id from movie where movie\_id in(

select movie\_id from rental where buyer\_id = 1

);

**-- a query to find all active orders for user id1**

select movie\_id, title, return\_date from movie, rental where movie.id = rental.id and buyr.id = 1 and return\_date>today;

**Display each movie name with its category name**

select mov.title, cat.name from movie mov left join movie\_category mcat on mov.movie\_id = mcat.movie\_id left join category cat on mcat.category\_id = cat.category\_id;

Following that week, we worked on sql queries and relational algebra as visible above.

Week 8 had us working on embedded SQL queries (present in accompanying folder). The language supported for the same is Python. It incorporates everything from insertion and updation to advanced queries involving different clauses.

The ER diagram (Crow’s feet notation) is visible below.

Apart from this, we have built a menu-driven app in Java (as explained below).

A **movie recommendation system** has also been implemented that uses the previously watched movies by the user and implements some machine learning algorithms to make the predictions. The data of previously watched movies is fed into the file named ml.py. Sklearn, Pandas and SQLAlchemy have been used in Python3 to implement the required functionality. We use the Random Forest Method which is an ensemble method for a generic classification problem. The data has been converted to categorical codes to improve the accuracy.

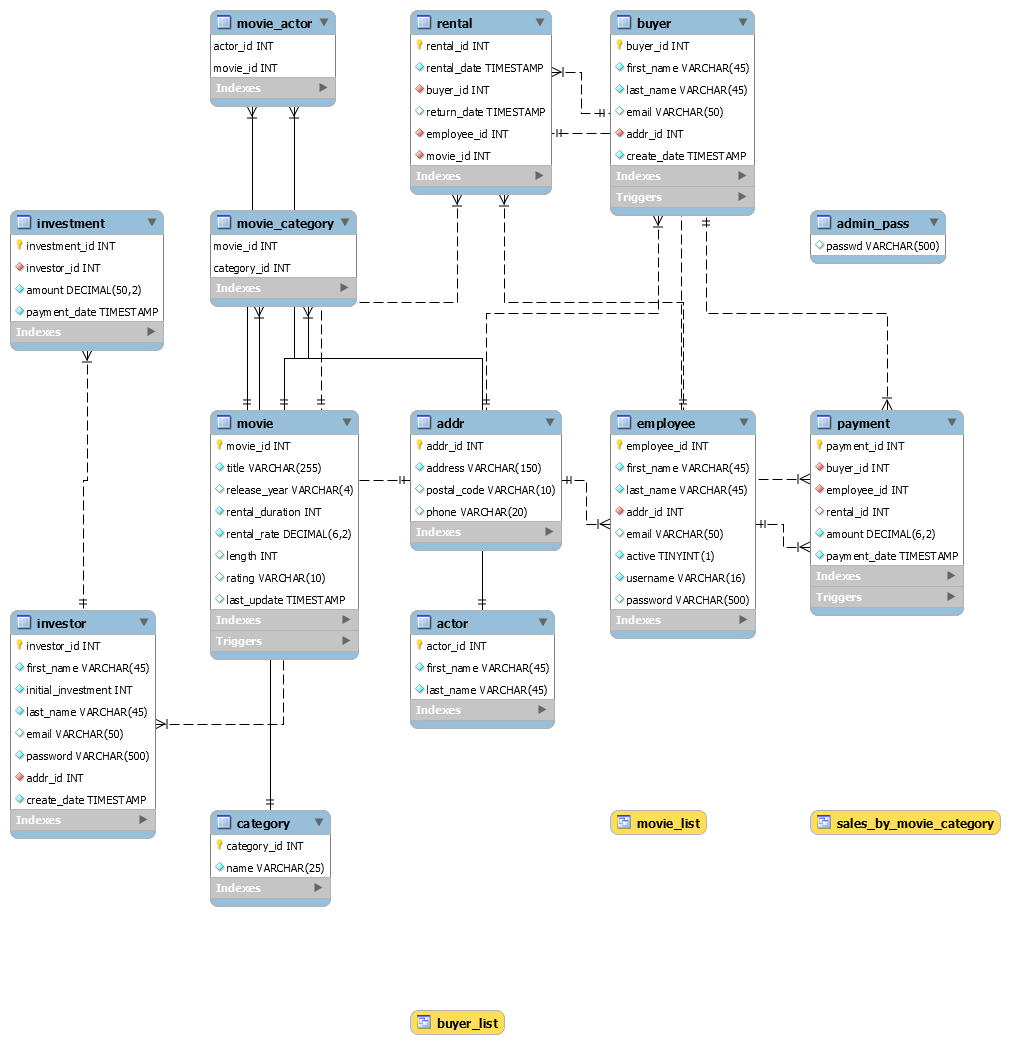
Also, we are using parameterized queries to prevent the failure/hacking of the database using **SQL Injection**.SQL injection is the placement of malicious code in SQL statements to either break into the database or to restrict it from functioning properly.

For a smooth functionality, we have implemented a **java menu driven** code where Admin/User/Employee/Investor can perform daily duties in a well defined manner.

Without loss of generality, we are assuming that only **ADMIN CAN ENTER A NEW INVESTOR OR EMPLOYEE and** can fetch important observations like movies trends or any actor in demand etc. We have made a panel where users can Login (if already registered) or signUP (if new user) and we’ll assign some id from which they can login. This ID is not changeable.

Storing passwords as plain text is a sin, so we’ve tried to **encode passwords using MD5** hashing tool and then storing them in our database. If an unauthorised person gains access to the database, he/she will not be able to fetch concrete information.

**ER DIAGRAM**

****

Note: The ER Diagram has been implemented via the Crow’s notation viz.

----- 1:1 (non-identifying relation)

-----<- 1:n (non-identifying relation)

\_\_\_\_\_\_ 1:1 (identifying relation)

-->----<-- n:m (non-identifying relation)

A straight line instead of dotted represents an identifying relation.

**Individual contribution**

* **Aayush Gupta- Creating the DDL/DML, Populating the tables, movie recommendation system and relational queries, Testing Java menu driven code**
* **Aarush Talwar- Populating the tables,relational and algebraic SQL queries, WIP**
* **Drishti Jain- Populating the tables, W-I-P, DDL, Indexes, creating views/triggers and writing queries**
* **Karamveer Singh- Populating the tables, integrating all components with Javas menu - driven code, Hashing, preventing SQL\_Injection, PL/SQL**
* **Sachin Nandal-Populating the tables, Creating the schema,relational and algebraic SQL queries, PL/SQL**

**Key Innovations:**

1. **Movie Recommendation System (Machine Learning)**
2. **Preventing SQL Injection**
3. **Hashing to secure useful information**