ONLINE MOVIE STORE

DBMS Final Report

Course: CPS 510

Section Number: 9

Team Number: 9

Group Members:

Julian Kuyumcu (500895647)

Jonathan Ma (500837227)

Chun Yiu Cheung (501029833)

Table of Contents

Introduction

Description of Project

The Online Movie Store DBMS is responsible for storing and maintaining information required for an online movie business. The database system stores data related to customer information, movie information, available movies for purchase, subscription tiers and rates and more. The primary end users of this system are administrators and customers, administrators are defined as users that maintain the database. Essentially, they are the users that perform common database functions like inserting, deleting and updating information about new movies and customers. Our main focus of this project is to create an easily accessible yet efficient system for purchasing and viewing movies.

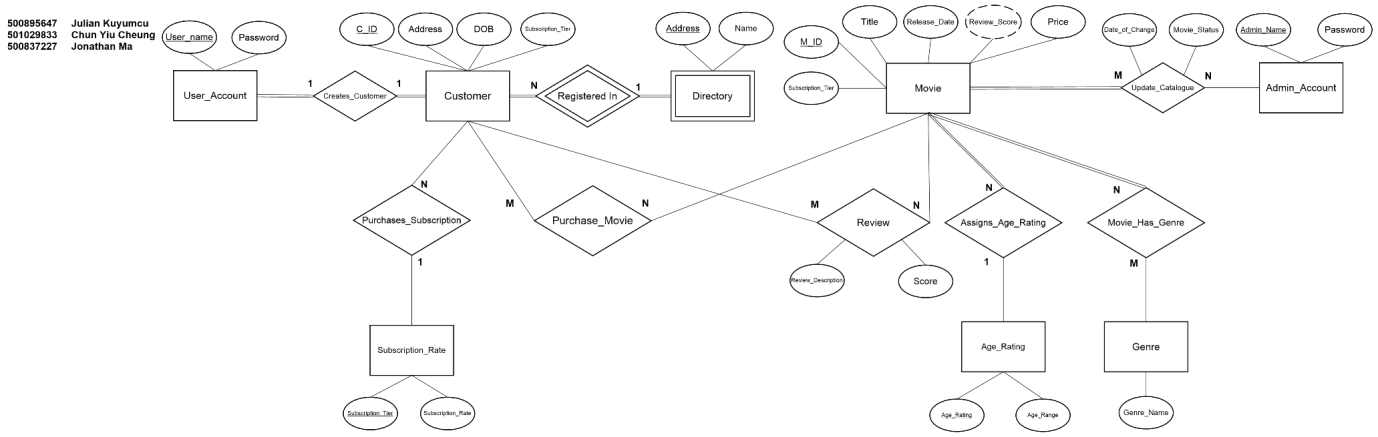
Basic Functions

Some of the basic functions that our system will be capable of doing:

Functions	Description
Insert new customer	Inserts information about a new customer into the database. Information such as their unique Customer ID, Address, DOB and subscription tier they registered with
Remove a customer	Remove an existing customer from our database
Adding new or old movies for customers	Inserts information about a new movie into the database for customer viewing, movies are rated and has a specific tier in which customers with that tier can purchase the movie.
Leaving a movie review	Customers can leave reviews on movies they purchased/viewed, these will be available for other customers to see, the movie review score is also averaged out between all reviews in the initial movie screen.
Viewing and Sorting movies based on specifics	Customers can view and sort movies based on genre, review score, alphabetical order of movie titles, release date and subscription tier/price
Retrieve movie catalogue updates	Administrators and publishers can add and remove movies, a table keeping track of all those transactions is available for viewing

Entity-Relationship Diagram

500895647 Julian Kuyumcu
501029833 Chun Yiu Cheung
500837227 Jonathan Ma



Entities	Relationships
- User_Account - Customer - Directory - Subscription_Rate - Movie - Admin_Account - Age_Rating - Genre	- User_Account (1) creates Customer (1) - A Subscription_Rate (1) can be applied to Multiple Customers (N) - Directory (1) contains Multiple Customers (N) - Multiple Customers (M) can purchase Multiple Movies (N) - Multiple Customers (M) can leave reviews for Multiple Movies (N) - Multiple Movies (N) can be assigned the same Age_Rating (1) - Multiple Movies (N) can have Multiple Genres (M) - Multiple Admin_Accounts (N) can update catalogue of Multiple Movies (M)

Schema Design

Creating Tables

```
CREATE TABLE User_Account (
    User_name VARCHAR(50) PRIMARY KEY,
    Password VARCHAR(20) CHECK (LENGTH(Password) > 7 AND LENGTH(Password) < 21) NOT NULL
);

CREATE TABLE Customer(
    C_ID INTEGER PRIMARY KEY,
    Address VARCHAR(1000) UNIQUE,
    DOB DATE,
    Subscription_Tier VARCHAR(20) NOT NULL CHECK (Subscription_Tier = 'Free' OR
        Subscription_Tier = 'Bronze' OR Subscription_Tier = 'Silver' OR
        Subscription_Tier = 'Gold' OR Subscription_Tier = 'Platinum')
);

CREATE TABLE Directory(
    Address VARCHAR(1000) REFERENCES Customer(Address) ON DELETE CASCADE,
    Name VARCHAR(1000)
);

CREATE TABLE Creates_Customer(
    Username VARCHAR(50) REFERENCES User_Account(User_name) ON DELETE CASCADE,
    C_ID INTEGER REFERENCES Customer(C_ID) ON DELETE CASCADE
);

CREATE TABLE Movie(
    M_ID INTEGER,
    Title VARCHAR(200) NOT NULL,
    Subscription_Tier VARCHAR(20) NOT NULL CHECK (Subscription_Tier = 'Free' OR
        Subscription_Tier = 'Bronze' OR Subscription_Tier = 'Silver' OR
        Subscription_Tier = 'Gold' OR Subscription_Tier = 'Platinum'),
    Release_Date INTEGER,
    Review_Score DECIMAL(3, 2),
    Price DECIMAL(5, 2) NOT NULL,
    PRIMARY KEY(M_ID)
);

CREATE TABLE Genre(
    Genre_Name VARCHAR(50) PRIMARY KEY
);
```

```
CREATE TABLE Movie_Has_Genre (
   Genre_Name VARCHAR(50) REFERENCES Genre(Genre_Name) ON DELETE CASCADE,
   M_ID INTEGER REFERENCES Movie(M_ID) ON DELETE CASCADE
);

CREATE TABLE Admin_Account  (
   Admin_Name VARCHAR(50) PRIMARY KEY,
   Password VARCHAR(20) CHECK (LENGTH(Password) > 7 AND LENGTH(Password) < 21) NOT NULL
);

CREATE TABLE Subscription_Rate (
   Subscription_Tier VARCHAR(20),
   Subscription_Rate DECIMAL(5,2) CHECK (Subscription_Rate >= 0),
   PRIMARY KEY(Subscription_Tier)
);



CREATE TABLE Purchase_Movie(
   C_ID INTEGER REFERENCES Customer(C_ID) ON DELETE CASCADE,
   M_ID INTEGER REFERENCES Movie(M_ID) ON DELETE CASCADE
);

CREATE TABLE Review (
   C_ID INTEGER REFERENCES Customer(C_ID) ON DELETE CASCADE,
   M_ID INTEGER REFERENCES Movie(M_ID) ON DELETE CASCADE,
   Review_Description VARCHAR2(4000),
   Score DECIMAL(2,1) NOT NULL CHECK (Score >= 1.0 AND Score <= 5.0 AND MOD(Score, 0.5) = 0)
);

CREATE TABLE Update_Catalogue (
   Admin_Name VARCHAR(50) REFERENCES Admin_Account(Admin_Name) ON DELETE CASCADE,
   M_ID INTEGER REFERENCES Movie(M_ID) ON DELETE CASCADE,
   Date_Change DATE NOT NULL,
   Movie_Status VARCHAR(6) NOT NULL CHECK (Movie_Status = 'Add' OR Movie_Status = 'Remove')
);

CREATE TABLE Age_Rating(
   Age_Rating VARCHAR(8) PRIMARY KEY,
   Age_Range VARCHAR(6) NOT NULL CHECK (REGEXP_LIKE(Age_Range, '^[0-9]{1,2}-[0-9]{1,3}$'))
);
```

```
CREATE TABLE Assign_Age_Rating(
    M_ID INTEGER REFERENCES Movie(M_ID) ON DELETE CASCADE,
    Age_Rating VARCHAR(8) REFERENCES Age_Rating(Age_Rating) ON DELETE CASCADE
);
```

Populating Tables

```
INSERT INTO User_Account (User_name, Password) VALUES ('username1', 'password1');
INSERT INTO User_Account (User_name, Password) VALUES ('username2', 'password2');
INSERT INTO User_Account (User_name, Password) VALUES ('username3', 'password3');
INSERT INTO User_Account (User_name, Password) VALUES ('username4', 'password4');
INSERT INTO User_Account (User_name, Password) VALUES ('username5', 'password5');

alter SESSION set NLS_DATE_FORMAT = 'YYYY-MM-DD';
INSERT INTO Customer VALUES(1, '1 Fake St. Toronto ON', '2015-01-01', 'Gold');
INSERT INTO Customer VALUES(2, '2 Unreal Ave. Montreal QC', '2002-12-01', 'Silver');
INSERT INTO Customer VALUES(3, '3 Fabrication Cres. Vancouver BC', '2009-02-05', 'Bronze');
INSERT INTO Customer VALUES(4, '21 Yonge St. Toronto ON', '1995-03-03', 'Gold');
INSERT INTO Customer VALUES(5, '34 Queen St. Vancouver BC', '1999-02-02', 'Bronze');

INSERT INTO Directory VALUES('1 Fake St. Toronto ON','John Doe');
INSERT INTO Directory VALUES('2 Unreal Ave. Montreal QC','Jane Naught');
INSERT INTO Directory VALUES('3 Fabrication Cres. Vancouver BC','James Stew');
INSERT INTO Directory VALUES('21 Yonge St. Toronto ON','Drake');
INSERT INTO Directory VALUES('34 Queen St. Vancouver BC','Alex Chan');

INSERT INTO Creates_Customer VALUES ('username1', 1);
INSERT INTO Creates_Customer VALUES ('username2', 2);
INSERT INTO Creates_Customer VALUES ('username3', 3);
INSERT INTO Creates_Customer VALUES ('username4', 4);
INSERT INTO Creates_Customer VALUES ('username5', 5);

INSERT INTO movie VALUES(1, 'Top Gun', 'Bronze', 1986, NULL, 15.99);
INSERT INTO movie VALUES(2, 'Old Movie A', 'Bronze', 1990, NULL, 20.99);
INSERT INTO movie VALUES(3, 'New movie A', 'Gold', 2022, NULL, 60.99);
INSERT INTO movie VALUES(4, 'New movie B', 'Silver', 2021, NULL, 45.99);
INSERT INTO movie VALUES(5, 'Indiana Jones', 'Bronze', 1989, NULL, 15.99);
INSERT INTO movie VALUES(6, 'The Avengers', 'Bronze', 2012, NULL, 15.99);

INSERT INTO Genre VALUES ('Action');
INSERT INTO Genre VALUES ('Horror');
INSERT INTO Genre VALUES ('Comedy');
INSERT INTO Genre VALUES ('Sci-Fi');

INSERT INTO Movie_Has_Genre VALUES ('Action', 1);
INSERT INTO Movie_Has_Genre VALUES ('Horror', 2);
INSERT INTO Movie_Has_Genre VALUES ('Sci-Fi', 3);
INSERT INTO Movie_Has_Genre VALUES ('Comedy', 4);
INSERT INTO Movie_Has_Genre VALUES ('Action', 5);
INSERT INTO Movie_Has_Genre VALUES ('Action', 6);
```

```
INSERT INTO Admin_Account VALUES ('siteadmin1', 'password1');
INSERT INTO Admin_Account VALUES ('publisher1', 'password2');

INSERT INTO subscription_rate VALUES('Gold', 20.25);
INSERT INTO subscription_rate VALUES('Silver', 10.25);
INSERT INTO subscription_rate VALUES('Bronze', 5.25);

INSERT INTO purchase_movie VALUES(1, 1);
INSERT INTO purchase_movie VALUES(1, 2);
INSERT INTO purchase_movie VALUES(1, 3);
INSERT INTO purchase_movie VALUES(1, 4);
INSERT INTO purchase_movie VALUES(1, 5);
INSERT INTO purchase_movie VALUES(1, 6);
INSERT INTO purchase_movie VALUES(2, 4);
INSERT INTO purchase_movie VALUES(2, 1);
INSERT INTO purchase_movie VALUES(3, 2);
INSERT INTO purchase_movie VALUES(3, 5);
INSERT INTO purchase_movie VALUES(4, 3);
INSERT INTO purchase_movie VALUES(4, 4);
INSERT INTO purchase_movie VALUES(5, 1);
INSERT INTO purchase_movie VALUES(5, 6);

INSERT INTO review VALUES(1, 1, 'Good movie, kinda long but overall enjoyable', 3.5);
INSERT INTO review VALUES(1, 2, 'This movie is so old!', 2.0);
INSERT INTO review VALUES(1, 3, 'Amazing, definitely worth the Gold tier!', 5.0);
INSERT INTO review VALUES(1, 4, 'Sort of boring at times, but it really picked up towards the end', 4.5);
INSERT INTO review VALUES(1, 5, 'A classic', 4.0);
INSERT INTO review VALUES(1, 6, 'I mean, its the Avengers', 5.0);
INSERT INTO review VALUES(2, 4, 'Meh', 1.5);
INSERT INTO review VALUES(3, 2, 'Decent!', 3.0);
INSERT INTO review VALUES(4, 3, 'Really good', 4.0);
INSERT INTO review VALUES(4, 4, 'I got the Silver tier just for this, super good', 5.0);
INSERT INTO review VALUES(5, 1, 'Trash', 1.0);
INSERT INTO review VALUES(5, 6, 'Typical Marvel stuff, which means its good', 3.5);

INSERT INTO Update_Catalogue VALUES ('siteadmin1', 1, '2022-09-28', 'Add');
INSERT INTO Update_Catalogue VALUES ('siteadmin1', 2, '2022-09-28', 'Add');
INSERT INTO Update_Catalogue VALUES ('siteadmin1', 3, '2022-09-28', 'Add');
INSERT INTO Update_Catalogue VALUES ('siteadmin1', 4, '2022-09-28', 'Add');
INSERT INTO Update_Catalogue VALUES ('siteadmin1', 5, '2022-09-28', 'Add');
INSERT INTO Update_Catalogue VALUES ('publisher1', 6, '2022-10-02', 'Add');
INSERT INTO Update_Catalogue VALUES ('publisher1', 6, '2022-10-04', 'Remove');
INSERT INTO Update_Catalogue VALUES ('siteadmin1', 4, '2022-09-04', 'Remove');
INSERT INTO Update_Catalogue VALUES ('publisher1', 2, '2022-10-10', 'Remove');

INSERT INTO age_rating VALUES ('G', '0-200');
INSERT INTO age_rating VALUES('PG', '8-12');
INSERT INTO age_rating VALUES('PG-13', '13-17');
INSERT INTO age_rating VALUES('R', '18-200');

INSERT INTO assign_age_rating VALUES(1, 'PG-13');
INSERT INTO assign_age_rating VALUES(2, 'R');
INSERT INTO assign_age_rating VALUES(3, 'PG');
```

```
INSERT INTO assign_age_rating VALUES(4, 'G');
INSERT INTO assign_age_rating VALUES(5, 'PG-13');
INSERT INTO assign_age_rating VALUES(6, 'PG-13');
```

Update Movie Review Scores

```
UPDATE Movie m SET m.review_score = (SELECT AVG(r.score) FROM Review r WHERE m.M_ID = r.M_ID);
```

Dropping Tables

```
DROP TABLE ASSIGN_AGE_RATING cascade constraints purge;
DROP TABLE CREATES_CUSTOMER cascade constraints purge;
DROP TABLE DIRECTORY cascade constraints purge;
DROP TABLE MOVIE_HAS_GENRE cascade constraints purge;
DROP TABLE PURCHASE_MOVIE cascade constraints purge;
DROP TABLE REVIEW cascade constraints purge;
DROP TABLE SUBSCRIPTION_RATE cascade constraints purge;
DROP TABLE UPDATE_CATALOGUE cascade constraints purge;
DROP TABLE USER_ACCOUNT cascade constraints purge;
DROP TABLE ADMIN_ACCOUNT cascade constraints purge;
DROP TABLE AGE_RATING cascade constraints purge;
DROP TABLE CUSTOMER cascade constraints purge;
DROP TABLE GENRE cascade constraints purge;
DROP TABLE MOVIE cascade constraints purge;
```

Simple Database Queries and Views

User Account Queries

SELECT * FROM user_account ORDER BY user_name ASC;

	USER_NAME	PASSWORD
1	username1	password1
2	username2	password2
3	username3	password3

(RA): $\tau_{\text{user\_name}}(\sigma(\text{user\_account}))$

Customer Queries

SELECT * FROM customer WHERE Address LIKE '%Toronto%' ORDER BY C_ID ASC;

	C_ID	ADDRESS	DOB	SUBSCRIPTION_TIER
1	1	1 Fake St. Toronto ON	15-01-01	Gold
2	4	21 Yonge St. Toronto ON	95-03-03	Gold

(RA): $\tau_{\text{C\_ID}}(\sigma_{\text{Address LIKE \%Toronto\%}}(\text{customer}))$

SELECT * FROM customer ORDER BY ASC;

	C_ID	ADDRESS	DOB	SUBSCRIPTION_TIER
1	4	21 Yonge St. Toronto ON	95-03-03	Gold
2	5	34 Queen St. Vancouver BC	99-02-02	Bronze
3	2	2 Unreal Ave. Montreal QC	02-12-01	Silver
4	3	3 Fabrication Cres. Vancouver BC	09-02-05	Bronze
5	1	1 Fake St. Toronto ON	15-01-01	Gold

(RA): $\tau_{\text{DOB}}(\sigma(\text{customer}))$

Movie Queries

SELECT Title, Price FROM Movie ORDER BY Price ASC;

	TITLE	PRICE
1	Top Gun	15.99
2	The Avengers	15.99
3	Indiana Jones	15.99
4	Old Movie A	20.99
5	New movie B	45.99
6	New movie A	60.99

(RA): $\tau_{\text{Price}}(\Pi_{\text{Title, Price}}(\sigma(\text{Movie})))$

SELECT * FROM movie WHERE subscription_tier LIKE 'Bronze' ORDER BY release_date DESC;

	M_ID	TITLE	SUBSCRIPTION_TIER	RELEASE_DATE	REVIEW_SCORE	PRICE
1	6	The Avengers	Bronze	2012	(null)	15.99
2	2	Old Movie A	Bronze	1990	(null)	20.99
3	5	Indiana Jones	Bronze	1989	(null)	15.99
4	1	Top Gun	Bronze	1986	(null)	15.99

(RA): $\tau_{\text{release\_date}\downarrow}(\sigma_{\text{subscription\_tier LIKE 'Bronze}}(\text{Movie}))$

Movie Genre Queries

SELECT * FROM Genre ORDER BY genre_name ASC;

	GENRE_NAME
1	Action
2	Comedy
3	Horror
4	Sci-Fi

(RA): $\tau_{\text{genre\_name}}(\sigma(\text{Genre }))$

SELECT * FROM movie_has_genre ORDER BY genre_name, M_ID ASC;

	GENRE_NAME	M_ID
1	Action	1
2	Action	5
3	Action	6
4	Comedy	4
5	Horror	2
6	Sci-Fi	3

(RA): $\tau_{\text{genre\_name, M\_ID}}\sigma(\text{movie\_has\_genre })$

SELECT COUNT(M_id), genre_name FROM movie_has_genre GROUP BY genre_name ORDER BY COUNT(M_id) DESC;

	COUNT(M_ID)	GENRE_NAME
1	3	Action
2	1	Horror
3	1	Sci-Fi
4	1	Comedy

(RA): $\tau_{\text{COUNT M\_id}\downarrow}$ (genre_name ^{F} COUNT M_id, genre_name (movie_has_genre))

Movie Review Queries

SELECT * FROM reviews WHERE score BETWEEN 2.0 AND 3.0;

	C_ID	M_ID	REVIEW_DESCRIPTION	SCORE
1	1	2	This movie is so old!	2
2	3	2	Decent!	3

(RA): $\sigma_{score >= 2.0 \text{ AND } score <= 3.0}$ (reviews)

Movie Age Rating Queries

SELECT age_range FROM age_rating WHERE age_rating = 'R';

	AGE_RANGE
1	18-200

(RA): $\Pi_{age\_range}(\sigma_{age\_rating = \text{'R'}}$ (age_rating)))

SELECT * FROM Assign_age_rating ORDER BY Age_Rating ASC, M_ID DESC;

	M_ID	AGE_RATING
1	4	G
2	3	PG
3	6	PG-13
4	5	PG-13
5	1	PG-13
6	2	R

(RA): $\tau_{\text{Age\_Rating , M\_id}\downarrow}(\sigma$(Assign_age_rating))

SELECT count(M_id), age_rating FROM assign_age_rating GROUP BY age_rating ORDER BY COUNT(M_id) DESC;

	COUNT(M_ID)	AGE_RATING
1	3	PG-13
2	1	G
3	1	PG
4	1	R

(RA): $\tau_{\text{COUNT M\_id}\downarrow}$(age_rating ^{F} COUNT M_id, age_rating (assign_age_rating))

Subscription Queries

SELECT * FROM subscription_rate ORDER BY subscription_rate DESC;

	SUBSCRIPTION_TIER	SUBSCRIPTION_RATE
1	Gold	20.25
2	Silver	10.25
3	Bronze	5.25
4	Free	0

(RA): $\sigma_{\text{score >= 2.0 AND score <= 3.0}}$ (reviews)

Administrative/Maintenance Queries

SELECT * FROM Update_Catalogue ORDER BY admin_name ASC, m_id ASC;

	ADMIN_NAME	M_ID	DATE_CHANGE	MOVIE_STATUS
1	publisher1	1	22-09-15	Remove
2	publisher1	2	22-09-28	Remove
3	publisher1	2	22-10-02	Add
4	publisher1	3	22-09-01	Remove
5	siteadmin1	1	22-09-28	Add
6	siteadmin1	2	22-09-28	Add
7	siteadmin1	3	22-09-28	Add
8	siteadmin1	4	22-09-28	Add
9	siteadmin1	4	22-09-04	Remove

(RA): $\tau_{\text{admin\_name, M\_id}}$ (σ(Update_Catalogue))

SELECT * FROM Update_Catalogue WHERE date_Change < '2022-09-30';

	ADMIN_NAME	M_ID	DATE_CHANGE	MOVIE_STATUS
1	siteadmin1	1	22-09-28	Add
2	siteadmin1	2	22-09-28	Add
3	siteadmin1	3	22-09-28	Add
4	siteadmin1	4	22-09-28	Add
5	publisher1	1	22-09-15	Remove
6	publisher1	3	22-09-01	Remove
7	siteadmin1	4	22-09-04	Remove
8	publisher1	2	22-09-28	Remove

(RA): $\sigma_{\text{date\_Change < '2022-09-30'}}$(Update_Catalogue)

Creating Views

Create views on movie availability for each subscription tier, as customers of each tier can only access movies of their tier or lower

```
CREATE VIEW movie_tier_bronze AS
(SELECT title, release_date, review_score, price
FROM movie
WHERE subscription_tier = 'Bronze')
WITH READ ONLY;
```

	TITLE	RELEASE_DATE	REVIEW_SCORE	PRICE
1	Top Gun	1986	(null)	15.99
2	Old Movie A	1990	(null)	20.99
3	Indiana Jones	1989	(null)	15.99
4	The Avengers	2012	(null)	15.99

```
CREATE VIEW movie_tier_silver AS
(SELECT title, release_date, review_score, price
FROM movie
WHERE subscription_tier = 'Bronze'
OR subscription_tier = 'Silver')
WITH READ ONLY;
```

	TITLE	RELEASE_DATE	REVIEW_SCORE	PRICE
1	Top Gun	1986	(null)	15.99
2	Old Movie A	1990	(null)	20.99
3	New movie B	2021	(null)	45.99
4	Indiana Jones	1989	(null)	15.99
5	The Avengers	2012	(null)	15.99

```
CREATE VIEW movie_tier_gold AS
(SELECT title, release_date, review_score, price
FROM movie
WHERE subscription_tier = 'Bronze'
OR subscription_tier = 'Silver'
OR subscription_tier = 'Gold')
WITH READ ONLY;
```

	TITLE	RELEASE_DATE	REVIEW_SCORE	PRICE
1	Top Gun	1986	(null)	15.99
2	Old Movie A	1990	(null)	20.99
3	New movie A	2022	(null)	60.99
4	New movie B	2021	(null)	45.99
5	Indiana Jones	1989	(null)	15.99
6	The Avengers	2012	(null)	15.99

Advanced Database Queries

Show the names of the customers who purchased any movie(s) in alphabetical order and display the title of the movie(s) they purchased also in alphabetical order.

```
SELECT purchase_movie.C_ID, purchase_movie.M_ID, directory.name, movie.title
FROM purchase_movie
JOIN customer ON purchase_movie.C_ID = customer.c_ID
JOIN directory ON customer.address = directory.address
JOIN movie ON purchase_movie.M_ID = movie.M_ID
ORDER BY Name ASC, Title ASC;
```

	C_ID	M_ID	NAME	TITLE
1	5	5	Alex Chan	Indiana Jones
2	5	2	Alex Chan	Old Movie A
3	5	6	Alex Chan	The Avengers
4	5	1	Alex Chan	Top Gun
5	4	5	Drake	Indiana Jones
6	4	3	Drake	New movie A
7	4	4	Drake	New movie B
8	4	6	Drake	The Avengers
9	3	5	James Stew	Indiana Jones
10	3	2	James Stew	Old Movie A
11	3	1	James Stew	Top Gun
12	2	3	Jane Naught	New movie A
13	2	4	Jane Naught	New movie B
14	2	1	Jane Naught	Top Gun
15	1	5	John Doe	Indiana Jones
16	1	3	John Doe	New movie A
17	1	4	John Doe	New movie B
18	1	2	John Doe	Old Movie A
19	1	6	John Doe	The Avengers
20	1	1	John Doe	Top Gun

(RA): $\tau_{\text{Name, Title}}$ ($\Pi_{\text{C\_ID, M\_ID, name, title}}$ (σ(purchase_movie $><_{\text{purchase\_movie.M\_ID = customer.C\_ID}}$ customer $><_{\text{customer.address = directory.address}}$ directory $><_{\text{purchae\_movie.M\_ID = movie.M\_ID}}$ movie)))

Order the movies by age rating, from general audiences to restricted, also, show the price and the appropriate age range for each respective movie.

```
SELECT assign_age_rating.M_ID, assign_age_rating.age_rating, age_rating.age_range, movie.title,
movie.price
FROM assign_age_rating
JOIN movie ON assign_age_rating.M_ID = movie.M_ID
```

JOIN age_rating ON assign_age_rating.age_rating = age_rating.age_rating

ORDER BY age_rating ASC, title ASC;

	M_ID	TITLE	AGE_RATING	AGE_RANGE	PRICE
1	4	New movie B	G	0–200	45.99
2	3	New movie A	PG	8–13	60.99
3	5	Indiana Jones	PG–13	13–18	15.99
4	6	The Avengers	PG–13	13–18	15.99
5	1	Top Gun	PG–13	13–18	15.99
6	2	Old Movie A	R	18–200	20.99

(RA):$\tau_{age\_rating,\ title}$ ($\Pi_{M\_ID,\ age\_rating,\ age\_range,\ title,\ price}$ (σ(assign_age_rating $><_{assign\_age\_rating.M\_ID = movie.M\_ID}$ movie $><_{assign\_age\_rating.age\_rating = age\_rating.age\_rating}$ age_rating)))

Shows description and score for all reviews for "New movie B" as well as the username responsible, sorted by score in descending order. Implicitly joins tables Movie, Review, and Creates_Customer.

SELECT m.Title, r.Review_Description, r.Score, cc.Username

FROM Movie m, Review r, Creates_Customer cc

WHERE m.Title = 'New movie B'

AND m.M_ID = r.M_ID

AND cc.C_ID = r.C_ID

ORDER BY Score DESC;

	TITLE	REVIEW_DESCRIPTION	SCORE	USERNAME
1	New movie B	I got the Silver tier just for this, super good	5	username4
2	New movie B	Sort of boring at times, but it really picked up towards the end	4.5	username1
3	New movie B	Meh	1.5	username2

(RA): $\tau_{Score\downarrow}$ ($\Pi_{Title,\ Review\_Description,\ Score,\ Username}$($\sigma_{movie.Title = 'New\ movie\ B'}$(Movie $><_{movie.M\_ID = review.M\_ID}$ Review $><_{cc.C\_ID = r.C\_ID}$ Creates_Customer)))

List the best and worst reviews for all movies. This is useful for gauging movie performance relative to the current market.

```
(SELECT m_id, review_description, score
FROM review
WHERE (score = 5))
UNION
(SELECT m_id, review_description, score
FROM review
WHERE (score = 1))
ORDER BY score DESC;
```

	M_ID	REVIEW_DESCRIPTION	SCORE
1	3	Amazing, definitely worth the Gold tier!	5
2	4	I got the Silver tier just for this, super good	5
3	6	I mean, its the Avengers	5
4	1	Trash	1

(RA): best_reviews $\leftarrow \Pi_{\text{m\_id, review\_description, score}}(\sigma_{\text{score} = 5}(\text{review}))$

worst_reviews $\leftarrow \Pi_{\text{m\_id, review\_description, score}}(\sigma_{\text{score} = 1}(\text{review}))$

R(M_ID, Review_Description, Score) $\leftarrow \tau_{\text{Score}\downarrow}$ (best_reviews $\cup$ worst_reviews)

Count sales number for each movie. This query is useful for market analysis.

```
SELECT M_ID, COUNT(C_ID) as Sales
FROM Purchase_movie
GROUP BY M_ID
ORDER BY Sales DESC;
```

	M_ID	SALES
1	1	3
2	4	3
3	5	2
4	3	2
5	6	2
6	2	2

(RA): R(M_ID, Sales) $\leftarrow \tau_{\text{COUNT C\_ID}\downarrow}$ (M_ID $^{\text{F}}$ M_ID, COUNT C_ID (Purchase_movie))

Show the list of movies with above average sales, in descending order. This query is useful for further market analysis.

```
SELECT M_ID, COUNT(C_ID) as Sales
FROM Purchase_movie
GROUP BY M_ID
HAVING COUNT(C_ID) > (
   SELECT AVG(Sales)
   FROM ( SELECT COUNT (C_ID) AS Sales
      FROM Purchase_movie
      GROUP BY M_ID )
)
ORDER BY Sales DESC;
```

	M_ID	SALES
1	4	3
2	1	3

Get all usernames of customers who have left reviews. This can be used in loyalty/rewards promotions on the storefront.

```
SELECT cc.Username FROM Creates_Customer cc
WHERE EXISTS (
   SELECT 1 FROM Review r
   WHERE cc.C_ID = r.C_ID
);
```

	USERNAME
1	username1
2	username2
3	username3
4	username4
5	username5

(RA): $R(Username) \leftarrow \Pi_{Username}(\sigma(Creates\_Customer)) \cap \Pi_{Username}(\sigma(Review))$

Database Normalization

Pre Normalization - Functional Dependencies

User_Account(User_name, Password)

- User_name → Password

Customer(C_ID, Name, Address, DOB)

- C_ID → Name
- C_ID → Address
- C_ID → DOB

Creates_Customer(Username, C_ID)

- Username → C_ID
- C_ID → Username

Movie(M_ID, Title, Subscription_Tier, Release_Date, Review_Score, Price)

- M_ID → Title
- M_ID → Subscription_Tier
- M_ID → Release_Date
- M_ID → Review_Score
- M_ID → Price

Genre(Genre_Name)

- None

Movie_Has_Genre(Genre_Name, M_ID)

- None (Many-to-many relationship)

Admin_Account(Admin_Name, Password)

- Admin_Name → Password

Subscription_Rate(Subscription_Tier, Subscription_Rate)

- Subscription_Tier → Subscription_Rate

Purchases_Subscription(C_ID, Subscription_Tier)

- C_ID → Subscription_Tier

Purchase_Movie(C_ID, M_ID)

- None (Many-to-many relationship)

Review(C_ID, M_ID, Review_Description, Score)

- C_ID, M_ID → Review_Description
- C_ID, M_ID → Score

Update_Catalogue(Admin_Name, M_ID, Date_Change, Movie_Status)

- Admin_Name, M_ID → Date_Change
- Admin_Name, M_ID → Movie_Status

Age_Rating(Age_Rating, Age_Range)

- Age_Rating → Age_Range

Assign_Age_Rating(M_ID, Age_Rating)

- M_ID → Age_Rating

Normalization into 3NF

Tables already in 3NF:

User_Account(User_name, Password)

- User_name → Password

Creates_Customer(Username, C_ID)-

- Username → C_ID
- C_ID → Username

Movie(M_ID, Title, Subscription_Tier, Release_Date, Review_Score, Price)

- M_ID → Title
- M_ID → Subscription_Tier
- M_ID → Release_Date
- M_ID → Review_Score
- M_ID → Price

Genre(Genre_Name)

- None

Movie_Has_Genre(Genre_Name, M_ID)

- None (Many-to-many relationship)

Admin_Account(Admin_Name, Password)

- Admin_Name → Password

Purchases_Subscription(C_ID, Subscription_Tier)

- C_ID → Subscription_Tier

Purchase_Movie(C_ID, M_ID)

- None (Many-to-many relationship)

Review(C_ID, M_ID, Review_Description, Score)

- C_ID, M_ID → Review_Description
- C_ID, M_ID → Score

Update_Catalogue(Admin_Name, M_ID, Date_Change, Movie_Status)

- Admin_Name, M_ID → Date_Change
- Admin_Name, M_ID → Movie_Status

Age_Rating(Age_Rating, Age_Range)

- Age_Rating → Age_Range

Assign_Age_Rating(M_ID, Age_Rating)

- M_ID → Age_Rating

Compound Functional Dependency + 2NF Decomposition

Customer(C_ID, Address, Name, DOB, Subscription_Tier, Subscription Rate)

- C_ID → Address
- C_ID → DOB
- C_ID → Subscription_Tier
- C_ID → Subscription_Rate
- C_ID, Address → Name *Compound dependency
- Address → Name

C_ID, Address → Name:
C_ID^+ = {C_ID, Address, Name, DOB, Subscription_Tier, Subscription_Rate}
$Address^+$ = {Address, Name}

Both C_ID^+ and $Address^+$ give Name, so this FD is partially dependent. Removing C_ID from the LHS of C_ID, Address → Name gives Address → Name.

Remaining FDs:

- C_ID → Address, DOB, Subscription_Tier, Subscription_Rate
- Address → Name

One relation for each FD:

- R1(C_ID, Address, DOB, Subscription_Tier, Subscription_Rate)
    - **Customer(C_ID, Address, DOB, Subscription_Tier, Subscription_Rate)**
- R2(Address, Name)
    - **Directory(Address, Name)**

Normalization into BCNF

Tables already in BCNF:

Subscription_Rate(Subscription_Tier, Subscription_Rate)

- Subscription_Tier → Subscription_Rate

Subscription_Tier+ = {Subscription_Tier, Subscription_Rate}

All attributes are within the closure for Subscription_Tier, therefore {Subscription_Tier} is a candidate key.

Relation is in 3NF, all attributes on L.H.S. of FDs contain a candidate key, so the relation is in BCNF.

User_Account(User_name, Password)

- User_name → Password

User_name+ = {User_name, Password}

All attributes are within the closure for User_name, therefore {User_name} is a candidate key.

Relation is in 3NF, all attributes on L.H.S. of FDs contain a candidate key, so the relation is in BCNF.

Creates_Customer(Username, C_ID)

- Username → C_ID
- C_ID → Username

Username+ = {Username, C_ID}

C_ID+ = {C_ID, Username}

All attributes are within the closure for Username and C_ID, therefore {Username} and {C_ID} are both candidate keys.

Relation is in 3NF, all attributes on L.H.S. of FDs contain a candidate key, so the relation is in BCNF.

Movie(M_ID, Title, Subscription_Tier, Release_Date, Review_Score, Price)

- M_ID → Title
- M_ID → Subscription_Tier
- M_ID → Release_Date
- M_ID → Review_Score
- M_ID → Price

M_ID+ = {M_ID, Title, Subscription_Tier, Release_Date, Review_Score, Price}

All attributes are within the closure for M_ID, therefore {M_ID} is a candidate key.

Relation is in 3NF, all attributes on L.H.S. of FDs contain a candidate key, so the relation is in BCNF.

Genre(Genre_Name)

- None

No FDs, but {Genre_Name} is a key to Genre, so the relation is in BCNF.

Movie_Has_Genre(Genre_Name, M_ID)

- None (Many-to-many relationship)

No FDs, but {Genre_Name, M_ID} is a key to Movie_Has_Genre, so the relation is in BCNF.

Admin_Account(Admin_Name, Password)

- Admin_Name → Password

Admin_Name+ = {Admin_Name, Password}

All attributes are within the closure for Admin_Name, therefore {Admin_Name} is a candidate key.

Relation is in 3NF, all attributes on L.H.S. of FDs contain a candidate key, so the relation is in BCNF.

Purchases_Subscription(C_ID, Subscription_Tier)

- C_ID → Subscription_Tier

C_ID+ = {C_ID, Subscription_Tier}

All attributes are within the closure for C_ID, therefore {C_ID} is a candidate key.

Relation is in 3NF, all attributes on L.H.S. of FDs contain a candidate key, so the relation is in BCNF.

Purchase_Movie(C_ID, M_ID)

- None (Many-to-many relationship)

No FDs, but {C_ID, M_ID} is a key to Purchase_Movie, so the relation is in BCNF.

Review(C_ID, M_ID, Review_Description, Score)

- C_ID, M_ID → Review_Description
- C_ID, M_ID → Score

C_ID, M_ID+ = {C_ID, M_ID, Review_Description, Score}

All attributes are within the closure for (C_ID, M_ID), therefore {C_ID, M_ID} is a candidate key.

Relation is in 3NF, all attributes on L.H.S. of FDs contain a candidate key, so the relation is in BCNF.

Update_Catalogue(Admin_Name, M_ID, Date_Change, Movie_Status)

- Admin_Name, M_ID → Date_Change
- Admin_Name, M_ID → Movie_Status

Admin_Name, M_ID+ = {Admin_Name, M_ID, Date_Change, Movie_Status}

All attributes are within the closure for (Admin_Name, M_ID), therefore {Admin_Name, M_ID} is a candidate key.

Relation is in 3NF, all attributes on L.H.S. of FDs contain a candidate key, so the relation is in BCNF.

Age_Rating(Age_Rating, Age_Range)

- Age_Rating → Age_Range

Age_Rating+ = {Age_Rating, Age_Range}

All attributes are within the closure for Age_Rating, therefore {Age_Rating} is a candidate key.

Relation is in 3NF, all attributes on L.H.S. of FDs contain a candidate key, so the relation is in BCNF.

Assign_Age_Rating(M_ID, Age_Rating)

- M_ID → Age_Rating

M_ID+ = {M_ID, Age_Rating}

All attributes are within the closure for M_ID, therefore {M_ID} is a candidate key.

Relation is in 3NF, all attributes on L.H.S. of FDs contain a candidate key, so the relation is in BCNF.

Normalization using Bernstein's Algorithm on Transitive & Partial Dependencies

Customer(C_ID, Address, Name, DOB, Subscription_Tier, Subscription Rate)

- C_ID → Address
- C_ID → DOB
- C_ID → Subscription_Tier
- C_ID → Subscription_Rate
- C_ID, Address → Name *Compound dependency
- Address → Name
- Subscription_Tier → Subscription_Rate *Transitive FD

Remove Partial Dependencies
C_ID, Address → Name:
C_ID^+ = {C_ID, Address, Name, DOB, Subscription_Tier, Subscription_Rate}
$Address^+$ = {Address, Name}

Both C_ID^+ and $Address^+$ give Name, so this FD is partially dependent. Removing C_ID from the LHS of C_ID, Address → Name gives Address → Name.

Thus, remove C_ID, Address → Name

Remove Redundant Dependencies
C_ID → Subscription_Rate: C_ID^+ = {C_ID, Address, DOB, Subscription_Tier, Subscription_Rate}
*Redundant since Subscription_Rate is included

Thus, remove C_ID → Subscription_Rate

Remaining FDs:

- C_ID → Address
- C_ID → DOB
- C_ID → Subscription_Tier
- Address → Name
- Subscription_Tier → Subscription_Rate

Combined FDs:

- C_ID → Address, DOB, Subscription_Tier
- Address → Name
- Subscription_Tier → Subscription_Rate

Decompose original relation into relations for each FD:

- R1(C_ID, Address, DOB, Subscription_Tier)
    - **Customer(C_ID, Address, DOB, Subscription_Tier)**
- R2(Address, Name)
    - **Directory(Address, Name)**
- R3(Subscription_Tier, Subscription_Rate)
    - **Subscription_Rate(Subscription_Tier, Subscription_Rate)**

Customer(C_ID, Address, DOB, Subscription_Tier)

- C_ID → Address
- C_ID → DOB
- C_ID → Subscription_Tier

C_ID+ = {Address, DOB, Subscription_Tier}

All attributes are within the closure for C_ID, therefore {C_ID} is a candidate key.

Relation is in 3NF, all attributes on L.H.S. of FDs contain a candidate key, so the relation is in BCNF.

Directory(Address, Name)

- Address → Name

Address+ = {Address, Name}

All attributes are within the closure for Address, therefore {Address} is a candidate key.

Relation is in 3NF, all attributes on L.H.S. of FDs contain a candidate key, so the relation is in BCNF.

UNIX Shell Implementation

An early implementation of a working user interactive interface was produced for this project. The interface was implemented using bash script on the TMU moons server connected to the Oracle database. The snapshots of the interface are provided below.

Shell Menu Snapshots

Main Menu

```
================================================================
| Online Movie Store DBMS - Oracle All Inclusive Tool
|
| Main Menu - Select Desired Operation(s):
|
| <CTRL-Z Anytime to Enter Interactive CMD Prompt>
|
----------------------------------------------------------------
 1) Drop Tables
 2) Drop Views
 3) Create Tables
 4) Create Views
 5) Populate Tables
 6) Query Tables

 E) End/Exit
Choose:
```

Dropping Tables

```
Choose:
1

SQL*Plus: Release 12.1.0.2.0 Production on Tue Oct 25 20:23:36 2022

Copyright (c) 1982, 2014, Oracle.  All rights reserved.


Connected to:
Oracle Database 11g Enterprise Edition Release 11.2.0.1.0 - 64bit Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options

SQL>
Table dropped.

SQL>
Table dropped.

SQL>
Table dropped.

SQL>
Table dropped.

SQL>
Table dropped.
```

Dropping Views

```
Choose:
2

SQL*Plus: Release 12.1.0.2.0 Production on Tue Oct 25 20:27:43 2022

Copyright (c) 1982, 2014, Oracle.  All rights reserved.


Connected to:
Oracle Database 11g Enterprise Edition Release 11.2.0.1.0 - 64bit Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options

SQL>
View dropped.

SQL>
View dropped.

SQL>
View dropped.
```

Creating Tables

```
Choose:
3

SQL*Plus: Release 12.1.0.2.0 Production on Tue Oct 25 20:26:09 2022

Copyright (c) 1982, 2014, Oracle.  All rights reserved.


Connected to:
Oracle Database 11g Enterprise Edition Release 11.2.0.1.0 - 64bit Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options

SQL>   2    3    4
Table created.

SQL> SQL>   2    3    4    5    6
Table created.

SQL> SQL>   2    3    4
Table created.

SQL> SQL>   2    3    4    5    6    7    8    9   10   11
Table created.

SQL> SQL>   2    3
Table created.
```

Creating Views

```
Choose:
4

SQL*Plus: Release 12.1.0.2.0 Production on Tue Oct 25 20:27:08 2022

Copyright (c) 1982, 2014, Oracle.  All rights reserved.


Connected to:
Oracle Database 11g Enterprise Edition Release 11.2.0.1.0 - 64bit Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options

SQL>   2    3    4    5
View created.

SQL> SQL>   2    3    4    5    6
View created.

SQL> SQL>   2    3    4    5    6    7
View created.
```

Populating Tables

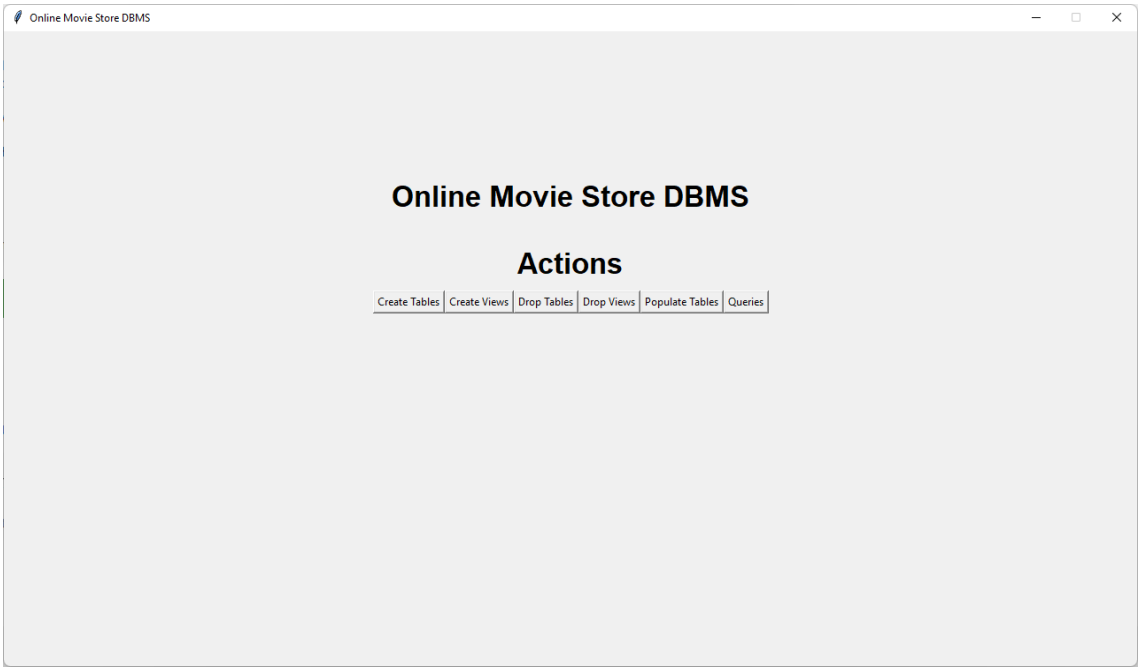
```
Choose:
5

SQL*Plus: Release 12.1.0.2.0 Production on Tue Oct 25 20:28:26 2022

Copyright (c) 1982, 2014, Oracle.  All rights reserved.


Connected to:
Oracle Database 11g Enterprise Edition Release 11.2.0.1.0 - 64bit Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options

SQL>
1 row created.

SQL>
1 row created.

SQL>
1 row created.

SQL>
1 row created.

SQL>
1 row created.

SQL> SQL>
Session altered.

SQL>
1 row created.
```

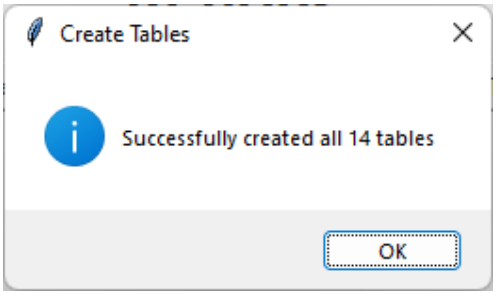
Querying Tables

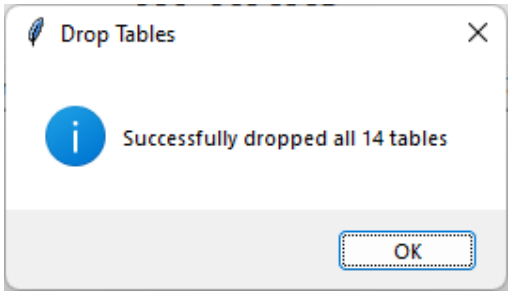
```
Choose:
6

SQL*Plus: Release 12.1.0.2.0 Production on Wed Oct 26 21:10:48 2022

Copyright (c) 1982, 2014, Oracle.  All rights reserved.


Connected to:
Oracle Database 11g Enterprise Edition Release 11.2.0.1.0 - 64bit Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options

SQL> SQL> SQL> SQL> SQL> SQL> SQL> SQL>   2    3    4    5
      C_ID NAME                                               ADDRESS                                            DOB
---------- ---------------------------------------------------- -------------------------------------------------- -----------
         2 Jane Naught                                         2 Unreal Ave. Montreal QC                          01-DEC-02
         3 James Stew                                          3 Fabrication Cres. Vancouver BC                   05-FEB-09
         5 Alex Chan                                           199 Wasabi Ave. Vancouver BC                       20-FEB-99

SQL> SQL> SQL>   2    3    4    5    6    7    8
      M_ID REVIEW_DESCRIPTION                                      SCORE
---------- ---------------------------------------------------- ----------
         3 Amazing, definitely worth the Gold tier!                    5
         4 I got the Silver tier just for this, super good             5
         6 I mean, its the Avengers                                    5
         1 Trash                                                       1

SQL> SQL>   2    3    4
      M_ID      SALES
---------- ----------
         1          3
         4          3
         5          2
         3          2
         6          2
         2          2

6 rows selected.

SQL> SQL>   2    3    4    5    6    7    8    9
      M_ID      SALES
---------- ----------
         4          3
         1          3

SQL> SQL> SQL>   2    3    4    5
USERNAME
--------------------
username1
username2
username3
```

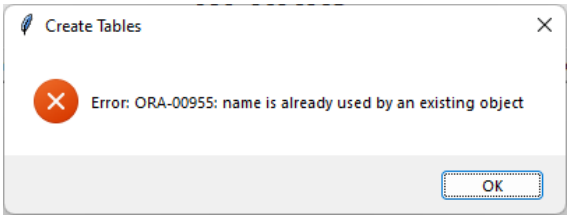
Python GUI

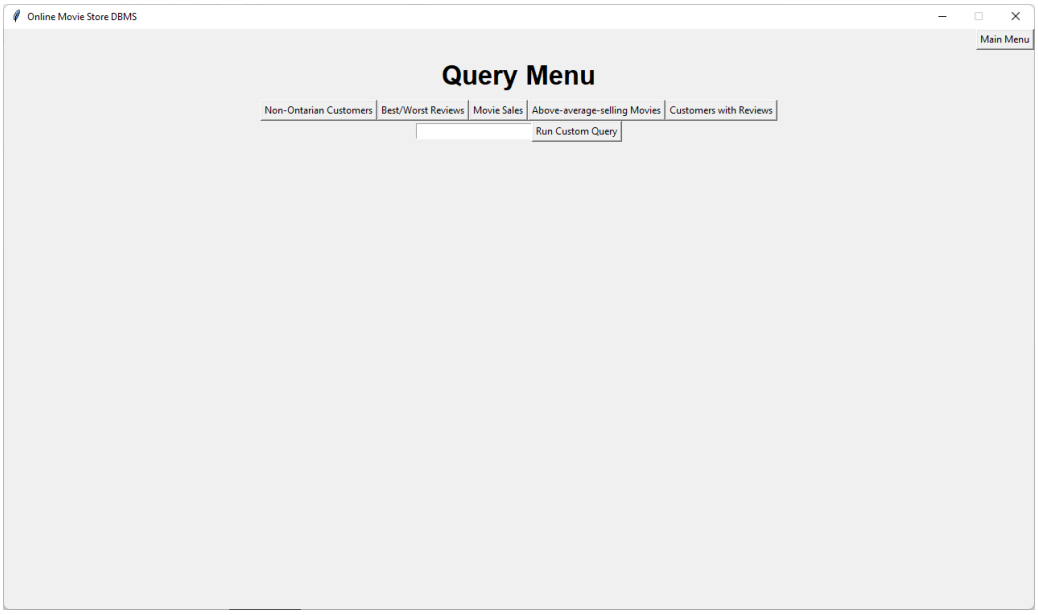
Snapshots

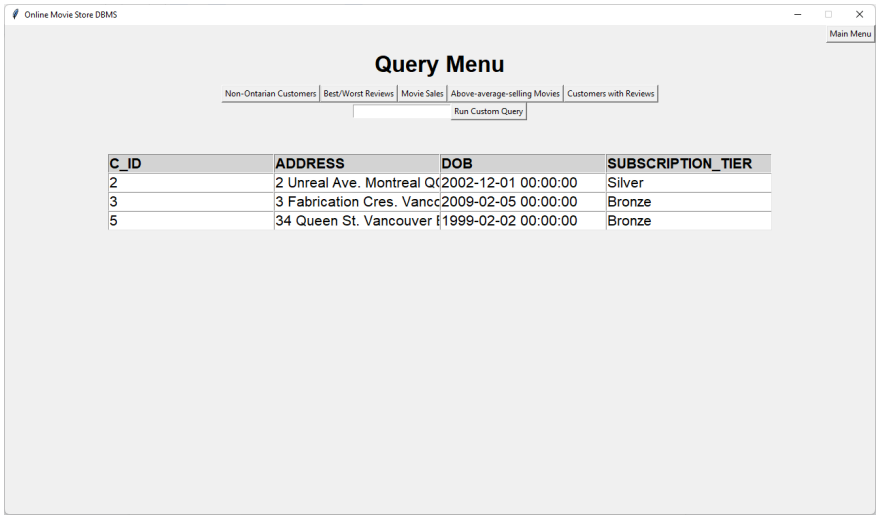


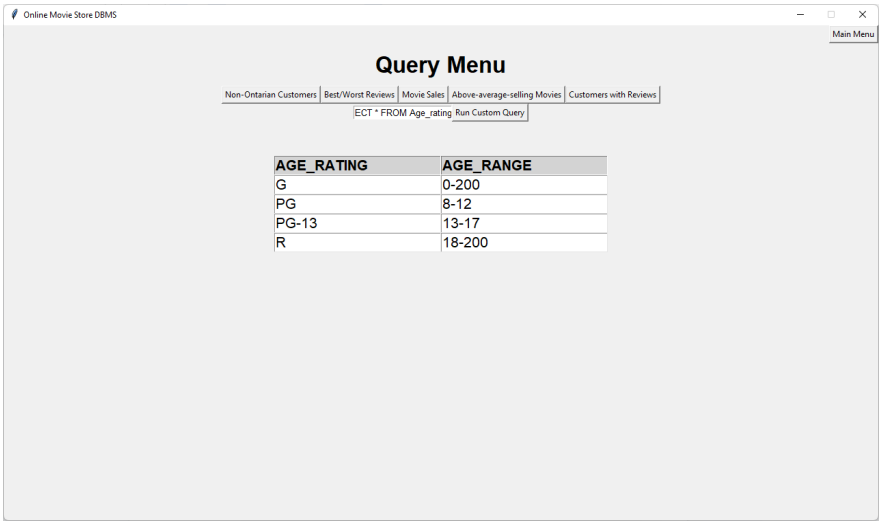







Online Movie Store DBMS
Main Menu
Query Menu
Non-Ontarian Customers
Best/Worst Reviews
Movie Sales
Above-average-selling Movies
Customers with Reviews
Run Custom Query


Online Movie Store DBMS
Main Menu
Query Menu
Non-Ontarian Customers
Best/Worst Reviews
Movie Sales
Above-average-selling Movies
Customers with Reviews
Run Custom Query
C_ID
ADDRESS
DOB
SUBSCRIPTION_TIER
2
2 Unreal Ave. Montreal Q
2002-12-01 00:00:00
Silver
3
3 Fabrication Cres. Vanc
2009-02-05 00:00:00
Bronze
5
34 Queen St. Vancouver
1999-02-02 00:00:00
Bronze


Online Movie Store DBMS
Main Menu
Query Menu
Non-Ontarian Customers
Best/Worst Reviews
Movie Sales
Above-average-selling Movies
Customers with Reviews
ECT * FROM Age_rating
Run Custom Query
AGE_RATING
AGE_RANGE
G
0-200
PG
8-12
PG-13
13-17
R
18-200

Description

The GUI is a local application coded in Python. It contains a main menu where the user can press buttons to instantaneously perform operations on the database. These operations include Drop Tables, Create Tables, Drop Views, Create Views, Populate Tables, and Query Tables. The user can also exit the DBMS application by closing the window. Additionally, there is a separate menu for query operations, where the user can choose to query for Non-Ontarian customers, the best/worst reviews, movie sales, above-average-selling movies, and customers who have left reviews. All results are returned in a graphical table. Finally, there is also a custom query field where the user can enter any Oracle SQL command to be performed on the database, including selecting, updating, and deleting records.

Installation (Windows only)

1. Download and install Python from https://www.python.org/
2. Run in a terminal ‘pip install tk’ and ‘pip install cx_oracle’
3. Download Oracle Instant Client v21.7 from https://www.oracle.com/database/technologies/instant-client/downloads.html
4. Extract a folder named ‘instantclient_21_7’ to the path ‘C:\Oracle’ from the downloaded Oracle Instant Client zip file
5. Download the Online Movie Store DBMS application from D2L and run main.py

Operation

The user can click on a given button to perform that command. If the command succeeds, a system dialog showing an informational message is displayed, letting the user know the command succeeded. If the command fails, a system dialog showing an error message is displayed instead.

For a custom query, a user can type an Oracle SQL command into the text entry box in the query menu. Once finished, they can click the ‘Run Custom Query’ button to execute the given instruction. If the query is a SELECT statement, the results are shown in a graphical table below the buttons, just as with the preset query buttons. If the query is an UPDATE or DELETE statement, dialog boxes are shown indicating the success or failure of the command. If the entered text is not a valid command, an error is displayed.

Conclusion

Working on this semester-long project based on an Online Movie Store provided us with a solid understanding of creating and managing a DBMS. We utilized and put into practice key concepts related to good database design and implementation. One of the key takeaways from this project was the importance of thinking about data in a clear and concise manner. Working through the different steps of the project helped us grasp the full development cycle of a normalized database, from creating entity-relationship diagrams to the normalization of relations. As a group, we were able to make data more accessible and operate in an efficient manner. Furthermore, working with SQL commands and the Oracle server familiarized us with common syntax and tools used in the workplace. In conclusion, working on this Online Movie Store was very educational and it was a perfect example of putting theory learned from lectures into practice.