Movie Inventory System

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 To further enhance this, each employee of the store has limited access to the database which is superseded by the admin.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
    a. Title,
    b. release date,
    c. actors name,
    d. Genre
    e. 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.
Later phase of the project had us working on embedded SQL queries. 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).

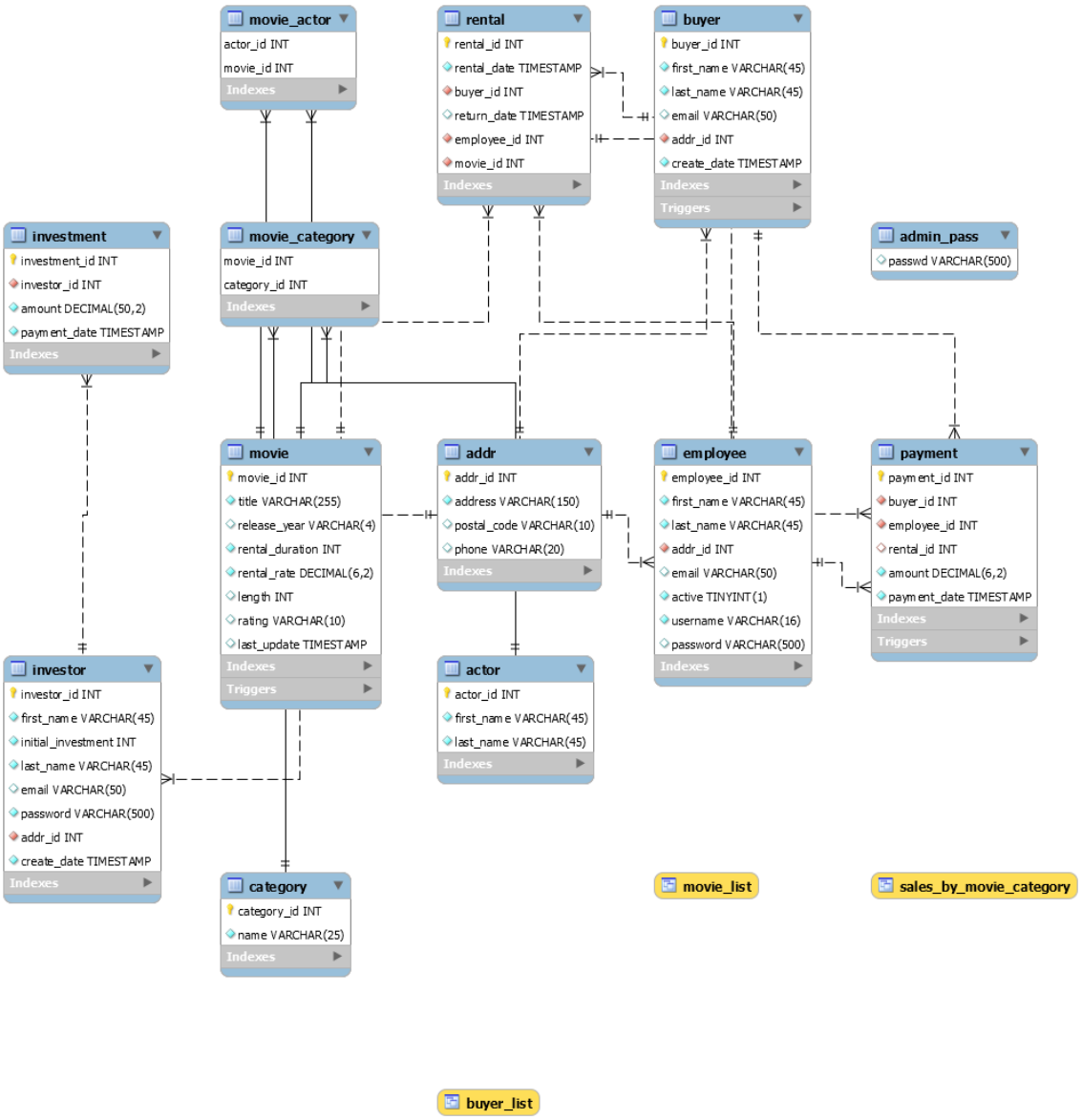
A **movie recommendation system** has also been implemented that uses the previously watched movies by the user and implements some machine learning algorithms to make the predictions. The data of previously watched movies is fed into the file named ml.py. Sklearn, Pandas and SQLAlchemy have been used 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.

Key Innovations:

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