CPS510-01: Database Systems I

# Movie Store DBMS

Group 5

Janice Zhu, Dalton Crowe, Fadi Al-Shabi

2022-11-30

## CONTENTS

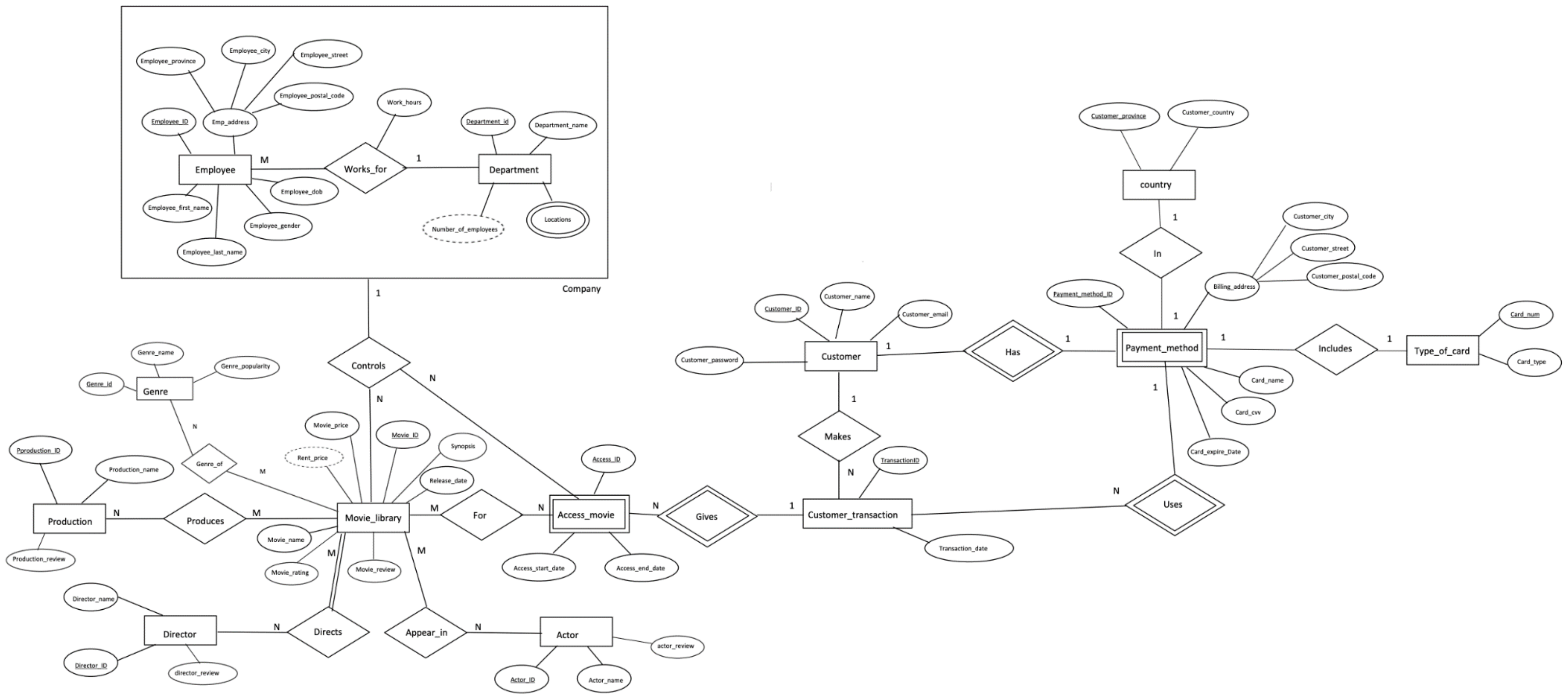
## APPLICATION DESCRIPTION

The online movie store allows people to search for their favorite movies with multiple filtering options based on genres, year released, director, and other criteria. The online store will enable people to buy or rent movies. Customers who choose to rent the movie have 30 days to watch the movie. Once they start streaming the movie, they have up to 48 hours to finish it. Customers who buy the movie can download it to their devices.

The employee can add, remove, and modify the digital movie entries, as well as customer table entries. The employee can add, remove or manage its movie metadata specifically genres, release date, actors, directors, etc.

A customer will have attributes like email, password, billing address, and payment info. They can search the database for movies they might be interested in purchasing by specifying a title, genre, or associated performers and directors. Upon selection of a desired movie title, the price will be displayed. Payment will be completed by charging the saved credit card on file before the user has access to the movie. Upon successful payment, a timestamp will be added to the customer's film to keep track of both purchase history and remaining rental time left if applicable.

## ER MODEL


Employee_province
Employee_city
Employee_street
Employee_postal_code
Employee_ID
Emp_address
Work_hours
Department_id
Department_name
Employee
M
Works_for
1
Department
Employee_dob
Employee_first_name
Employee_gender
Employee_last_name
Number_of_employees
Locations
Company
1
Controls
N
N
Genre_name
Genre_popularity
Genre_id
Genre
N
Genre_of
M
Movie_price
Movie_ID
Synopsis
Rent_price
Release_date
Pproduction_ID
Production_name
Production
N
Produces
M
Movie_library
M
For
N
Access_ID
Access_movie
N
Gives
1
Customer_transaction
Production_review
Movie_name
Movie_rating
Movie_review
M
M
Access_start_date
Access_end_date
Transaction_date
TransactionID
N
Makes
1
Customer
Customer_ID
Customer_name
Customer_email
Customer_password
1
Has
1
Payment_method
Payment_method_ID
Billing_address
Customer_city
Customer_street
Customer_postal_code
1
In
1
country
Customer_province
Customer_country
1
Includes
1
Type_of_card
Card_num
Card_type
Card_name
Card_cvv
Card_expire_Date
1
N
Uses
Director_name
Director
N
Directs
Director_ID
director_review
Appear_in
N
Actor
actor_review
Actor_ID
Actor_name

## SQL SOURCE CODE

### CREATE TABLES

#### DEPARTMENT TABLE

```
CREATE TABLE department (
  department_id NUMBER PRIMARY KEY,
  department_name VARCHAR2(50) NOT NULL,
  department_location VARCHAR2(50) NOT NULL
);
```

#### GENRE TABLE

```
CREATE TABLE genre (
  genre_name VARCHAR2(50) PRIMARY KEY,
  genre_popularity NUMBER DEFAULT 0
);
```

#### PRODUCTION TABLE

```
CREATE TABLE production (
  production_id NUMBER PRIMARY KEY,
  production_name VARCHAR2(50) NOT NULL,
  production_review NUMBER DEFAULT 0
);
```

#### DIRECTOR TABLE

```
CREATE TABLE director (
  director_id NUMBER PRIMARY KEY,
  director_first_name VARCHAR2(50) NOT NULL,
  director_last_name VARCHAR2(50) NOT NULL,
  director_review NUMBER DEFAULT 0
);
```

#### ACTOR TABLE

```
CREATE TABLE actor (
  actor_id NUMBER PRIMARY KEY,
  actor_first_name VARCHAR2(50) NOT NULL,
  actor_last_name VARCHAR2(50) NOT NULL,
  actor_review NUMBER DEFAULT 0
);
```

#### CARD_TYPE TABLE

```
CREATE TABLE card_type(
  card_num NUMBER PRIMARY KEY,
  card_type varchar(50) NOT NULL
);
```

## EMPLOYEE TABLE

```
CREATE TABLE employee (
  employee_id NUMBER,
  department_id NUMBER NOT NULL,
  work_hours NUMBER NOT NULL,
  employee_first_name VARCHAR2(50) NOT NULL,
  employee_last_name VARCHAR2(50) NOT NULL,
  employee_gender VARCHAR2(50),
  employee_dob DATE,
  employee_province VARCHAR2(50) NOT NULL,
  employee_city VARCHAR2(50) NOT NULL,
  employee_street VARCHAR2(50) NOT NULL,
  employee_postal_code VARCHAR2(50) NOT NULL,
  CONSTRAINT employee_id PRIMARY KEY (employee_id),
  CONSTRAINT department_fk FOREIGN KEY (department_id) REFERENCES department
(department_id)
);
```

## MOVIE_LIBRARY TABLE

```
CREATE TABLE movie_library (
  movie_id NUMBER,
  director_id NUMBER NOT NULL,
  movie_name VARCHAR2(50) NOT NULL,
  movie_rating VARCHAR2(50) NOT NULL,
  movie_review NUMBER DEFAULT 0,
  movie_price NUMBER NOT NULL,
  release_date DATE,
  synopsis VARCHAR2(200),
  CONSTRAINT movie_pk PRIMARY KEY (movie_id),
  CONSTRAINT director_fk FOREIGN KEY (director_id) REFERENCES director
(director_id)
);
```

## GENRE_OF TABLE

```
CREATE TABLE genre_of (
  movie_id NUMBER NOT NULL,
  genre_name VARCHAR2(50) NOT NULL,
  CONSTRAINT movie_genre_fk FOREIGN KEY (movie_id) REFERENCES movie_library
(movie_id),
  CONSTRAINT genre_fk FOREIGN KEY (genre_name) REFERENCES genre (genre_name)
);
```

## PRODUCES TABLE

```
CREATE TABLE produces (
  movie_id NUMBER NOT NULL,
  production_id NUMBER NOT NULL,
  CONSTRAINT movie_produces_fk FOREIGN KEY (movie_id) REFERENCES
movie_library (movie_id),
  CONSTRAINT production_fk FOREIGN KEY (production_id) REFERENCES production
(production_id)
);
```

## APPEAR_IN TABLE

```
CREATE TABLE appear_in (
  movie_id NUMBER NOT NULL,
  actor_id NUMBER NOT NULL,
  CONSTRAINT movie_actor_fk FOREIGN KEY (movie_id) REFERENCES movie_library
(movie_id),
  CONSTRAINT actor_fk FOREIGN KEY (actor_id) REFERENCES actor (actor_id)
);
```

## PAYMENT_METHOD TABLE

```
CREATE TABLE payment_method (
  payment_method_id NUMBER,
  customer_province VARCHAR2(50) NOT NULL,
  customer_city VARCHAR2(50) NOT NULL,
  customer_street VARCHAR2(50) NOT NULL,
  customer_postal_code VARCHAR2(50) NOT NULL,
  card_name VARCHAR2(50) NOT NULL,
  card_num NUMBER NOT NULL,
  card_cvv NUMBER NOT NULL,
  card_expire_date DATE NOT NULL,
  CONSTRAINT payment_method_pk PRIMARY KEY (payment_method_id)
);
```

## CUSTOMER TABLE

```
CREATE TABLE customer (
  customer_id NUMBER,
  payment_method_id NUMBER NOT NULL,
  customer_name VARCHAR2(50) NOT NULL,
  customer_email VARCHAR2(50) NOT NULL,
  customer_password VARCHAR2(50) NOT NULL,
  CONSTRAINT customer_pk PRIMARY KEY (customer_id),
  CONSTRAINT payment_method_fk FOREIGN KEY (payment_method_id) REFERENCES
payment_method (payment_method_id)
);
```

## CUSTOMER_TRANSACTION TABLE

```
CREATE TABLE customer_transaction (
  transaction_id NUMBER,
  payment_method_id NUMBER NOT NULL,
  customer_id NUMBER NOT NULL,
  transaction_date DATE NOT NULL,
  CONSTRAINT transaction_pk PRIMARY KEY (transaction_id),
  CONSTRAINT payment_method_id_fk FOREIGN KEY (payment_method_id) REFERENCES
payment_method (payment_method_id),
  CONSTRAINT customer_id_fk FOREIGN KEY (customer_id) REFERENCES customer
(customer_id)
);
```

## ACCESS_MOVIE TABLE

```
CREATE TABLE access_movie (
  access_id NUMBER,
  department_id NUMBER NOT NULL,
  transaction_id NUMBER NOT NULL,
  movie_id NUMBER NOT NULL,
  accesss_start_date DATE NOT NULL,
  access_end_date DATE,
  CONSTRAINT access_pk PRIMARY KEY (access_id),
  CONSTRAINT department_id_acc_fk FOREIGN KEY (department_id) REFERENCES
department (department_id),
  CONSTRAINT transaction_acc_fk FOREIGN KEY (transaction_id) REFERENCES
customer_transaction (transaction_id),
  CONSTRAINT movie_acc_fk FOREIGN KEY (movie_id) REFERENCES movie_library
(movie_id)
);
```

## POPULATE TABLES

### DEPARTMENT TABLE

```
--(department_id,'department_name','location')
INSERT INTO department
VALUES
  (01, 'Marketing', 'Toronto');
INSERT INTO department
VALUES
  (02, 'Human Resources', 'Ottawa');
```

### EMPLOYEE TABLE

```
--Insert values in employee table
INSERT INTO employee
VALUES
  (
    00001, 01, 1440, 'Igor', 'Mathewson',
    'Male', '1993 - 11 - 03', 'Ontario',
    'Toronto', '1637 Eglinton Avenue',
    'M4P 1A6'
  );
INSERT INTO employee
VALUES
  (
    00002, 01, 7488, 'Lidia', 'Ardelean',
    'Female', '1974-06-24', 'Alberta',
    'Veteran', '3046 Pine Street', 'T0C 2S0'
  );
INSERT INTO employee
VALUES
  (
    00003, 02, 12480, 'Hilda', 'Danell',
    'Female', '1975-01-06', 'British Columbia',
    'Victoria', '4716 Burdett Avenue',
    'V8Y 1Y7'
  );
INSERT INTO employee
VALUES
  (
    00004, 01, 1920, 'Jordan', 'Okonkwo',
    'Male', '2000-10-27', 'Ontario',
    'Etobicoke', '267 Queen Elizabeth Boulevard',
    'M8Z 1M3'
  );
INSERT INTO employee
VALUES
  (
    00005, 01, 3744, 'Cadice', 'Lambert',
    'Male', '1977-01-22', 'Quebec', 'Notre Dame De La Salette',
    '792 rue des Églises Est', 'J0X 2L0'
  );
```

## CARD_TYPE TABLE

```
INSERT INTO card_type
VALUES
   (4, 'Visa');
INSERT INTO card_type
VALUES
   (5, 'Mastercard');
INSERT INTO card_type
VALUES
   (6, 'Discover Card');
```

## GENRE TABLE

```
--INSERT INTO genre VALUES ('genre_name',genre_popularity)
INSERT INTO genre
VALUES
   ('Action', 10);
INSERT INTO genre
VALUES
   ('Adventure', 7);
INSERT INTO genre
VALUES
   ('Fantasy', 8);
INSERT INTO genre
VALUES
   ('Comedy', 9);
INSERT INTO genre
VALUES
   ('Drama', 6);
INSERT INTO genre
VALUES
   ('Romance', 5);
INSERT INTO genre
VALUES
   ('Crime', 6);
INSERT INTO genre
VALUES
   ('Horror', 5);
INSERT INTO genre
VALUES
   ('Family', 6);
```

## ACTOR TABLE

```
--INTO actor VALUES (actor_id, 'first_name', 'last_name', actor_review)
--Actors in Docter Strange
INSERT INTO actor
VALUES
  (
    000001, 'Benedict', 'Cumberbatch',
    9
  );
INSERT INTO actor
VALUES
  (000002, 'Chiwetel', 'Ejiofor', 8);
INSERT INTO actor
VALUES
  (000003, 'Rachel', 'McAdams', 8);

--Actors in Jumanji
INSERT INTO actor
VALUES
  (000004, 'Dwayne', 'Johnson', 10);
INSERT INTO actor
VALUES
  (000005, 'Karen', 'Gillan', 7);
INSERT INTO actor
VALUES
  (000006, 'Kevin', 'Hart', 10);

--Actors in Guardians of Galaxy
INSERT INTO actor
VALUES
  (0007, 'Chris', 'Pratt', 9);
INSERT INTO actor
VALUES
  (0008, 'Vin', 'Diesel', 7);
INSERT INTO actor
VALUES
  (0009, 'Bradley', 'Cooper', 6);

--Actors in Beauty and the Beast
INSERT INTO actor
VALUES
  (0010, 'Emma', 'Watson', 9);
INSERT INTO actor
VALUES
  (0011, 'Dan', 'Stevans', 6);
INSERT INTO actor
VALUES
  (0012, 'Luke', 'Evans', 6);
```

## DIRECTOR TABLE

```sql
--INTO director VALUES (director_id,'first_name', 'last_name')
--Director for Docter Strange
INSERT INTO director
VALUES
  (0001, 'Scott', 'Derrickson', 8);

--Director for Jumanji
INSERT INTO director
VALUES
  (0002, 'Jake', 'Kasdan', 7);

--Director for Guardian of the Galaxy
INSERT INTO director
VALUES
  (0003, 'James', 'Gunn', 9);

--Director for Beauty and the Beast
INSERT INTO director
VALUES
  (0004, 'Bill', 'Condon', 6);
```

## PRODUCTION TABLE

```sql
--INTO production VALUES (production_id,'production_name', production_review)
--Producer for Docter strange, Guardian of the Galaxy
INSERT INTO production
VALUES
  (0001, 'Marvel Studios', 10);

--Producers for Jumanji
INSERT INTO production
VALUES
  (0002, 'Columbia Pictures', 9);
INSERT INTO production
VALUES
  (
    0003, 'Matt Tolmach Productions',
    7
  );
INSERT INTO production
VALUES
  (
    0004, 'Seven Bucks Productions',
    6
  );
INSERT INTO production
VALUES
  (0005, 'Radar Pictures', 6);

--Producer for Beauty and the Beast
INSERT INTO production
VALUES
  (0006, 'Walt Disney Pictures', 10);
```

## PRODUCES TABLE

```sql
INSERT INTO produces
VALUES
  (00001, 0001);
INSERT INTO produces
VALUES
  (00002, 0002);
INSERT INTO produces
VALUES
  (00002, 0003);
INSERT INTO produces
VALUES
  (00002, 0004);
INSERT INTO produces
VALUES
  (00003, 0001);
INSERT INTO produces
VALUES
  (00004, 0006);
```

## PAYMENT_METHOD TABLE

```sql
INSERT INTO payment_method
VALUES
  (
    0001, 'Manitoba', 'Dauphin', '585 Main St',
    'R7N 2T3', 'Ema Wallis', 5024007180587064,
    665, '2025-05-01'
  );
INSERT INTO payment_method
VALUES
  (
    0002, 'British Columbia', 'Vancouver',
    '1011 Tolmie St', 'V6R 4C5', 'Marcelino Metz',
    4539613673997583, 554, '2022-03-01'
  );
```

## CUSTOMER TABLE

```sql
INSERT INTO customer
VALUES
  (
    00001, 0001, 'Ema Wallis', 'EmaW@email.com',
    'EmaW123pw'
  );
INSERT INTO customer
VALUES
  (
    00002, 0002, 'Marcelino Metz', 'MarcelinoM@email.com',
    'MarcelinoM123pw'
  );
```

## GENRE_OF TABLE

```
INSERT INTO genre_of
VALUES
  (00001, 'Action');
INSERT INTO genre_of
VALUES
  (00001, 'Adventure');
INSERT INTO genre_of
VALUES
  (00001, 'Fantasy');
INSERT INTO genre_of
VALUES
  (00002, 'Action');
INSERT INTO genre_of
VALUES
  (00002, 'Adventure');
INSERT INTO genre_of
VALUES
  (00002, 'Comedy');
INSERT INTO genre_of
VALUES
  (00003, 'Action');
INSERT INTO genre_of
VALUES
  (00003, 'Adventure');
INSERT INTO genre_of
VALUES
  (00003, 'Comedy');
INSERT INTO genre_of
VALUES
  (00004, 'Adventure');
INSERT INTO genre_of
VALUES
  (00004, 'Family');
INSERT INTO genre_of
VALUES
  (00004, 'Fantasy');
```

## CUSTOMER_TRANSACTION TABLE

```
INSERT INTO customer_transaction
VALUES
  (000001, 0001, 00001, '2015-01-16');
INSERT INTO customer_transaction
VALUES
  (000002, 0001, 00001, '2016-12-23');
INSERT INTO customer_transaction
VALUES
  (000003, 0002, 00002, '2020-08-06');
INSERT INTO customer_transaction
VALUES
  (000004, 0001, 00001, '2021-08-06');
INSERT INTO customer_transaction
VALUES
  (000005, 0002, 00002, '2022-08-06');
```

## APPEAR_IN TABLE

```
INSERT INTO appear_in
VALUES
   (00001, 000001);
INSERT INTO appear_in
VALUES
   (00001, 000002);
INSERT INTO appear_in
VALUES
   (00001, 000003);
INSERT INTO appear_in
VALUES
   (00002, 000004);
INSERT INTO appear_in
VALUES
   (00002, 000005);
INSERT INTO appear_in
VALUES
   (00002, 000006);
INSERT INTO appear_in
VALUES
   (00003, 000007);
INSERT INTO appear_in
VALUES
   (00003, 000008);
INSERT INTO appear_in
VALUES
   (00003, 000009);
INSERT INTO appear_in
VALUES
   (00004, 000010);
INSERT INTO appear_in
VALUES
   (00004, 000011);
INSERT INTO appear_in
VALUES
   (00004, 000012);
```

## MOVIE_LIBRARY TABLE

```
--Insert values in movie library table (Docter Strange)
INSERT INTO movie_library
VALUES
  (
    00001, 0001, 'Docter Strange', 'PG-13',
    9, 50, '2016 - 11 - 04', 'While on a journey of physical and spiritual
healing, a brilliant neurosurgeon is drawn into the world of the mystic
arts.'
  );

--Insert values in movie library table (Jumanji)
INSERT INTO movie_library
VALUES
  (
    00002, 0002, 'Jumanji: Welcome to the Jungle',
    'PG-13', 7, 30, '2017 - 12 - 20',
    'Four teenagers are sucked into a magical video game, and the only way
they can escape is to work together to finish the game.'
  );

--Insert values in movie library table (Guardians of Galaxy)
INSERT INTO movie_library
VALUES
  (
    00003, 0003, 'Guardians of the Galaxy',
    'PG-13', 7, 50, '2014 - 07 - 01',
    'A group of intergalactic criminals must pull together to stop a
fanatical warrior with plans to purge the universe.'
  );

--Insert values in movie library table (Beauty and the Beast)
INSERT INTO movie_library
VALUES
  (
    00004, 0004, 'Beauty and the Beast',
    'PG', 9, 60, '2017 - 03 - 17', 'A selfish Prince is cursed to become a
monster for the rest of his life, unless he learns to fall in love with a
beautiful young woman he keeps prisoner.'
  );
```

## ACCESS_MOVIE TABLE

```
--customer 1 purchase movie 3 (guardians of galaxy)
INSERT INTO access_movie
VALUES
  (
    000001, 01, 000001, 00003, '2015-01-16',
    NULL
  );

--customer 1 rent movie 1 (doc strange) for 1 month
INSERT INTO access_movie
VALUES
  (
    000002, 01, 000002, 00001, '2016-12-23',
    '2017-01-23'
  );

--customer 2 rent movie 2 (jumanji) for 3 month
INSERT INTO access_movie
VALUES
  (
    000003, 01, 000003, 00002, '2020-08-06',
    '2020-11-06'
  );

--customer 1 purchase movie 2 (jumanji) now 4
INSERT INTO access_movie
VALUES
  (
    000004, 01, 000004, 00004, '2021-08-06',
    NULL
  );

--customer 2 purchase movie 3 (jumanji)
INSERT INTO access_movie
VALUES
  (
    000005, 01, 000005, 00003, '2021-08-06',
    NULL
  );
```

## SIMPLE QUERIES

```
--List hours worked
SELECT
  employee_first_name,
  SUM (work_hours)
FROM
  employee
GROUP BY
  employee_first_name;

--List all employees in marketing by seniority (most hours)
SELECT
  employee_first_name,
  employee_last_name,
  work_hours as seniority_in_marketing_dep
FROM
  employee
WHERE
  department_id = 1
ORDER BY
  work_hours DESC;

--List all movies and their release date
SELECT
  movie_name,
  'released on:',
  release_date
FROM
  movie_library;

--List all movies accessed
SELECT
  access_id,
  transaction_id,
  movie_id,
  accesss_start_date,
  access_end_date
FROM
  access_movie
WHERE
  accesss_start_date <= '2020-12-01'
ORDER BY
  accesss_start_date DESC;

--List customers in alphabetical order
SELECT
  *
FROM
  customer
ORDER BY
  customer_name;
```

```
 --List all actors from highest review
SELECT
  *
FROM
  actor
ORDER BY
  actor_review DESC;

--List all movies and with a price of 50
SELECT
  *
FROM
  movie_library
WHERE
  movie_price >= 50;

--List all movies with a review of 8 or more, order by descending
SELECT
  *
FROM
  movie_library
WHERE
  movie_review >= 8
ORDER BY
  movie_review DESC;

--List only movie id and actor id
SELECT
  movie_id,
  actor_id
FROM
  appear_in
ORDER BY
  movie_id DESC,
  actor_id asc;

--List all employees in marketing department sorted by name in ascending
order
SELECT
  employee_first_name as employees_in_marketing
FROM
  employee
WHERE
  department_id = 001
ORDER BY
  employee_first_name;

--List all movie (movie name and rating) of a director
SELECT
  movie_name,
  movie_rating
FROM
  movie_library
WHERE
  movie_rating = 'PG-13'
  AND director_id = 001;
```

```sql
--List all movies (movie name and review only) in movie rating PG-13 with a
review of 9 or more or PG with eview of 5 or more
SELECT
  movie_name,
  movie_review
FROM
  movie_library
WHERE
  (
    movie_review >= 9
    AND movie_rating = 'PG-13'
  )
  OR (
    movie_review >= 5
    AND movie_rating = 'PG'
  );

--Q8: List all movies (name and director) who are not PG-13
SELECT
  movie_name,
  director_id
FROM
  movie_library
WHERE
  movie_rating <> 'PG-13';
```

## ADVANCED QUERIES

```
---lists num of movies for each genre with popularity more than 7
SELECT
  genre_name,
  COUNT(movie_id) AS num_movies
FROM
  genre_of
WHERE
  EXISTS (
    SELECT
      genre_popularity
    FROM
      genre
    WHERE
      genre_of.genre_name = genre.genre_name
      AND genre_popularity > 7
  )
GROUP BY
  genre_name
HAVING
  COUNT(movie_id) > 0;

---list for each movie, the number of transactions on a movie before december
1st 2020
SELECT
  movie_id,
  COUNT(transaction_id) AS num_transactions
FROM
  access_movie
WHERE
  EXISTS (
    SELECT
      transaction_id
    FROM
      customer_transaction
    WHERE
      customer_transaction.transaction_id = access_movie.transaction_id
      AND transaction_date < '2020-12-01'
  )
GROUP BY
  movie_id;
```

```
--this MINUS returns all movie_id values that are in the movie_library table
--group by statement groups the rows that have same values into summary rows
---list number of departments who did not access start dates before 2020-10-
01
SELECT
  COUNT(department_id) AS num_departments
FROM
  (
    SELECT
      department_id
    FROM
      department MINUS
    SELECT
      department_id
    FROM
      access_movie
    WHERE
      accesss_start_date < '2020-10-01'
  )
GROUP BY
  (department_id);

--the UNION operator is used to combine the result-set of two or more SELECT
statements
---list a count of everyone (employees and customers) in each province
SELECT
  employee_province,
  COUNT(*) AS total_num_people
FROM
  (
    SELECT
      employee_id,
      employee_province
    FROM
      employee
    UNION
    SELECT
      payment_method_id,
      customer_province
    FROM
      payment_method
  )
GROUP BY
  employee_province;
```

```
---lists all movies by id with num of actors with a review more than 8
SELECT
  movie_id,
  COUNT(actor_id) AS num_actors
FROM
  appear_in
WHERE
  EXISTS (
    SELECT
      actor_id
    FROM
      actor
    WHERE
      actor.actor_id = appear_in.actor_id
      AND actor_review > 8
  )
GROUP BY
  movie_id
ORDER BY
  movie_id;
```

## VIEWS

```sql
CREATE VIEW potential_awarded (
  ActorID, First_Name, Last_Name, Review
) AS (
  SELECT
    actor_id,
    actor_first_name,
    actor_last_name,
    actor_review
  FROM
    actor
  WHERE
    actor_review > 8
) WITH READ ONLY;

CREATE VIEW customer_card_type (
  customer_id, customer_name, card_type,
  card_expire_date
) AS (
  SELECT
    customer_id,
    customer_name,
    payment_method.card_type,
    payment_method.card_expire_date
  FROM
    customer,
    payment_method
  WHERE
    customer.payment_method_id = payment_method.payment_method_id
);

CREATE VIEW directed_by (
  movie_name, director_first_name,
  director_last_name
) AS (
  SELECT
    movie_name,
    director.director_first_name,
    director.director_last_name
  FROM
    movie_library,
    director
  WHERE
    movie_library.director_id = director.director_id
);
```

### DROP VIEW

```sql
DROP
  VIEW potential_awarded;
DROP
  VIEW customer_card_type;
DROP
  VIEW directed_by;
```

## LINUX SHELL SOURCE CODE

```
#!/bin/sh

MainMenu()
{
while [ "$CHOICE" != "START" ]
do
clear
echo
"=================================================================="
echo "| Oracle All Inclusive Tool
|"
echo "| Main Menu - Select Desired Operation(s):
|"
echo "| <CTRL-Z Anytime to Enter Interactive CMD Prompt>
|"
echo "--------------------------------------------------------------
----"
echo " $IS_SELECTEDM M) View Manual"
echo " "
echo " $IS_SELECTED1 1) Drop Tables"
echo " $IS_SELECTED2 2) Create Tables"
echo " $IS_SELECTED3 3) Populate Tables"
echo " $IS_SELECTED4 4) Query Join Tables"
echo " $IS_SELECTED5 5) Query Count Tables"
echo " $IS_SELECTED6 6) Drop Views"
echo " $IS_SELECTED7 7) Create Views"
echo " $IS_SELECTED8 8) Query Views"
echo " "
echo " $IS_SELECTEDX X) Force/Stop/Kill Oracle DB"
echo " "
echo " $IS_SELECTEDE E) End/Exit"
echo "Choose: "
read CHOICE
if [ "$CHOICE" == "0" ]
then
echo "Nothing Here"
read -p "Press any key to resume ..."

elif [ "$CHOICE" == "1" ]
then
bash drop_tables.sh
Pause
read -p "Press any key to resume ..."

elif [ "$CHOICE" == "2" ]
then
bash create_tables.sh
Pause
read -p "Press any key to resume ..."
```

```
elif [ "$CHOICE" == "3" ]
then
bash populate_tables.sh
Pause
read -p "Press any key to resume ..."

elif [ "$CHOICE" == "4" ]
then
bash join_queries.sh
Pause
read -p "Press any key to resume ..."

elif [ "$CHOICE" == "5" ]
then
bash count_queries.sh
Pause
read -p "Press any key to resume ..."

elif [ "$CHOICE" == "6" ]
then
bash drop_views.sh
Pause
read -p "Press any key to resume ..."

elif [ "$CHOICE" == "7" ]
then
bash create_views.sh
Pause
read -p "Press any key to resume ..."

elif [ "$CHOICE" == "8" ]
then
bash view_queries.sh
Pause
read -p "Press any key to resume ..."

elif [ "$CHOICE" == "E" ]
then
exit
fi
done
}

#--COMMENTS BLOCK--
# Main Program
#--COMMENTS BLOCK--
ProgramStart()
{
StartMessage
while [ 1 ]
do
MainMenu
done
}

ProgramStart
```

## NORMALIZATION

### FUNCTIONAL DEPENDENCIES

- department (department_id, department_name, department_location)
  - department_id → department_name, department_location
- employee (employee_id, department_id, work_hours, employee_first_name, employee_last_name, employee_gender, employee_dob, employee_province, employee_city, employee_street, employee_postal_code )
  - employee_id → department_id, work_hours, employee_first_name, employee_last_name, employee_gender, employee_dob, employee_province, employee_city, employee_street, employee_postal_code
- genre (genre_name, genre_popularity)
  - genre_name → genre_popularity
- production (production_id, production_name, production_review)
  - production_id → production_name, production_review
- director (director_id, director_first_name, director_last_name, director_review)
  - director_id → director_first_name, director_last_name, director_review
- actor (actor_id, actor_first_name, actor_last_name, actor_review)
  - actor_id → actor_first_name, actor_last_name, actor_review
- movie_library (movie_id,  director_id, movie_name, movie_rating, movie_review, movie_price, release_date, synopsis)
  - movie_id → director_id, movie_name, movie_rating, movie_review, movie_price, release_date, synopsis
- payment_method (payment_method_id, customer_province, customer_city, customer_street, customer_postal_code, card_name, card_type, card_num, card_cvv, card_expire_date)
  - payment_method_id → customer_province, customer_city, customer_street, customer_postal_code, card_name, card_type, card_num, card_cvv, card_expire_date
- customer (customer_id, payment_method_id,  customer_name, customer_email, customer_password)
  - customer_id → payment_method_id, customer_name, customer_email, customer_password
- customer_transaction (transaction_id, customer_id, payment_method_id, transaction_date)
  - transaction_id → transaction_date, customer_id, payment_method_id
- access_movie (access_id, department_id, transaction_id, movie_id, accesss_start_date, access_end_date)
  - access_id → transaction_id, department_id, movie_id, access_start_date, access_end_date

### 3NF

The transitive dependency card_num → card_type needed to be decomposed into a table named card_type, containing attributes card_num and card_type. This table would be referenced to determine the card type based on the first digit of the card number. The resultant tables are as follows:

- department (department_id, department_name, department_location)
  - department_id → department_name, department_location

- employee (employee_id, department_id, work_hours, employee_first_name, employee_last_name, employee_gender, employee_dob, employee_province, employee_city, employee_street, employee_postal_code )
    - employee_id → department_id, work_hours, employee_first_name, employee_last_name, employee_gender, employee_dob, employee_province, employee_city, employee_street, employee_postal_code
- genre (genre_name, genre_popularity)
    - genre_name → genre_popularity
- production (production_id, production_name, production_review)
    - production_id → production_name, production_review
- director (director_id, director_first_name, director_last_name, director_review)
    - director_id → director_first_name, director_last_name, director_review
- actor (actor_id, actor_first_name, actor_last_name, actor_review)
    - actor_id → actor_first_name, actor_last_name, actor_review
- movie_library (movie_id,  director_id, movie_name, movie_rating, movie_review, movie_price, release_date, synopsis)
    - movie_id → director_id, movie_name, movie_rating, movie_review, movie_price, release_date, synopsis
- payment_method (payment_method_id, customer_province, customer_city, customer_street, customer_postal_code, card_name, card_num, card_cvv, card_expire_date)
    - payment_method_id → customer_province, customer_city, customer_street, customer_postal_code, card_name, card_num, card_cvv, card_expire_date
- card_type (card_num, card_type)
    - card_num → card_type
- customer (customer_id, payment_method_id,  customer_name, customer_email, customer_password)
    - customer_id → payment_method_id, customer_name, customer_email, customer_password
- customer_transaction (transaction_id, customer_id, payment_method_id, transaction_date)
    - transaction_id → transaction_date, customer_id, payment_method_id
- access_movie (access_id, department_id, transaction_id, movie_id, accesss_start_date, access_end_date)
    - access_id → transaction_id, department_id, movie_id, access_start_date, access_end_date

## BCNF

To achieve BCNF, an algorithm was used as follows:

**Relation:**
payment_method (payment_method_id, customer_country, customer_province, customer_city, customer_street, customer_postal_code, card_name, card_type, card_num, card_cvv, card_expire_date)

**Step 1**: Find out facts about the real world and list attributes and FD's
payment_method_id → customer_country, customer_province, customer_city, customer_street, customer_postal_code, card_name, card_type, card_num, card_cvv, card_expire_date

card_num → card_type

customer_province → customer_country

customer_postal_code → customer_country, customer_province

**Step 2:** Reduce the list of functional dependencies using polynomial algorithm

payment_method_id → customer_country

payment_method_id → customer_province

payment_method_id → customer_city

payment_method_id → customer_street

payment_method_id → customer_postal_code

payment_method_id → card_name

payment_method_id → card_type

payment_method_id → card_num

payment_method_id → card_cvv

payment_method_id → card_expire_date

card_num → card_type

customer_province → customer_country

customer_postal_code → customer_country

customer_postal_code → customer_province

**Redundancy:**

payment_method+ = {payment_method, customer_province, customer_country, …}

payment_method+ = {payment_method, customer_postal_code, customer_country, customer_province, …}

payment_method+ = {payment_method, customer_country, customer_province, customer_city, customer_street, customer_postal_code, card_name, card_num, card_type, …}

customer_postal_code+ = {customer_postal_code, customer_province, customer_country}

**Step 3**:  Find the keys

LHS & NRHS = {payment_method_id}

NLHS & NRHS = {}

NLHS & RHS = {customer_city, customer_street, card_name, card_cvv, card_expire_date, card_type, customer_country}

(payment_method_id)+ = {payment_method_id, customer_city,  customer_street, customer_postal_code, customer_province, customer_country, card_name, card_num, card_type, card_cvv, card_expire_date}

**It's a KEY**

(payment_method_id, customer_province)+ = {payment_method_id, customer_province, customer_country, customer_city, customer_street, customer_postal_code, card_name, card_num, card_type, card_cvv, card_expire_date}

**Its a KEY**

(payment_method_id, customer_postal_code)+ = {payment_method_id, customer_postal_code, customer_province, customer_country, customer_city, customer_street, customer_postal_code, card_name, card_num,  card_type, card_cvv,  card_expire_date}

**It is a KEY**

(payment_method_id, card_num)+ = {payment_method_id, card_num, customer_city, customer_street, customer_postal_code,  customer_province, customer_country, card_name, card_num, card_type, card_cvv, card_expire_date}

**It is a KEY**

**Step 4**: Derive the final schema by combining the FDs with its respective left-hand side

**R1** (payment_method_id, customer_city, customer_street, customer_postal_code, card_name, card_num, card_cvv,  card_expire_date)

FD's:

{payment_method_id → customer_city, customer_street, customer_postal_code, card_name, card_num, card_cvv, card_expire_date}

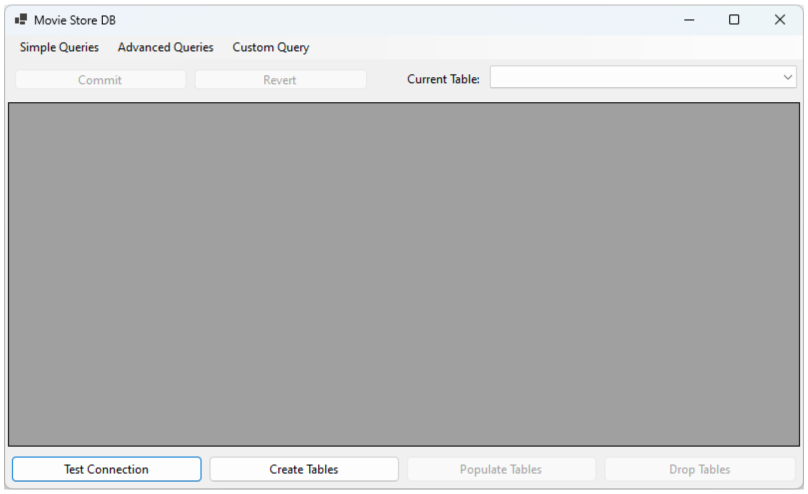
**R2** (card_num, card_type)
FD's:
{card_num → card_type}

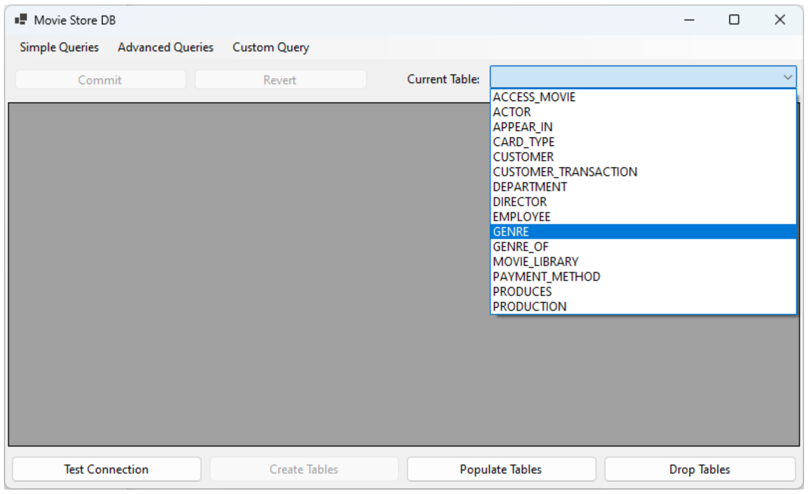
**R3** (customer_province, customer_country)
FD's:
{customer_province → customer_country}

**R4** (customer_postal_code, customer_province)
FD's
{customer_postal_code →  customer_province}

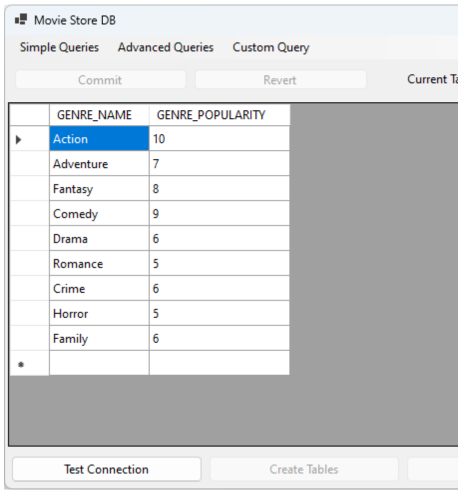
## GUI-BASED APPLICATION

The application for this project was written in .net C#. This simple program allows the user to explore the database system, allowing for records to be added, modified, or deleted. To the right is an image of the main window that the user can interact with.

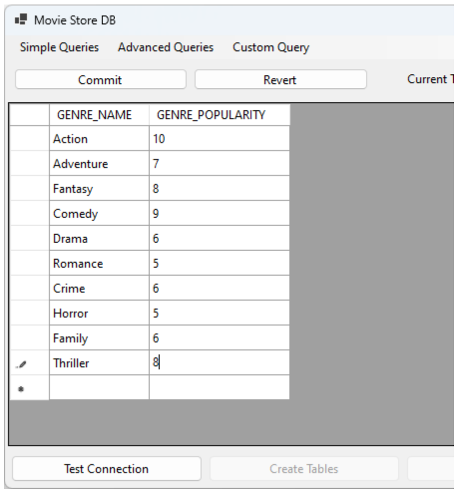




On first launch, the DBMS tables should not yet exist, enabling the "create tables" button located at the bottom of the window. If any tables do exist, then the "drop tables" button will be enabled instead, allowing for the removal of all related tables. The "populate tables" button is only active after the current instance of the program has created the tables for the first time. This function fills the tables with sample data. Along with these three buttons is a fourth button labeled "test connection," that will display a message if the connection is valid or not. On the upper right side of the window is a drop-down box containing all tables in the database. Selecting one from the list will populate the data grid allowing for data review or manipulation. Following this are two cropped images of the populated data grid and the insertion of a single row.



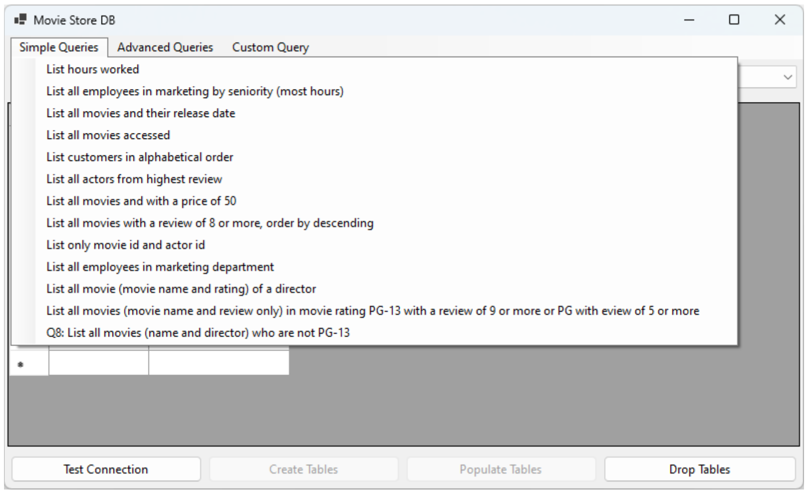
| GENRE_NAME | GENRE_POPULARITY |
|---|---|
| Action | 10 |
| Adventure | 7 |
| Fantasy | 8 |
| Comedy | 9 |
| Drama | 6 |
| Romance | 5 |
| Crime | 6 |
| Horror | 5 |
| Family | 6 |
| | |

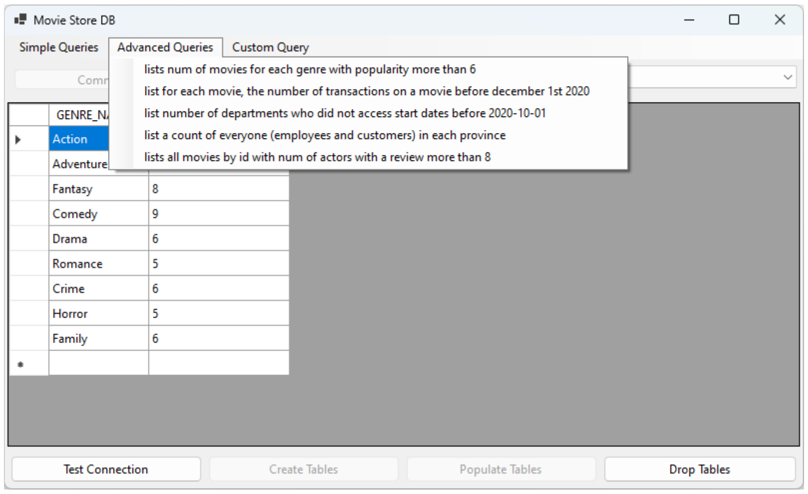


| GENRE_NAME | GENRE_POPULARITY |
|---|---|
| Action | 10 |
| Adventure | 7 |
| Fantasy | 8 |
| Comedy | 9 |
| Drama | 6 |
| Romance | 5 |
| Crime | 6 |
| Horror | 5 |
| Family | 6 |
| Thriller | 8 |
| | |

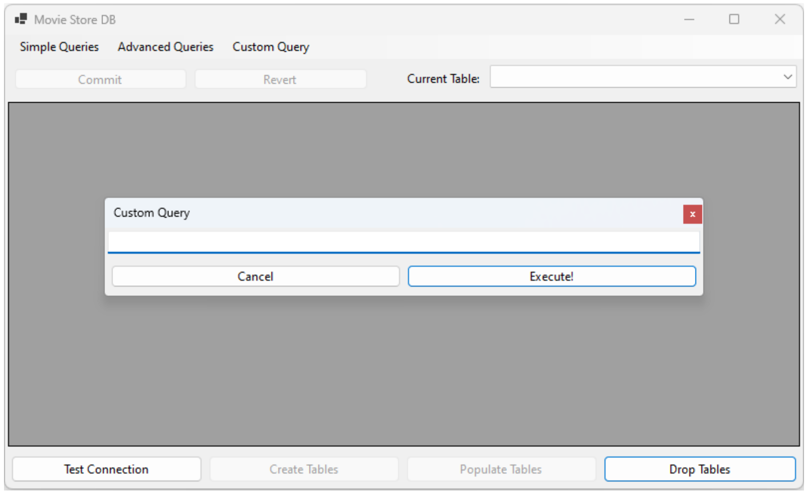
The next three images demonstrate the query functionality of the application. The menu-strip found at the top of the window shows option for simple, advanced, and custom queries. Clicking on either the simple or advanced queries will show a list of pre-programmed options that were loaded from the scripts directory. Loading these scripts externally allows for the simple addition, modification, or removal of queries without the requirement to recompile the main executable. Within these two menus, selecting any item will execute the script query, and display the resultant data (if any) in the data grid.

If the user requires a custom query (typically for a search, but any command is accepted), then the custom query option can be selected in the menu-strip. When this menu item is selected, a dialog window opens, and the user can type their query in the provided textbox, executing the command with the appropriate button. This can be seen in the final image below.


Movie Store DB
Simple Queries
Advanced Queries
Custom Query
List hours worked
List all employees in marketing by seniority (most hours)
List all movies and their release date
List all movies accessed
List customers in alphabetical order
List all actors from highest review
List all movies and with a price of 50
List all movies with a review of 8 or more, order by descending
List only movie id and actor id
List all employees in marketing department
List all movie (movie name and rating) of a director
List all movies (movie name and review only) in movie rating PG-13 with a review of 9 or more or PG with eview of 5 or more
Q8: List all movies (name and director) who are not PG-13
Test Connection
Create Tables
Populate Tables
Drop Tables


Movie Store DB
Simple Queries
Advanced Queries
Custom Query
lists num of movies for each genre with popularity more than 6
list for each movie, the number of transactions on a movie before december 1st 2020
list number of departments who did not access start dates before 2020-10-01
list a count of everyone (employees and customers) in each province
lists all movies by id with num of actors with a review more than 8
Comr
Test Connection
Create Tables
Populate Tables
Drop Tables

| GENRE_N |  |
|---|---|
| Action |  |
| Adventure |  |
| Fantasy | 8 |
| Comedy | 9 |
| Drama | 6 |
| Romance | 5 |
| Crime | 6 |
| Horror | 5 |
| Family | 6 |



## INSTALLATION INSTRUCTIONS (WINDOWS)

To run the application on your own computer, first make sure the local system is connected through the TMU VPN (vpn.scs.ryerson.ca). Without this connection, the application will fail to launch, displaying a connection error message notification. Once the required connection has been made, locate the “binaries” directory that was included alongside this document. Inside this folder will be two items: a “Scripts” directory containing the SQL scripts used by the application, and the main executable named “Movie Store.exe”. Double click on the executable to launch the application.

## RELATIONAL ALGEBRA

### SIMPLE QUERIES

- List hours worked
  - $\Pi_{\text{employee\_first\_name, F sum work\_hours}}$ (employee)
- List all employees in marketing by seniority (most hours)
  - $\Pi_{\text{employee\_first\_name, employee\_last\_name, pwork\_hours/seniority\_in\_marketing\_dep}}$ ($\sigma_{\text{department\_id=1}}$ (employee))
- List all movies and their release date
  - $\Pi_{\text{movie\_name, release\_date}}$ (movie_library)
- List all movies accessed
  - $\Pi_{\text{access\_id, transaction\_id, movie\_id, access\_start\_date, access\_end\_date}}$ ($\sigma_{\text{access\_start\_date <= '2020-12-01'}}$ (access_movie))
- List customers in alphabetical order
  - customer
- List all actors from highest review
  - actor
- List all movies and with a price of 50
  - $\sigma_{\text{movie\_price >= 50}}$ (movie_library)
- List all movies with a review of 8 or more, order by descending
  - $\sigma_{\text{movie\_review >= 8}}$ (movie_library)
- List only movie id and actor id
  - $\Pi_{\text{movie\_id, actor\_id}}$ (appear_in)
- List all employees in marketing department
  - $\rho_{\text{employees\_in\_marketing/employee\_first\_name}}$ ($\sigma_{\text{department\_id=001}}$ (employee))
- List all movie (movie name and rating) of a director
  - $\Pi_{\text{movie\_name, movie\_rating}}$ ($\sigma_{\text{movie\_rating = 'PG-13' AND director\_id = 001}}$ (movie_library))
- List all movies (movie name and review only) in movie rating PG-13 with a review of 9 or more or PG with a review of 5 or more
  - $\Pi_{\text{movie\_name, movie\_review}}$ ($\sigma_{\text{(movie\_rating = 'PG-13' AND movie\_review >= 9) OR (movie\_rating = 'PG' AND movie\_review >= 5)}}$ (movie_library))
- Q8: List all movies (name and director) who are not PG-13
  - $\Pi_{\text{movie\_name, director\_id}}$ ($\sigma_{\text{movie\_rating <> 'PG-13'}}$ (movie_library))

### ADVANCED QUERIES

- lists num of movies for each genre with popularity more than 7
  - $\Pi_{\text{genre\_name, ρ(F count movie\_id)/num\_movies}}$ ($\sigma_{\text{genre\_popularity > 7}}$ (genre ⋈ genre_of))
- list for each movie, the number of transactions on a movie before December 1st 2020
  - $\Pi_{\text{movie\_id, ρ(F count transaction\_id)/num\_transactions}}$ ($\sigma_{\text{transaction\_date < '2020-12-01'}}$ (customer_transaction ⋈ access_movie))
- list number of departments who did not access start dates before 2020-10-01
  - $\rho_{\text{(F count department\_id)/num\_departments}}$ ((department - ($\sigma_{\text{access\_start\_date < '2020-10-01'}}$ (access_movie)))
- list a count of everyone (employees and customers) in each province
  - $\Pi_{\text{employee\_province, ρ(F count *)/total\_num\_people}}$ (employee ∪ payment_method)
- lists all movies by id with num of actors with a review more than 8
  - $\Pi_{\text{employee\_province, ρ(F count actor\_id)/num\_actors}}$ ($\sigma_{\text{actor\_review > 8}}$ (actor ⋈ appear_in))

## CLOSING STATEMENTS

The design process throughout this project was a valuable learning experience. It demonstrated the advantages of following a structured approach toward developing a DBMS with access through a client application. During this project, our group identified several concepts that required refining. One notable change was the continual modification of the ER-diagram and associated application description. As our group learned about the intricacies of a DBMS, we discovered that this project would benefit from the added knowledge that we had obtained. The most influential stage for change was database normalization. During this stage, tables were modified and deleted to accommodate higher normalization views.