

## CSE 305 Project Assignment 1

**Team:** Trappist-1f

**Members:**

Zhe Lin, 109369879, [zhe.lin@stonybrook.edu](mailto:zhe.lin@stonybrook.edu)

Sean Pesce, 107102508, [sean.pesce@stonybrook.edu](mailto:sean.pesce@stonybrook.edu)

Weichao Zhao, 109957656, [weichao.zhao@stonybrook.edu](mailto:weichao.zhao@stonybrook.edu)

**Collaboration Plan:**

Zhe Lin will work on movie and actor data.

Sean Pesce will work on account and order (rental) data.

Weichao Zhao will work on customer and employee data.

**Stage 1:**

Zhe created the tables for **Actor** and **Movie** entity type and Weichao created the tables for **Person**, **Customer**, and **Employee** entity type. Sean worked on the 3 relationship types **Queued**, **Casted**, and **Rented**.

**Stage 2:**

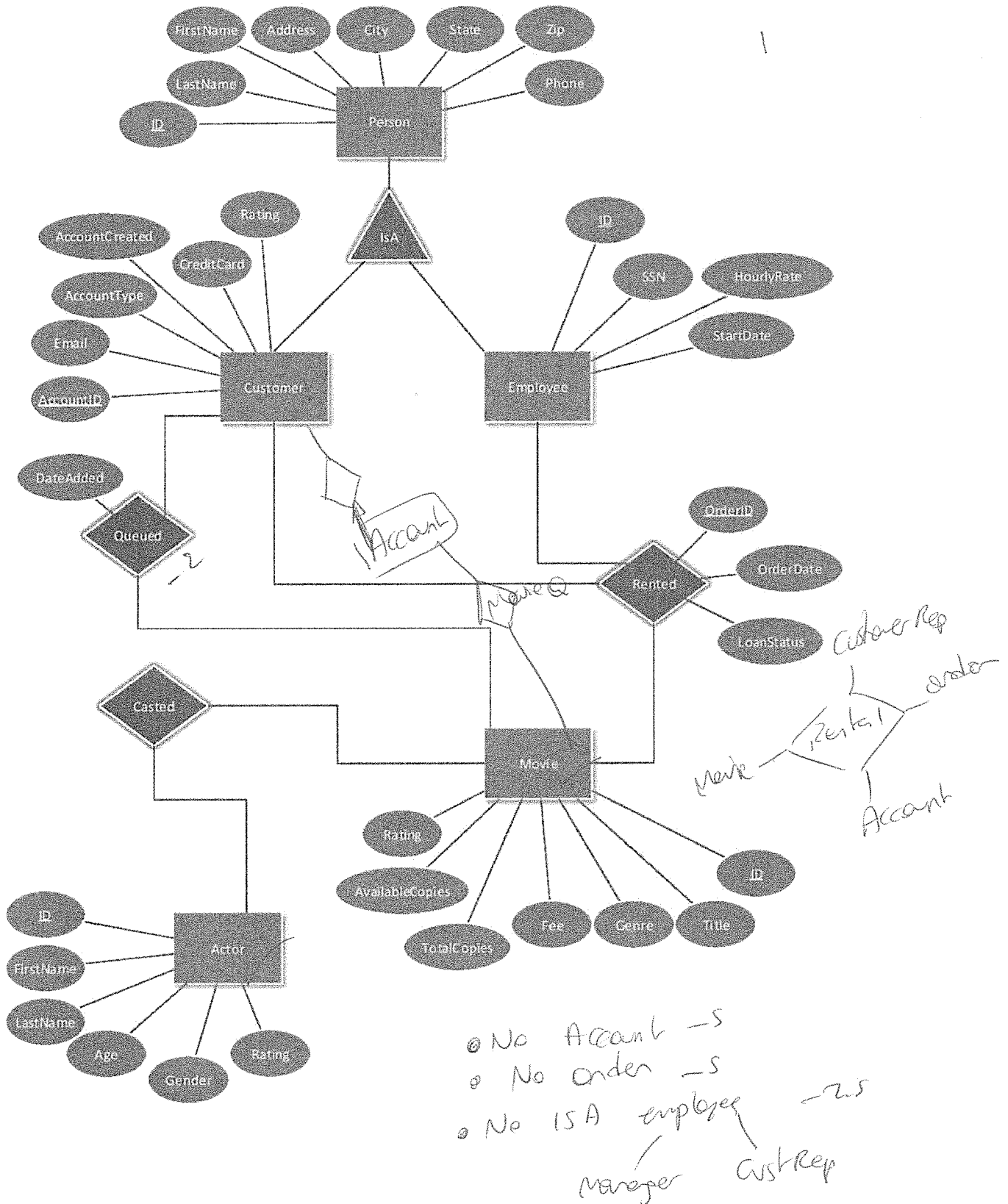
Sean checked and included the integrity constraints and then drew the ER diagram.

Weichao created the views for customer information (Movie Queued, RentalHistory, CurrentLoan). Zhe created the views for actor and movie information (CastList, Roles, ComedyMovie, etc). Sean and Zhe worked together for the triggers and procedures.

**Stage 3:**

Weichao finished the brief rationale for the E-R model and relational model.

# 1: E-R Diagram:



## **1.1 : Rationale for E-R Diagram:**

The online movie rental system have four major entity types: **Customer**, **Employee**, **Actor** and **Movie**. The attributes are shown on the diagram. Besides, Customer and Employee are inheritances of **Person** Entity Type so they both have attributes that **Person** has, such as FirstName, LastName, and primary key ID. (For Customer, ID is used as AccountID.) However, Actor, whose primary key is also ID, is not inheritance of **Person** since attributes like Zip or Address are not in **Actor** Entity Type, since those attributes are not required for **Actor**.

Those 4 major entity types are linked through 3 relationship types: **Queued**, **Rented** and **Casted**. **Queued**(Customer, Movie; DateAdded) relationship types relates the **Actor** and **Movie** with the DateAdded attribute. For example, Queued(John, "Superman"; 02/23/2017) describes a relationship that John added the movie "Superman" to his wish list on 02/23/2017.

**Rented**(Customer/Employee, Movie; OrderID, OrderDate, LoanStatus) relationship type relates **Customer/Employee** with **Movie** with attribute OrderDate. For example, John is the name of **Customer/Employee**, "Superman" name of **Movie**, then (John, "Superman"; 123, 02/23/2017, Active) describes a relationship that John rented a copy of "Superman" on 02/23/2017, and this order is active with id 123.

**Casted**(Actor, Movie): Queued relationship type relates an element of Actor entity type and an element of Movie entity type with attribute Role. For example, John is a value of **Actor** role, "Superman" a value of **Movie** role, then (John, "Superman") describes a relationship that John was in movie "Superman".

## 2. Relational Model (1):

CREATE TABLE Person (

*SSN* ID INT, ✓  
LastName VARCHAR(64),  
FirstName VARCHAR(64),  
Address VARCHAR(64),  
City VARCHAR(64),  
State ENUM('AK','AL','AR','AZ','CA','CO','CT',  
          'DE','FL','GA','HI','IA','ID','IL','IN','KS',  
          'KY','LA','MA','MD','ME','MI','MN','MO','MS',  
          'MT','NC','ND','NE','NH','NJ','NM','NV','NY',  
          'OH','OK','OR','PA','RI','SC','SD','TN','TX',  
          'UT','VA','VT','WA','WI','WV','WY',  
          'AS','DC','FM','GU','MH','MP','PR','PW','VI',  
          'AA','AE','AP'),  
Zip VARCHAR(10),  
Phone VARCHAR(20),  
PRIMARY KEY (ID), ✓  
UNIQUE KEY NameAddress (LastName, FirstName, Address, City, State, Zip),  
CONSTRAINT chk\_Zip CHECK (Zip RLIKE '^[0-9]{5}\$'),  
CONSTRAINT chk\_Phone CHECK (Phone RLIKE '^[0-9]{10}\$'));

CREATE TABLE Employee ( # IsA Person

ID INT, # References Person(ID) ✓  
SSN CHAR(9),  
StartDate DATE, # Start of employment  
HourlyRate FLOAT NOT NULL DEFAULT 20.00,  
PRIMARY KEY (ID), ✓  
FOREIGN KEY (ID) REFERENCES Person(ID) ✓  
ON DELETE CASCADE  
ON UPDATE CASCADE,  
UNIQUE KEY (SSN), # No two people have the same SSN  
CONSTRAINT chk\_Pay CHECK (HourlyRate >= 9.00),  
CONSTRAINT chk\_SSN CHECK (SSN RLIKE '^[0-9]{9}\$'));

## 2. Relational Model (2)

CREATE TABLE Customer (

✓ AccountID INT, # References Person(ID)

Email VARCHAR(64) NOT NULL,

AccountType ENUM('Limited', 'Unlimited', 'Unlimited+', 'Unlimited++') NOT NULL  
DEFAULT 'Limited',

AccountCreated DATE,

CreditCard CHAR(16) NOT NULL,

Rating INT DEFAULT 1,

✓ PRIMARY KEY (AccountID),

✓ FOREIGN KEY (AccountID) REFERENCES Person(ID)

ON DELETE CASCADE NO ACTION

ON UPDATE CASCADE,

UNIQUE KEY (Email),

CONSTRAINT chk\_Rating CHECK (Rating IN (1, 2, 3, 4, 5)),

CONSTRAINT chk\_Email CHECK (Email LIKE '%\_@\_%.\_%'),

CONSTRAINT chk\_CC CHECK (CreditCard RLIKE '[0-9]{16}\$')

);

CREATE TABLE Movie (

ID INT, ✓

Title VARCHAR(64) NOT NULL,

Genre ENUM('Comedy', 'Drama', 'Action', 'Foreign'),

Fee FLOAT DEFAULT 0.00,

TotalCopies INT DEFAULT 0,

AvailableCopies INT DEFAULT 0,

Rating INT,

PRIMARY KEY (ID), ✓

CONSTRAINT chk\_Rating CHECK (Rating IN (1, 2, 3, 4, 5)),

CONSTRAINT chk\_TotalCopies CHECK (TotalCopies >= 0),

✓ CONSTRAINT chk\_AvailableCopies CHECK (AvailableCopies >= 0),

CONSTRAINT chk\_AvailableVsTotalCopies CHECK (AvailableCopies <= TotalCopies),

CONSTRAINT chk\_Fee CHECK (Fee >= 0.0)

);

## 2. Relational Model (3)

CREATE TABLE Actor (

✓ ID INT,

FirstName VARCHAR(64) NOT NULL,

LastName VARCHAR(64) NOT NULL,

Gender ENUM('M','F'), # ENUM acts as a domain

Age INT,

✓ # Rating INT,

✓ PRIMARY KEY (ID),

# CONSTRAINT chk\_Rating CHECK (Rating IN (1, 2, 3, 4, 5)), # CONSTRAINT acts as a domain

CONSTRAINT chk\_Age CHECK (Age >= 0)

);

CREATE TABLE Casted (

✓ ActorID INT,

✓ MovieID INT,

Role VARCHAR(128),

ActorRating INT,

✓ PRIMARY KEY (ActorID, MovieID),

✓ FOREIGN KEY (ActorID) REFERENCES Actor(ID)

ON DELETE CASCADE

ON UPDATE CASCADE,

✓ FOREIGN KEY (MovieID) REFERENCES Movie(ID)

ON DELETE CASCADE

ON UPDATE CASCADE,

CONSTRAINT chk\_Rating CHECK (ActorRating IN (1, 2, 3, 4, 5))

);

CREATE TABLE Queued (

CustomerID INT, ✓

MovieID INT, ✓

DateAdded DATETIME,

PRIMARY KEY (CustomerID, MovieID), ✓

FOREIGN KEY (CustomerID) REFERENCES Customer(AccountID)

ON DELETE CASCADE ✓

ON UPDATE CASCADE,

FOREIGN KEY (MovieID) REFERENCES Movie(ID)

ON DELETE CASCADE ✓

ON UPDATE CASCADE );

NO ACTION

## 2. Relational Model (4)

CREATE TABLE Rented (

✓ OrderID INT,

✓ CustomerID INT NOT NULL,

✓ MovieID INT,

✓ EmployeeID INT,

✓ OrderDate DATETIME,

LoanStatus ENUM('Expired', 'Active') NOT NULL DEFAULT 'Active',

PRIMARY KEY (OrderID),

FOREIGN KEY (CustomerID) REFERENCES Customer(AccountID)

ON DELETE CASCADE

ON UPDATE CASCADE,

✓ FOREIGN KEY (MovieID) REFERENCES Movie(ID)

ON DELETE SET NULL

ON UPDATE CASCADE,

✓ FOREIGN KEY (EmployeeID) REFERENCES Employee(ID)

ON DELETE SET NULL

ON UPDATE CASCADE,

UNIQUE KEY (CustomerID, MovieID, OrderDate)

);

NO ACTION

## **2. Relational Model (5)**

### *Views of Movie Information:*

```
CREATE VIEW CastList (MovieID, ActorID, FirstName, LastName, Gender, Age, ActorRating) AS (  
    SELECT MovieID, ActorID, FirstName, LastName, Gender, Age, Actor.Rating  
    FROM (Actor JOIN Casted ON (Actor.ID = ActorID)) JOIN Movie ON (MovieID = Movie.ID)  
);
```

```
CREATE VIEW Roles (ActorID, MovieID, Title, Genre, MovieRating) AS (  
    SELECT ActorID, MovieID, Title, Genre, Movie.Rating  
    FROM (Actor JOIN Casted ON (Actor.ID = ActorID)) JOIN Movie ON (MovieID = Movie.ID)  
);
```

```
CREATE VIEW ComedyMovies (MovieId, Title, MovieRating, AvailableCopies, Fee) AS (  
    SELECT MovieID, ActorID, Title, MovieRating, AvailableCopies, Fee  
    FROM Movie  
    WHERE Genre = 'Comedy'  
);
```

```
CREATE VIEW DramaMovies (MovieId, Title, MovieRating, AvailableCopies, Fee) AS (  
    SELECT MovieID, ActorID, Title, MovieRating, AvailableCopies, Fee  
    FROM Movie  
    WHERE Genre = 'Drama'  
);
```

```
CREATE VIEW ActionMovies (MovieId, Title, MovieRating, AvailableCopies, Fee) AS (  
    SELECT MovieID, ActorID, Title, MovieRating, AvailableCopies, Fee  
    FROM Movie  
    WHERE Genre = 'Action'  
);
```

```
CREATE VIEW ForeignMovies (MovieId, Title, MovieRating, AvailableCopies, Fee) AS (  
    SELECT MovieID, ActorID, Title, MovieRating, AvailableCopies, Fee  
    FROM Movie  
    WHERE Genre = 'Foreign'  
);
```



## **2. Relational Model (6)**

*Views of Customer Information:*

```
CREATE VIEW MovieQueue (CustomerID, MovieID, DateAdded) AS (  
    SELECT CustomerID, MovieID, DateAdded  
    FROM Queued JOIN Customer ON (CustomerID = AccountID)  
    ORDER BY DateAdded ASC  
);
```

```
CREATE VIEW RentalHistory (CustomerID, MovieID, Title, Genre, Rating, OrderDate, LoanStatus) AS (  
    SELECT CustomerID, MovieID, Title, Genre, Movie.Rating, OrderDate, LoanStatus  
    FROM Rented JOIN Movie ON (MovieID = ID)  
    ORDER BY OrderDate DESC  
);
```

```
CREATE VIEW CurrentLoans (CustomerID, MovieID, Title, OrderDate) AS (  
    SELECT CustomerID, MovieID, Title, OrderDate  
    FROM Rented JOIN Movie ON (MovieID = ID)  
    WHERE LoanStatus = 'Active'  
);
```

## **2. Relational Model (7)**

DELIMITER \$\$

CREATE PROCEDURE CustomerExistsBeforeOrder (IN New\_CustomerID INT, New\_OrderDate DATETIME)

BEGIN

```
    IF DATE(New_OrderDate) < (SELECT AccountCreated
                              FROM Customer
                              WHERE New_CustomerID = AccountID
                              )
```

THEN

SIGNAL SQLSTATE 'E0451'

SET MESSAGE\_TEXT = 'Date conflict: Order date precedes Customer's account creation.';

END IF;

END;

\$\$

DELIMITER ;

DELIMITER \$\$

CREATE PROCEDURE EmployeeExistsBeforeOrder (IN New\_EmployeeID INT, New\_OrderDate DATETIME)

BEGIN

```
    IF DATE(New_OrderDate) < (SELECT StartDate
                              FROM Employee
                              WHERE New_EmployeeID = Employee.ID
                              )
```

THEN

SIGNAL SQLSTATE 'E0451'

SET MESSAGE\_TEXT = 'Date conflict: Order date precedes Employee start date.';

END IF;

END;

\$\$

DELIMITER ;

## **2. Relational Model (8)**

DELIMITER \$\$

CREATE PROCEDURE CantRentUnavailable (IN New\_MovieID INT, New\_LoanStatus ENUM('Expired', 'Active'))

BEGIN

```
    IF      New_LoanStatus = 'Active' AND
           1 > (SELECT AvailableCopies
                FROM Movie M
                WHERE New_MovieID = M.ID
              )
```

THEN

SIGNAL SQLSTATE 'E0928'

SET MESSAGE\_TEXT = 'Rental conflict: There are no available copies of this movie.';

END IF;

END;

\$\$

DELIMITER ;

DELIMITER \$\$

CREATE PROCEDURE CantHaveTwoCopies (

IN New\_LoanStatus ENUM('Expired', 'Active'), New\_CustomerID INT, New\_MovieID INT)

BEGIN

```
    IF      New_LoanStatus = 'Active' AND
           1 <= (SELECT COUNT(*)
                FROM Rented R1
                WHERE
                    New_CustomerID = R1.CustomerID AND
                    New_MovieID = R1.MovieID AND
                    R1.LoanStatus = 'Active'
              )
```

THEN

SIGNAL SQLSTATE 'E0928'

SET MESSAGE\_TEXT = 'Rental conflict: Customer is already renting a copy of this movie.';

END IF;

END;

\$\$

DELIMITER ;

## **2. Relational Model (9)**

DELIMITER \$\$

CREATE PROCEDURE CustomerExistsBeforeQueue (IN New\_CustomerID INT, New\_DateAdded DATETIME)

BEGIN

IF DATE(New\_DateAdded) < (SELECT AccountCreated  
FROM Customer  
WHERE New\_CustomerID = AccountID  
)

THEN

SIGNAL SQLSTATE 'E0451'

SET MESSAGE\_TEXT = 'Date conflict: Date of queue insertion precedes Customer's account creation.';

END IF;

END;

\$\$

DELIMITER ;

DELIMITER \$\$

CREATE PROCEDURE CantExceedTotal (IN New\_TotalCopies INT, New\_AvailableCopies INT)

BEGIN

IF New\_TotalCopies < New\_AvailableCopies

THEN

SIGNAL SQLSTATE 'E0928'

SET MESSAGE\_TEXT = 'Logical Error: Available copies number cannot be exceed Total copies number.';

END IF;

END;

\$\$

DELIMITER ;

## **2. Relational Model (10)**

DELIMITER \$\$

```
CREATE PROCEDURE deleteFromQueue (IN New_LoanStatus ENUM('Expired', 'Active'), New_CustomerID
INT, New_MovieID INT)
```

```
BEGIN
```

```
    IF      New_LoanStatus = 'Active' AND
           1 = (SELECT COUNT(*)
                FROM QUEUED Q
                WHERE New_MovieID = Q.MovieID AND
                      New_CustomerID = Q.CustomerID
              )
```

```
    THEN
```

```
        DELETE FROM QUEUED
        WHERE MovieID = New_MovieID AND
              CustomerID = New_CustomerID;
```

```
    END IF;
```

```
END;
```

\$\$

DELIMITER ;

# Pre-INSERT trigger for Rented:

DELIMITER \$\$

```
CREATE TRIGGER Rented_PreInsert_Checks BEFORE INSERT ON Rented
FOR EACH ROW BEGIN
```

```
    CALL CantRentUnavailable(NEW.MovieID, NEW.LoanStatus);
    CALL CantHaveTwoCopies(NEW.LoanStatus, NEW.CustomerID, NEW.MovieID);
    CALL CustomerExistsBeforeOrder (NEW.CustomerID, NEW.OrderDate);
    CALL EmployeeExistsBeforeOrder(NEW.EmployeeID, NEW.OrderDate);
```

```
END;
```

\$\$

DELIMITER ;

DELIMITER \$\$

```
CREATE TRIGGER Queued_PreInsert_Checks BEFORE INSERT ON Queued
FOR EACH ROW BEGIN
```

```
    CALL CustomerExistsBeforeQueue(NEW.CustomerID, NEW.DateAdded);
```

END;

\$\$

## **2. Relational Model (11)**

DELIMITER \$\$

CREATE TRIGGER Queued\_PreUpdate\_Checks BEFORE UPDATE ON Queued  
FOR EACH ROW BEGIN

CALL CustomerExistsBeforeQueue(NEW.CustomerID, NEW.DateAdded);

END;

\$\$

DELIMITER ;

# If customer rents a movie, delete it from their queue (if it exists)

DELIMITER \$\$

CREATE TRIGGER Rent\_Out\_Queue AFTER INSERT ON RENTED  
FOR EACH ROW BEGIN

CALL deleteFromQueue(NEW.LoanStatus, NEW.CustomerID, NEW.MovieID);

END;

\$\$

DELIMITER ;

DELIMITER \$\$

CREATE TRIGGER Rent\_Again\_Out\_Queue AFTER UPDATE ON RENTED  
FOR EACH ROW BEGIN

CALL deleteFromQueue(NEW.LoanStatus, NEW.CustomerID, NEW.MovieID);

END;

\$\$

DELIMITER ;

DELIMITER \$\$

CREATE TRIGGER Available\_Copies\_Check BEFORE UPDATE ON Movie  
FOR EACH ROW BEGIN

CALL CantExceedTotal(NEW.TotalCopies, NEW.AvailableCopies);

END;

\$\$

DELIMITER ;

## **2.1 Rationale for Relational Model:**

We created corresponding tables for all the entity types and relationship types designed in our E-R diagram. We created the views and granted the permission to the customers such that the following information is available to the customers: list of available movies of a particular type; their currently held movies; and their rental history.

We also created procedures together with triggers to control changes like update value or insert rows in the tables. For example, we have a trigger Queued\_PreUpdate\_Checks and a procedure CustomerExistsBeforeQueue that prevent that the date (DateAdded in Queued table) a movie added to wish list (Queued table) is before the date that the customer account was created (AccountCreated in Customer table). By creating the triggers and procedures, we make sure that the system is working as intended (no invalid order is placed).