Edith Cowan University
School of Science

CSI6207 – Systems Analysis and Database Design

Assessment 3: Database Design and Implementation

Due Date:
9am 27th May 2019

Assignment Marks: 35% of unit

General Assignment Information:

The assignment consists of the implementation of a collection of SQL scripts which create and populate a database and query the data it contains.

Table of Contents

1. Scenario Details

You are required to design and create a database for a movie theatre. You have the following information about the way the movie theatre operates:

- Details of the **movies** that the theatre has must be recorded. This includes a movie ID number, the name of the movie, its duration in minutes and a description of the movie.
- the **classification** of each movie must also be recorded. To do this, the database should also contain a table containing the details of classifications as shown below:

Classification	Name	Minimum Age
G	General	Not Applicable
PG	Parental Guidance	Not Applicable
M	Mature	Not Applicable
MA	Mature Audiences	15
R	Restricted	18

(The classification column can be used as the primary key of the table)

- The movie table will need a foreign key to identify the classification of each movie.
- The **genres** of each movie must also be recorded. The only details needed to be stored are an ID number and name. Each movie can have many genres (but must have at least one) and there can be many movies of the same genre.
- The database must record details of the **Cinemas** within the movie theatre – i.e., the rooms in which movies are viewed. The details that need to be stored about the cinemas are an ID number, the name of the cinema (e.g. “Cinema 3”) and the seating capacity.
- To allow for better organisation and advertising of sessions, the theatre wants to keep track of their **cinema types**. Add an entity to store the list of cinema types shown below and make sure that the cinema entity includes a foreign key to identify the type of each cinema.

Cinema Type ID	Name
1	2D
2	3D
3	Gold Class 2D
4	Gold Class 3D

- The database must also record details of movie **sessions** – i.e., the screening of a movie in a cima which customers can attend. The details of a session must include a session ID number, the date/time of the session (as one attribute), the cost of a ticket for the session, and foreign keys identifying the movie being played and cinema that it is in.
    - Each movie that the theatre owns must have at least one session associated with it.

- The movie theatre only allows tickets to be purchased by **customers** who have registered online. The database must record the email address, password, first name, last name and date of birth of customers. The email address must be unique and can be the primary key.
- The database must record the details of tickets that customers have purchased. A customer can buy many tickets, and there can be many tickets purchased for as session. No additional details need to be recorded about tickets – seating is not reserved.
- Customers can write reviews of movies, and these reviews must be stored in the database. The text of the review, a rating between 0 and 5 and the date/time must be recorded for each review. Use the current date/time as the default value for the date/time column.
    - A customer can review mane different movies and a movie can be reviewed by many different customers.
    - A single customer should only be able to review a single movie once.

2. Implementation requirements

The database system described above should be designed and implemented in Microsoft SQL Server 2014 or above. Create your scripts as three ".sql" files, following the templates provided on Blackboard.

Your SQL code should be formatted for readability using comments for headings and to provide detail or information about your code if needed.

Your scripts should accomplish the following tasks:

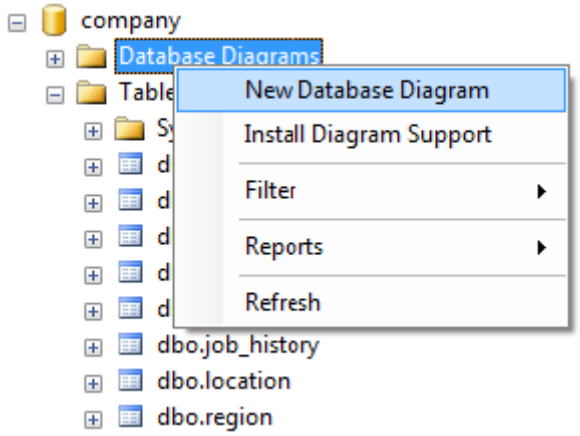
1. Database creation and population

Produce a script to create the database specified above. Be sure to give your columns appropriate data types and use consistent naming conventions. Include any suitable default values and any CHECK or UNIQUE constraints that you feel are appropriate.

Make sure this script can be run multiple times without resulting in any errors (hint: drop the database if it exists before trying to create it). You can use/adapt the code at the start of the creation scripts of the sample databases available in the unit materials to implement this. You will need to follow an appropriate creation order when creating your tables – you cannot create a table with a foreign key constraint that refers to a table which does not yet exist.

HINT: Draw an ER diagram first to design the layout and structure of the database you are attempting to create. Once you have created your database, it is recommended that you use SSMS to create an ER diagram and use this to verify that your implementation matches your design. This can be done by

right clicking on the “Database Diagrams” folder of the database in the Object Explorer in SSMS.



Following the SQL statements to create your database and its tables, you must include statements to populate the database with sufficient test data.

You are only required to populate the database with enough data to make sure that all views and queries return meaningful results.

Make sure referential integrity is observed – you cannot add data to a column with a foreign key constraint if you do not yet have data in the column it references.

Remember that when using an auto-incrementing integer, you cannot specify a value for that column when inserting a row of data. Simply pretend the column does not exist when inserting data – do not try to specify a value for it.

The data you add is simply for testing purposes, and therefore does not need to be particularly realistic, cohesive or consistent. Avoid spending unnecessary amounts of time writing sample data.

To assist with this, I have included data for a number of tables in the create_TEMPLATE.sql file.

2. *Views*

Views allow you to refer to the result of a SELECT statement as if it was a table, making query writing easier.

Create a **Cinema view** that selects the cinema ID number, cinema name, seating capacity and the name of the cinema type for all cinemas. This will involve joining the cinema and cinema type tables.

	cinema_id	cinema_name	seat_cap	c_type_name
1	1	Cinema 1	100	2D
2	2	Cinema 2	250	3D
3	3	Cinema 3	70	Gold Class 2D
4	4	Cinema 4	200	Gold Class 3D
5	5	Cinema 5	125	2D

(example output of the Cinema View – your data may vary, example illustrates structure only)

Create a **Session view** that selects the following details of all movie sessions:

- The session ID number, session date/time and cost of the session.
- The movie ID number, movie name and classification of the movie (e.g. "PG") being shown.
- The cinema ID number, cinema name, seating capacity and cinema type name of the cinema that the session is in.

This statement requires multiple JOINs. Using the Cinema View in this view is recommended.

	session_id	session_time	cost	movie_id	movie_name	class	cinema_id	cinema_name	seat_cap	c_type_name
1	1	2018-06-02 09:00:00	15.00	10	Metropolis	PG	1	Cinema 1	100	2D
2	2	2018-06-02 12:00:00	15.00	3	Forrest Gump	M	1	Cinema 1	100	2D
3	3	2018-06-02 15:00:00	15.00	2	Pulp Fiction	R	1	Cinema 1	100	2D
4	4	2018-06-02 18:00:00	15.00	6	Eternal Sunshine of the Spotless Mind	M	1	Cinema 1	100	2D
5	5	2018-06-02 09:00:00	15.00	9	Gran Torino	M	3	Cinema 3	70	Gold Class 2D
6	6	2018-06-02 12:00:00	15.00	7	Monty Python and the Holy Grail	PG	3	Cinema 3	70	Gold Class 2D
7	7	2018-06-02 15:00:00	15.00	4	Star Wars: Episode IV - A New Hope	PG	3	Cinema 3	70	Gold Class 2D
8	8	2018-06-02 18:00:00	15.00	1	The Shawshank Redemption	MA	3	Cinema 3	70	Gold Class 2D
9	9	2018-10-01 10:00:00	18.50	1	The Shawshank Redemption	MA	1	Cinema 1	100	2D
10	10	2018-10-01 10:00:00	24.50	8	Up	PG	2	Cinema 2	250	3D

(example output of the Session View – your data may vary, example illustrates structure only)

You are encouraged to use the views to simplify the queries which follow - You can use a view in a SELECT statement in exactly the same way as you can use a table, often avoiding the need to write the same joins and calculations over and over.

3. Queries

Write SELECT statements to complete the following queries. If you do not understand or are not sure about exactly what a query requires, contact your lecturer or tutor.

Create a **Child Friendly Movies** query that selects the movie name, duration and classification of all movies that have a duration of less than 100 minutes and a classification of "G" or "PG". Order the results by duration.

	movie_name	duration	class
1	Monty Python and the Holy Grail	91	PG
2	Up	96	PG
3	WALL-E	98	G

(expected query results, if provided data used)

Create a **Movie Search** query that selects the movie name, session date/time, cinema type name and cost of all upcoming sessions (i.e. session date/time is later than the current date/time) showing movies that have "star wars" anywhere in the movie name. Order the results by session date/time. Using the Session View in this query is recommended.

	movie_name	session_time	cost	c_type_name
1	Star Wars: Episode IV - A New Hope	2018-10-01 10:00:00	36.50	Gold Class 3D
2	Star Wars: Episode IV - A New Hope	2018-10-01 14:00:00	24.50	3D
3	Star Wars: Episode IV - A New Hope	2018-10-01 18:00:00	24.50	3D

(example of query results – your data may vary, example illustrates structure only)

Create a **Review Details** query that selects the details of all reviews for the movie with a movie ID number of 5. The results should include the text of the review, the date/time the review was posted, the rating given, and the first name and age (calculated from the date of birth) of the customer who posted the review. Order the results by the review date, in descending order.

	review_text	review_date	star_rating	first_name	age
1	Drags on a bit	2018-09-24 10:47:00	3.0	Mary	19
2	I laughed	2018-07-20 10:05:00	3.0	Sam	25
3	so funny	2018-03-03 09:14:00	4.0	Fred	24
4	My fav Python movie!!	2017-05-25 16:21:00	5.0	Joe	43

(example of query results – your data may vary, example illustrates structure only)

Create a **Top Selling Movies** query that selects the name and total number of tickets sold of the three most popular movies (determined by total ticket sales). Using the Session View in this query is recommended.

	movie_name	tickets_sold
1	Forrest Gump	7
2	Metropolis	6
3	Eternal Sunshine of the Spotless Mind	4

(example of query results – your data may vary, example illustrates structure only)

Create a **Customer Ticket Stats** query that selects the full names (by concatenating their first name and last name) of all customers in the customer table, how many tickets they have each purchased, and the total cost of these tickets. Be sure to include all customers, even if they have never purchased a ticket. Order the results by total ticket cost, in descending order.
Hint: This will involve using OUTER JOINs, GROUP BY, COUNT and SUM.

	full_name	tickets_purchased	total_spent
1	John Smith	9	179.50
2	Mary Sue	6	121.00
3	Janine Burgess	7	118.50
4	Fred Pickles	5	118.00
5	Sam Wood	5	111.50
6	Joe Bloggs	4	83.00
7	Garry Irvine	0	NULL

(example of query results – your data may vary, example illustrates structure only)

Create an **Age Appropriate Movies** query that selects the name, duration and description of all movies that a certain customer (chosen by you) can legally watch, based on the customer's date of birth and the minimum age required by the movie's classification. Select a customer whose date of birth makes them 15-17 years old for this query, so that the results include all movies except those classified "R".
Hint: Use a subquery to select the age of your chosen customer.

	movie_name	duration	movie_desc
1	The Shawshank Redemption	142	Two imprisoned men bond over a number of years,...
2	Forrest Gump	142	Forrest Gump, while not intelligent, has accidentall...
3	Star Wars: Episode IV - A New Hope	121	Luke Skywalker joins forces with a Jedi Knight, a ...
4	WALL-E	98	In the distant future, a small waste collecting robot ...
5	Eternal Sunshine of the Spotless Mind	108	When their relationship turns sour, a couple under...
6	Monty Python and the Holy Grail	91	King Arthur and his knights embark on a low-budg...
7	Up	96	Seventy-eight year old Carl Fredricksen travels to ...
8	Gran Torino	116	Disgruntled Korean War veteran Walt Kowalski se...
9	Metropolis	153	In a futuristic city sharply divided between the wor...

(example of query results – your data may vary, example illustrates structure only)

Create a **Session Revenue** query that selects the session ID, session date/time, movie name, cinema name, tickets sold and total revenue of all sessions that occurred in the past. Total revenue is the session cost multiplied by the number of tickets sold. Ensure that sessions that had no tickets sold appear in the results (with 0 tickets sold and 0 revenue). Order the results by total revenue, in descending order.

	session_id	session_time	movie_name	cinema_name	tickets_sold	total_revenue
1	1	2018-06-02 09:00:00	Metropolis	Cinema 1	6	90.00
2	2	2018-06-02 12:00:00	Forrest Gump	Cinema 1	4	60.00
3	4	2018-06-02 18:00:00	Eternal Sunshine of the Spotless Mind	Cinema 1	4	60.00
4	5	2018-06-02 09:00:00	Gran Torino	Cinema 3	3	45.00
5	3	2018-06-02 15:00:00	Pulp Fiction	Cinema 1	3	45.00
6	6	2018-06-02 12:00:00	Morty Python and the Holy Grail	Cinema 3	2	30.00
7	8	2018-06-02 18:00:00	The Shawshank Redemption	Cinema 3	1	15.00
8	7	2018-06-02 15:00:00	Star Wars: Episode IV - A New Hope	Cinema 3	0	0.00

(example of query results – your data may vary, example illustrates structure only)

3. Marking Guide

Marking Criteria	Marks
Database creation and population script	6
Cinema View	1
Session View	2
Query 1	2
Query 2	2
Query 3	2
Query 4	2
Query 5	2
Query 6	3
Query 7	3
Query 8	4
Query 9	4
Comments and formatting	2
Total	/35