DATABASE MANGAEMNET INTERN

AS A PART OF

Brainwave
MATRIX SOLUTIONS

PRESENTING TASK 2

Submitted By:

Shafak

Date: 14 Jan. 25

Develop a database for entity movie rental. This project involves more queries and database design. Write SQL queries to handle customer orders.

Designing and implementing a relational database for a **Movie Rental System** involves several steps, including defining the requirements, designing the schema, and implementing the database. Below is a comprehensive guide to help you through this process:

Step 1: Requirements Gathering

Before designing the database, we need to identify the key entities, their attributes, and the relationships between them. Here are the common requirements for a Movie Rental System:

Key Entities and Their Attributes

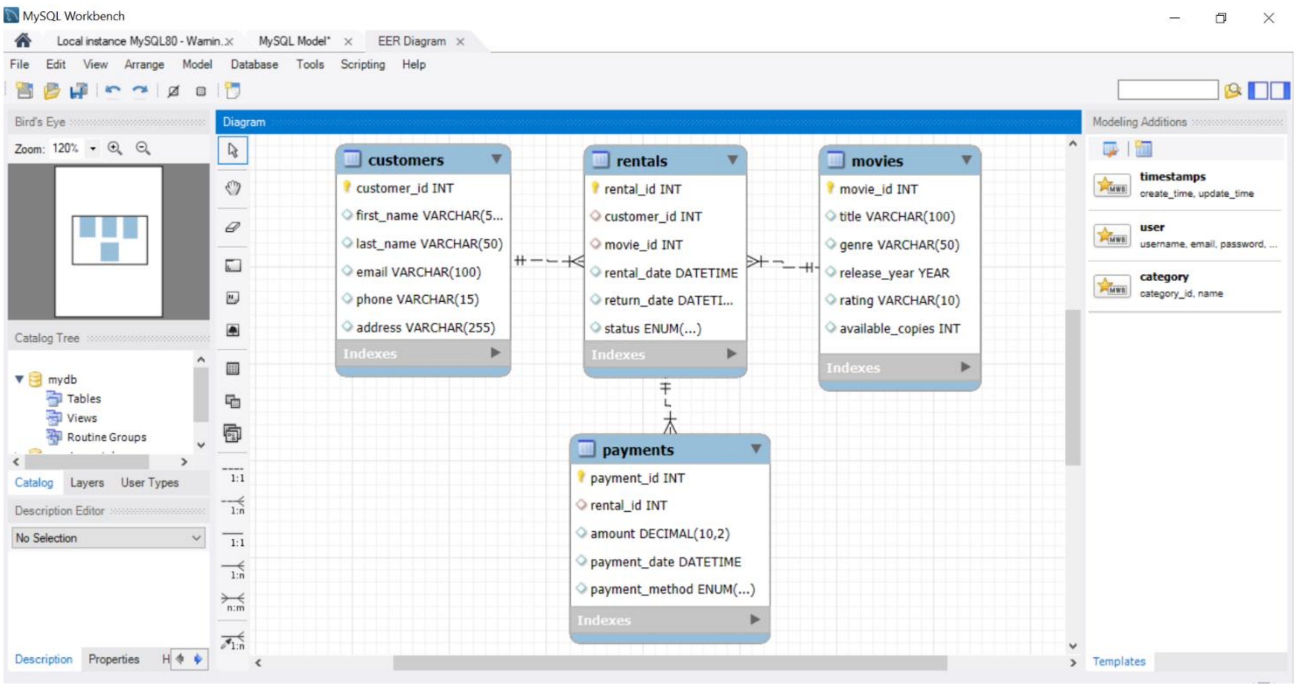
1. **Movies**
    - **Attributes:**
        - **movie_id:** Unique identifier for each movie (Primary Key)
        - **title:** Title of the movie
        - **genre:** Genre of the movie (e.g., Action, Comedy, Drama)
        - **release_year:** Year the movie was released
        - **rating:** Rating of the movie (e.g., PG, PG-13, R)
        - **available_copies:** Number of copies available for rent
        - **duration:** Duration of the movie (in minutes)
        - **description:** Brief description or synopsis of the movie
2. **Customers**
    - **Attributes:**
        - **customer_id:** Unique identifier for each customer (Primary Key)
        - **first_name:** First name of the customer
        - **last_name**: Last name of the customer
        - **email:** Email address of the customer (Unique)
        - **phone:** Contact number of the customer
        - **address:** Residential address of the customer
        - **membership_status:** Status of the customer's membership (e.g., Active, Inactive)
3. **Rentals**
    - **Attributes:**

        - **rental_id:** Unique identifier for each rental transaction (Primary Key)
        - **customer_id:** Identifier for the customer who rented the movie (Foreign Key)
        - **movie_id:** Identifier for the rented movie (Foreign Key)
        - **rental_date:** Date and time when the movie was rented
        - **return_date:** Date and time when the movie was returned
        - **due_date:** Date by which the movie should be returned
        - **status:** Status of the rental (e.g., Rented, Returned, Overdue)

4. **Payments**
    - **Attributes:**
        - **payment_id:** Unique identifier for each payment transaction (Primary Key)
        - **rental_id:** Identifier for the rental associated with the payment (Foreign Key)
        - **amount:** Amount paid for the rental
        - **payment_date:** Date and time when the payment was made
        - **payment_method:** Method of payment (e.g., Credit Card, PayPal, Cash)
5. **Categories**
    - **Attributes:**
        - **category_id:** Unique identifier for each category (Primary Key)
        - **category_name:** Name of the category (e.g., Action, Comedy, Drama)
6. **Movie_Categories (Join Table)**
    - **Attributes:**
        - **movie_id:** Identifier for the movie (Foreign Key)
        - **category_id:** Identifier for the category (Foreign Key)

Relationships

1. **Movies and Categories**
    - A movie can belong to multiple categories, and a category can have multiple movies. This is a many-to-many relationship, which will be implemented using a join table called Movie_Categories.
2. **Customers and Rentals**
    - A customer can rent multiple movies, but each rental is associated with only one customer. This is a one-to-many relationship.
3. **Movies and Rentals**
    - A movie can be rented multiple times, but each rental is associated with only one movie. This is a one-to-many relationship.
4. **Rentals and Payments**
    - Each rental can have one payment associated with it, but a payment is linked to only one rental. This is a one-to-one relationship.
5. **Movies and Actors (Optional)**
    - If we want to include actors, we can add an Actors table with a many-to-many relationship to Movies.



customers
customer_id INT
first_name VARCHAR(5...
last_name VARCHAR(50)
email VARCHAR(100)
phone VARCHAR(15)
address VARCHAR(255)
Indexes

rentals
rental_id INT
customer_id INT
movie_id INT
rental_date DATETIME
return_date DATETI...
status ENUM(...)
Indexes

movies
movie_id INT
title VARCHAR(100)
genre VARCHAR(50)
release_year YEAR
rating VARCHAR(10)
available_copies INT
Indexes

payments
payment_id INT
rental_id INT
amount DECIMAL(10,2)
payment_date DATETIME
payment_method ENUM(...)
Indexes

Step 2: Database Schema Design

Based on the requirements gathered, we can design the database schema. Below is the SQL code to create the necessary tables:

1. **Create the Database**
```
CREATE DATABASE movie_rental;
USE movie_rental;
```
2. **Create the Tables**
```
CREATE TABLE Customers (
    customer_id INT AUTO_INCREMENT PRIMARY KEY,
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    email VARCHAR(100) UNIQUE,
    phone VARCHAR(15),
    address VARCHAR(255)
    membership_status ENUM('Active', 'Inactive')
);

CREATE TABLE Movies (
    movie_id INT AUTO_INCREMENT PRIMARY KEY,
    title VARCHAR(100),
    genre VARCHAR(50),
    release_year YEAR,
    rating VARCHAR(10),
    available_copies INT
);

CREATE TABLE Rentals (
    rental_id INT AUTO_INCREMENT PRIMARY KEY,
    customer_id INT,
    movie_id INT,
    rental_date DATETIME,
    return_date DATETIME,
    status ENUM('rented', 'returned'),
    FOREIGN KEY (customer_id) REFERENCES Customers(customer_id),
    FOREIGN KEY (movie_id) REFERENCES Movies(movie_id)
);


CREATE TABLE Payments (
    payment_id INT AUTO_INCREMENT PRIMARY KEY,
    rental_id INT,
    amount DECIMAL(10, 2),
    payment_date DATETIME,
    payment_method ENUM('credit_card', 'paypal', 'cash'),
    FOREIGN KEY (rental_id) REFERENCES Rentals(rental_id)
);
```

3. **Insert Sample Data**

Insert Customers

```
INSERT INTO Customers (first_name, last_name, email, phone, address) VALUES
('John', 'Doe', 'john.doe@example.com', '1234567890', '123 Elm St'),
('Jane', 'Smith', 'jane.smith@example.com', '0987654321', '456 Oak St');
```

Insert Movies

```
INSERT INTO Movies (title, genre, release_year, rating, available_copies) VALUES
('Inception', 'Sci-Fi', 2010, 'PG-13', 5),
('The Godfather', 'Crime', 1972, 'R', 3),
('The Dark Knight', 'Action', 2008, 'PG-13', 4);
```

Insert Rentals

```
INSERT INTO Rentals (customer_id, movie_id, rental_date, status) VALUES
(1, 1, NOW(), 'rented'),
(2, 2, NOW(), 'rented');
```

Insert Payments

```
INSERT INTO Payments (rental_id, amount, payment_date, payment_method)
VALUES
(1, 4.99, NOW(), 'credit_card'),
(2, 3.99, NOW(), 'paypal');
```

Step 3. Queries to Handle Customer Orders

1. **Get All Movies Available for Rent**

   SELECT * FROM Movies WHERE available_copies > 0;

2. **Rent a Movie**

   Assuming customer_id and movie_id are provided

   SET @customer_id = 1;

   SET @movie_id = 1;

   Check if the movie is available

   SELECT available_copies FROM Movies WHERE movie_id = @movie_id;

   If available, insert rental and update available copies

   INSERT INTO Rentals (customer_id, movie_id, rental_date, status) VALUES

   (@customer_id, @movie_id, NOW(), 'rented');

   UPDATE Movies SET available_copies = available_copies - 1 WHERE movie_id = @movie_id;

3. **Update Customer Information**

   UPDATE Customers

   SET phone = '555-5678', address = '101 Maple St'

   WHERE customer_id = 1; **-- Assuming customer_id 1 is the customer to update**

4. **Retrieve All Customers**

   SELECT * FROM Customers;

5. **Retrieve a Specific Customer by ID**

   SELECT * FROM Customers

   WHERE customer_id = 1; -- Replace with the desired customer_id

6. **Retrieve Customers with Active Membership**

   SELECT * FROM Customers

   WHERE membership_status = 'Active';

7. **Retrieve Customer Count by Membership Status**

```
SELECT membership_status, COUNT(*) AS count
FROM Customers
GROUP BY membership_status;
```