3.1

Add, Edit and Delete movies

- Type and Definitions of Parameters:
  -

Column Name	Datatype
Id	INT
Name	CHAR(20)
Type	CHAR(20)
Rating	INT
DistrFee	DOUBLE
NumCopies	INT

(Movie)

- SQL Statement:
  -
```
INSERT INTO Movie (Id, Name, Type, Rating, DistrFee, NumCopies)
VALUES (?,?,?,?,?,?);
```
  -
```
UPDATE Movie
SET ? = ?
WHERE Id = ?';
```
  -
```
DELETE FROM AppearedIn
Where MovieId = ?;
DELETE FROM Rental
WHERE MovieId = ?;
DELETE FROM Movie
WHERE Id = ?;
```
- Execution
  -
```
mysql> INSERT INTO Movie (Id, Name, Type, Rating, DistrFee, NumCopies)
    -> VALUES (7,'Movie Name', 'Movie Type', 4, 3.99, 10);
Query OK, 1 row affected (0.01 sec)

mysql> SELECT * FROM Movie;
+----+----------------------+------------+--------+----------+-----------+
| Id | Name                 | Type       | Rating | DistrFee | NumCopies |
+----+----------------------+------------+--------+----------+-----------+
|  1 | The Godfather        | Drama      |      5 |    10000 |         3 |
|  2 | Shawshank Redemption | Drama      |      4 |     1000 |         2 |
|  3 | Borat                | Comedy     |      3 |      500 |         1 |
|  7 | Movie Name           | Movie Type |      4 |     3.99 |        10 |
+----+----------------------+------------+--------+----------+-----------+
```
  -
```
mysql> UPDATE Movie
    -> SET Rating = 5
    -> WHERE Name = 'Movie Name';
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> SELECT * FROM Movie;
+----+----------------------+------------+--------+----------+-----------+
| Id | Name                 | Type       | Rating | DistrFee | NumCopies |
+----+----------------------+------------+--------+----------+-----------+
|  1 | The Godfather        | Drama      |      5 |    10000 |         3 |
|  2 | Shawshank Redemption | Drama      |      4 |     1000 |         2 |
|  3 | Borat                | Comedy     |      3 |      500 |         1 |
|  7 | Movie Name           | Movie Type |      5 |     3.99 |        10 |
+----+----------------------+------------+--------+----------+-----------+
4 rows in set (0.00 sec)
```
  -

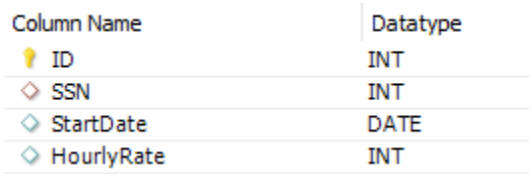
```
mysql> DELETE FROM AppearedIn
    -> Where MovieId = 3;
Query OK, 1 row affected (0.01 sec)
```

```
mysql> DELETE FROM Rental
    -> WHERE MovieId = 3;
Query OK, 2 rows affected (0.01 sec)

mysql> DELETE FROM Movie
    -> WHERE Id = 3;
Query OK, 1 row affected (0.00 sec)
```

- Short Description and Concerns:
  - These are simple insertion, update, and delete statements. However, a simple deletion from the movie table would not be allowed as it would violate a foreign key constraint in the AppearsIn table and the Rental table.

Add, Edit and Delete information for an employee

- Type and Definitions of Parameters:
  - 

Column Name	Datatype
ID	INT
SSN	INT
StartDate	DATE
HourlyRate	INT

(Employee)
- SQL Statement:
  - ```
    INSERT INTO Employee (ID, SSN, StartDate, HourlyRate)
    VALUES (?,?,?,?);
    ```
  - ```
    UPDATE Employee
    SET ? = ?
    WHERE ID = ?;
    ```
  - ```
    DELETE FROM Employee
    WHERE ID = ?;
    ```
- Execution
  - ```
    mysql> INSERT INTO Employee (ID, SSN, StartDate, HourlyRate)
        -> VALUES (10, 123456789, '2023-11-02', 15.0);
    Query OK, 1 row affected (0.01 sec)

    mysql> Select * FROM Employee;
    +----+-----------+------------+------------+
    | ID | SSN       | StartDate  | HourlyRate |
    +----+-----------+------------+------------+
    |  1 | 123456789 | 2011-01-05 |         60 |
    |  2 | 789123456 | 0002-02-06 |         50 |
    |  9 | 123456789 | 2023-11-02 |         15 |
    | 10 | 123456789 | 2023-11-02 |         15 |
    +----+-----------+------------+------------+
    4 rows in set (0.00 sec)
    ```

```
mysql> UPDATE Employee
    -> SET HourlyRate = 18.0
    -> WHERE ID = 1;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

```
mysql> DELETE FROM Employee
    -> WHERE ID = 9;
Query OK, 1 row affected (0.01 sec)
```

- Short Description and Concerns:
    - These are simple insertion, update, and delete statements.

Obtain a sales report (i.e. the overall income from all active subscriptions) for a particular month:

- Type and Definitions of Parameters:
    - MONTH(A.DateOpened): INTEGER
- SQL Statement

```
SELECT
   SUM(
      CASE
         WHEN A.Type = 'Limited' THEN 10
         WHEN A.Type = 'Unlimited-1' THEN 15
         WHEN A.Type = 'Unlimited-2' THEN 20
         WHEN A.Type = 'Unlimited-3' THEN 25
         ELSE 0
      END
   ) AS TotalIncome
FROM
   Account A
WHERE
   MONTH(A.DateOpened) = ?;
```

- Execution

```
mysql> SELECT
    ->     SUM(
    ->         CASE
    ->             WHEN A.Type = 'Limited' THEN 10
    ->             WHEN A.Type = 'Unlimited-1' THEN 15
    ->             WHEN A.Type = 'Unlimited-2' THEN 20
    ->             WHEN A.Type = 'Unlimited-3' THEN 25
    ->             ELSE 0
    ->         END
    ->     ) AS TotalIncome
    -> FROM
    ->     Account A
    -> WHERE
    ->     MONTH(A.DateOpened) = 6;
+-------------+
| TotalIncome |
+-------------+
|          60 |
+-------------+
1 row in set (0.00 sec)
```

- Short Description and Concerns:
  - This SQL statement essentially queries the Account table for all the types for each row that is during a specific month and calculates a sum of values corresponding to the cost of each subscription type. Some concerns are if there are any new types or changes in the types, the SQL statement itself has to be modified.

Produce a comprehensive listing of all movies:

- SQL Statement
  - ```
    SELECT * From Movie;
    ```
- Execution
  - ```
    mysql> SELECT * From Movie;
    +----+----------------------+--------+--------+----------+-----------
    | Id | Name                 | Type   | Rating | DistrFee | NumCopies
    +----+----------------------+--------+--------+----------+-----------
    |  1 | The Godfather        | Drama  |      5 |    10000 |         3
    |  2 | Shawshank Redemption | Drama  |      4 |     1000 |         2
    |  3 | Borat                | Comedy |      3 |      500 |         1
    +----+----------------------+--------+--------+----------+-----------
    3 rows in set (0.00 sec)
    ```
- Short Description and Concerns:
  - This SQL statement essentially just returns every row from the Movie table. A concern is it does not properly organize any of the data in a different or more digestible form.

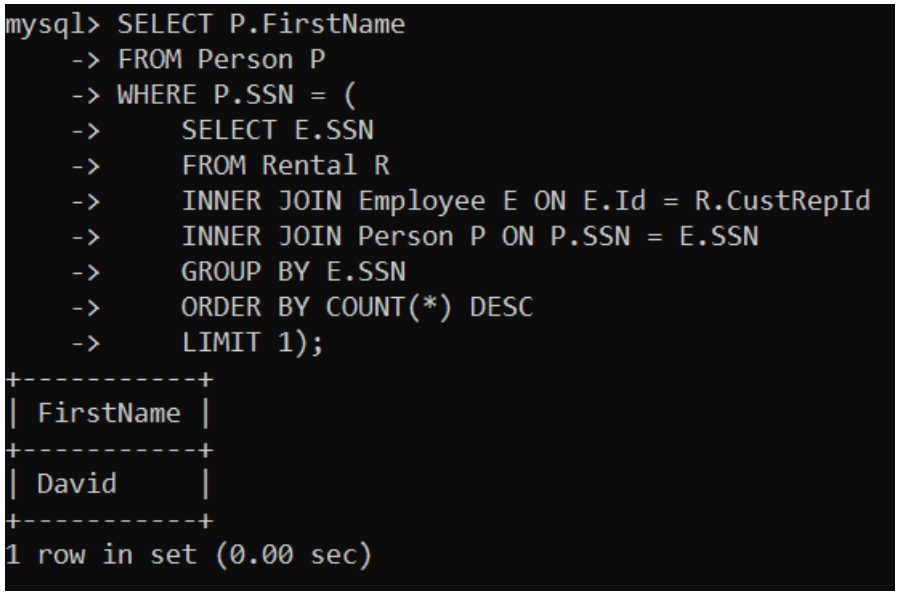
Produce a list of movie rentals by movie name, movie type or customer name: (don't know what the fuck this is specifically asking, just joined everything we need basically).

- Type and Definitions of Parameters:
  - M.Name: CHAR(20)
  - M.Type: CHAR(20)
  - P.FirstName: CHAR(20)
- SQL Statement
  -
```
SELECT R.AccountId, R.CustRepId, R.OrderId, R.MovieId, M.Name
FROM Rental R
INNER JOIN `Order` O ON R.OrderId = O.Id
INNER JOIN Movie M ON R.MovieId = M.Id
INNER JOIN Account A ON A.Id = R.AccountId
INNER JOIN Customer C ON A.Customer = C.Id
INNER JOIN Person P ON C.Id = P.SSN
WHERE M.Name = ? OR M.Type = ? or P.FirstName = ?;
```
- Execution
  -
```
mysql>
mysql> SELECT R.AccountId, R.CustRepId, R.OrderId, R.MovieId, M.Name
    -> FROM Rental R
    -> INNER JOIN `Order` O ON R.OrderId = O.Id
    -> INNER JOIN Movie M ON R.MovieId = M.Id
    -> INNER JOIN Account A ON A.Id = R.AccountId
    -> INNER JOIN Customer C ON A.Customer = C.Id
    -> INNER JOIN Person P ON C.Id = P.SSN
    -> WHERE M.Name = "The Godfather" OR M.Type = "" or P.FirstName = "";
+-----------+-----------+---------+---------+---------------+
| AccountId | CustRepId | OrderId | MovieId | Name          |
+-----------+-----------+---------+---------+---------------+
|         1 |         1 |       1 |       1 | The Godfather |
+-----------+-----------+---------+---------+---------------+
1 row in set (0.00 sec)
```
- Short Description and Concerns:
  - This query essentially joins many tables to connect all the data and filters the rows on different parameters such as the Movie Name, movie type, and or customer name and then returns the information for each related Rental entry. One of the major concerns is the sheer amount of joins involved in the query (a lot of computation). The query also requires direct modification if different combinations of filtering needs to be used. (For example, rather than all three parameters, only two or one.)

Determine which customer representative oversaw the most transactions (rentals):

- SQL Statement
  -

```
SELECT P.FirstName
FROM Person P
WHERE P.SSN = (
   SELECT E.SSN
   FROM Rental R
   INNER JOIN Employee E ON E.Id = R.CustRepId
   INNER JOIN Person P ON P.SSN = E.SSN
   GROUP BY E.SSN
   ORDER BY COUNT(*) DESC
   LIMIT 1);
```

- Execution
  - 
```
mysql> SELECT P.FirstName
    -> FROM Person P
    -> WHERE P.SSN = (
    ->     SELECT E.SSN
    ->     FROM Rental R
    ->     INNER JOIN Employee E ON E.Id = R.CustRepId
    ->     INNER JOIN Person P ON P.SSN = E.SSN
    ->     GROUP BY E.SSN
    ->     ORDER BY COUNT(*) DESC
    ->     LIMIT 1);
+-----------+
| FirstName |
+-----------+
| David     |
+-----------+
1 row in set (0.00 sec)
```
- Short Description and Concerns:
  - This query fetches all of the names of the people who are employees that have ties to the most rows in the Rental table. One concern about this specific query is the case in which multiple representatives have overseen the most amount of transactions.

<u>Produce a list of most active customers:</u>

- SQL Statement
  -

```
SELECT C.Id AS CustomerId,
      P.FirstName AS FirstName,
      P.LastName AS LastName,
      COUNT(R.AccountId) AS RentalCount
FROM Customer C
INNER JOIN Person P ON C.Id = P.SSN
LEFT JOIN Account A ON C.Id = A.Customer
```

```
LEFT JOIN Rental R ON A.Id = R.AccountId
GROUP BY C.Id, P.FirstName, P.LastName
ORDER BY RentalCount DESC;
```

- Execution
  -
```
mysql> SELECT C.Id AS CustomerId,
    ->        P.FirstName AS FirstName,
    ->        P.LastName AS LastName,
    ->        COUNT(R.AccountId) AS RentalCount
    -> FROM Customer C
    -> INNER JOIN Person P ON C.Id = P.SSN
    -> LEFT JOIN Account A ON C.Id = A.Customer
    -> LEFT JOIN Rental R ON A.Id = R.AccountId
    -> GROUP BY C.Id, P.FirstName, P.LastName
    -> ORDER BY RentalCount DESC;
+------------+-----------+----------+-------------+
| CustomerId | FirstName | LastName | RentalCount |
+------------+-----------+----------+-------------+
|  444444444 | Lewis     | Philip   |           4 |
|  111111111 | Shang     | Yang     |           0 |
|  222222222 | Victor    | Du       |           0 |
|  333333333 | John      | Smith    |           0 |
+------------+-----------+----------+-------------+
4 rows in set (0.00 sec)
```

- Short Description and Concerns:
  - This query fetches all of the customers who have rented the most movies. It joins the Person table, with the Account table, and the Rental table on the proper columns, groups them by the Customer Id's and names, and orders the rows on the count of the account id's to accumulate how much they've rented movies. A concern with this query is that it does not filter out the customers that have not rented any movies so there is seemingly useless information.

<u>Produce a list of most actively rented movies:</u>

- SQL Statement
  -
```
SELECT M.Id AS MovieId,
    M.Name AS MovieName,
    COUNT(R.MovieId) AS RentalCount
FROM Movie M
LEFT JOIN Rental R ON M.Id = R.MovieId
GROUP BY M.Id, M.Name
ORDER BY RentalCount DESC;
```

- Execution

-
```
mysql> SELECT M.Id AS MovieId,
    ->        M.Name AS MovieName,
    ->        COUNT(R.MovieId) AS RentalCount
    -> FROM Movie M
    -> LEFT JOIN Rental R ON M.Id = R.MovieId
    -> GROUP BY M.Id, M.Name
    -> ORDER BY RentalCount DESC;
+---------+---------------------+-------------+
| MovieId | MovieName           | RentalCount |
+---------+---------------------+-------------+
|       3 | Borat               |           2 |
|       1 | The Godfather       |           1 |
|       2 | Shawshank Redemption |           1 |
+---------+---------------------+-------------+
3 rows in set (0.00 sec)
```

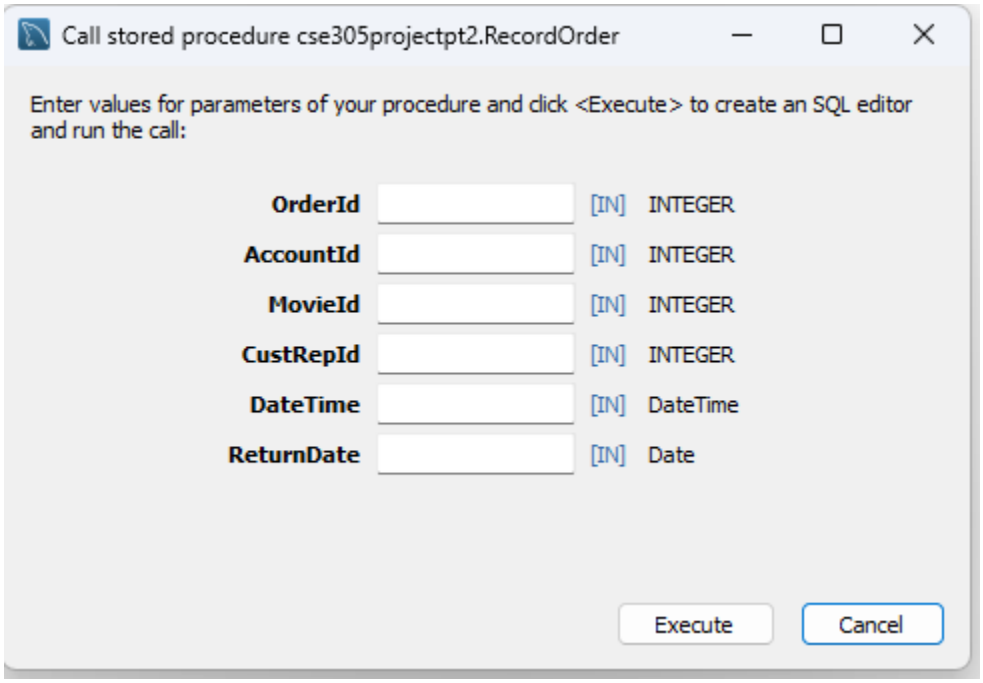
- Short Description and Concerns:
  - This query joins the Rental table to the Movie table and groups by the movie id's and names and creates a count of the movie ids to represent the amount of times they have been rented. One major concern about this query is that it does not limit the amount of results, and as such all the different movies would be returned.

3.2 and 3.3

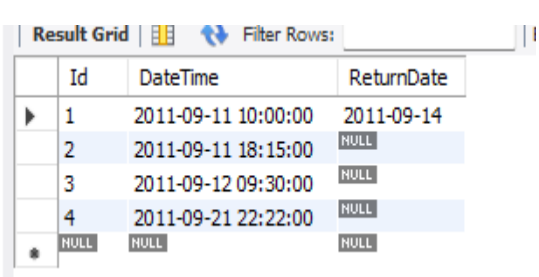
3.2

Record Order

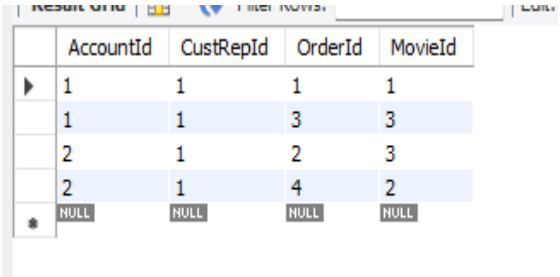
```
CREATE DEFINER=`root`@`localhost` PROCEDURE `RecordOrder`(
    IN OrderId INTEGER,
        AccountId INTEGER,
        MovieId INTEGER,
        CustRepId INTEGER,
        `DateTime` DateTime,
        ReturnDate Date
    )
BEGIN
    INSERT INTO `order` (`DateTime`,ReturnDate,Id)
    VALUES (`DateTime`,ReturnDate,OrderId);
    INSERT INTO `rental` (CustRepId,AccountId,MovieId,OrderId)
    VALUES (CustRepId,AccountId,MovieId,OrderId);
END
```



OutPut:



Id	DateTime	ReturnDate
1	2011-09-11 10:00:00	2011-09-14
2	2011-09-11 18:15:00	NULL
3	2011-09-12 09:30:00	NULL
4	2011-09-21 22:22:00	NULL
NULL	NULL	NULL

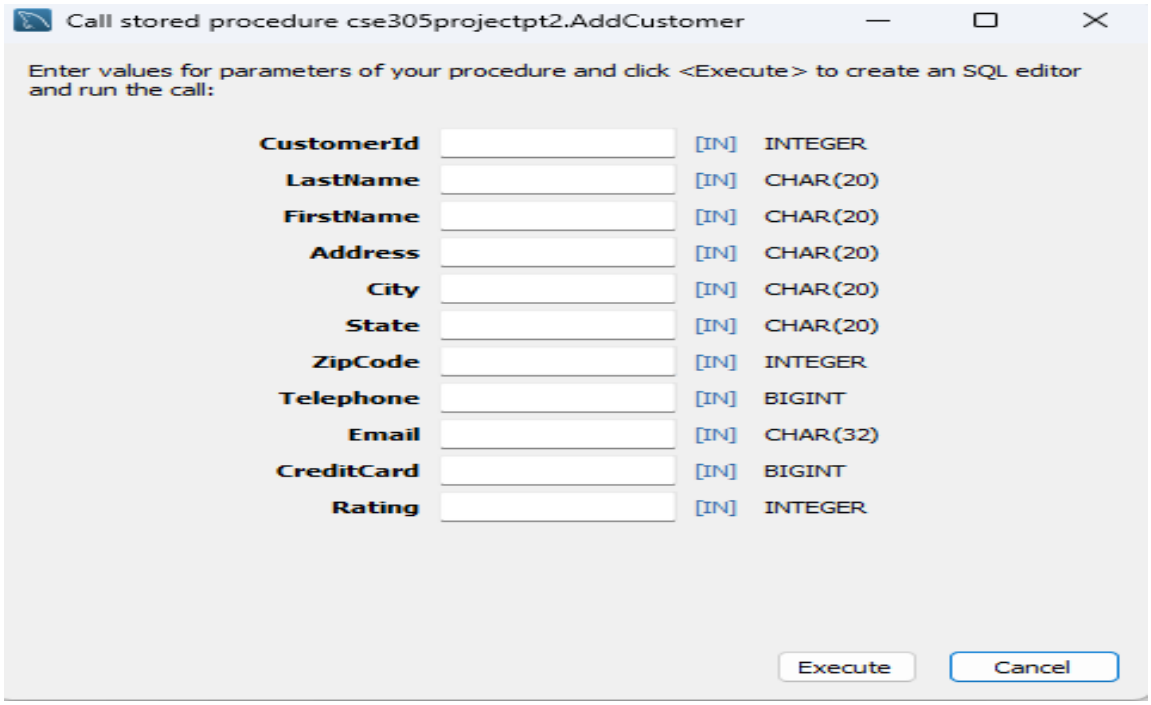


AccountId	CustRepId	OrderId	MovieId
1	1	1	1
1	1	3	3
2	1	2	3
2	1	4	2
NULL	NULL	NULL	NULL

Add customer:

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `AddCustomer`(
    IN CustomerId INTEGER,
        LastName CHAR(20),
        FirstName CHAR(20),
        Address CHAR(20),
        City CHAR(20),
        State CHAR(20),
        ZipCode INTEGER,
        Telephone BIGINT,
        Email CHAR(32),
        CreditCard BIGINT,
        Rating INTEGER
    )
BEGIN
    INSERT INTO `person`(SSN,LastName,FirstName,Address,ZipCode,Telephone)
    VALUES(CustomerId,LastName,FirstName,Address,ZipCode,Telephone);
    INSERT INTO `customer`(Id,Email,Rating,CreditCardNumber)
    VALUES(CustomerId,Email,Rating,CreditCard);
    INSERT INTO `Location`(City,State,ZipCode)
    VALUES(City,State,ZipCode);

END
```



Output:

Id	Email	Rating	CreditCardNumber
111111111	syang@cs.sunysb.edu	1	1234567812345678
222222222	vicdu@cs.sunysb.edu	1	5678123456781234
333333333	jsmith@ic.sunysb.edu	1	2345678923456789
444444444	pml@cs.sunysb.edu	1	6789234567892345
NULL	NULL	NULL	NULL

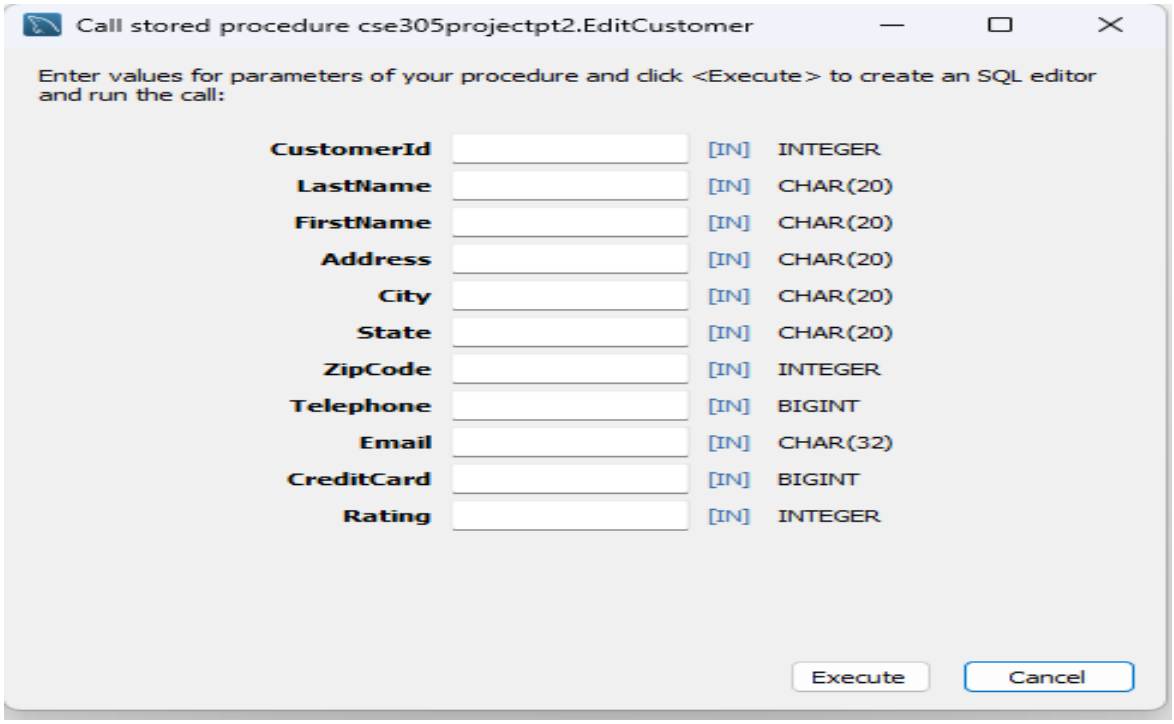
SSN	LastName	FirstName	Address	ZipCode	Telephone
111111111	Yang	Shang	123 Success Street	11790	5166328959
123456789	Smith	David	123 College road	11790	5162152345
222222222	Du	Victor	456 Fortune Road	11790	5166324360
333333333	Smith	John	789 Peace Blvd.	93536	3154434321
444444444	Philip	Lewis	135 Knowledge Lane	11794	5166668888
789123456	Warren	David	456 Sunken Street	11794	6316329987
NULL	NULL	NULL	NULL	NULL	NULL

ZipCode	City	State
11790	Stony Brook	NY
11794	Stony Brook	NY
93536	Los Angeles	CA
NULL	NULL	NULL

EditCustomer:

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `EditCustomer`(
    IN CustomerId INTEGER,
        LastName CHAR(20),
        FirstName CHAR(20),
        Address CHAR(20),
        City CHAR(20),
        State CHAR(20),
        ZipCode INTEGER,
        Telephone BIGINT,
        Email CHAR(32),
        CreditCard BIGINT,
        Rating INTEGER
    )
BEGIN
        Update `person`
        SET `person`.LastName = LastName,
            `person`.FirstName = FirstName,
            `person`.Address = Address,
            `person`.ZipCode = ZipCode,
            `person`.TelePhone = LastName,
            `person`.SSN = CustomerId
        WHERE `person`.SSN = CustomerId;
        Update `customer`
        SET `customer`.Email = Email,
            `customer`.Rating = Rating,
            `customer`.CreditCardNumber = CreditCard
        Where `customer`.Id = CustomerId;
        INSERT INTO `Location`(City,State,ZipCode)
        VALUES(City,State,ZipCode);

END
```



Output:(change customer id with 111111111's info all to 123)

Result Grid | Filter Rows: | Edit:

Id	Email	Rating	CreditCardNumber
111111111	123	123	123
222222222	vicdu@cs.sunysb.edu	1	5678123456781234
333333333	jsmith@ic.sunysb.edu	1	2345678923456789
444444444	pml@cs.sunysb.edu	1	6789234567892345
NULL	NULL	NULL	NULL

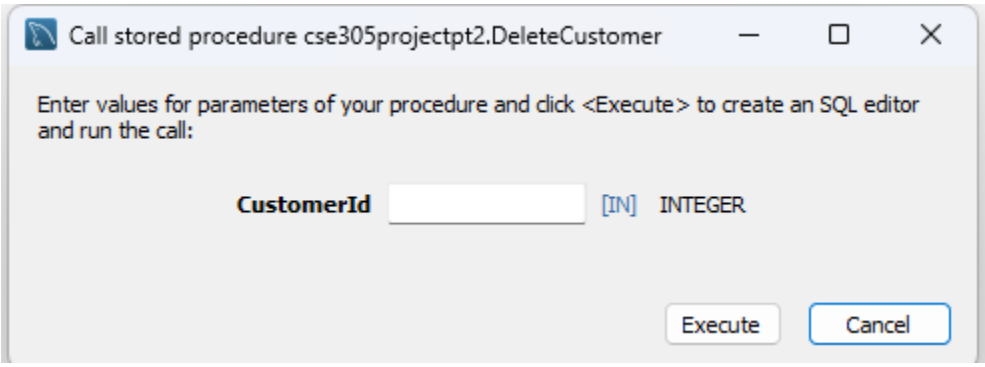
SSN	LastName	FirstName	Address	ZipCode	Telephone
111111111	123	123	123	123	123
123456789	Smith	David	123 College road	11790	5162152345
222222222	Du	Victor	456 Fortune Road	11790	5166324360
333333333	Smith	John	789 Peace Blvd.	93536	3154434321
444444444	Philip	Lewis	135 Knowledge Lane	11794	5166668888
789123456	Warren	David	456 Sunken Street	11794	6316329987
NULL	NULL	NULL	NULL	NULL	NULL

ZipCode	City	State
123	123	123
11790	Stony Brook	NY
11794	Stony Brook	NY
93536	Los Angeles	CA
NULL	NULL	NULL

Delete customer:

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `DeleteCustomer`(
    IN CustomerId INTEGER
    )
BEGIN
        DELETE FROM `customer` WHERE (`Id` = CustomerId);
        DELETE FROM `person` WHERE (`SSN` = CustomerId);

    END
```



Output:(we delete customer id with 111111111)

Id	Email	Rating	CreditCardNumber
222222222	vicdu@cs.sunysb.edu	1	5678123456781234
333333333	jsmith@ic.sunysb.edu	1	2345678923456789
444444444	pml@cs.sunysb.edu	1	6789234567892345
NULL	NULL	NULL	NULL

Produce mailing list:

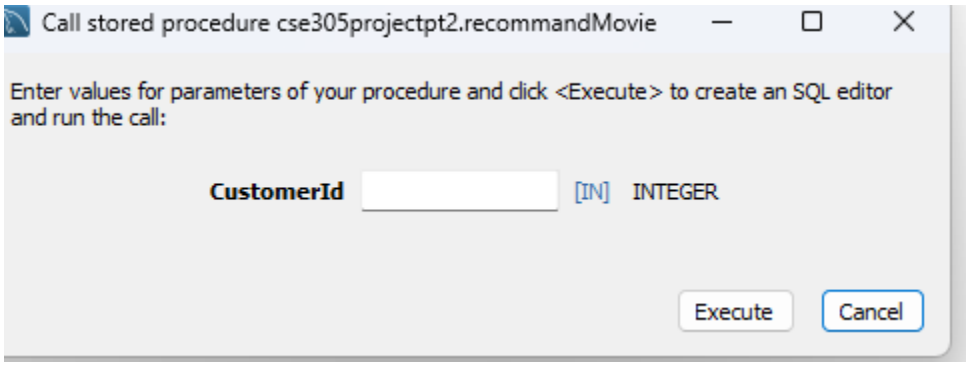
```
CREATE DEFINER=`root`@`localhost` PROCEDURE `CustomerMailingList`()
BEGIN
        SELECT SSN, LastName, FirstName, Email, Address, City, State, `location`.ZipCode AS ZipCode
        From `customer`,`person`,`location`
        WHERE `customer`.Id = `person`.SSN AND `person`.ZipCode = `location`.ZipCode;
    END
```

Output:

SSN	LastName	FirstName	Email	Address	City	State	ZipCode
111111111	Yang	Shang	syang@cs.sunysb.edu	123 Success Street	Stony Brook	NY	11790
222222222	Du	Victor	vicdu@cs.sunysb.edu	456 Fortune Road	Stony Brook	NY	11790
333333333	Smith	John	jsmith@ic.sunysb.edu	789 Peace Blvd.	Los Angeles	CA	93536
444444444	Philip	Lewis	pml@cs.sunysb.edu	135 Knowledge Lane	Stony Brook	NY	11794

Produce movie suggestion:

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `recommandMovie`(
        IN CustomerId INTEGER)
BEGIN
        SELECT DISTINCT `Name`
        From `account`,`rental`,`movie`
        WHERE CustomerId = `account`.Customer AND  `rental`.accountId = `account`.Id AND `rental`.movieId != `movie`.Id ;
    END
```



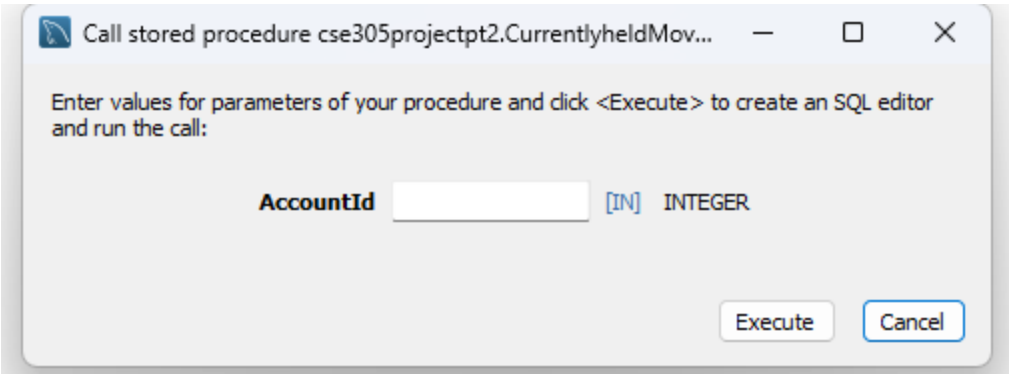
Output:(for customer id is 444444444)

Name
The Godfather
Shawshank Redemption
Borat

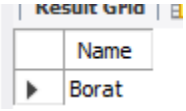
3.3

Customer currently held movies:

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `CurrentlyheldMovies`(
        IN AccountId INTEGER)
BEGIN
        SELECT `Name`
        From `rental`,`movie`,`order`
        WHERE AccountId=`rental`.AccountId AND `order`.Id=`rental`.OrderId  AND `movie`.Id=`rental`.MovieId  AND isnull(`order`.ReturnDate);
    END
```



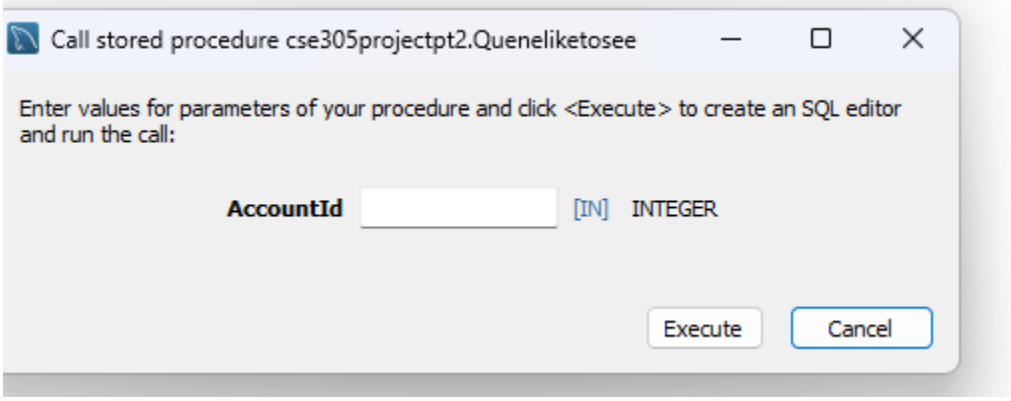
Output:



	Name
▶	Borat

customer's queue of movies it would like to see:

```
1  CREATE DEFINER=`root`@`localhost` PROCEDURE `Queneliketosee`(
2          IN AccountId INTEGER)
3  BEGIN
4          SELECT `Name`
5          From `movie`,`movieq`
6          WHERE AccountId=`movieq`.AccountId  AND `movie`.Id=`movieq`.MovieId;
7      END
```

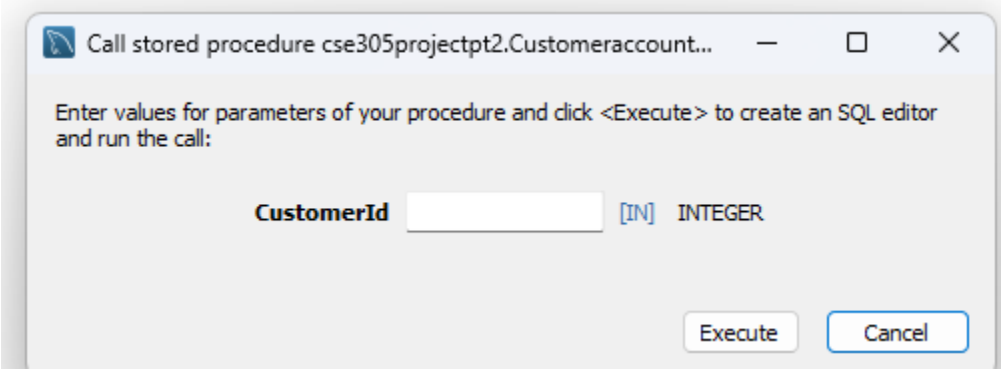


Output:

Name
Shawshank Redemption

customer's account settings:

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `CustomeraccountSetting`(
        IN CustomerId INTEGER)
BEGIN
        SELECT *
        From `account`
        WHERE `account`.Customer=CustomerId;
END
```



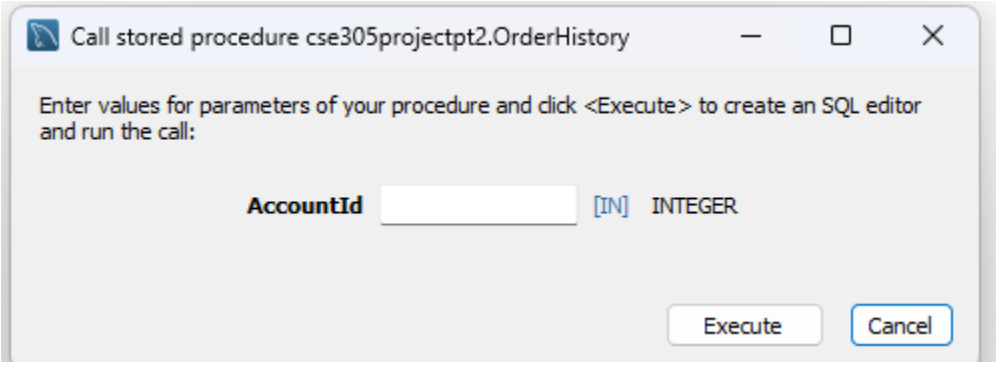
Output:

Result Grid | Filter Rows: | Expor

Id	DateOpened	Type	Customer
3	2010-06-15	limited	222222222
4	2010-06-15	limited	222222222

history of all current and past orders a customer has placed:

```
1  CREATE DEFINER=`root`@`localhost` PROCEDURE `OrderHistory`(
2          IN AccountId INTEGER)
3  BEGIN
4          SELECT distinct `order`.*
5          From `rental`,`movie`,`order`
6          WHERE `order`.Id=`rental`.OrderId AND `rental`.AccountId = AccountId;
7  END
```



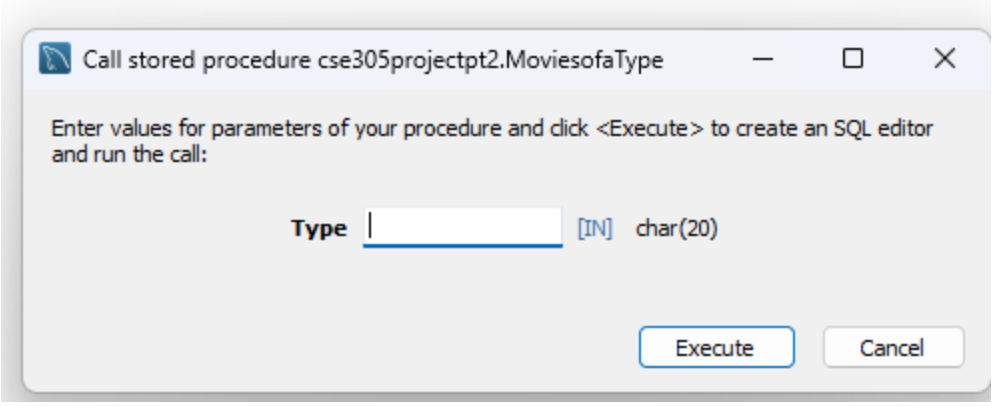
Output:

Result Grid | Filter Rows: | Export:

Id	DateTime	ReturnDate
3	2011-09-12 09:30:00	NULL
1	2011-09-11 10:00:00	2011-09-14

Movies available of a particular type:

```
1  CREATE DEFINER=`root`@`localhost` PROCEDURE `MoviesofaType`(
2          IN `Type` char(20))
3  BEGIN
4          SELECT *
5          From `movie`
6          WHERE `movie`.`Type` = `Type`;
7  END
```

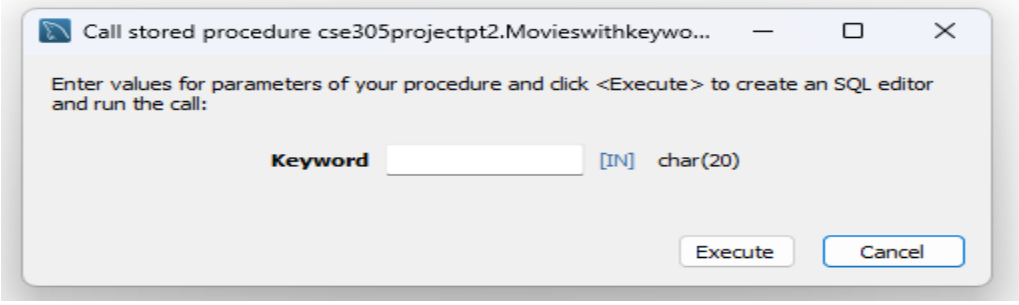


Output:

Id	Name	Type	Rating	DistrFee	NumCopies
1	The Godfather	Drama	5	10000	3
2	Shawshank Redemption	Drama	4	1000	2

Movies available with a particular keyword or set of keywords in the movie name:

```
1  CREATE DEFINER=`root`@`localhost` PROCEDURE `MovieswithkeywordSearch`(
2          IN Keyword char(20))
3  BEGIN
4          SELECT o1.* From movie as o1
5          where o1.Name in
6          (select Name from movie where `Name` REGEXP (CONCAT('.*',Keyword,'.*')) );
7  END
```

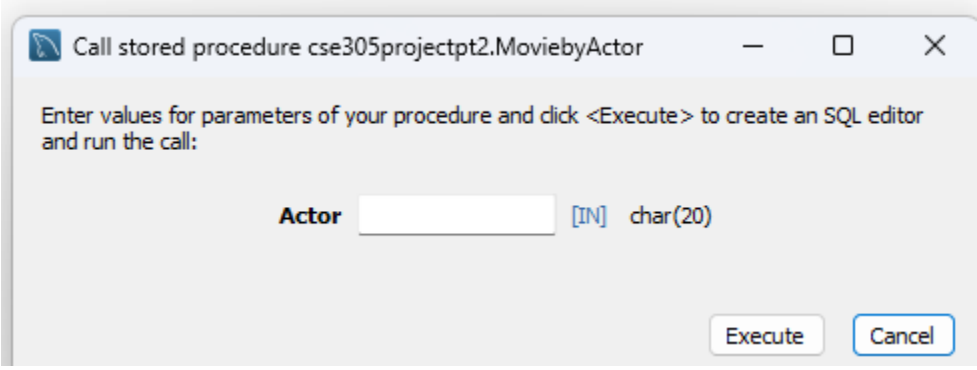


Output:(with keyword “the”)

	Id	Name	Type	Rating	DistrFee	NumCopies
▶	1	The Godfather	Drama	5	10000	3

Movies available starring a particular actor or group of actors:

```
1  CREATE DEFINER=`root`@`localhost` PROCEDURE `MoviebyActor`(
2          IN Actor char(20))
3  BEGIN
4          SELECT movie.`Name`
5          From actor, movie, appearedin
6          WHERE actor.Id = appearedin.ActorId and actor.Name = Actor and appearedin.MovieId = movie.Id;
7  END
```



Output:(Al Pacino)

	Name
▶	The Godfather
	Borat

Best-Seller list of movies:

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `bestSeller`(
        )
BEGIN
        SELECT count(rental.OrderId), movie.`Name`
        From rental, movie
        Where movie.Id = rental.MovieId
        Group by movie.`Name`
        Order by count(rental.OrderId) desc;
END
```

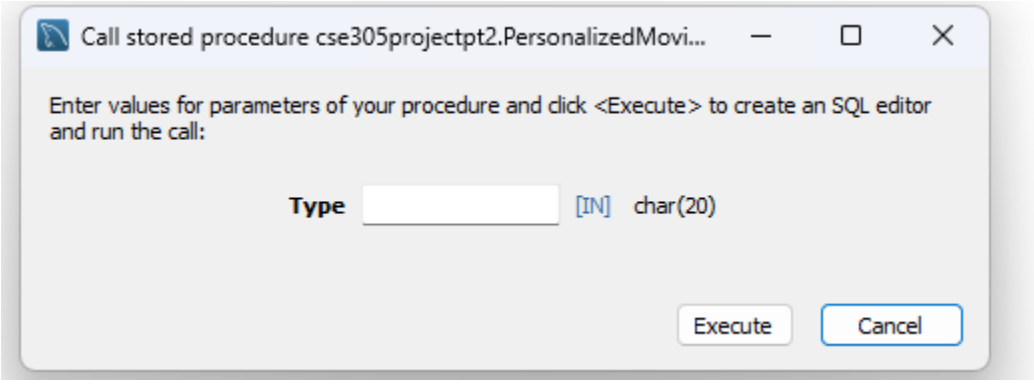
Output:

Result Grid | Filter Rows: | Export:

count(rental.OrderId)	Name
2	Borat
1	The Godfather
1	Shawshank Redemption

Personalized movie suggestion list:

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `PersonalizedMovieSuggestionlist`(
        IN `Type` char(20))
BEGIN
        SELECT *
        From `movie`
        WHERE `movie`.`Type` = `Type`;
END
```



Output:(Drama)

Id	Name	Type	Rating	DistrFee	NumCopies
1	The Godfather	Drama	5	10000	3
2	Shawshank Redemption	Drama	4	1000	2

Queries in text (left for our sake)

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `AddCustomer`(
   IN CustomerId INTEGER,
                  LastName CHAR(20),
      FirstName CHAR(20),
      Address CHAR(20),
      City CHAR(20),
      State CHAR(20),
      ZipCode INTEGER,
      Telephone BIGINT,
      Email CHAR(32),
      CreditCard BIGINT,
      Rating INTEGER
   )
BEGIN
        INSERT INTO `person`(SSN,LastName,FirstName,Address,ZipCode,Telephone)
   VALUES(CustomerId,LastName,FirstName,Address,ZipCode,Telephone);
        INSERT INTO `customer`(Id,Email,Rating,CreditCardNumber)
   VALUES(CustomerId,Email,Rating,CreditCard);
   INSERT INTO `Location`(City,State,ZipCode)
   VALUES(City,State,ZipCode);

END



CREATE DEFINER=`root`@`localhost` PROCEDURE `CustomerMailingList`()
BEGIN
                  SELECT SSN, LastName, FirstName, Email, Address, City, State,
`location`.ZipCode AS ZipCode
      From `customer`,`person`,`location`
      WHERE `customer`.Id = `person`.SSN AND `person`.ZipCode = `location`.ZipCode;
END



CREATE DEFINER=`root`@`localhost` PROCEDURE `DeleteCustomer`(
   IN CustomerId INTEGER
   )
```

```
BEGIN
		DELETE FROM `customer` WHERE (`Id` = CustomerId);
	DELETE FROM `person` WHERE (`SSN` = CustomerId);

END


CREATE DEFINER=`root`@`localhost` PROCEDURE `EditCustomer`(
  IN CustomerId INTEGER,
		LastName CHAR(20),
	FirstName CHAR(20),
	Address CHAR(20),
	City CHAR(20),
	State CHAR(20),
	ZipCode INTEGER,
	Telephone BIGINT,
	Email CHAR(32),
	CreditCard BIGINT,
	Rating INTEGER
  )
BEGIN
		Update `person`
	SET `person`.LastName = LastName,
			`person`.FirstName = FirstName,
			`person`.Address = Address,
			`person`.ZipCode = ZipCode,
			`person`.TelePhone = LastName,
			`person`.SSN = CustomerId
	WHERE `person`.SSN = CustomerId;
	Update `customer`
	SET `customer`.Email = Email,
			`customer`.Rating = Rating,
	   `customer`.CreditCardNumber = CreditCard
	Where `customer`.Id = CustomerId;
	INSERT INTO `Location`(City,State,ZipCode)
		VALUES(City,State,ZipCode);

END



CREATE DEFINER=`root`@`localhost` PROCEDURE `recommandMovie`(
		IN CustomerId INTEGER)
BEGIN
```

```
                SELECT DISTINCT `Name`
    From `account`,`rental`,`movie`
    WHERE CustomerId = `account`.Customer AND  `rental`.accountId = `account`.Id AND
`rental`.movieId != `movie`.Id ;
END


CREATE DEFINER=`root`@`localhost` PROCEDURE `RecordOrder`(
  IN OrderId INTEGER,
                AccountId INTEGER,
                MovieId INTEGER,
    CustRepId INTEGER,
    `DateTime` DateTime,
    ReturnDate Date
  )
BEGIN
        INSERT INTO `order` (`DateTime`,ReturnDate,Id)
  VALUES (`DateTime`,ReturnDate,OrderId);
  INSERT INTO `rental` (CustRepId,AccountId,MovieId,OrderId)
  VALUES (CustRepId,AccountId,MovieId,OrderId);
END
```

3.3

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `CurrentlyheldMovies`(
                IN AccountId INTEGER)
BEGIN
                SELECT `Name`
    From `rental`,`movie`,`order`
    WHERE AccountId=`rental`.AccountId AND `order`.Id=`rental`.OrderId  AND
`movie`.Id=`rental`.MovieId  AND isnull(`order`.ReturnDate);
END


CREATE DEFINER=`root`@`localhost` PROCEDURE `Queneliketosee`(
                IN AccountId INTEGER)
BEGIN
                SELECT `Name`
    From `movie`,`movieq`
    WHERE AccountId=`movieq`.AccountId  AND `movie`.Id=`movieq`.MovieId;
END
```

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `CustomeraccountSetting`(
		IN CustomerId INTEGER)
BEGIN
		SELECT *
	From `account`
	WHERE `account`.Customer=CustomerId;
END


CREATE DEFINER=`root`@`localhost` PROCEDURE `OrderHistory`(
		IN AccountId INTEGER)
BEGIN
		SELECT distinct `order`.*
	From `rental`,`movie`,`order`
	WHERE `order`.Id=`rental`.OrderId AND `rental`.AccountId = AccountId;
END


CREATE DEFINER=`root`@`localhost` PROCEDURE `MoviesofaType`(
		IN `Type` char(20))
BEGIN
		SELECT *
	From `movie`
	WHERE `movie`.`Type` = `Type`;
END


CREATE DEFINER=`root`@`localhost` PROCEDURE `MovieswithkeywordSearch`(
		IN Keyword char(20))
BEGIN
		SELECT o1.* From movie as o1
	where o1.Name in
	(select Name from movie where `Name` REGEXP (CONCAT('.*',Keyword,'.*')) );
END



CREATE DEFINER=`root`@`localhost` PROCEDURE `MoviebyActor`(
```