Movie Rental SQL Project

README.md

````
#  Movie Rental SQL Project

This is a full SQL database project for a movie rental store. It simulates operations similar to Blockbuster or Netflix DVD services.

##  Database Structure

### Tables:
- **Customers**: Stores customer info.
- **Movies**: Stores details about each movie.
- **Inventory**: Tracks available quantity for each movie.
- **Rentals**: Manages rentals and returns.
- **Payments**: Stores payment information.

##  ERD
(Use a tool like dbdiagram.io to create and add an ERD image here.)

##  Folder Structure
```
movie-rental-sql/
 schema.sql
 data.sql
 queries.sql
 views.sql
 README.md
```

##  How to Use
1. Create a database in MySQL or PostgreSQL.
2. Run `schema.sql` to create tables.
3. Run `data.sql` to populate data.
4. Run `queries.sql` to execute queries.
5. Run `views.sql` to create views.

##  Sample Queries
- Top 5 most rented movies.
- Movies currently rented by a customer.
- Revenue generated per month.
- Late returns.

##  Author
This project is created for portfolio/resume use.
````

schema.sql

```
CREATE TABLE Customers (
    CustomerID INT PRIMARY KEY,
    Name VARCHAR(100),
```

```
    Email VARCHAR(100),
    Phone VARCHAR(15),
    Address TEXT
);

CREATE TABLE Movies (
    MovieID INT PRIMARY KEY,
    Title VARCHAR(100),
    Genre VARCHAR(50),
    ReleaseYear INT,
    Rating VARCHAR(10)
);

CREATE TABLE Inventory (
    InventoryID INT PRIMARY KEY,
    MovieID INT,
    QuantityAvailable INT,
    FOREIGN KEY (MovieID) REFERENCES Movies(MovieID)
);

CREATE TABLE Rentals (
    RentalID INT PRIMARY KEY,
    CustomerID INT,
    MovieID INT,
    RentalDate DATE,
    ReturnDate DATE,
    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID),
    FOREIGN KEY (MovieID) REFERENCES Movies(MovieID)
);

CREATE TABLE Payments (
    PaymentID INT PRIMARY KEY,
    RentalID INT,
    Amount DECIMAL(5,2),
    PaymentDate DATE,
    FOREIGN KEY (RentalID) REFERENCES Rentals(RentalID)
);
```

data.sql

```
INSERT INTO Customers VALUES (1, 'Alice Smith', 'alice@example.com', '1234567890', '123
Main St');
INSERT INTO Customers VALUES (2, 'Bob Johnson', 'bob@example.com', '0987654321', '456
Oak St');

INSERT INTO Movies VALUES (1, 'The Matrix', 'Sci-Fi', 1999, 'R');
INSERT INTO Movies VALUES (2, 'Titanic', 'Drama', 1997, 'PG-13');

INSERT INTO Inventory VALUES (1, 1, 3);
INSERT INTO Inventory VALUES (2, 2, 2);

INSERT INTO Rentals VALUES (1, 1, 1, '2025-07-01', NULL);
INSERT INTO Rentals VALUES (2, 2, 2, '2025-06-20', '2025-06-25');
```

```
INSERT INTO Payments VALUES (1, 1, 4.99, '2025-07-01');
INSERT INTO Payments VALUES (2, 2, 3.99, '2025-06-20');
```

queries.sql

```
-- List all movies by genre
SELECT Genre, Title FROM Movies ORDER BY Genre;

-- Top 5 most rented movies
SELECT MovieID, COUNT(*) AS RentalCount FROM Rentals GROUP BY MovieID ORDER BY RentalCount DESC LIMIT 5;

-- Current rentals by customer
SELECT c.Name, m.Title, r.RentalDate FROM Rentals r
JOIN Customers c ON r.CustomerID = c.CustomerID
JOIN Movies m ON r.MovieID = m.MovieID
WHERE r.ReturnDate IS NULL;

-- Late returns
SELECT * FROM Rentals WHERE ReturnDate IS NULL AND RentalDate < CURDATE() - INTERVAL 7 DAY;

-- Revenue for July 2025
SELECT SUM(Amount) FROM Payments WHERE MONTH(PaymentDate) = 7 AND YEAR(PaymentDate) = 2025;
```

views.sql

```
-- View for available movies
CREATE VIEW AvailableMovies AS
SELECT m.Title, i.QuantityAvailable
FROM Movies m JOIN Inventory i ON m.MovieID = i.MovieID;

-- View for active rentals
CREATE VIEW ActiveRentals AS
SELECT c.Name, m.Title, r.RentalDate
FROM Rentals r
JOIN Customers c ON r.CustomerID = c.CustomerID
JOIN Movies m ON r.MovieID = m.MovieID
WHERE r.ReturnDate IS NULL;
```