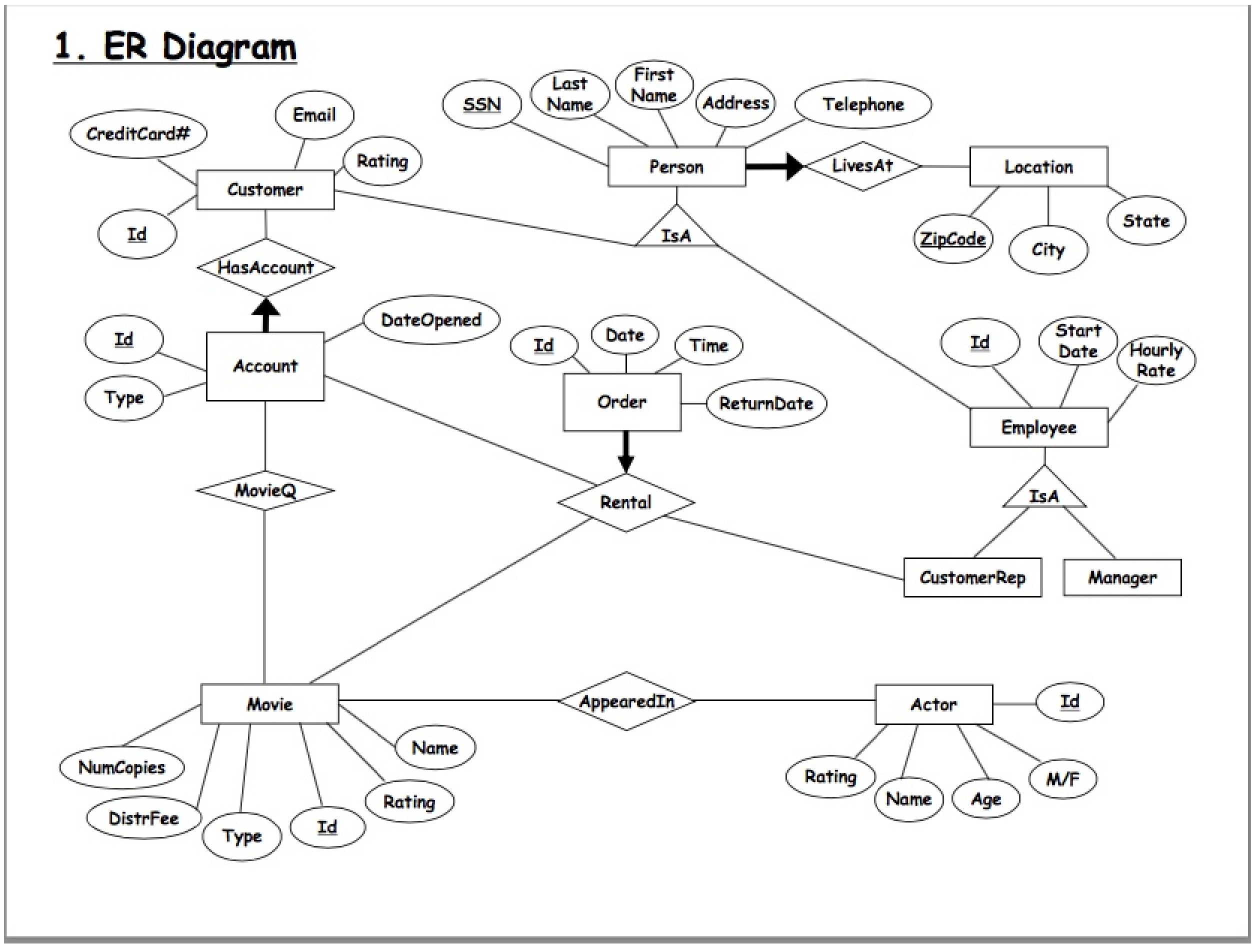
Sunflix
Programmer's Guide

Aaron Lin
Adam Guerin
Daniel Khuu

1 ER Diagram:


1. ER Diagram
CreditCard#
Email
Rating
Customer
Id
HasAccount
SSN
Last Name
First Name
Address
Telephone
Person
LivesAt
Location
ZipCode
City
State
IsA
Account
Id
Type
DateOpened
Id
Date
Time
Order
ReturnDate
Id
Start Date
Hourly Rate
Employee
MovieQ
Rental
IsA
CustomerRep
Manager
Movie
AppearedIn
Actor
Id
NumCopies
DistrFee
Type
Id
Rating
Name
Rating
Name
Age
M/F

2 Relational Model

```
CREATE TABLE `Account` (
  `Id` int(11) NOT NULL AUTO_INCREMENT,
  `DateOpened` date DEFAULT NULL,
  `Type` varchar(20) DEFAULT NULL,
  `Customer` varchar(11) DEFAULT NULL,
  PRIMARY KEY (`Id`),
  KEY `Customer` (`Customer`),
  CONSTRAINT `Account_ibfk_1` FOREIGN KEY (`Customer`) REFERENCES `Customer` (`Id`) ON DELETE NO ACTION
ON UPDATE CASCADE
)

CREATE TABLE `Actor` (
  `Id` int(11) NOT NULL DEFAULT '0',
  `Name` varchar(20) NOT NULL,
  `Age` int(11) NOT NULL,
  `Gender` char(1) NOT NULL,
  `Rating` int(11) DEFAULT NULL,
  PRIMARY KEY (`Id`)
)


DROP TABLE IF EXISTS `AppearedIn`;

CREATE TABLE `AppearedIn` (
  `ActorId` int(11) NOT NULL DEFAULT '0',
  `MovieId` int(11) NOT NULL DEFAULT '0',
  PRIMARY KEY (`ActorId`,`MovieId`),
  KEY `MovieId` (`MovieId`),
  CONSTRAINT `AppearedIn_ibfk_1` FOREIGN KEY (`ActorId`) REFERENCES `Actor` (`Id`) ON DELETE NO ACTION ON
UPDATE CASCADE,
  CONSTRAINT `AppearedIn_ibfk_2` FOREIGN KEY (`MovieId`) REFERENCES `Movie` (`Id`) ON DELETE NO ACTION
ON UPDATE CASCADE
)


DROP TABLE IF EXISTS `Customer`;

CREATE TABLE `Customer` (
  `Id` varchar(11) NOT NULL DEFAULT '',
  `Email` varchar(40) DEFAULT NULL,
  `Rating` int(11) DEFAULT NULL,
  `CreditCardNumber` varchar(19) DEFAULT NULL,
  PRIMARY KEY (`Id`),
  CONSTRAINT `Customer_ibfk_1` FOREIGN KEY (`Id`) REFERENCES `Person` (`SSN`) ON DELETE NO ACTION ON
UPDATE CASCADE
) ;


DROP TABLE IF EXISTS `Employee`;
```

```
CREATE TABLE `Employee` (
  `ID` int(11) NOT NULL DEFAULT '0',
  `SSN` varchar(11) DEFAULT NULL,
  `StartDate` date DEFAULT NULL,
  `HourlyRate` int(11) DEFAULT NULL,
  `ManagerStatus` tinyint(1) NOT NULL,
  PRIMARY KEY (`ID`),
  KEY `SSN` (`SSN`),
  CONSTRAINT `Employee_ibfk_1` FOREIGN KEY (`SSN`) REFERENCES `Person` (`SSN`) ON DELETE NO ACTION ON
UPDATE CASCADE
)


DROP TABLE IF EXISTS `Location`;

CREATE TABLE `Location` (
  `ZipCode` int(11) NOT NULL DEFAULT '0',
  `City` varchar(20) NOT NULL,
  `State` varchar(20) NOT NULL,
  PRIMARY KEY (`ZipCode`)
)


DROP TABLE IF EXISTS `Movie`;

CREATE TABLE `Movie` (
  `Id` int(11) NOT NULL DEFAULT '0',
  `Name` varchar(20) NOT NULL,
  `Type` varchar(20) NOT NULL,
  `Rating` int(11) DEFAULT NULL,
  `DistrFee` int(11) DEFAULT NULL,
  `NumCopies` int(11) DEFAULT NULL,
  PRIMARY KEY (`Id`)
)


DROP TABLE IF EXISTS `MovieQ`;

CREATE TABLE `MovieQ` (
  `AccountId` int(11) NOT NULL DEFAULT '0',
  `MovieId` int(11) NOT NULL DEFAULT '0',
  PRIMARY KEY (`AccountId`,`MovieId`),
  KEY `MovieId` (`MovieId`),
  CONSTRAINT `MovieQ_ibfk_1` FOREIGN KEY (`AccountId`) REFERENCES `Account` (`Id`) ON DELETE NO ACTION
ON UPDATE CASCADE,
  CONSTRAINT `MovieQ_ibfk_2` FOREIGN KEY (`MovieId`) REFERENCES `Movie` (`Id`) ON DELETE NO ACTION ON
UPDATE CASCADE
)


DROP TABLE IF EXISTS `Order`;

CREATE TABLE `Order` (
  `Id` int(11) NOT NULL DEFAULT '0',
  `DateTime` datetime DEFAULT NULL,
  `ReturnDate` date DEFAULT NULL,
  PRIMARY KEY (`Id`)
)


DROP TABLE IF EXISTS `Person`;

CREATE TABLE `Person` (
  `SSN` varchar(11) NOT NULL DEFAULT '',
  `LastName` varchar(20) NOT NULL,
  `FirstName` varchar(20) NOT NULL,
  `Address` varchar(20) DEFAULT NULL,
  `ZipCode` int(11) DEFAULT NULL,
  `Telephone` varchar(12) DEFAULT NULL,
  PRIMARY KEY (`SSN`),
  KEY `ZipCode` (`ZipCode`),
  CONSTRAINT `Person_ibfk_1` FOREIGN KEY (`ZipCode`) REFERENCES `Location` (`ZipCode`) ON DELETE NO ACTION
ON UPDATE CASCADE
)


DROP TABLE IF EXISTS `Rental`;

CREATE TABLE `Rental` (
  `AccountId` int(11) NOT NULL DEFAULT '0',
  `CustRepId` int(11) NOT NULL DEFAULT '0',
  `OrderId` int(11) NOT NULL DEFAULT '0',
  `MovieId` int(11) NOT NULL DEFAULT '0',
  PRIMARY KEY (`AccountId`,`CustRepId`,`OrderId`,`MovieId`),
  KEY `CustRepId` (`CustRepId`),
  KEY `OrderId` (`OrderId`),
  KEY `MovieId` (`MovieId`),
  CONSTRAINT `Rental_ibfk_1` FOREIGN KEY (`AccountId`) REFERENCES `Account` (`Id`) ON DELETE NO ACTION ON
UPDATE CASCADE,
  CONSTRAINT `Rental_ibfk_2` FOREIGN KEY (`CustRepId`) REFERENCES `Employee` (`ID`) ON DELETE NO ACTION
ON UPDATE CASCADE,
  CONSTRAINT `Rental_ibfk_3` FOREIGN KEY (`OrderId`) REFERENCES `Order` (`Id`) ON DELETE NO ACTION ON
UPDATE CASCADE,
  CONSTRAINT `Rental_ibfk_4` FOREIGN KEY (`MovieId`) REFERENCES `Movie` (`Id`) ON DELETE NO ACTION ON
UPDATE CASCADE
)


DROP TABLE IF EXISTS `Users`;

CREATE TABLE `Users` (
  `username` varchar(20) NOT NULL DEFAULT '',
  `password` varchar(20) DEFAULT NULL,
  `ssn` varchar(20) DEFAULT NULL,
  PRIMARY KEY (`username`)
)
```

3 SQL

3.1 Manager-Level Transactions

1. Add, Edit and Delete movies

- **ADD**

```
INSERT INTO Movie(Id, Name, Type, Rating, DistrFee, NumCopies) VALUES (?, ?, ?, ?, ?,
?);

INSERT INTO Movie(Id, Name, Type, Rating, DistrFee, NumCopies) VALUES (4, 'Star Wars
Rogue One', 'Action', 5, 9001, 10);
```

- **EDIT**

```
UPDATE Movie
SET ? = ?
WHERE Id = ?;

UPDATE Movie
SET NumCopies = 20
WHERE id = 1;
```

- **DELETE**

```
DELETE FROM AppearedIn
WHERE MovieId = ?;

DELETE FROM MovieQ
WHERE MovieId = ?;

DELETE FROM Rental
WHERE MovieId = ?

DELETE FROM Movie
WHERE Id = ?;

DELETE FROM AppearedIn
WHERE MovieId = 3;

DELETE FROM MovieQ
WHERE MovieId = 3;

DELETE FROM Rental
WHERE MovieId = 3;

DELETE FROM Movie
WHERE Id = 3;
```

2. Add, Edit, and Delete information from an employee

- **ADD**

```
INSERT INTO Employee (ID, SSN, StartDate, ManagerStatus) VALUES (?, ?, ?, ?)
```

- **EDIT**

```
UPDATE Employee
SET ? = ?
```

```
WHERE ID = ?;

UPDATE Employee
SET HourlyRate = 10
WHERE ID = 1;
```

- **DELETE**

```
DELETE FROM Employee WHERE SSN = ?
DELETE FROM Person WHERE SSN = ?
```

3. Obtain a sales report for a particular month

```
CREATE VIEW Overall_Income AS
SELECT Type, case when Type = 'limited' THEN 10
WHEN Type = 'unlimited-1' THEN 15
WHEN Type = 'unlimited-2' THEN 20
WHEN Type = 'unlimited-3' THEN 25
END AS SubFee FROM Account;


SELECT SUM(O.SubFee) AS Sales FROM Overall_Income O, Account A WHERE
A.DateOpened > ?;

DROP VIEW Overall_Income;
```

4. Produce a comprehensive listing of all movies

```
SELECT Id, Name, Type, Rating FROM Movie;
```

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

- **Movie name**

```
SELECT R.AccountId, R.CustRepId, R.OrderId, M.Name
from Movie M, Rental R
where M.Id = R.MovieId and M.Name = ?;


SELECT R.AccountId, R.CustRepId, R.OrderId, M.Name
from Movie M, Rental R
where M.Id = R.MovieId and M.Name = 'The Godfather';
```

- **Movie Type**

```
SELECT R.AccountId, R.CustRepId, R.OrderId, M.Name
from Movie M, Rental R
where M.Id = R.MovieId and M.Type = ?;

SELECT R.AccountId, R.CustRepId, R.OrderId, M.Name
from Movie M, Rental R
where M.Id = R.MovieId and M.Type = 'Comedy';
```

- **Customer Name**

```
SELECT R.AccountId, R.CustRepId, R.OrderId, M.Name
FROM Movie M, Rental R, Account A, Customer C, Person P
where M.Id = R.MovieId AND R.AccountId = A.Id AND A.Customer = C.Id AND C.Id = P.SSN
AND P.SSN = ?
```

6. Determine which customer representative oversaw the most transactions (rentals)

```
create view CustRepOrders as
SELECT R.CustRepId, COUNT(*) as NumRentals
FROM Rental R
GROUP BY R.CustRepId;
```

```
select O.CustRepId, O.NumRentals
From CustRepOrders O
where NumRentals = (SELECT MAX(NumRentals) FROM CustRepOrders);

Drop view CustRepOrders;
```

7. Product a list of most active customers

```
DROP VIEW IF EXISTS MostActiveCustomers;
CREATE VIEW MostActiveCustomers AS SELECT R.AccountId, COUNT(*) as NumOrders
FROM Rental R GROUP BY R.AccountId;
SELECT * FROM MostActiveCustomers GROUP BY NumOrders ORDER BY NumOrders DESC;
DROP VIEW MostActiveCustomers
```

8. Product a list of most actively rented movies

```
DROP VIEW IF EXISTS MovieOrder
CREATE VIEW MovieOrder(MovieId, NumOrders) AS SELECT R.MovieId, COUNT(R.MovieId) FROM
Rental R GROUP BY R.MovieId;
SELECT M.ID, M.Name, M.Rating, O.NumOrders FROM MovieOrder O, Movie M
WHERE O.MovieId = M.ID AND O.NumOrders >= (SELECT MAX(R.NumOrders FROM MovieOrder R);
DROP VIEW MovieOrder
```

3.2 Customer-Representative-Level Transactions

1. Record an Order

```
INSERT INTO `Order` (Id, DateTime, ReturnDate) VALUES (? , ? , ?);
INSERT INTO Rental (AccountId, CustRepId, OrderId, MovieId) VALUES (? ,  ? ,  ? , ?);

INSERT INTO `Order` (Id, DateTime, ReturnDate) VALUES (10, '2017-03-18 10:00:00',
'2017-03-18');
INSERT INTO Rental (AccountId, CustRepId, OrderId, MovieId) VALUES (1 ,  1 ,  10 , 2);
```

The proper format for DateTime is `YYYY-MM-DD HH:MM:SS`
The proper format for ReturnDate is `YYYY-MM-DD`

2. Add, Edit and Delete information for a customer

- **ADD**

```
INSERT INTO Location( ZipCode, City, State) VALUES (?, ?, ?)
INSERT INTO Person (SSN, LastName, FirstName, Address, ZipCode, Telephone) VALUES (?,
?, ?, ?, ?, ?);
INSERT INTO Customer(Id, Email, Rating, CreditCardNumber) VALUES (?, ?, ?, ?)
INSERT INTO Account (DateOpened, Type, Customer) VALUES (?, ?, ?)
```

- **EDIT**

```
UPDATE Customer
SET ? = ?
WHERE Id = ?;

UPDATE Customer
SET Email = "edited@email.com"
WHERE Id = '111-11-1111';
```

- **DELETE**

```
UPDATE Customer
SET ? = null
WHERE Id = ?;

UPDATE Customer
SET CreditCardNumber = null
WHERE Id = '111-11-1111';
```

3. Produce customer mailing lists

Definition: Get a list of emails from all customers.

```
select P.SSN, P.FirstName, P.LastName, P.Address, C.Email FROM Person P, Customer C
WHERE P.SSN = C.Id
```

4. 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)

```
DROP VIEW IF EXISTS PastOrder;
CREATE VIEW PastOrder(CustId, MovieId, MovieType) AS SELECT A.Customer, R.MovieId,
M.Type FROM Account A, Rental R, Movie M WHERE A.Id = R.AccountId AND R.MovieId = M.Id
SELECT M.Id, M.Name, M.Type FROM Movie M WHERE M.Type IN (SELECT O.MovieType FROM
PastOrder O WHERE O.CustId = ?
AND M.Id NOT IN (SELECT O.MovieId FROM PastOrder O
WHERE O.CustId = ?
DROP VIEW PastOrder
```

3.3 Customer-Level Transactions

1. A customer's currently held movies

```
create view Currently_Held as
select R.AccountId, R.MovieId, O.DateTime, O.ReturnDate
from Rental R, `Order` O
where R.OrderId = O.Id AND R.AccountId = ?;

select * from Currently_Held where CURDATE() < ReturnDate;
DROP VIEW Currently_Held;

create view Currently_Held as
select R.AccountId, R.MovieId, O.DateTime, O.ReturnDate
from Rental R, `Order` O
where R.OrderId = O.Id AND R.AccountId = 1;

select * from Currently_Held where CURDATE() < ReturnDate;
DROP VIEW Currently_Held;
```

2. A customer's queue of movies it would like to see

Definition: A customer will be able to see what movies she has in her queue.

```
select * from MovieQ where AccountId = ?;

select * from MovieQ where AccountId = 1;
```

3. A customer's account settings

```
SELECT A.Id, A.DateOpened, A.Type, C.Email, C.CreditCardNumber FROM
Customer C, Account A where A.Customer = C.Id AND C.Id = ?
```

4. A history of all current and past orders a customer has placed

Definition: A customer will be able to get an order log of current and past orders.

```
select R.AccountId, R.MovieId, O.DateTime, O.ReturnDate
from Rental R, `Order` O
where R.OrderId = O.Id AND R.AccountId = ?;

select R.AccountId, R.MovieId, O.DateTime, O.ReturnDate
from Rental R, `Order` O
where R.OrderId = O.Id AND R.AccountId = 1;
```

5. Movies available of a particular type

```
Select * from Movie where Type = ?;

Select * from Movie where Type = 'Comedy';
```

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

```
select * from Movie WHERE name REGEXP ?;

select * from Movie WHERE name REGEXP 'God';
```

7. Movies available starring a particular actor or a group of actors.

```
DROP VIEW IF EXISTS ActorId_Movie
DROP VIEW IF EXISTS Actors_Movie
CREATE VIEW ActorId_Movie AS
select M.Name, M.Type, M.DistrFee, M.NumCopies, M.Rating, A.ActorId
from Movie M, AppearedIn A
where M.Id = A.MovieId;

create view Actors_Movie as
select AM.Name as MovieName, AM.Type, AM.DistrFee, AM.NumCopies, AM.Rating, A.Name as
ActorName
from ActorId_Movie AM, Actor A
where AM.ActorId = A.Id;


select *
from Actors_Movie
where NumCopies > 0 AND ActorName = ?;

DROP VIEW ActorId_Movie;
DROP VIEW Actors_Movie;


CREATE VIEW ActorId_Movie AS
select M.Name, M.Type, M.DistrFee, M.NumCopies, M.Rating, A.ActorId
from Movie M, AppearedIn A
where M.Id = A.MovieId;

create view Actors_Movie as
select AM.Name as MovieName, AM.Type, AM.DistrFee, AM.NumCopies, AM.Rating, A.Name as
ActorName
from ActorId_Movie AM, Actor A
where AM.ActorId = A.Id;


select *
from Actors_Movie
where NumCopies > 0 AND ActorName = 'Al Pacino';

DROP VIEW ActorId_Movie;
DROP VIEW Actors_Movie;
```

8. Best-Seller list of movies

```
DROP VIEW IF EXISTS MovieOrder
```

```
CREATE VIEW MovieOrder(MovieId, NumOrders) AS SELECT R.MovieId, COUNT(R.MovieId) FROM
Rental R GROUP BY R.MovieId
SELECT M.Id, M.Name, M.Type, M.Rating, N.NumOrders FROM MovieOrder N, Movie M WHERE
N.MovieId = M.Id ORDER BY N.NumOrders DESC
DROP VIEW MovieOrder
```

9. Personalized movie suggestion list

```
DROP VIEW IF EXISTS PastOrder
CREATE VIEW PastOrder(CustId, MovieId, MovieType) AS SEleCT A.Customer, R.MovieId,
M.Type FROM Account A, Rental R, Movie M WHERE A.Id = R.AccountId AND R.MovieId = M.Id
SELECT M.Id, M.Name, M.Type FROM Movie M WHERE M.Type IN (SELECT O.MovieType FROM
PastOrder O WHERE O.CustId = ?
DROP VIEW PastOrder
```

10. Rate the movies they have rented

```
UPDATE Movie SET Rating = ? WHERE Name = ?

UPDATE Movie SET Rating = 1 WHERE Name = 'The Godfather';
```

4 User Interface Description

The front-end was written in HTML/CSS, and the back-end and APIs were written in Java.

The front-end interface utilizes the bootstrap library. It communicates to the back-end of the project via the REST api. The REST api utilizes the Spring Framework and the thymeleaf template engine to handle ajax calls.

The front-end calls a javascript function when a HTML form is submitted. The function converts the HTML form-fields into JSON and calls AJAX, which then sends JSON to the URL mapping implemented in the back-end. The mapping takes the JSON and puts it into a JSON object. The back-end grabs values based on keys from the JSON object and inserts them into SQL statements. The back-end converts the SELECT statements from the SQL statements into List<Map<String, Object>> objects, which are then converted into JSON objects and sent to the front-end.

For UPDATE, INSERTION, and DELETION statements, the back-end sends a response body object that contains done or error for the first field, and a JSON object in the second field. The front end then parses this object and stores the values onto a table to show to the user.

5 Architectural Diagram