Online Movie Store Database

CSS 475: Database Systems Final Project | Summer 2017 University Of Washington

Designed and Developed by Team ProjectX

Dawit Y Abera | Reshma Maduri Sivakumar | Haimanot Zerkel | Lok Him Tam

August 17, 2017

TABLE OF CONTENTS

INTRODUCTION

This document provides an overview of the Online Movie Store database developed by the ProjectX team for the final project in Database Systems class at University of Washington. The following sections outline the database application, lists the entities that the database stores and provides some SQL queries that were used to build the database. The document concludes with a reflection of the project.

DATABASE APPLICATION

The application developed is an Online Movie Store which stores several movies, that customers are able to watch depending on the kind of Subscription plan they paid for. The application allows Customers to create a profile and choose among three subscription plans: HD, Non HD and TV-Shows that are varying in prices and the kind of content that is available. The Customer is able to login using their name and the their unique account number as the password. The Customer is able to rent movies or tv-shows as long as it exists in the database. The Online Movie Store holds a maximum of 20 copies for the movies and tv-shows. The administrator has authority to manage checking in and checking out of movies and tv shows, keep track of the movie inventory and track Customers along with the type of subscription they have signed up for.

USER REQUIREMENT

CUSTOMER:

- Each customer's will have a first name, middle initial and last name.
- Each customer has unique ID number provided by the store. The unique ID is 6 digits total.
- Each customer's will have an address comprising of street, city, state and zip code.
- Each customer's subscription date will be stored.
- Each customer's phone and email address is stored.
- Each customer has a date of birth.

- Each customer must have a credit card that can be used to pay for the subscription.

MOVIE:

- Each movie has a unique ID
- Each movie has a title, genre, actors and director's name, release date, movie and a type (HD, Non-HD or TV-Shows)
- Each movie has a copy (1-20). A movie copy does not exist if the corresponding movie does not exist

CHECKOUT:

- A checkout is a unique transaction occurring any particular time a customer checks out a movie.
- A customer makes a rental, the customer he/she is at least one copy, any may rent up to 5 copies.
- Each rental is made by exactly one customer
- A customer can renew their rental and submit their rental PAYMENT:
- Each transaction/payment is uniquely identified by the payment ID
- Only the credit card type is stored in the database ** due project limitation
- Each payments credit card may belong to exactly one customer
- The payment due date is stored in database. Payments will be due after 29 days of subscription

SUBSCRIPTION:

- Each subscription is uniquely identified by the subscription ID.
- Each subscription offers three types of subscription plan: HD, Non-HD and Tv- shows.
- Each subscription package has its own price.

SYSTEM DESCRIPTION AND FUNCTIONALITIES

The DBMS will be a database with a user interface that will allow the Movie Store's data to be viewed, stored and updated using SQL queries to the database. These

will allow the store to maintain and have easy and efficient access to accurate and up-to-date records relevant to the Movie Store information.

The system will facilitate the rental of movie copies by updating database tables. It will also allow the store to easily look for a customer and all sort of queries by using complex queries.

SYSTEM FUNCTIONALITY

The system is mainly a DBMS build around a SQL database and a Java GUI. The system has several main functions and those will be delivered.

- The system will let a user connect to the database with the mean of a username and password
- The system will let a user search or see a movie information
- The system will allow a user to subscribe to our HD, Non-HD or TV-Shows offers and also allow the user to rent a movie
- The system will allow a user to update their personal customer information
- The system will allow the admin/employee to add and update a movie to the database
- The system will allow the admin/employee to search and update the record for a specific customer using the customer's ID
- The system will allow the admin/employee to add credit card type to the customer record
- The system will allow a customer to rent only 5 copies at one time

Due to time constraints and project limitation:

- the system assumes only people(employees) with administrative access can delete customer information
- The system only assumes only one admin has access to the DBMS
- The system does not perform any form of transactions. The credit card information is only saved on the DB.

ENTITIES

The following section outlines all the Entities that the database tracks and lists its primary key, Foreign keys if any, and all it's attributes.

CUSTOMER

- Primary key: Customer_ID
- Foreign key: Subscription_id references SUBSCRIPTION record.
- Attributes: Customer_id, Customer_fname, Customer_minit, Customer_lname, Customer_phone, Customer_email, Address, City, State, Zip, DOB, Subscription_date

MOVIE

- Primary key: Movie_id
- Foreign key: Movie_type , Movie_genre
- Attributes: Movie_id, Movie_title, Movie_year, Movie_genre, Moive_director, Movie_actor, Movie_copy, Movie_type

SUB_TYPE

- Primary key: sub_ID
- Foreign key: None
- Attributes: sub_id, type, price

GENRE

- Primary key: Genre_ID
- Foreign key: None
- Attributes: Genre_ID, Genre_type

PAYMENT

- Primary key: Payment_id
- Foreign key: Customer_id references CUSTOMER record, Subscription_id
- references SUBSCRIPTION record.

- Attributes: Payment_id, Payment_method, Payment_date, Customer_id,
- (Customer_id), Subscription_id

SUBSCRIPTION

- Primary key: Subscription_id
- Foreign key: Customer_id
- Attributes: Subscription_id, Customer_id, Sub_plan, sub_type

CHECKOUT

- Foreign key: Customer_id, Movie_id
- Attributes: Customer_id, Movie_id, renew_count, checkout_time

NORMALIZATION

Upon designing the ER Diagram and Relational Data Model, the process of Normalization was adopted to assess any deficiencies in the derived tables. The goal was to develop third normal form for all tables. In order to achieve third normal form, each tables must be free from multi values attributes and transient dependencies and must have full functional dependencies. This section outlines all the functional dependencies for the final relations.

CUSTOMER

- Customer ID → Customer_fname
- Customer ID → Customer_minit
- Customer ID → Customer_lname
- Customer ID → Customer_phone
- Customer ID → Customer_email
- Customer ID → Address
- Customer ID → City
- Customer ID → State
- Customer ID → Zip
- Customer ID → DOB
- Customer ID → Subscription_date

The Customer table is in third normal form. All of the attributes are uniquely identified by the Customer ID and no transient dependencies exist.

MOVIE

- Movie ID → Movie_title
- Movie ID → Movie_year
- Movie ID → Movie_genre
- Movie ID → Moive_director
- Movie ID → Movie_actor
- Movie ID → Movie_copy
- Movie ID → Movie_type

The Movie table is in third normal form. All of the attributes are uniquely identified by the Movie ID and no transient dependencies exist.

SUB_TYPE

- Sub ID → Type
- Sub ID → Price

The Sub_Type table is in third normal form. All of the attributes are uniquely identified by the Sub ID and no transient dependencies exist.

GENRE

- Genre ID → Genre_type

The Genre table is in third normal form. The Genre_type attribute is uniquely identified by the Genre ID and no transient dependencies exist.

PAYMENT

- Payment ID → Payment_method
- Payment ID → Payment_date
- Payment ID → Customer_id
- Payment ID → Subscription_id

The Payment table is in third normal form. All of the attributes are uniquely identified by the Payment ID and no transient dependencies exist.

SUBSCRIPTION

- Subscription ID → Customer_id
- Subscription ID → Sub_plan
- Subscription ID → sub_type

The Subscription table is in third normal form. All of the attributes are uniquely identified by the Subscription ID and no transient dependencies exist.

CHECKOUT

- Subscription ID → Customer_id
- Subscription ID → Sub_plan
- Subscription ID → sub_type

The Subscription table is in third normal form. All of the attributes are uniquely identified by the Subscription ID and no transient dependencies exist.

- Foreign key: Customer_id, Movie_id
- Attributes: Customer_id, Movie_id, renew_count, checkout_time

CONSTRAINTS

The three main types of integrity constraints are: Key, Entity integrity and Referential integrity constraints.

CUSTOMER

Key: customer_ID is the primary key and must hold “unique” values. Values cannot be repeated for customer_ID.

Entity Integrity:

- Customer_ID is the primary key and hence cannot be a NULL value.
- Customer name, phone number, email, address, city, state, zip, date of birth and subscription id can’t be null.
- State has a Check constraints and it has to be one of this symbols (AL,AK,AS,AZ,AR,CA,CO,CT,DE,DC,FM,FL,GA,GU,HI,ID,IL,IN,IA,KS,KY,LA, ME,MH,MD

,MA,MI,MN,MS,MO,MT,NE,NV,NH,NJ,NM,NY,NC,ND,MP,OH,OK,OR,PW, PA,PR,RI,SC,SD TN, TX, UT, VT, VI, VA, WA, WV, WI, WY)

- Date of birth has a Check constraints (Current_Date - 18) to make sure that the person is old enough to rent a movie
- Subscription date has a Check constraints that checks the subscription date to be on the date that the person is signing up for subscription.

Referential Integrity: None. No foreign keys in Customer table.

MOVIE

Key: Movie_ID is the primary key and must hold "unique" values. Values cannot be repeated for Movie_ID.

Entity Integrity:

- Movie_ID is the primary key and hence cannot be a NULL value. Movie title, actor and director cannot be null.
- Movie copy has a Check constraints <= 20 since the store only has 20 copies of each movie.
- Movie year has a Check constraints <= Current Date to make sure the year is correct.

Referential Integrity: Movie_type has references to Sub_Type's table sub_id.

SUBSCRIPTION

Key: Subscription_ID is the primary key and must hold "unique" values. Values cannot be repeated for subscription_ID.

Entity Integrity: Subscription_ID is the primary key and hence cannot be a NULL value.

Referential Integrity: Customer_id references CUSTOMER table's Customer_id, Sub_plan references SUB_TYPE table's sub_id. Customer_id and Sub_id cannot be NULL and must be a value that already exists in the parent table.

SUB_TYPE

Key: Sub_ID is the primary key and must hold "unique" values. Values cannot be repeated for Sub_ID.

Entity Integrity:

- Sub_ID is the primary key and hence cannot be a NULL value.
- Type has Check constraints and can only be 'HD' 'Non-HD' and 'TV shows'.

Referential Integrity: None. No foreign keys in Sub_type table.

GENRE

Key: Genre_ID is the primary key and must hold "unique" values. Values cannot be repeated for genre_ID.

Entity Integrity: Genre_ID is the primary key and hence cannot be a NULL value.
Referential Integrity: None. No foreign keys in Genre table.

PAYMENT

Key: Payment_ID is the primary key and must hold "unique" values. Values cannot be repeated for Payment_ID.

Entity Integrity: Payment_ID is the primary key and hence cannot be a NULL value. Payment method has a Check constraints to check the payment method is one of those (VISA, MASTER, and AMEX).

Referential Integrity: Customer_id references CUSTOMER table, Subscription_id references SUBSCRIPTION table. Customer_id and Subscription_id cannot be NULL and must be a value that already exists in the parent table.

CHECKOUT

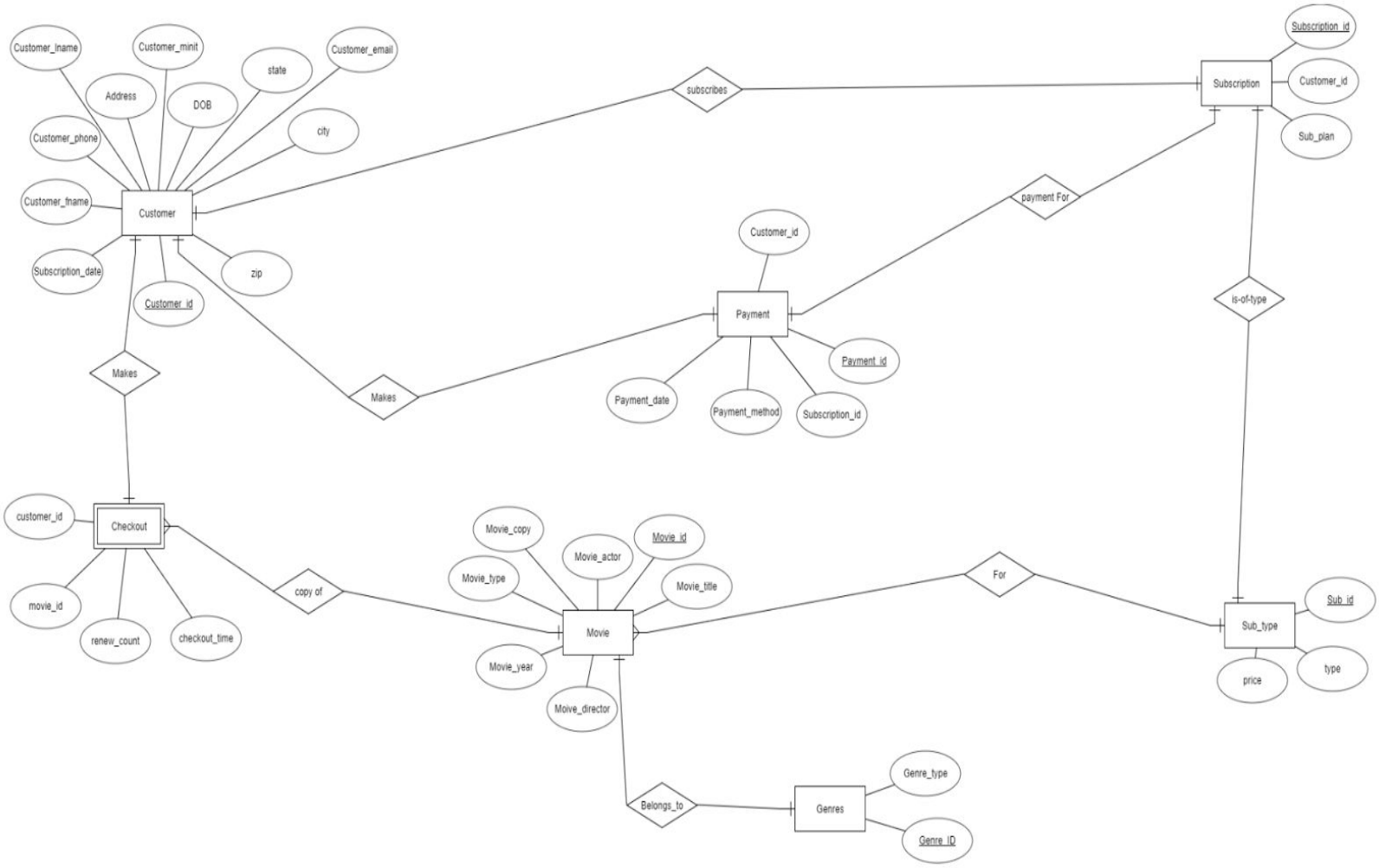
Key: CHECKOUT is weak entity and doesn't have primary key
Entity Integrity: None
Referential Integrity: Customer_id references customer_id in Customer table and movie_id references Movie_id in Movie table.

RELATIONSHIPS

- One CUSTOMER record has one and only one SUBSCRIPTION record.
- One CUSTOMER makes one PAYMENT record.
- One SUBSCRIPTION only have one SUBSCRIPTION type.
- One SUB_TYPE is for many movies.
- One PAYMENT record for one SUBSCRIPTION record.
- One MOVIE record belongs to one and only one GENRE record.
- One CUSTOMER can CHECKOUT up to five MOVIES

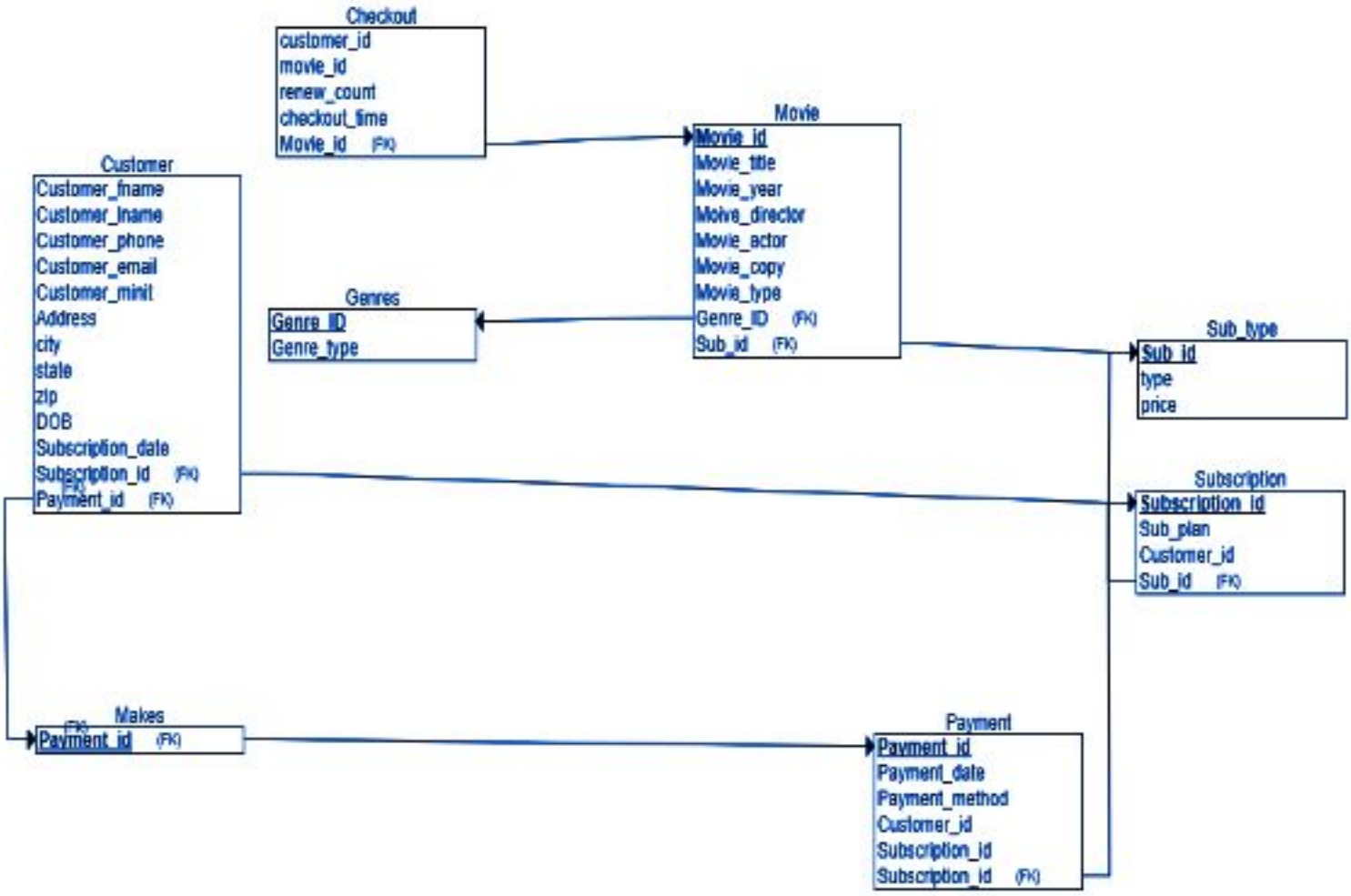
ENTITY RELATIONSHIP DIAGRAM

Figure 1: Entity Relationship Diagram for the Online Movie Store Database



RELATIONAL DATA MODEL

Figure 2: Relational Data Model for the Online Movie Store Database



DATABASE CREATION WITH SQL SCRIPTS

The database is created using SQLite and JavaTM database connectivity (JDBC) using NetBeans IDE.

CREATE SQL STATEMENTS

Customer

```
CREATE TABLE Customer (
    Customer_id       INTEGER       PRIMARY KEY
                                    NOT NULL
                                    UNIQUE,
    Customer_fname    VARCHAR (50)  NOT NULL,
    Customer_minit    CHAR (1),
    Customer_lname    VARCHAR (50)  NOT NULL,
    Customer_phone    VARCHAR (30)  NOT NULL,
    Customer_email    VARCHAR (100) NOT NULL,
    Address           VARCHAR (100) NOT NULL,
    City              VARCHAR (30)  NOT NULL,
    State             CHAR (2)      NOT NULL
                                    CHECK (State =
'AL''AK''AS''AZ''AR''CA''CO''CT''DE''DC''FM''FL''GA''GU''HI''ID''IL''IN''IA''KS''KY'
'LA''ME''MH''MD''MA''MI''MN''MS''MO''MT''NE''NV''NH''NJ''NM''NY''NC''ND''MP''OH''OK'
'OR''PW''PA''PR''RI''SC''SD''TN''TX''UT''VT''VI''VA''WA''WV''WI''WY'),
    Zip               INTEGER       NOT NULL,
    DOB               DATE          NOT NULL
                                    CHECK (DOB = CURRENT_DATE - 18),
    Subscription_date DATE          NOT NULL
                                    CHECK (Subscription_date = CURRENT_DATE)
);
```

Checkout

```
CREATE TABLE Checkout (
    Customer_id   INTEGER REFERENCES Customer (Customer_id)
                          NOT NULL,
    Movie_id      INTEGER REFERENCES Movie (Movie_id)
                          NOT NULL,
    renew_count   INTEGER,
    checkout_time TIME    DEFAULT (CURRENT_TIMESTAMP)
);
```

Genre

```
CREATE TABLE Genre (
    Genre_ID   INTEGER       PRIMARY KEY
                             UNIQUE
                             NOT NULL,
    Genre_type VARCHAR (15)  NOT NULL
);
```

Movie

```
CREATE TABLE Movie (
    Movie_id       INTEGER       PRIMARY KEY
                                 UNIQUE
                                 NOT NULL,
    Movie_title    VARCHAR (20)  NOT NULL,
    Movie_year     DATE,
    Movie_genre    INTEGER       REFERENCES Genre (Genre_ID),
    Moive_director VARCHAR (20)  NOT NULL,
    Movie_actor    VARCHAR (20)  NOT NULL,
    Movie_copy     INTEGER,
    Movie_type     INTEGER       REFERENCES sub_type (sub_id)
);
```

Payment

```
CREATE TABLE Payment (
    Payment_id      INTEGER  PRIMARY KEY
                             UNIQUE,
    Payment_method  CHAR (5) NOT NULL
                             CHECK (Payment_method = 'VISA''MASTER''AMEX'),
    Payment_date    DATE,
    Customer_id     INTEGER  REFERENCES CUSTOMER (Customer_id),
    Subscription_id INTEGER  REFERENCES Subscription (Subscription_id)
);
```

Sub_Type

```
CREATE TABLE Sub_type (
    sub_id INTEGER      PRIMARY KEY,
    type   VARCHAR (8),
    price  INTEGER
);
```

Subscription

```
CREATE TABLE Subscription (
    Subscription_id INTEGER        PRIMARY KEY
                                   NOT NULL,
    Customer_id     INTEGER        REFERENCES Customer (Customer_id)
                                   NOT NULL,
    Sub_plan        DECIMAL (5, 2) NOT NULL
);
```

POPULATE DATABASE: INSERT SQL STATEMENTS

INSERTING INTO CUSTOMER TABLE:

```
INSERT INTO CUSTOMER VALUES
(11110,'Helen',NULL,'James','206-122-3221','helen@gmail.com','14
5th Pine St','Seattle','WA',98012,'2/13/1978','2/1/2017');


INSERT INTO CUSTOMER VALUES
(11111,'Erik','M','Flint','323-646-6334','ErikBNolen@armyspy.com
'1111 Evergreen Lane','San
Francisco','CA',90040,'12/6/1964','5/25/2017');

INSERT INTO CUSTOMER VALUES
(11112,'Gary', 'A', 'Freeman', '910-236-7112',
'GaryAMcCarthy@superrito.com','3803 Clarence Court', 'Appleton',
'WI',28540,'6/26/1943','3/29/2016');
```

INSERTING INTO SUB_TYPE TABLE:

```
INSERT INTO Sub_type VALUES (3455, 'Non-HD', 20);

INSERT INTO Sub_type VALUES (3456, 'HD', 40);

INSERT INTO Sub_type VALUES (3457, 'TV-Shows', 10);
```

INSERTING INTO SUBSCRIPTION TABLE

```
INSERT INTO Subscription VALUES (45300, 11110, '2/1/17', 3455);

INSERT INTO Subscription VALUES (45301, 11111, '5/25/17', 3455);
```

INSERTING INTO PAYMENT TABLE

```
INSERT INTO Payment VALUES (21200, 11110, 45300);

INSERT INTO Payment VALUES (21201, 11111, 45301);
```

INSERTING INTO MOVIE TABLE

```
INSERT INTO Movie VALUES (1001, 'James Bond Story', 1999, 'James
Bond', 'Chris Hunt', 'Daniel Craig', 10,3456);

INSERT INTO Movie VALUES(1002,'World Is Not Enough',1999,'James
Bond','Michael Apted','Roger Moore',10,3456);

INSERT INTO Movie VALUES (1003,'Die Another Day',2002,'James
Bond','Lee Tamahori','Roger Moore',10,3456);
```

INSERTING INTO GENRE TABLE

```
INSERT INTO GENRE VALUES (400, 'James Bond');

INSERT INTO GENRE VALUES (401, 'Sci-Fi');
```

INSERTING INTO CHECKOUT TABLE

```
INSERT INTO CHECKOUT VALUES (11110,1001,0,CURRENT_TIME);
```

QUERIES

This section lists some examples of queries that were ran against the database.

- How many customers were subscribed to HD, Non HD and Tv Show services ?
- How many customers were from Seattle?
- List all the customer's subscription details.
- What movies can each customer watch based on their subscription type : HD, Non-HD, TV Shows ?
- How many people subscribed for the HD services?
- List Payment details of customers who subscribed for HD services.

The queries listed are examples of the ones that the Admin would implement in order to monitor the online movie store. The result helps the Admin keep track of all the customers, their subscription record, the kind of service they are subscribed for, the payment details and all the movies they can watch/checkout based on the service they pay for.

Query: Show customers who subscribed for HD, Non-HD and TV-Show services respectively?

Result :

```
sqlite> SELECT c.Customer_id FROM CUSTOMER c JOIN
Subscription s  ON c.Customer_id = s.Customer_id JOIN
sub_type t ON s.Sub_plan = t.sub_id WHERE type = 'HD';
```

```
Customer_id
-----------
1011
1156

sqlite> SELECT c.Customer_id FROM CUSTOMER c JOIN
Subscription s  ON c.Customer_id = s.Customer_id JOIN
sub_type t ON s.Sub_plan = t.sub_id WHERE type = 'Non-HD';

Customer_id
-----------
2113

sqlite> SELECT c.Customer_id FROM CUSTOMER c JOIN
Subscription s  ON c.Customer_id = s.Customer_id JOIN
sub_type t ON s.Sub_plan = t.sub_id WHERE type =
'TV-Shows';

Customer_id
-----------
1111
```

Purpose: The above query is used by the Admin to determine the number of customers that are using the various services that the online movie store provides. This information can be used for several things like what service is the most and least popular among customers, which can help us determine how can we change them around to better serve the customers. This query is used by the admin to keep an accurate list of all the customers who have subscribed for HD, Non-HD and TV Shows respectively. The result of the query is just the Customer ID.

Query: Show customers who have subscription for HD, Non-HD or TV-Shows services who are from 'Seattle'?

Result:

```
sqlite> SELECT c.Customer_id,c.City FROM CUSTOMER c JOIN
Subscription s  ON c.Customer_id = s.Customer_id JOIN
```

```
sub_type t ON s.Sub_plan = t.sub_id WHERE type = 'HD' AND
c.City = 'Seattle';
```

Customer_id	City
1011	Seattle

Purpose: The above query is used by the Admin to determine the number of customers that are using the various services that the online movie store provides in Seattle. This example shows that the admin is able to perform a query on all the customers who have signed up for various services based on a location. The result of the query is the Customer ID and the City.

Query: List all customer with their subscription plan and subscription date

Result:

```
sqlite> SELECT c.Customer_id, s.Sub_plan,
s.Subscription_date FROM CUSTOMER c JOIN Subscription s ON
c.Customer_id = s.Customer_id JOIN payment p ON
s.Subscription_id = p.Subscription_id;
```

Customer_id	Sub_plan	Subscription_date
1011	3456	2017-07-21
2113	3455	2017-09-14
1111	3457	2016-01-09
1011	3456	2017-07-21

Purpose: The above query is used by the Admin to retrieve all the customer's subscription plan and the date of their subscription. This can be used to address any billing issues or plan related inquiries that may arise. The query returns the Customer ID along with its subscription plan ID and the date of Subscription.

Query: List all customers with their subscription plan, subscription date, subscription type and the price they are paying?

Result:

```
sqlite> SELECT c.Customer_id, s.Sub_plan,
s.Subscription_date,st.type,st.price FROM CUSTOMER c JOIN
Subscription s ON c.Customer_id = s.Customer_id JOIN
payment p ON s.Subscription_id = p.Subscription_id JOIN
sub_type st ON s.Sub_plan = st.sub_id;
```

Customer_id	Sub_plan	Subscription_date	type	price
1011	3456	2017-07-21	HD	40
2113	3455	2017-09-14	Non-HD	20
1111	3457	2016-01-09	TV-Shows	10
1011	3456	2017-07-21	HD	40

Purpose: The above query is used by the Admin to retrieve more information regarding customer's subscription plan, the data of subscription, type of service they signed up for and the amount they are paying. This can be used to address any billing issues or plan related inquiries that may arise or just to retrieve all the information regarding the customer's interaction with the online store. The query returns the Customer ID, the ID of the kind of service they chose (HD, Non HD or TV shows), date of subscription, type and its price.

Query: List all the name of the movies that a customer subscribed with their sub plan and they price they are paying?

Result:

```
sqlite> SELECT c.Customer_id,m.Movie_id, m.Movie_type,
s.sub_id,s.type,s.price FROM Movie m Join sub_type s ON
m.Movie_type = s.sub_id JOIN Subscription su ON su.sub_plan
= s.sub_id JOIN CUSTOMER c ON c.Customer_id =
su.Customer_id;
```

Customer_id	Movie_id	Movie_type	sub_id	type	price
1011	1290	3456	3456	HD	40
1011	1344	3456	3456	HD	40
2113	1289	3455	3455	Non-HD	20
1111	1345	3457	3457	TV-Shows	10
1156	1290	3456	3456	HD	40
1156	1344	3456	3456	HD	40

Purpose: The above query can be used by both admins and customers to retrieve information regarding the list of movies/shows they have access to based on their subscription plan as well as the price they are paying for it. The query returns Customer ID, the Movie's ID and their type (HD or Non HD) and the price.

Query: Add Payment details as well as movie details, the subscription plan and the price they are paying for the services.

Result:

```
sqlite> SELECT c.Customer_id,m.Movie_id, m.Movie_type,
s.sub_id,p.Payment_id,p.Payment_method,p.Payment_date,s.typ
e,s.price FROM Movie m Join sub_type s ON m.Movie_type =
s.sub_id JOIN Subscription su ON su.sub_plan = s.sub_id
JOIN CUSTOMER c ON c.Customer_id = su.Customer_id JOIN
Payment p ON p.Customer_id = c.Customer_id;
```

Customer_id	Movie_id	Movie_type	sub_id	Payment_id	Payment_method	Payment_date	type	price
1011	1290	3456	3456	1	AMEX	2017-09-19	HD	40
1011	1344	3456	3456	1	AMEX	2017-09-19	HD	40
2113	1289	3455	3455	2	VISA	2017-09-28	Non-HD	20
1111	1345	3457	3457	3	MASTER	2017-09-20	TV-Shows	10
1156	1290	3456	3456	4	AMEX	2017-09-12	HD	40
1156	1344	3456	3456	4	AMEX	2017-09-12	HD	40

sqlite>

Purpose: The above query can be used by admin information regarding the list of movies/shows they have access to based on their subscription plan as well as the price they are paying for it and add the customer's corresponding payment details. This can be used to show the customers information regarding the movies they can

watch/checkout, the kind of plan they subscribed for, the subscription date as well the price and all their payment details.

METHODOLOGY

DESIGN PATTERN

For our project, we used DAO (Data Access Object) design pattern to do the Movie Store project. By using the DAO design pattern, we were able to abstract the detail of our application and communicate with the database indirectly. The design pattern provides a DAO layer, and the DAO layer handles all the database operations and communicates with the domain. The reason we chose to follow this design pattern is that if we want to change the underlying persistence mechanism, we can just use the DAO layer, without altering the business/domain layer.

SOFTWARE AND DATABASE DEVELOPMENT APPROACH

For our project, we used both spiral and agile software development methods. The spiral model helped us develop the project iteratively by incorporating instructors feedback at each iteration stage and testing the software before the next incremental refinement. And the agile approach helped us to adapt to the specification and design changes quickly.

TESTING

Testing is the most crucial part of our software development, and we took the time to come up with test data to the software program by considering all the

various use cases. All the test data are generated manually and uses the White Box Testing approach.

The white box testing helped us to directly examine both the SQL program and the code we wrote in Java to be tested one code at a time. These means all branches of the code and SQL programs must be at least tested once but using many different possible cases. For example, in the Movie class, we stated that the store only has 20 copies of a single movie and these needs to be enforced both in the SQL program and the actual GUI Java program.

TEST DATA GENERATION

The test data are generated manually based on the user requirements (refer page 2 to see the user requirements) and use cases.

TABLE 1: A SAMPLE TEST DATA

ID#	Test Condition	Expected result	Procedure	Pass/fail	Defect ID
1	Customer tries to return a movie that was not in the system	The system must flag this and report to the customer saying the item does not belong to the store	1. A customer login with their credential 2. Goes to return a movie section and types the movie id		
2	New customer subscription	customer is signed up and added to the DB	1. The store admin will use the customer's information and 2. provides the customer with a unique id		
3	Existing customers try to sign up again	The system should flag these if the same email and phone number is used	1. the store admin should get an error based on the email and phone number		

PROJECT EVALUATION

The following section gives a brief reflection on our journey towards building our database.

The work distribution varied from time to time but everyone in the group worked on certain aspect of the project. Our weekly meetings were very productive and made an agenda prior to meeting to best optimize on time. After the meeting, we would adjourn individual work that needed to be done for every group member. All team members were easily available to meet up and our sessions ranged from 2 - 5 hours either once or twice a week.

We split the work into three main areas. One group member worked on the UI, one on SQL scripts and data modeling and the others on the design document. We were flexible with this plan and often worked on other parts if another group member needed help. We used Git and google docs to collaborate online and not worrying about merging our work together. We were able to collaborate online more leading to lesser in person meetings and a better use of our time.

IMPROVEMENTS

- Customer login screen needs improvement. The login page does not save the credential correctly, therefore the information is lost. A customer can checkout a movie using other customer's unique code.
- We would have done the actual transaction
- The ER and RD can be improved to make things a lot easier.
- We could have used the actual email address to confirm customer's info.

REFERENCES

- Elmasri, Ramez, and Shamkant B. Navathe. Fundamentals of database system: Ramez Elmasri, Shamkant B. Navathe. Readwood, CA: Benjamin/Cumming, 1989. Print.

- Jenkov, Jakob. "The DAO Design Pattern." *Jenkov.com*. N.p., n.d. Web. 10 Aug. 2017.

- Rice, Written By Randall. "How to Develop Test Cases and Test Scripts for Web Testing."

- *Software Testing Training and Software Testing Consulting - ISTQB Software Testing Certification Training*. N.p., n.d. Web. 10 Aug. 2017.