INFS1200 Module 3 Assignment

Due: Monday 17 October 2022 @ 1PM AEST

Weighting: 30% Version: 1.0

	Full Name	Student ID
Group Member 1 (This person should submit on Gradescope)	**Cao Quoc Thang Hoang**	**47594876**
Group Member 2	**Hoang Viet Nguyen**	**46769659**

1. Overview

The purpose of this assignment is to test your ability to use and apply SQL concepts to complete tasks in a real-world scenario. Specifically, this assessment will examine your ability to use SQL Data Manipulation Language to return specific subsets of information which exist in a database and Data Definition Language to create new relational schema.

This assignment can be completed in **group of two** or **individually**. If done in a group, your partner must be enrolled in the same course.

2. Submission

All submissions must be made through an electronic marking tool called Gradescope, which will also be used for providing feedback and automated marking. You will need to submit two types of files to the autograder:

- **Query Files**: For each question in sections A, B and C, you are required to submit a *.sql* or *.txt* file which contains your SQL query solution for that question (only one of these files, if you submit both, the *.sql* file will be graded). The file should only contain the SQL query(s), no additional text. The file should be named as per the description in the question. Additionally, the number of queries allowed to be run per question is also specified in each question's description.
- **Assignment PDF**: After you have completed all the questions for this assignment, you should insert your answers into the template boxes where appropriate. Then you should export this document to a pdf and upload it to the respective Gradescope portal. Please note that this portal is simply a backup for Sections A, B and C. **Only** section D will be hand marked from your pdf submission.

When submitting to the autograder, please select all your *.txt* or *.sql* individually **instead** of uploading a zip file. Additionally, for student working in a group, only one group member should submit via Gradescope. The student submitting on behalf of their group must add their group member to their submission via Gradescope.

3. Marking

The module 3 assignment is worth **30 course marks** (of 100 course marks total for all assessment). The marking distribution per section is as follows:

- Section A – SQL DDL: 4 marks
- Section B – SQL DML (UPDATE, INSERT, DELETE): 5 marks
- Section C – SQL DML (SELECT): 16 marks
- Section D – Critical Thinking: 2.5 marks
- Section E – RiPPLE Task: 2.5 marks

Grading and autograder feedback

Sections A, B and C of this assignment will be graded via an autograder deployed on Gradescope; however we reserve the right to revert to hand marking using the pdf submission should the need arise. Specifically, your assignment will be graded against several data instances, which may include a simple (and small) data instance, a large data instance or an instance containing curated edge cases. The correctness of your queries will be judged by comparing your queries' return values to those of our solutions, because there is usually more than one equivalent way to execute a given query.

Note that solutions to each question will be limited to contain a maximum of 3 queries.

When you submit your code, the autograder will provide you with two forms of immediate feedback:

- **File existence and compilation tests**: Your code will be checked to see if it compiles correctly. If it fails one or more compilation test, the errors returned by the autograder will help you debug. Note that code that fails to compile will receive 0 marks. No marks are given for passing the compilation tests.
- **Simple instance data tests:** The autograder will return your degree of success on the simple data instance, so that you can judge your progress (i.e. 90% of simple instance tests passed). Individual test results will not be revealed, and your submission's performance on the more difficult instances will remain hidden until grades are released. Final weightings on the different test instances will also remain hidden until grades are released.

More details will be provided regarding how you can interpret the results of these tests and what it means for your assignment grade during practicals.

The final details of the Gradescope autograder will be released closer to the assignment deadline. Note that you will be able to resubmit to the autograder an unlimited number of times before the deadline.

Materials provided:

You will be provided with the DB schema and the simple data instance. Because the autograder uses the same DBMS as your zones, you are encouraged to use your zones to develop your assignment answers.

4. Plagiarism

The University has strict policies regarding plagiarism. Penalties for engaging in unacceptable behaviour can range from loss of grades in a course through to expulsion from UQ. You are required to read and understand the policies on academic integrity and plagiarism in the course profile (ECP Section 6.1).

If you have any questions regarding acceptable level of collaboration with your peers, please see either the lecturer or your tutor for guidance. Remember that ignorance is not a defence!

The task is described on the next page

5. Task

For this assignment you will be presented with the simplified schema of an online movie streaming service. You will be required to write a combination of SQL DML and DDL queries which answer higher level questions about the data constrained in the database or perform operations against the database's schema and instance data.

Note: Your queries must compile using MySQL version 8.0. This is the same DBMS software as is used on your zones. You may use any MySQL function that have been used in class in addition to those specified in the questions. You may also use other MySQL functions not covered in this course to assist with manipulating the data if needed, however please ensure you read the MySQL documentation page first to ensure the functions works as intended.

Assignment Specification

A relational database has been setup to track customer browsing activity for an online movie streaming service called *SurfTheStream*. Movies are identified by a unique code that consists of a four-character prefix and four-digit suffix. Additionally, each movie is assigned a content rating which must be one of the following options: "G", "PG", "M", "MA15+" or "R18+". The first time a customer previews a movie is captured by the database. Customers may preview a movie before they stream it, however, they cannot preview a movie after they have started to stream it. You may assume "Duration" refers to the time in seconds a customer has spent streaming a particular movie after the "Timestamp".

A simplified version of their database schema has been provided below including foreign key constraints. You should consult the provided blank database import file for further constraints which may be acting within the system.

Relational Schema

Customer [id, name, dob, bestFriend, subscriptionLevel]
Customer.bestFriend references Customer.id
Customer.subscriptionLevel references Subscription.level

Movie [prefix, suffix, name, description, rating, releaseDate]

Previews [customer, moviePrefix, movieSuffix, timestamp]
Previews.customer references Customer.id
Previews.{moviePrefix, movieSuffix} reference Movie.{prefix, suffix}

Streams [customer, moviePrefix, movieSuffix, timestamp, duration]
Streams.customer reference Customer.id
Streams.{moviePrefix, movieSuffix} reference Movie.{prefix, suffix}

Subscription [level, price]

A file containing this schema and a small data instance are included on Blackboard. For this assignment you will be required to write SQL queries to complete the following questions. Please use the submission boxes provided to record your answers, but **do not forget to submit to the autograder as well!**

Section A – SQL DDL

Example	
Task	The company has decided to stop recording and tracking how many customers preview movies. Write an SQL query to reflect this change in the database schema.
Explanation	This change implies that the Previews table will no longer be needed.
SQL Solution	**DROP TABLE** Previews;

Question 1	
Task	Write a SQL DDL query to implement the following relational schema and associated foreign keys.
Explanation	The relational schema for this the new table is as follows: **Table: MovieEmployee** **Column** / **Data Type** / **Allow Nulls?** / **Primary Key?** moviePrefix / VARCHAR(4) / No / Yes movieSuffix / VARCHAR(4) / No / Yes employeeName / VARCHAR(100) / No / Yes role / {“Actor”, “Production”, “Other”} / No / Yes startDate / DATE / Yes / No Additionally, no employee should be able to start two roles in the same or different movies on the same day. The foreign keys for this new table are as follows: MovieEmployee.{moviePrefix, movieSuffix} references Movie.{prefix, suffix} Foreign key constraints should be implemented such that: • Updates to a *Movie*’s prefix and/or suffix are automatically updated in *MovieEmployee* as well. • A *Movie* cannot be deleted if there is an employee recorded as having worked on that movie. **Note**: You may wish to consult the MySQL documentation on the enum datatype.
File Name	*a1.txt* or a*1.sql*
SQL Solution	CREATE TABLE MovieEmployee (moviePrefix VARCHAR(4) NOT NULL, movieSuffix VARCHAR(4) NOT NULL, employeeName VARCHAR(100) NOT NULL, role ENUM('Actor','Production','Other') NOT NULL, startDate DATE NULL, PRIMARY KEY(moviePrefix, movieSuffix, employeeName, role), FOREIGN KEY (moviePrefix, movieSuffix) REFERENCES Movie(prefix, suffix) ON UPDATE CASCADE ON DELETE RESTRICT);

Question 2	
Task	The Company has decided to keep track of the country of origin for each movie. Write an SQL query(s) to capture this change.
Explanation	This query should add an attribute “CountryOfOrigin”, which captures the country of origin via a country code of a maximum of three letters (e.g., AUS for Australia). The attribute should be null for existing tuples.
File Name	*a2.txt* or *a2.sql*
SQL Solution	ALTER TABLE Movie ADD CountryOfOrigin VARCHAR(3) NULL;

Section B – SQL DML (UPDATE, DELETE, INSERT)

Note: Modification made to the database schema made in previous questions are **not** persisted.

Example	
Task	Set the content rating of all movies called “Bad Day” to be “PG”.
Explanation	The rating for each movie is captured using the *rating* attribute in the *Movie* table.
SQL Solution	**UPDATE** Movie **SET** rating = “PG” **WHERE** name = “Bad Day”

Question 1	
Task	In an effort to purge the system of fake customer accounts the Chief Information Officer of *SurfTheStream* has authorised all customer accounts to be removed that meet either (or both) of the following conditions: • The account id has less than 3 characters. (e.g., “AA”) • The customer is older than 110 or younger than 10 years old.
Explanation	You may assume that all the fake customer accounts which meet one (or both) of the above criteria are not referenced by foreign keys in any other tables. Additionally, you may find the MySQL function **CHAR_LENGTH** helpful for this question. The MySQL documentation page for this function can be viewed here.
File Name	*b1.txt* or *b1.sql*
SQL Solution	DELETE FROM Customer C WHERE CHAR_LENGTH(C.id) < 3 OR YEAR(DATEDIFF(CURRENT_DATE, C.dob)) > 110 OR YEAR(DATEDIFF(CURRENT_DATE, C.dob)) < 10; -- Second method: DELETE FROM Customer C WHERE CHAR_LENGTH(C.id) < 3 OR FLOOR(DATEDIFF(CURRENT_DATE, C.dob)/365) > 110 OR FLOOR(DATEDIFF(CURRENT_DATE, C.dob)/365) < 10;

Question 2	
Task	Make the necessary modification to the database so that previews of “Harry Potter” movies are not shown to current customers.
Explanation	The system will not show a movie preview if the customer has already previewed the movie. Therefore, for all movies with “Harry Potter” in the title create a fake preview record with the *timestamp* being the current time the query is run.
File Name	*b2.txt* or *b2.sql*
SQL Solution	INSERT INTO Previews (customer, moviePrefix, movieSuffix, timestamp) SELECT C.id, M.prefix, M.suffix, NOW() FROM Customer C, Movie M WHERE M.name LIKE "%Harry Potter%" AND C.id NOT IN (SELECT P.customer FROM Previews P JOIN Movie M2 ON (M2.prefix = P.moviePrefix AND M2.suffix = P.movieSuffix) WHERE M2.name LIKE "%Harry Potter%") -- Second method INSERT INTO Previews (customer, moviePrefix, movieSuffix, timestamp) SELECT C.id, M.prefix, M.suffix, NOW() FROM Movie M, Customer C WHERE M.name LIKE '%Harry Potter%' AND NOT EXISTS (SELECT * FROM Previews P WHERE P.customer = C.id AND P.moviePrefix = M.prefix AND P.movieSuffix = M.suffix)

Section C – SQL DML (SELECT)

Example	
Task	Return the prefix and suffix of all movies with a rating of “M”.
Explanation	This query should return a table with two columns. The first column should correspond to the movie’s prefix and the second column to the movie’s suffix.
SQL Solution	SELECT prefix, suffix FROM Movie WHERE rating = “M”

Question 1	
Task	Return the id and name of all costumers that hold a ‘basic” subscription level in ascending order by age.
Explanation	This query should return a table with two columns, the first containing the id of a customer and the second their name.
File Name	*c1.txt* or *c1.sql*
SQL Solution	SELECT C.id, C.name FROM Customer C WHERE C.subscriptionLevel LIKE "Basic" ORDER BY YEAR(C.dob) DESC, MONTH(C.dob) DESC, DAY(C.dob) DESC -- Second method SELECT C.id, C.name FROM Customer C WHERE C.subscriptionLevel = "Basic" ORDER BY TIMESTAMPDIFF(YEAR , C.dob, NOW()) ASC;

Question 2	
Task	Return the movies that have been streamed at least once during the past seven days.
Explanation	This query should return a table with three columns of prefix, suffix, name of movies that have been streamed at least once. The 7-day time period is inclusive and should be correct to the second the query is run.
File Name	*c2.txt* or *c2.sql*
SQL Solution	SELECT S.moviePrefix, S.movieSuffix, M.name FROM Streams S, Movie M WHERE DATEDIFF(CURRENT_DATE, DATE(S.timestamp)) < 7 AND (M.prefix = S.moviePrefix AND M.suffix = S.movieSuffix)

Question 3	
Task	Return the number of movies which were released per year.
Explanation	This query should return a table with two columns, the first containing a year, the second containing the number of movies released in that year. You do not need to represent years which had zero movies released.
File Name	*c3.txt* or *c3.sql*
SQL Solution	SELECT YEAR(M.releaseDate), COUNT(*) FROM Movie M GROUP BY YEAR(M.releaseDate);

Question 4	
Task	For each customer who has a best friend, return their id, name and age difference in comparison to their best friend.
Explanation	This query should return a table with fours columns, the first for the id of the customer, the second for the name of the customer, the third for the name of their best friend and the four for the age difference (in days). The age difference should be calculated using: *Customer's DOB – Customer's best friend's DOB* Additionally, you may find the MySQL function **DATEDIFF** helpful for this question. The MySQL documentation page for this function can be viewed here.
File Name	*c4.txt* or *c4.sql*
SQL Solution	SELECT C.id, C.name, F.name, DATEDIFF(C.dob,F.dob) FROM Customer C JOIN Customer F ON (C.bestFriend = F.id);

Question 5	
Task	Return the list of customer ids who have watched the same number of previews as their best friends.
Explanation	This query should return 3 columns. The first two columns display the ids of the two customers. The third column displays the number of previews they have watched. You may ignore cases where the number of previews was 0. if two people are each other's best friends we return twice (with the id's switched) and don't remove them.
File Name	*c5.txt* or *c5.sql*
SQL Solution	SELECT C.id, C.bestFriend, COUNT(P.customer) FROM Customer C JOIN Previews P ON (C.id = P.customer) GROUP BY C.id HAVING COUNT(P.customer) = (SELECT COUNT(P2.customer) FROM Previews P2 WHERE(P2.customer = C.bestFriend))

Question 6	
Task	Return a list of customers who have streamed exactly 5 distinct movies, of which at least one was streamed for over an hour.
Explanation	This query should return a table with two columns, the first being customer ids and the second being customer names. **Hint: Non-Correlated Subquery**
File Name	*c6.txt* or *c6.sql*
SQL Solution	SELECT C.id, C.name FROM Customer C JOIN Streams S ON (S.customer = C.id) WHERE S.duration >= 3600 AND C.id IN (SELECT S2.customer FROM Streams S2 GROUP BY S2.customer HAVING COUNT(S2.customer) = 5) GROUP BY S.customer

Question 7	
Task	Return the id and name of all customers who have previewed at least all the “Harry Potter” movies.
Explanation	This query should return a table with two columns, the first being customer ids and the second being customer names. Any movie with “Harry Potter” in the title is considered a Harry Potter movie. **Hint: Correlated Subquery**
File Name	*c7.txt* or *c7.sql*
SQL Solution	SELECT C.id, C.name FROM Customer C JOIN Previews P ON P.customer = C.id JOIN Movie M ON M.prefix = P.moviePrefix AND M.suffix = P.movieSuffix WHERE M.name LIKE "%Harry Potter%" GROUP BY P.customer HAVING COUNT(*) = (SELECT COUNT(*) FROM Movie M2 WHERE M2.name LIKE "%Harry Potter%");

Question 8	
Task	For each customer, return the content rating that is most representative of the movies they have seen.
Explanation	This query should return a table with two columns. The first containing the id of the customer and the second containing the content rating that is most representative of the movies they have seen. You can ignore customers who have not watched any movies. In cases that there is a tie between ratings, any would be ok to be returned. **Hint**: **VIEW**.
File Name	*c8.txt* or *c8.sql*
SQL Solution	CREATE VIEW `CustomerRating` AS SELECT S.customer, M.rating, COUNT(M.rating) AS content FROM Streams S JOIN Movie M ON (M.prefix = S.moviePrefix AND M.suffix = S.movieSuffix) GROUP BY M.rating,S.customer ORDER BY S.customer; CREATE VIEW `MaxRating` AS SELECT CR.customer, MAX(CR.content) AS RR FROM CustomerRating CR GROUP BY CR.customer HAVING MAX(CR.content); SELECT CR.customer, CR.rating FROM CustomerRating CR JOIN MaxRating MR ON (CR.customer = MR.customer) WHERE CR.content = MR.RR;

Section D – Critical Thinking

1. In this section you will be presented with an abstract scenario(s) relating to the UoD provided in the task description.

Example	
Task	*SurfTheStream* is looking for customers who can work for the company as movie critics and write blogs on new releases. Propose two different strategies to complete the given task. Pick one of those two strategies and write an SQL query(s) that implements that strategy.
Strategies	1. Customers who watch a lot of movies are likely to meet the criterion defined in the question. We can go for customers who watched say at least 50 movies. 2. Customers who watch movies when they come out are likely to meet the criterion defined in the question. We can go for customers who watched several (say 10) movies within a week of the movie's release.
SQL Solution	**SELECT** C.* **FROM** Customer C **JOIN** Previews P **ON** P.customer = C.id **JOIN** Movie M **ON** M.prefix = P.moviePrefix **AND** M.suffix = P.movieSuffix **WHERE DATEDIFF**(P.timestamp, M.releaseDate) < 8 **GROUP BY** C.id **HAVING COUNT**(*) > 9

Question 1	
Task	*SurfTheStream* wants to provide a free subscription level upgrade to loyal customers. Propose two different strategies to complete the given task. Pick one of those two strategies and write an SQL query(s) that implements that strategy.
Strategies	1. In order to be a loyal customer, a customer who watch a lot of movies in a fix and long duration (a month, a year...) and have high subscription level. We can go for customer who watch at least 10 movies in a month (approximately 2 times per week) and have. 2. Customer who has large total stream time is large. For instance, total stream time is 50 hours (equivalent to 180000 seconds) or more to have a free subscription to loyal customer.
SQL Solution	SELECT C.id FROM Customer C JOIN Streams S ON (C.id = S.customer) GROUP BY C.id HAVING SUM(S.duration) > 180000;

Section E – RiPPLE Task

Using the RiPPLE online software, you must complete the following activities before the assignment due date:

- **Resource Creation**: Create one or more effective resource. For a learning resource to be considered as effective it needs to pass a moderation process which is administered by your peers and the teaching team. Teaching staff will be spot-checking to review moderations performed by just peers and change the outcome if necessary.

- **Resource Moderation**: Moderate 4 or more resources effectively. An effective moderation means that you have completed the moderation rubric and have provided a detailed justification for your judgement as well as constructive feedback on how the resource can be improved. Simply saying a resource is “good” does not qualify. Again, teaching staff will be spot-checking the quality of moderations and change the outcome when necessary.

- **Answering Questions**: Answer 10 or more questions correctly. To answer a resource correctly your first response must be correct. You can attempt as many questions as you want, and incorrect answers do not count against you. Only answers from the Practice tab are counted. Answering in-class RiPPLE activity questions does not count towards questions answers.

These tasks are to be completed **individually** through the RiPPLE platform, via the link available on Blackboard.

Note: For the above three activities, the resources you create, moderate and answer **must** be in the following categories on RiPPLE:

- SQL

- Functional-dependency

- Normalization

Creating, moderating or answering questions from other categories will not be counted towards your mark for the RiPPLE component of this assignment.