CSE 305 Project Assignment 2

Team: Trappist-1f
Members:
Zhe Lin, 109369879, zhe.lin@stonybrook.edu
Sean Pesce, 107102508, sean.pesce@stonybrook.edu
Weichao Zhao, 109957656, weichao.zhao@stonybrook.edu

1. List of tables with the initial Demo Data loaded:

Customers (Obtained by joining Person table and Customer table)

```
mysql> SELECT * FROM Person JOIN Customer ON (Person.ID=Customer.ID);
+-----------+----------+-----------+-----------------+-------------+-------+-------+------------+-----------+--------------------+-----------------+--------+
| ID        | LastName | FirstName | Address         | City        | State | Zip   | Phone      | ID        | Email              | CreditCard      | Rating |
+-----------+----------+-----------+-----------------+-------------+-------+-------+------------+-----------+--------------------+-----------------+--------+
| 111111111 | Yang     | Shang     | 123 Success Street | Stony Brook | NY    | 11790 | 5166328959 | 111111111 | syang@cs.sunysb.edu | 1234567812345678 |      1 |
| 222222222 | Du       | Victor    | 456 Fortune Road | Stony Brook | NY    | 11790 | 5166324360 | 222222222 | vicdu@cs.sunysb.edu | 5678123456781234 |      1 |
| 333333333 | Smith    | John      | 789 Peace Blvd. | Los Angeles | CA    | 93536 | 3154434321 | 333333333 | jsmith@ic.sunysb.edu | 2345678923456789 |      1 |
| 444444444 | Philip   | Lewis     | 135 Knowledge Lane | Stony Brook | NY    | 11794 | 5166668888 | 444444444 | pml@cs.sunysb.edu | 6789234567892345 |      1 |
+-----------+----------+-----------+-----------------+-------------+-------+-------+------------+-----------+--------------------+-----------------+--------+
4 rows in set (0.00 sec)
```

Accounts (Obtained from Account table)

```
mysql> SELECT * FROM Account;
+----+------------+--------------+------------+
| ID | CustomerID | Subscription | Created    |
+----+------------+--------------+------------+
|  1 |  444444444 | Unlimited+   | 2006-10-01 |
|  2 |  222222222 | Limited      | 2006-10-15 |
+----+------------+--------------+------------+
2 rows in set (0.00 sec)
```

Actors (Obtained by joining Actor table and Casted table)

```
mysql> SELECT * FROM Actor JOIN Casted ON (ID = ActorID) ORDER BY ActorID;
+----+-----------+----------+--------+------+--------+---------+---------+------+
| ID | FirstName | LastName | Gender | Age  | Rating | ActorID | MovieID | Role |
+----+-----------+----------+--------+------+--------+---------+---------+------+
|  1 | Al        | Pacino   | M      |   63 |      5 |       1 |       1 | NULL |
|  1 | Al        | Pacino   | M      |   63 |      5 |       1 |       3 | NULL |
|  2 | Tim       | Robbins  | M      |   53 |      2 |       2 |       1 | NULL |
+----+-----------+----------+--------+------+--------+---------+---------+------+
3 rows in set (0.00 sec)
```

Employees (Obtained by joining Person table and Employee table)

```
mysql> SELECT * FROM Person JOIN Employee ON (Person.ID=Employee.SSN);
+-----------+----------+-----------+-----------------+-------------+-------+-------+------------+-----------+-----------+-------------+------------+
| ID        | LastName | FirstName | Address         | City        | State | Zip   | Phone      | SSN       | StartDate | Position    | HourlyRate |
+-----------+----------+-----------+-----------------+-------------+-------+-------+------------+-----------+-----------+-------------+------------+
| 123456789 | Smith    | David     | 123 College road | Stony Brook | NY    | 11790 | 5162152345 | 123456789 | 2005-11-01 | Customer Rep |         60 |
| 789123456 | Warren   | David     | 456 Sunken Street | Stony Brook | NY    | 11794 | 6316329987 | 789123456 | 2006-02-02 | Manager     |         50 |
+-----------+----------+-----------+-----------------+-------------+-------+-------+------------+-----------+-----------+-------------+------------+
2 rows in set (0.00 sec)
```

Movies (Obtained from Movie table)

```
mysql> SELECT * FROM Movie;
+----+-----------------------+---------+-------+-------------+--------+
| ID | Title                 | Genre   | Fee   | TotalCopies | Rating |
+----+-----------------------+---------+-------+-------------+--------+
|  1 | The Godfather         | Drama   | 10000 |           3 |      5 |
|  2 | Shawshank Redemption  | Drama   |  1000 |           2 |      4 |
|  3 | Borat                 | Comedy  |   500 |           1 |      3 |
+----+-----------------------+---------+-------+-------------+--------+
3 rows in set (0.00 sec)
```

Orders (Obtained by joining Rental table and _Order table)

```
mysql> SELECT * FROM Rental JOIN _Order ON (ID=OrderID) ORDER BY OrderID;
+---------+-----------+---------+------------+----+---------------------+---------------------+
| OrderID | AccountID | MovieID | EmployeeID | ID | OrderDate           | ReturnDate          |
+---------+-----------+---------+------------+----+---------------------+---------------------+
|       1 |         1 |       1 |  123456789 |  1 | 2009-11-11 10:00:00 | 2009-11-14 00:00:00 |
|       2 |         2 |       3 |  123456789 |  2 | 2009-11-11 18:15:00 | NULL                |
|       3 |         1 |       3 |  123456789 |  3 | 2009-11-12 09:30:00 | NULL                |
|       4 |         2 |       2 |  123456789 |  4 | 2009-11-21 22:22:00 | NULL                |
+---------+-----------+---------+------------+----+---------------------+---------------------+
4 rows in set (0.00 sec)
```

2. Transactions

2.1 Manager-Level Transactions

a. Add/Edit movie

In-Parameters:

new_Title: Its type is VARCHAR, and it's the title of the movie added to the database.

new_Genre: Its type is ENUM ('Comedy', 'Drama', 'Action', 'Foreign'), and it's the genre of the movie added to the database.
new_Fee: Its type is FLOAT, and it's the fee of the movie added to the database.
new_TotalCopies: Its type is INT UNSIGNED, and it's the number of total copies of the movie added to the database.

SQL Code:

```
DELIMITER $$
CREATE PROCEDURE AddMovie (IN new_Title VARCHAR(64), new_Genre ENUM
('Comedy', 'Drama', 'Action', 'Foreign'), new_Fee FLOAT, new_TotalCopies INT UNSIGNED)
BEGIN
        START TRANSACTION;
      INSERT INTO Movie (Title, Genre, Fee, TotalCopies)
      VALUES (new_Title, new_Genre, new_Fee, new_TotalCopies);
  COMMIT;
END;
$$
DELIMITER ;
```

Sample Output:

```
mysql> CALL AddMovie('TestMovie', 'Drama', 1.00, 5);
Query OK, 0 rows affected (0.07 sec)

mysql> SELECT * FROM Movie;
+----+----------------------+--------+-------+-------------+--------+
| ID | Title                | Genre  | Fee   | TotalCopies | Rating |
+----+----------------------+--------+-------+-------------+--------+
|  1 | The Godfather        | Drama  | 10000 |           3 |      5 |
|  2 | Shawshank Redemption | Drama  |  1000 |           2 |      4 |
|  3 | Borat                | Comedy |   500 |           1 |      3 |
|  4 | TestMovie            | Drama  |     1 |           5 |   NULL |
+----+----------------------+--------+-------+-------------+--------+
4 rows in set (0.00 sec)
```

Description:
Rating is set to be null for a new movie. The ID was generated automatically.

b. Edit movie

In-Parameters:
MovieID: Its type is INT UNSIGNED, and it's the id of the movie being edited.
_Attribute: Its type is VARCHAR, and it's the attribute(title, genre,fee or number of total copies) that we wanted to change for that movie.

new_Value: Its type is VARCHAR, and it's the new value of the attribute being edited.

SQL Code:

```
DELIMITER $$
CREATE PROCEDURE EditMovie (IN MovieID INT UNSIGNED, _Attribute
VARCHAR(64), new_Value VARCHAR(256))
BEGIN
   START TRANSACTION;
      SET @editmovie_str = CONCAT('UPDATE Movie SET ', _Attribute, '=', new_Value, '
WHERE ID=', MovieID);
      PREPARE editmovie_stmt FROM @editmovie_str;
      EXECUTE editmovie_stmt;
      DEALLOCATE PREPARE editmovie_stmt;
   COMMIT;
END;
$$
DELIMITER ;
```

Sample Output:

```
mysql> CALL EditMovie(4, 'Fee', 10.00);
Query OK, 0 rows affected (0.07 sec)

mysql> CALL EditMovie(4, 'Rating', 5);
Query OK, 0 rows affected (0.07 sec)

mysql> SELECT * FROM Movie;
+----+----------------------+--------+-------+-------------+--------+
| ID | Title                | Genre  | Fee   | TotalCopies | Rating |
+----+----------------------+--------+-------+-------------+--------+
|  1 | The Godfather        | Drama  | 10000 |           3 |      5 |
|  2 | Shawshank Redemption | Drama  |  1000 |           2 |      4 |
|  3 | Borat                | Comedy |   500 |           1 |      3 |
|  4 | TestMovie            | Drama  |    10 |           5 |      5 |
+----+----------------------+--------+-------+-------------+--------+
4 rows in set (0.00 sec)
```

c. Delete movie

In-Parameters:

MovieID: Its type is INT UNSIGNED, and it's the id of the movie being deleted.

SQL Code:

```
DELIMITER $$
```

```
CREATE PROCEDURE DeleteMovie (IN MovieID INT UNSIGNED)
BEGIN
   START TRANSACTION;
      DELETE FROM Movie
      WHERE ID=MovieID;
   COMMIT;
END;
$$
DELIMITER ;
```

Sample Output:

```
mysql> CALL DeleteMovie(4);
Query OK, 0 rows affected (0.12 sec)

mysql> SELECT * FROM Movie;
+----+----------------------+--------+-------+-------------+--------+
| ID | Title                | Genre  | Fee   | TotalCopies | Rating |
+----+----------------------+--------+-------+-------------+--------+
|  1 | The Godfather        | Drama  | 10000 |           3 |      5 |
|  2 | Shawshank Redemption | Drama  |  1000 |           2 |      4 |
|  3 | Borat                | Comedy |   500 |           1 |      3 |
+----+----------------------+--------+-------+-------------+--------+
3 rows in set (0.00 sec)
```

d.Add information for an employee

In-Parameters:

new_Position: Its type is ENUM('Manager', 'Customer Rep'), and it's the position of the new employee.
new_SSN: Its type is INT UNSIGNED, and it's the SSN of the new employee.
new_FirstName: Its type is VARCHAR, and it's the first name of the new employee.
new_LastName: Its type is VARCHAR, and it's the last name of the new employee.
new_Address: Its type is VARCHAR, and it's the address of the new employee.
new_City: Its type is VARCHAR, and it's the city that the new employee from.
new_State: Its type is VARCHAR, and it's the state that the new employee from.
new_Zip: Its type is INT UNSIGNED, and it's the zip of the city that the new employee from.
new_Phone: Its type is INT UNSIGNED, and it's the phone number of the new employee.
new_StartDate: Its type is DATE, and it's the date that the new employee starts working.
new_Wage: Its type is FLOAT, and it's the hourly wage of the new employee.

SQL Code:

```
DELIMITER $$
```

```
CREATE PROCEDURE AddEmployee (IN    new_Position ENUM('Manager', 'Customer Rep'),
new_SSN INT UNSIGNED, new_FirstName VARCHAR(64),
                              new_LastName VARCHAR(64), new_Address VARCHAR(64),
new_City VARCHAR(64), new_State CHAR(2),
                              new_Zip INT(5) UNSIGNED, new_Phone BIGINT(10) UNSIGNED,
new_StartDate DATE, new_Wage FLOAT)
BEGIN
   DECLARE current_person_index INT; # Needed if a customer exists with the new
Employee's SSN
   SET new_SSN = IF(new_SSN=0, (SELECT AUTO_INC FROM PersonData LIMIT 1),
new_SSN);
          START TRANSACTION;
   IF NOT EXISTS (SELECT * FROM Person WHERE ID = new_SSN) THEN
      INSERT INTO Person (ID, LastName, FirstName, Address, City, State, Zip, Phone)
      VALUES (new_SSN, new_LastName, new_FirstName, new_Address, new_City,
UPPER(new_State), new_Zip, new_Phone);
   ELSEIF NOT EXISTS(SELECT * FROM Employee WHERE SSN=new_SSN) AND (
         new_FirstName != (SELECT FirstName from Person WHERE ID = new_SSN) OR    #
Customer exists with the needed SSN; change the customer's ID:
         new_LastName != (SELECT LastName from Person WHERE ID = new_SSN)) THEN
            SET current_person_index = (SELECT AUTO_INC FROM PersonData WHERE
ID='1'); # Get first free Person ID
      UPDATE Person SET ID=current_person_index WHERE ID=new_SSN; # Change the
customer's CustomerID to the free Person ID
      INSERT INTO Person (ID, LastName, FirstName, Address, City, State, Zip, Phone)
      VALUES (new_SSN, new_LastName, new_FirstName, new_Address, new_City,
UPPER(new_State), new_Zip, new_Phone);
   END IF;
   COMMIT;
   START TRANSACTION;
   INSERT INTO Employee (SSN, Position, StartDate, HourlyRate) VALUES (new_SSN,
new_Position, new_StartDate, new_Wage);
   COMMIT;
END;
$$
DELIMITER ;
```

Sample Output:

```
mysql> CALL AddEmployee('Customer Rep', 987654321, 'Test', 'Case', '123 Hello road', 'Stony Brook', 'NY', 11790, 1234123412, '2005-11-11', 60.00);
Query OK, 0 rows affected (0.08 sec)

mysql> SELECT * FROM Person JOIN Employee ON (Person.ID=Employee.SSN);
+-----------+----------+-----------+------------------+-------------+-------+-------+------------+-----------+-----------+-------------+------------+
| ID        | LastName | FirstName | Address          | City        | State | Zip   | Phone      | SSN       | StartDate | Position    | HourlyRate |
+-----------+----------+-----------+------------------+-------------+-------+-------+------------+-----------+-----------+-------------+------------+
| 123456789 | Smith    | David     | 123 College road | Stony Brook | NY    | 11790 | 5162152345 | 123456789 | 2005-11-01 | Customer Rep |         60 |
| 789123456 | Warren   | David     | 456 Sunken Street | Stony Brook | NY    | 11794 | 6316329987 | 789123456 | 2006-02-02 | Manager     |         50 |
| 987654321 | Case     | Test      | 123 Hello road   | Stony Brook | NY    | 11790 | 1234123412 | 987654321 | 2005-11-11 | Customer Rep |         60 |
+-----------+----------+-----------+------------------+-------------+-------+-------+------------+-----------+-----------+-------------+------------+
3 rows in set (0.00 sec)
```

Description:
The ID was the same as SSN.

e.Edit information for an employee

In-Parameters:
EmployeeSSN: Its type is INT UNSIGNED, and it's the SSN of the employee being edited.
_Attribute: Its type is VARCHAR, and it's the attribute(title, genre,fee or number of total copies) that we wanted to change for that employee.
new_Value: Its type is VARCHAR, and it's the new value of the attribute being edited.

SQL Code:

```
DELIMITER $$
CREATE PROCEDURE EditEmployee (IN EmployeeSSN INT UNSIGNED, _Attribute
VARCHAR(64), new_Value VARCHAR(256))
BEGIN
   START TRANSACTION;
      IF _Attribute IN ('ID', 'FirstName', 'LastName', 'Address', 'City', 'State', 'Zip', 'Phone')
THEN
         SET @editEmployee_str = CONCAT('UPDATE Person SET ', _Attribute, '=',
new_Value, ' WHERE ID=', EmployeeSSN);
         PREPARE editEmployee_stmt FROM @editEmployee_str;
         EXECUTE editEmployee_stmt;
         DEALLOCATE PREPARE editEmployee_stmt;
      ELSE
         SET @editEmployee_str = CONCAT('UPDATE Employee SET ', _Attribute, '=',
new_Value, ' WHERE SSN=', EmployeeSSN);
         PREPARE editEmployee_stmt FROM @editEmployee_str;
         EXECUTE editEmployee_stmt;
```

```
        DEALLOCATE PREPARE editEmployee_stmt;
    END IF;
  COMMIT;
END;
$$
DELIMITER ;
```

Sample Output:

```
mysql> CALL EditEmployee(987654321, 'HourlyRate', 65.00);
Query OK, 0 rows affected (0.07 sec)

mysql> SELECT * FROM Person JOIN Employee ON (Person.ID=Employee.SSN);
+-----------+----------+-----------+------------------+-------------+-------+-------+------------+-----------+------------+-------------+------------+
| ID        | LastName | FirstName | Address          | City        | State | Zip   | Phone      | SSN       | StartDate  | Position    | HourlyRate |
+-----------+----------+-----------+------------------+-------------+-------+-------+------------+-----------+------------+-------------+------------+
| 123456789 | Smith    | David     | 123 College road | Stony Brook | NY    | 11790 | 5162152345 | 123456789 | 2005-11-01 | Customer Rep |         60 |
| 789123456 | Warren   | David     | 456 Sunken Street | Stony Brook | NY    | 11794 | 6316329987 | 789123456 | 2006-02-02 | Manager      |         50 |
| 987654321 | Case     | Test      | 123 Hello road   | Stony Brook | NY    | 11790 | 1234123412 | 987654321 | 2005-11-11 | Customer Rep |         65 |
+-----------+----------+-----------+------------------+-------------+-------+-------+------------+-----------+------------+-------------+------------+
3 rows in set (0.00 sec)
```

f.Delete information for an employee

In-Parameters:

EmployeeSSN: Its type is INT UNSIGNED, and it's the SSN of the employee being deleted.

SQL Code:

```
DELIMITER $$
CREATE PROCEDURE DeleteEmployee (IN EmployeeSSN INT UNSIGNED)
BEGIN
  IF EXISTS(SELECT * FROM Employee WHERE SSN = EmployeeSSN) THEN
    START TRANSACTION;
      DELETE FROM Person
      WHERE ID=EmployeeSSN;
    COMMIT;
  ELSE
    SIGNAL SQLSTATE 'EI928'
      SET MESSAGE_TEXT = 'Invalid Parameter: Employee does not exist.';
  END IF;
END;
$$
DELIMITER ;
```

Sample Output:

```
mysql> CALL DeleteEmployee(987654321);
Query OK, 0 rows affected (0.07 sec)

mysql> SELECT * FROM Person JOIN Employee ON (Person.ID=Employee.SSN);
+-----------+----------+-----------+------------------+-------------+-------+-------+------------+-----------+-----------+-------------+------------+
| ID        | LastName | FirstName | Address          | City        | State | Zip   | Phone      | SSN       | StartDate | Position    | HourlyRate |
+-----------+----------+-----------+------------------+-------------+-------+-------+------------+-----------+-----------+-------------+------------+
| 123456789 | Smith    | David     | 123 College road | Stony Brook | NY    | 11790 | 5162152345 | 123456789 | 2005-11-01 | Customer Rep |         60 |
| 789123456 | Warren   | David     | 456 Sunken Street | Stony Brook | NY    | 11794 | 6316329987 | 789123456 | 2006-02-02 | Manager     |         50 |
+-----------+----------+-----------+------------------+-------------+-------+-------+------------+-----------+-----------+-------------+------------+
2 rows in set (0.00 sec)
```

g.Obtain a sales report for a particular month

SQL Code:

```
CREATE VIEW SalesReport (AccountID, AccountType, AccountCreated, Income) AS (
   SELECT A1.ID, A1.Subscription, A1.Created, 0.00
   FROM Account A1
   WHERE A1.Subscription = 'Limited')
   UNION
   (SELECT A2.ID, A2.Subscription, A2.Created, 5.00
   FROM Account A2
   WHERE A2.Subscription = 'Unlimited')
   UNION
   (SELECT A3.ID, A3.Subscription, A3.Created, 10.00
   FROM Account A3
   WHERE A3.Subscription = 'Unlimited+')
   UNION
   (SELECT A4.ID, A4.Subscription, A4.Created, 15.00
   FROM Account A4
   WHERE A4.Subscription = 'Unlimited++'
);
```

Sample Output:

```
mysql> SELECT * FROM SalesReport;
+-----------+-------------+----------------+--------+
| AccountID | AccountType | AccountCreated | Income |
+-----------+-------------+----------------+--------+
|         2 | Limited     | 2006-10-15     |   0.00 |
|         1 | Unlimited+  | 2006-10-01     |  10.00 |
+-----------+-------------+----------------+--------+
2 rows in set (0.00 sec)
```

Description:
Only the sales report for the latest month can be viewed, since we have nothing to track the rental plan history of the customers.

h.Produce a comprehensive listing of all movies

SQL Code:

```
SELECT * FROM Movie;
```

Sample Output:

```
mysql> SELECT * FROM Movie;
+----+----------------------+--------+-------+-------------+--------+
| ID | Title                | Genre  | Fee   | TotalCopies | Rating |
+----+----------------------+--------+-------+-------------+--------+
|  1 | The Godfather        | Drama  | 10000 |           3 |      5 |
|  2 | Shawshank Redemption | Drama  |  1000 |           2 |      4 |
|  3 | Borat                | Comedy |   500 |           1 |      3 |
+----+----------------------+--------+-------+-------------+--------+
3 rows in set (0.00 sec)
```

i.Produce a list of movie rentals by movie name, movie type or customer name

SQL Code:

```
# Produce a list of movie rentals by movie name:
CREATE VIEW RentalsByMovie (OrderID, AccountID, EmployeeID, MovieID, Title) AS (
   SELECT R.OrderID, R.AccountID, R.EmployeeID, R.MovieID, M.Title
   FROM Rental R JOIN Movie M ON (R.MovieID = M.ID)
   WHERE M.Title = ?  # When a manager uses this transaction, title needs to be specified
);

# Produce a list of movie rentals by customer name:
CREATE VIEW RentalsByCustomer (OrderID, AccountID, EmployeeID, MovieID,
CustomerName) AS (
   SELECT R.OrderID, R.AccountID, R.EmployeeID, R.MovieID, CONCAT(P.FirstName, " ",
P.LastName)
```

```
    FROM Rental R, Person P, Account A
    WHERE P.ID = A.CustomerID
    AND A.ID = R.AccountID
    AND CONCAT(P.FirstName, " ", P.LastName) = ? #When a manager uses this transaction, customer's name needs to be specified
        GROUP BY R.OrderID
);

# Produce a list of movie rentals by movie genre/type:
CREATE VIEW RentalsByGenre (OrderID, AccountID, EmployeeID, MovieID, Genre) AS (
        SELECT R.OrderID, R.AccountID, R.EmployeeID, R.MovieID, M.Genre
        FROM Rental R JOIN Movie M ON (R.MovieID = M.ID)
        WHERE M.Genre = ? #When a manager uses this transaction, movies' genre needs to be specified

);
```

Sample Output:

With ? = 'The Godfather'

```
mysql> SELECT * FROM RentalsByMovie;
+---------+-----------+------------+---------+---------------+
| OrderID | AccountID | EmployeeID | MovieID | Title         |
+---------+-----------+------------+---------+---------------+
|       1 |         1 |  123456789 |       1 | The Godfather |
+---------+-----------+------------+---------+---------------+
1 row in set (0.00 sec)
```

With ? = 'Victor Du'

```
mysql> SELECT * FROM RentalsByCustomer;
+---------+-----------+------------+---------+--------------+
| OrderID | AccountID | EmployeeID | MovieID | CustomerName |
+---------+-----------+------------+---------+--------------+
|       2 |         2 |  123456789 |       3 | Victor Du    |
|       4 |         2 |  123456789 |       2 | Victor Du    |
+---------+-----------+------------+---------+--------------+
2 rows in set (0.00 sec)
```

With ? = 'Drama'

```
mysql> SELECT * FROM RentalsByGenre;
+---------+-----------+------------+---------+-------+
| OrderID | AccountID | EmployeeID | MovieID | Genre |
+---------+-----------+------------+---------+-------+
|       1 |         1 |  123456789 |       1 | Drama |
|       4 |         2 |  123456789 |       2 | Drama |
+---------+-----------+------------+---------+-------+
2 rows in set (0.00 sec)
```

j.Determine which customer representative oversaw the most transactions

SQL Code:

```
CREATE VIEW RepRentalCount (Count, SSN, FullName) AS (
        SELECT C.RentalCount, E.SSN, P.FullName
        FROM (Employee E JOIN (SELECT ID, CONCAT(FirstName, ' ', LastName) FullName
                                FROM Person) P
                                ON E.SSN = P.ID)
                                JOIN (SELECT COUNT(*) RentalCount, R.EmployeeID ID
                                FROM Rental R
                                GROUP BY R.EmployeeID) C
                                ON C.ID = E.SSN
        ORDER BY C.RentalCount DESC
        LIMIT 1
);
```

Sample Output:

```
mysql> SELECT * FROM RepRentalCount;
+-------+-----------+-------------+
| Count | SSN       | FullName    |
+-------+-----------+-------------+
|     4 | 123456789 | David Smith |
+-------+-----------+-------------+
1 row in set (0.00 sec)
```

k.Produce a list of most active customers

SQL Code:

```
CREATE VIEW MostActiveCustomers (AccountID, CustomerID, Rating, JoinDate) AS (
        SELECT A.ID, A.CustomerID, C.Rating, A.Created
        FROM Customer C JOIN Account A ON C.ID = A.CustomerID
        ORDER BY C.Rating DESC, A.Created DESC
        LIMIT 10
);
```

Sample Output:

```
mysql> SELECT * FROM MostActiveCustomers;
+-----------+------------+--------+------------+
| AccountID | CustomerID | Rating | JoinDate   |
+-----------+------------+--------+------------+
|         2 |  222222222 |      1 | 2006-10-15 |
|         1 |  444444444 |      1 | 2006-10-01 |
+-----------+------------+--------+------------+
2 rows in set (0.00 sec)
```

Description:

Top 10 most active customers are listed.

l.Produce a list of most actively rented movies

SQL Code:

```
CREATE VIEW PopularMovies (RentalCount, MovieID, Title) AS (
        SELECT R.TotalRentals, M.ID, M.Title
        FROM Movie M JOIN (SELECT COUNT(*) TotalRentals, MovieID
                                        FROM Rental
                                        GROUP BY MovieID) R ON M.ID = R.MovieID
        ORDER BY R.TotalRentals DESC, M.Title ASC
        LIMIT 10
);
```

Sample Output:

```
mysql> SELECT * FROM PopularMovies;
+-------------+---------+----------------------+
| RentalCount | MovieID | Title                |
+-------------+---------+----------------------+
|           2 |       3 | Borat                |
|           1 |       2 | Shawshank Redemption |
|           1 |       1 | The Godfather        |
+-------------+---------+----------------------+
3 rows in set (0.00 sec)
```

Description:

Top 10 most popular movies are listed.

2.2Customer-Representative-Level Transactions

a.Record an order

In-Parameters:

new_OrderDate: Its type is DATETIME , and it's the date of the new order.
new_AccountID: Its type is INT UNSIGNED, and it's the id of the customer who place the order.
new_movieID: Its type is INT UNSIGNED, and it's the id of the movie that being ordered.
new_employeeID: Its type is INT UNSIGNED, and it's the id of the employee who process the order.

SQL Code:

```
DELIMITER $$
CREATE PROCEDURE CreateOrder(IN new_OrderDate DATETIME, new_AccountID INT UNSIGNED, new_MovieID INT UNSIGNED, new_EmployeeID INT(9) UNSIGNED ZEROFILL)
BEGIN
   START TRANSACTION;
      INSERT INTO _Order (OrderDate)
      VALUES (new_OrderDate); # If new_OrderDate is NULL, the current date/time is generated for this record

      INSERT INTO Rental (AccountID, MovieID, EmployeeID, OrderID)
      VALUES (new_AccountID, new_MovieID, new_EmployeeID, (SELECT LAST_INSERT_ID()));
   COMMIT;
END;
$$
DELIMITER ;
```

Sample Output:

```
mysql> CALL CreateOrder('2009-11-21 22:22:00', 2, 2, 123456789);
Query OK, 0 rows affected (0.03 sec)

mysql> SELECT * FROM Rental JOIN _Order ON (ID=OrderID) ORDER BY OrderID;
+---------+-----------+---------+------------+----+---------------------+---------------------+
| OrderID | AccountID | MovieID | EmployeeID | ID | OrderDate           | ReturnDate          |
+---------+-----------+---------+------------+----+---------------------+---------------------+
|       1 |         1 |       1 |  123456789 |  1 | 2009-11-11 10:00:00 | NULL                |
|       2 |         2 |       3 |  123456789 |  2 | 2009-11-11 18:15:00 | NULL                |
|       3 |         1 |       3 |  123456789 |  3 | 2009-11-12 09:30:00 | NULL                |
|       4 |         2 |       2 |  123456789 |  4 | 2009-11-21 22:22:00 | NULL                |
+---------+-----------+---------+------------+----+---------------------+---------------------+
4 rows in set (0.00 sec)
```

b.Add information for a customer

In-Parameters:

new_FirstName: Its type is VARCHAR, and it's the first name of the new customer.
new_LastName: Its type is VARCHAR, and it's the last name of the new customer.
new_Address: Its type is VARCHAR, and it's the address of the new customer live.
new_City: Its type is VARCHAR, and it's the city that the new customer from.
new_State: Its type is VARCHAR, and it's the state that the new customer from.
new_Zip: Its type is INT UNSIGNED, and it's the zip of the city that the new customer from.
new_Phone: Its type is INT UNSIGNED, and it's the phone number of the new customer.
new_Email: Its type is VARCHAR, and it's the email address of the new customer.
new_CreditCard: Its type is BIGINT UNSIGNED, and it's the credit card number that the new customer holds.

SQL Code:

```
CREATE PROCEDURE AddCustomer (IN new_FirstName VARCHAR(64), new_LastName
VARCHAR(64), new_Address VARCHAR(64),
                    new_City VARCHAR(64), new_State CHAR(2), new_Zip INT(5)
UNSIGNED, new_Phone BIGINT(10) UNSIGNED,
                    new_Email VARCHAR(64), new_CreditCard BIGINT(16) UNSIGNED
ZEROFILL)
BEGIN
      START TRANSACTION;
    INSERT INTO Person (FirstName, LastName, Address, City, State, Zip, Phone)
    VALUES (new_FirstName, new_LastName, new_Address, new_City, new_State, new_Zip,
new_Phone);

    INSERT INTO Customer (Email, CreditCard, ID)
    VALUES (new_Email, new_CreditCard, (SELECT ID
```

```
                        FROM Person P
                        WHERE P.LastName = new_LastName
                        AND P.FirstName = new_FirstName
                        AND P.Address = new_Address
                        AND P.City = new_City
                        AND P.State = new_State
                        AND P.Zip = new_Zip));
   COMMIT;
END;
$$
DELIMITER ;
```

Sample Output:

```
mysql> CALL AddCustomer('Lewis', 'Philip', '135 Knowledge Lane', 'Stony Brook', 'NY', 11794, 5166668888, 'pml@cs.sunysb.edu', 6789234567892345);
Query OK, 0 rows affected (0.01 sec)

mysql> SELECT * FROM Person JOIN Customer ON (Person.ID=Customer.ID);
+-----------+----------+-----------+--------------------+-------------+-------+-------+------------+-----------+---------------------+------------------+--------+
| ID        | LastName | FirstName | Address            | City        | State | Zip   | Phone      | ID        | Email               | CreditCard       | Rating |
+-----------+----------+-----------+--------------------+-------------+-------+-------+------------+-----------+---------------------+------------------+--------+
| 000000001 | Yang     | Shang     | 123 Success Street | Stony Brook | NY    | 11790 | 5166328959 | 000000001 | syang@cs.sunysb.edu | 1234567812345678 |      1 |
| 000000002 | Du       | Victor    | 456 Fortune Road   | Stony Brook | NY    | 11790 | 5166324360 | 000000002 | vicdu@cs.sunysb.edu | 5678123456781234 |      1 |
| 000000003 | Smith    | John      | 789 Peace Blvd.    | Los Angeles | CA    | 93536 | 3154434321 | 000000003 | jsmith@ic.sunysb.edu | 2345678923456789 |      1 |
| 000000004 | Philip   | Lewis     | 135 Knowledge Lane | Stony Brook | NY    | 11794 | 5166668888 | 000000004 | pml@cs.sunysb.edu   | 6789234567892345 |      1 |
+-----------+----------+-----------+--------------------+-------------+-------+-------+------------+-----------+---------------------+------------------+--------+
4 rows in set (0.00 sec)
```

c.Edit information for a customer

In-Parameters:

CustomerId**:** Its type is INT UNSIGNED, and it's the id of the customer being modified.
_Attribute: Its type is VARCHAR, and it's the attribute in the table being modified.
new_Value: Its type is VARCHAR, and it's the new value in the customer table.

SQL Code:

```
DELIMITER $$
CREATE PROCEDURE EditCustomer (IN CustomerID INT UNSIGNED, _Attribute
VARCHAR(64), new_Value VARCHAR(256))
BEGIN
   START TRANSACTION;
      IF _Attribute IN ('ID', 'FirstName', 'LastName', 'Address', 'City', 'State', 'Zip', 'Phone')
THEN
         SET @editCustomer_str = CONCAT('UPDATE Person SET ', _Attribute, '=',
new_Value, ' WHERE ID=', CustomerID);
```

```
        PREPARE editCustomer_stmt FROM @editCustomer_str;
        EXECUTE editCustomer_stmt;
        DEALLOCATE PREPARE editCustomer_stmt;
    ELSE
        SET @editCustomer_str = CONCAT('UPDATE Customer SET ', _Attribute, '=',
new_Value, ' WHERE ID=', CustomerID);
        PREPARE editCustomer_stmt FROM @editCustomer_str;
        EXECUTE editCustomer_stmt;
        DEALLOCATE PREPARE editCustomer_stmt;
    END IF;
  COMMIT;
END;
$$
DELIMITER ;
```

Sample Output:

```
mysql> CALL EditCustomer(4, 'ID', 444444444);
Query OK, 0 rows affected (0.01 sec)

mysql> SELECT * FROM Person JOIN Customer ON (Person.ID=Customer.ID);
+-----------+----------+-----------+--------------------+-------------+-------+-------+------------+-----------+---------------------+-----------------+--------+
| ID        | LastName | FirstName | Address            | City        | State | Zip   | Phone      | ID        | Email               | CreditCard      | Rating |
+-----------+----------+-----------+--------------------+-------------+-------+-------+------------+-----------+---------------------+-----------------+--------+
| 111111111 | Yang     | Shang     | 123 Success Street | Stony Brook | NY    | 11790 | 5166328959 | 111111111 | syang@cs.sunysb.edu | 1234567812345678 |      1 |
| 222222222 | Du       | Victor    | 456 Fortune Road   | Stony Brook | NY    | 11790 | 5166324360 | 222222222 | vicdu@cs.sunysb.edu | 5678123456781234 |      1 |
| 333333333 | Smith    | John      | 789 Peace Blvd.    | Los Angeles | CA    | 93536 | 3154434321 | 333333333 | jsmith@ic.sunysb.edu | 2345678923456789 |      1 |
| 444444444 | Philip   | Lewis     | 135 Knowledge Lane | Stony Brook | NY    | 11794 | 5166668888 | 444444444 | pml@cs.sunysb.edu   | 6789234567892345 |      1 |
+-----------+----------+-----------+--------------------+-------------+-------+-------+------------+-----------+---------------------+-----------------+--------+
4 rows in set (0.00 sec)
```

d.Delete information for a customer

In-Parameters:

CustomerId: Its type is INT UNSIGNED, and it's the id of the customer being deleted.

SQL Code:

```
DELIMITER $$
CREATE PROCEDURE DeleteCustomer (IN CustomerID INT UNSIGNED)
BEGIN
  IF EXISTS(SELECT * FROM Customer WHERE ID = CustomerID) THEN
    START TRANSACTION;
      DELETE FROM Person
      WHERE ID=CustomerID;
    COMMIT;
```

```
    ELSE
        SIGNAL SQLSTATE 'EI928'
            SET MESSAGE_TEXT = 'Invalid Parameter: Customer does not exist.';
    END IF;
END;
$$
DELIMITER ;
```

Sample Output:

```
mysql> CALL DeleteCustomer(333333333);
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT * FROM Person JOIN Customer ON (Person.ID = Customer.ID);
+-----------+----------+-----------+-------------------+-------------+-------+-------+------------+-----------+--------------------+------------------+--------+
| ID        | LastName | FirstName | Address           | City        | State | Zip   | Phone      | ID        | Email              | CreditCard       | Rating |
+-----------+----------+-----------+-------------------+-------------+-------+-------+------------+-----------+--------------------+------------------+--------+
| 111111111 | Yang     | Shang     | 123 Success Street | Stony Brook | NY    | 11790 | 5166328959 | 111111111 | syang@cs.sunysb.edu | 1234567812345678 |      1 |
| 222222222 | Du       | Victor    | 456 Fortune Road  | Stony Brook | NY    | 11790 | 5166324360 | 222222222 | vicdu@cs.sunysb.edu | 5678123456781234 |      1 |
| 444444444 | Philip   | Lewis     | 135 Knowledge Lane | Stony Brook | NY    | 11794 | 5166668888 | 444444444 | pml@cs.sunysb.edu  | 6789234567892345 |      1 |
+-----------+----------+-----------+-------------------+-------------+-------+-------+------------+-----------+--------------------+------------------+--------+
3 rows in set (0.00 sec)
```

e.Produce customer mailing lists

SQL Code:

```
CREATE VIEW MailingList (AccountID, CustomerID, CustomerName, Email, Subscription)
AS (
        SELECT A.ID, C.ID, C.FullName, C.Email, A.Subscription
        FROM Account A JOIN (SELECT C1.ID, CONCAT(P.FirstName, ' ', P.LastName)
                        FullName, C1.Email FROM Customer C1 JOIN Person P ON
                        C1.ID = P.ID) C ON A.CustomerID = C.ID
);
```

Sample Output:

```
mysql> SELECT * FROM MailingList
    -> ;
+-----------+------------+--------------+---------------------+--------------+
| AccountID | CustomerID | CustomerName | Email               | Subscription |
+-----------+------------+--------------+---------------------+--------------+
|         1 |  444444444 | Lewis Philip | pml@cs.sunysb.edu   | Unlimited+   |
|         2 |  222222222 | Victor Du    | vicdu@cs.sunysb.edu | Limited      |
+-----------+------------+--------------+---------------------+--------------+
2 rows in set (0.00 sec)
```

f.Produce a list of movie suggestions for a given customer (using the recommender system which uses information about the customer's past orders and that of nearest neighbors)

SQL Code:

```
# ? is the account id of the given customer, and ?? is the total number of accounts in the data base
CREATE VIEW SuggestionOfMovie(MovieTitle) AS (
        SELECT M.Title
        FROM Rental R, Movie M, Casted C1, Casted C2
        WHERE M.ID NOT IN (SELECT MovieID FROM Rental R WHERE R.AccountID = ?)
        AND ( (C1.MovieID = M.ID AND C2.MovieID = R.MovieID AND C1.ActorID =
                C2.ActorID) OR ( M.ID IN (SELECT MovieID FROM Rental R WHERE
                R.AccountID = MOD((?-1), ??) OR R.AccountID = MOD((?+1), ??)) ) )
        GROUP BY M.Title
);
```

Sample Output: (With ? = 2, and ?? = 2)

```
mysql> SELECT * FROM SuggestionOfMovie;
+---------------+
| MovieTitle    |
+---------------+
| The Godfather |
+---------------+
1 row in set (0.01 sec)
```

2.3 Customer-Level Transactions

a.View a customer's currently held movies

SQL Code:

```
CREATE VIEW CurrentRentals (AccountID, MovieID, Title, OrderDate) AS (
        SELECT AccountID, MovieID, Title, OrderDate
        FROM (Rental JOIN _Order ON OrderID = _Order.ID) JOIN Movie ON (MovieID =
Movie.ID)
        WHERE ReturnDate IS NULL
        ORDER BY OrderDate ASC
);
```

Sample Output:

```
mysql> SELECT * FROM CurrentRentals;
+-----------+---------+----------------------+---------------------+
| AccountID | MovieID | Title                | OrderDate           |
+-----------+---------+----------------------+---------------------+
|         2 |       3 | Borat                | 2009-11-11 18:15:00 |
|         1 |       3 | Borat                | 2009-11-12 09:30:00 |
|         2 |       2 | Shawshank Redemption | 2009-11-21 22:22:00 |
+-----------+---------+----------------------+---------------------+
3 rows in set (0.00 sec)
```

b.View a customer's queue of movies it would like to see

SQL Code:

```
CREATE VIEW MovieQueue (AccountID, CustomerID, MovieID, Title, DateAdded) AS (
        SELECT Q.AccountID, A.CustomerID, Q.MovieID, M.Title, Q.DateAdded
        FROM (Queued Q JOIN Movie M ON (Q.MovieID = M.ID)) JOIN Account A ON
(Q.AccountID = A.ID)
        ORDER BY DateAdded ASC
);
```

Sample Output:

```
mysql> SELECT * FROM MovieQueue;
Empty set (0.00 sec)
```

Description:
The Queued table is empty.

c.View a customer's account settings

SQL Code:

```
CREATE VIEW AccountSetting (AccountID, CustomerID, Subscription) AS (
        SELECT A.ID, A.CustomerID, A.Subscription
        FROM Account A
   WHERE A.CustomerID = ?   # ? is the customer id of the given customer
);
```

Sample Output: (With ? = 222222222)

```
mysql> SELECT * FROM AccountSetting;
+-----------+------------+--------------+
| AccountID | CustomerID | Subscription |
+-----------+------------+--------------+
|         2 |  222222222 | Limited      |
+-----------+------------+--------------+
1 row in set (0.00 sec)
```

d.View a history of all current and past orders a customer has placed

SQL Code:

```
CREATE VIEW RentalHistory (AccountID, OrderID, MovieID, Title, Genre, Rating,
        OrderDate, ReturnDate) AS (
        SELECT AccountID, OrderID, MovieID, Title, Genre, Movie.Rating, OrderDate,
                ReturnDate
        FROM (Rental JOIN _Order ON OrderID = _Order.ID) JOIN Movie ON (MovieID =
                Movie.ID)
        WHERE AccountID = ?  # ? is the account id of the given customer
        ORDER BY OrderDate DESC
);
```

Sample Output:

```
mysql> SELECT * FROM RentalHistory;
+-----------+---------+---------+----------------------+--------+--------+---------------------+------------+
| AccountID | OrderID | MovieID | Title                | Genre  | Rating | OrderDate           | ReturnDate |
+-----------+---------+---------+----------------------+--------+--------+---------------------+------------+
|         2 |       4 |       2 | Shawshank Redemption | Drama  |      4 | 2009-11-21 22:22:00 | NULL       |
|         2 |       2 |       3 | Borat                | Comedy |      3 | 2009-11-11 18:15:00 | NULL       |
+-----------+---------+---------+----------------------+--------+--------+---------------------+------------+
2 rows in set (0.00 sec)
```

e.View if movies available of a particular type

SQL Code:

```
# List of the number of copies rented for each movie:
CREATE VIEW RentedCopies (MovieID, Copies) AS (
        SELECT MovieID, SUM(ReturnDate IS NULL)
        FROM (Rental JOIN _Order ON OrderID = _Order.ID) JOIN Movie ON (MovieID =
Movie.ID)
        GROUP BY MovieID
);

# List of the number of copies available for each movie:
CREATE VIEW AvailableCopies (MovieID, Copies) AS (
        SELECT MovieID, CAST(TotalCopies AS SIGNED) - Copies
        FROM RentedCopies JOIN Movie ON (MovieID = Movie.ID)
        GROUP BY MovieID
);

# List of available movies by genre:
CREATE VIEW AvailableMoviesByGenre(MovieID, Title, Genre, Rating, AvailableCopies,
TotalCopies) AS (
        SELECT MovieID, Title, Genre, Rating, A.Copies, TotalCopies
        FROM Movie JOIN AvailableCopies A ON (ID=A.MovieID)
        WHERE A.Copies > 0
   AND Genre = ? # ? is the genre of the movie
   GROUP BY MovieID
   ORDER BY MovieID
);
```

Sample Output: (With ? = ‘Drama’)

```
mysql> SELECT * FROM AvailableMoviesByGenre;
+---------+----------------------+-------+--------+-----------------+-------------+
| MovieID | Title                | Genre | Rating | AvailableCopies | TotalCopies |
+---------+----------------------+-------+--------+-----------------+-------------+
|       1 | The Godfather        | Drama |      5 |               3 |           3 |
|       2 | Shawshank Redemption | Drama |      4 |               1 |           2 |
+---------+----------------------+-------+--------+-----------------+-------------+
2 rows in set (0.00 sec)
```

Description:

There is a bug in the given demo data. The total copies of movie “Borat” is 1, but 2 customers are renting it.

f.View if movies available with a particular keyword or set of keywords in the movie name

In Parameter:

word**:** Its type is VARCHAR, and it’s the keyword of the customer is searching.

SQL Code:

```
DELIMITER $$
CREATE PROCEDURE Search (IN word VARCHAR(64))
BEGIN
        SELECT M.Title, A.Copies
   FROM Movie M JOIN AvailableCopies A ON (M.ID = A.MovieID)
   WHERE M.Title LIKE CONCAT('%', word, '%');
END;
$$
DELIMITER ;
```

Sample Output:

```
mysql> CALL Search("God");
+---------------+--------+
| Title         | Copies |
+---------------+--------+
| The Godfather |      3 |
+---------------+--------+
1 row in set (0.00 sec)
```

Description:

This implementation can search only consecutive keywords. For example, if a movie is called “Batman vs. Superman”, a user can only get the movie by searching “Batman vs” but not “Batman Superman”.

g.View if movies available starring a particular actor or group of actors

In Parameter:

starName**:** Its type is VARCHAR, and it's the actor's name of the customer is searching.

SQL Code:

```
DELIMITER $$
CREATE PROCEDURE SearchStar (IN starName VARCHAR(64))
BEGIN
        SELECT M.Title, CONCAT(S.FirstName, " ", S.LastName) AS Star,A.Copies
   FROM ((Movie M JOIN AvailableCopies A ON (M.ID = A.MovieID)) JOIN Casted C ON
(C.MovieID = M.ID)) JOIN Actor S ON (S.ID = C.ActorID)
   WHERE CONCAT(S.FirstName, " ", S.LastName) LIKE starName
   GROUP BY S.ID;
END;
$$
DELIMITER ;
```

Sample Output:

```
mysql> CALL SearchStar('Al Pacino');
+---------------+-----------+--------+
| Title         | Star      | Copies |
+---------------+-----------+--------+
| The Godfather | Al Pacino |      3 |
| Borat         | Al Pacino |     -1 |
+---------------+-----------+--------+
2 rows in set (0.00 sec)
```

Description:

This implementation can search an actor at a time only.

Also as mentioned above, there is a bug in the given demo data. The total copies of movie "Borat" is 1, but 2 customers are renting it.

h.View the Best-Seller list of movies

Description:

It is the same as part i in Manager-Level Transactions: Produce a list of most actively rented movies.

i.View the personalized movie suggestion list (by selecting a actor or movie name)

Description:

It is the same as part f: search keyword and part g search actor(star).

j.Rate the movies they have rented

In Parameter:

mvTitle: Its type is VARCHAR, and it is the title of the movie that the customer wants to rate.
rate: Its type is INT UNSIGNED, and it is the rate the customer put.

SQL Code:

```
DELIMITER $$
CREATE PROCEDURE rateRented (IN mvTitle VARCHAR(64), rate INT UNSIGNED)
BEGIN
   START TRANSACTION;
                IF mvTitle IN (SELECT Title FROM RentalsByCustomer C JOIN (Rental R
JOIN Movie M ON (R.MovieID = M.ID)) ON (C.AccountID = R.AccountID))
THEN
                        UPDATE Movie
                        SET Rating = ((rate + (? * Rating)) / (1 + ?)) # ? is the total number of
times that the movie is rented
                        WHERE Title LIKE mvTitle;
                END IF;
   COMMIT;
END;
$$
DELIMITER ;
```

Sample Output: (With ? = 1)

```
mysql> SELECT * FROM Movie;
+----+----------------------+--------+-------+-------------+--------+
| ID | Title                | Genre  | Fee   | TotalCopies | Rating |
+----+----------------------+--------+-------+-------------+--------+
|  1 | The Godfather        | Drama  | 10000 |           3 |      5 |
|  2 | Shawshank Redemption | Drama  |  1000 |           2 |      4 |
|  3 | Borat                | Comedy |   500 |           1 |      3 |
+----+----------------------+--------+-------+-------------+--------+
3 rows in set (0.00 sec)

mysql> CALL rateRented('Borat', 5);
Query OK, 0 rows affected (0.08 sec)

mysql> SELECT * FROM Movie;
+----+----------------------+--------+-------+-------------+--------+
| ID | Title                | Genre  | Fee   | TotalCopies | Rating |
+----+----------------------+--------+-------+-------------+--------+
|  1 | The Godfather        | Drama  | 10000 |           3 |      5 |
|  2 | Shawshank Redemption | Drama  |  1000 |           2 |      4 |
|  3 | Borat                | Comedy |   500 |           1 |      4 |
+----+----------------------+--------+-------+-------------+--------+
3 rows in set (0.00 sec)
```

Description:

Same as above in sample output part i in Manager-Level Transactions, the RentalsByCustomer is the view for rentals by Victor Du. And in further implementation, we need to add attribute to keep track of the total times a movie has been rated so that we can get rid of the ? in this implementation.