INFS2200/7903 PROJECT ASSIGNMENT

Semester Two 2021

Total marks:	100 marks (25%)
Due Date:	11:59PM, 28 October 2021
What to Submit:	SQL script file + short report
Where to Submit:	Electronic submission via Blackboard

The goal of this project is to gain practical experience in applying several database management concepts and techniques using the Oracle DBMS.

Your main task is to first populate your database with appropriate data, then design, implement, and test the appropriate queries to perform the tasks explained in the next sections.

You must work on this project **individually**. Academic integrity policies apply. Please refer to *3.60.04 Student Integrity and Misconduct* of the University Policy for more information.

Roadmap: Section 1 describes the database schema for your project and it also provides instructions on downloading the script file needed to create and populate your database. Section 2 describes the tasks to be completed for this project. Finally, Section 3 provides you with all the necessary submission guidelines.

Enjoy the project!

--------☙---------

SECTION 1. THE MOVIES DATABASE

The Database: The MOVIES database (Figure 1) captures the information regarding movies and the actors in these movies. The database includes six tables: **film**, **actor**, **category**, **language**, **film_actor**, and **film_category**. **Film** keeps track of film details. **Actor** stores information about all actors in the movie industry. **Category** stores the information about the different types of film categories. **Language** stores the different languages in which these movies are released. **Film_actor** and **film_category** keep track of which actors have acted in which films, and which films are classified under which categories, respectively.

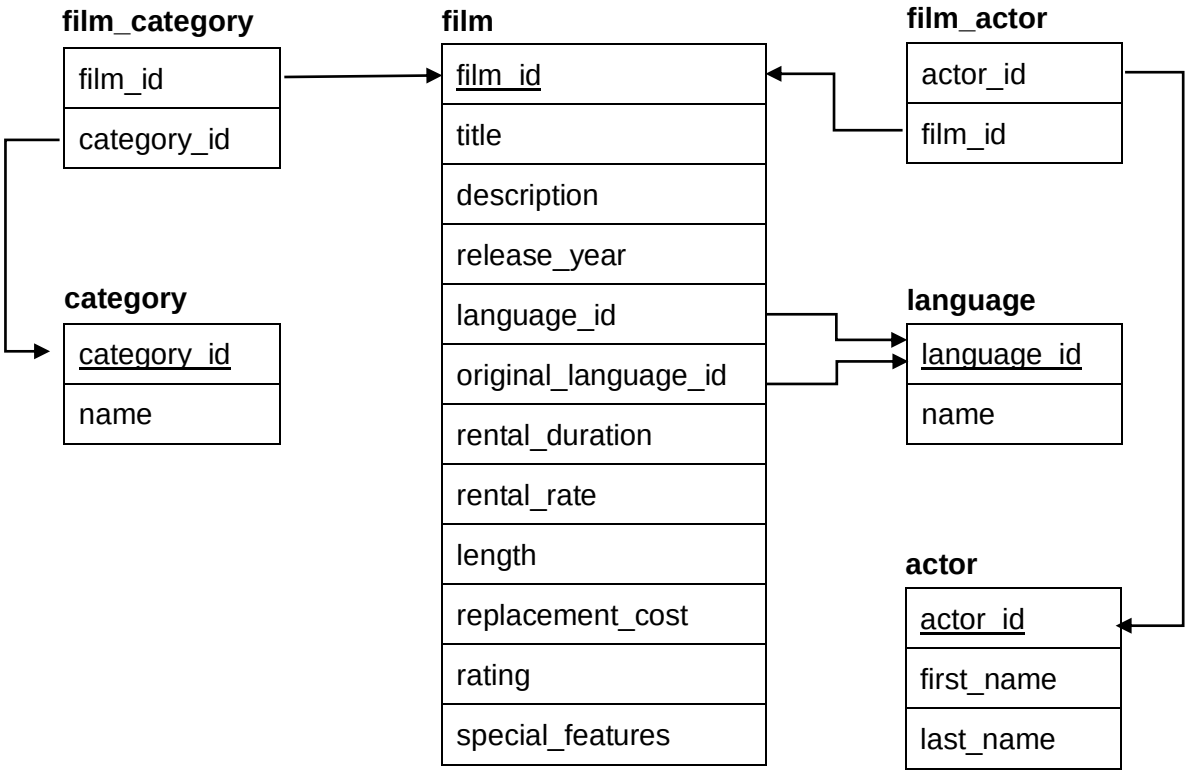

Figure 1 Database schema

The Script File: Please go to Blackboard and download the supplementary script file for this project “prjScript.sql”.

The Database Constraints: The following table lists all the constraints that should be created on the MOVIES database.

No	Constraint Name	Table.Column	Description
1	PK_ACTORID	actor.actor_id	actor_id is the primary key of actor
2	PK_CATEGORYID	category.category_id	category_id is the primary key of category
3	PK_FILMID	film.film_id	film_id is the primary key of film
4	PK_LANGUAGEID	language.language_id	language_id is the primary key of language
5	UN_DESCRIPTION	film.description	Film description values are unique
6	CK_FNAME	actor.first_name	Actor's first name must not be empty (not null)
7	CK_LNAME	actor.last_name	Actor's last name must not be empty (not null)
8	CK_CATNAME	category.name	Category name must not be empty (not null)
9	CK_LANNAME	language.name	Language name must not be empty (not null)
10	CK_TITLE	film.title	Film title must not be empty (not null)
11	CK_RELEASEYR	film.release_year	film.release_year is less than or equal to current year (Hardcode the current year 2020)
12	CK_RATING	film.rating	Rating type must be one of the following: 'G', 'PG', 'PG-13', 'R', 'NC-17'
13	CK_SPLFEATURES	film.special_features	Special features type must be either empty or one of the following: 'Trailers', 'Commentaries', 'Deleted Scenes', 'Behind the Scenes'
14	FK_LANGUAGEID	film.language_id and language.language_id	film.language_id refers to language.language_id
15	FK_ORLANGUAGEID	film.original_language_id and language.language_id	film.original_language_id refers to language.language_id
16	FK_ACTORID	film_actor.actor_id and actor.actor_id	film_actor.actor_id refers to actor.actor_id
17	FK_CATEGORYID	film_category.category_id and category.category_id	film_category.category_id refers to category.category_id
18	FK_FILMID1	film_actor.film_id and film.film_id	film_actor.film_id refers to film.film_id
19	FK_FILMID2	film_category.film_id and film.film_id	film_category.film_id refers to film.film_id

Table 1. Database constraints

--------☙---------

SECTION 2. ASSIGNMENT TASKS

Create and Populate Database: You need to execute the script file "prjScript.sql" to create and populate your database before working on the following tasks. Wait till you see the message "DONE! All data has been inserted." It should only take one minute. The script will also drop related tables.

Task 1 – *Constraints*

1. After running the script file, you will notice that only some of the constraints listed in Table 1 were created. Write a SQL statement to find out what constraints have been created on the six tables. (***Note:*** *some table names may need to be in capitals, e.g., 'FILM' instead of 'film'*)
   Question: Which constraints in Table 1 have been created on these six tables?

2. Write the SQL statements to create all the missing constraints.

Task 2 – *Triggers*

1. Assume that the film_id should be automatically populated when a new film is added. Write a SQL statement to create a sequence object to generate values for this column. The sequence, named FILM_ID_SEQ, should start from 20,010 and increment by 10.

2. Write a SQL statement to create an Oracle trigger called BI_FILM_ID that binds the sequence object FILM_ID_SEQ to the film_id column, i.e., the trigger populates values of FILM_ID_SEQ to the film_id column when a new film is added.

3. Write a SQL statement to create an Oracle trigger BI_FILM_DESP that appends text to the description of every new film inserted into the database. The text is based on the rating, the language, and the original language of the film. The format of the text you append should be as follows (replacing tokens):

   <rating>-<seq>: Originally in <original language>. Re-released in <language>.

   Here, <seq> is the sequence number of the film with that <rating>, and <original language> and <language> should be the name of the language from the language table.

   Hint: *You might need to use some built-in functions for string manipulation such as TO_CHAR, CONCAT, SUBSTR, INSTR, etc.*

Example: Assume the following film is inserted into the database, which is the 5th film with a rating 'PG' (i.e., there are already 4 films with the rating 'PG' in the database), and the current FILM_ID_SEQ value is 20,010.

INSERT INTO film (title, description, language_id, original_language_id, rating) VALUES ('B Movie', 'Movie about wasps.', 1, 2, 'PG');

It should produce the following result when the following SQL statement is run:

```
SQL> SELECT description FROM film WHERE film_id = 20010;

DESCRIPTION
--------------------------------------------------------------------------------
Movie about wasps.PG-5: Originally in Italian. Re-released in English.
```

Notes for Task 2.3:

- *The new description must match the expected output **exactly** in order to receive marks. For example,*
  - *Do not add extra space or line break;*
  - *Do not change capitalisation of the rating or the language names.*
- *If either rating, language_id, or original_language_id of the new film is null, then the trigger should do nothing, i.e., the new film uses the original description.*
- *You do not need to handle the cases where the resulting text after the trigger exceeds the description length. Let the trigger fail.*
- *Do not use hardcode. Your trigger should be able to handle other languages beyond those provided to you in the SQL script.*
  - *For example, if the language 'SQL' is added to the language table, then the trigger should be able to handle a movie in 'SQL'.*

Task 3 – *Views*

1. Write a SQL statement to find the 'Action' (category) films with the shortest running time (length). Your query should output the titles and lengths of the films.

2. Write a SQL statement to create a (virtual) view called MIN_ACTION_ACTORS that contains all the actors who have acted in the films you obtained in Task 3.1. The view should include the columns actor_id, first_name, and last_name. (***Note:*** *Each actor should only appear once in the view, even if they may have acted in multiple films*)

3. Write a SQL statement to create a (virtual) view called V_ACTION_ACTORS_2012 that lists the ids, first names and last names of all the actors who have acted in an 'Action' film released in the year 2012. (***Note:*** *There should be no duplicate rows in the view, similar to Task 3.2*)

4. Write a SQL statement to create a materialized view MV_ACTION_ACTORS_2012 that lists the same information as in Task 3.3.

5. Execute the following two SQL statements and report their query execution time. ***Question:*** Did the materialized view speed up the query processing? Explain your answer. (***Hint:*** *You should look at both the elapsed time and the cost in the query execution plan*)

```
SELECT * FROM V_ACTION_ACTORS_2012;
SELECT * FROM MV_ACTION_ACTORS_2012;
```

Note: *For any task mentioning the execution time, please run the queries on a computer with a HDD rather than an SSD, so that the timing difference is noticeable. All lab computers have HDDs and are appropriate for such task.*

Task 4 – *Indexes*

1. Write a SQL statement to find the first 100 films (in ascending alphabetical order of the film titles) that take place in a 'Boat', i.e., the word 'Boat' appears in the film description. (***Note:*** *You should avoid using LIKE in the SQL statement and instead use string manipulation functions*)

2. In order to potentially speed up the query in Task 4.1, a function-based index could be created on the film table. Write a SQL statement to create an index IDX_BOAT that best fits the task and justify your choice.

3. Report the execution time and execution plan of the query statement you wrote in Task 4.1 before and after creating the index in Task 4.2.
   Question: Did the index speed up query processing*?* Explain your answer. (***Hint:*** *You should look at both the elapsed time and the cost in the query execution plan)*

4. Write a SQL statement to count the number of films for which there are at least 40 other films with the same release_year, rating, and special_features values.

5. In order to potentially speed up the query in Task 4.4, indexes should be created on the release_year, rating, and special_features columns.
   Question: In your opinion, what is the most suitable index type to create on those columns, and why? (***Note:*** *Do not include any SQL to create the index in your script file; just provide your answer in the report*)

Task 5 – *Execution Plan*

1. A B+ tree index PK_FILMID has been generated automatically for the primary key film_id of the table film. Write SQL statements to answer the following ***Questions:***

- What is the height of the B+ tree index?
- What is the number of leaf blocks in the B+ tree index?
- What is the number of block access needed for a full table scan of the film table?

Hint: *You may find the following documents from Oracle helpful for Task 5.1:*

- *https://docs.oracle.com/cd/B28359_01/server.111/b28320/statviews_5119.htm#REFRN29025*
- *https://docs.oracle.com/cd/B19306_01/server.102/b14237/statviews_4473.htm#REFRN26286*
- *https://docs.oracle.com/cd/B19306_01/server.102/b14237/statviews_2105.htm#REFRN20286*

2. The following SQL statement lists all the films with a film_id larger than 100:

```
SELECT * FROM FILM WHERE FILM_ID > 100;
```

Report the rule-based execution plan chosen by the Oracle optimizer for executing this query.
Question: Explain the query processing steps taking place in this plan.

3. Report the cost-based execution plan chosen by the Oracle optimizer for executing the query in Task 5.2.
Question: Explain the query processing steps taking place in this plan. In your opinion, what are the main differences between the plans you obtained in Task 5.2 and Task 5.3, based on the statistics from Task 5.1 and your calculation?

4. The following SQL statement lists all the films with a film_id larger than 19,990:

```
SELECT * FROM FILM WHERE FILM_ID > 19990;
```

Report the cost-based execution plan chosen by the Oracle optimizer for executing this query.
Question: Explain the query processing steps taking place in this plan. In your opinion, what are the main differences between the plans you obtained in Task 5.3 and Task 5.4, based on the statistics from Task 5.1 and your calculation?

5. The following SQL statement lists all information for the film with a film_id of 100:

```
SELECT * FROM FILM WHERE FILM_ID = 100;
```

Report the cost-based execution plan chosen by the Oracle optimizer for executing this query.
Question: Explain the query processing steps taking place in this plan. In your opinion, what are the main differences between the plans you obtained in Task 5.3 and Task 5.5, based on the statistics from Task 5.1 and your calculation?

--------☙---------

SECTION 3. DELIVERABLES

The project is due **11:59PM, 28 October 2021**. Late submissions will not be accepted unless you are approved for an extension (refer to Section 5.3 of the ECP).

You are required to turn in two files (use StudentID to name your files):

1. StudentID.pdf: (replacing StudentID) – *Submit on Blackboard via the Turnitin link "Report Submission"*
   A report that answers all the questions in Section 2 including all the necessary SQL statements and screenshots of their outputs.

2. StudentID.sql: (replacing StudentID) – *Submit on Blackboard via the standard upload link "SQL Script Submission"*
   A plaintext script file that includes all your SQL statements.

Your **report** file should include the following content:

- Answers to all the **Questions** in Section 2.

- If you are asked to write SQL statements, you need to include those statements in your report.

- After you execute a SQL statement, if Oracle produces any output (e.g. query result, query execution time, query plan, etc), you should also include a screenshot of the output as well. (***Note:*** *Please be sensible when including query output. Any output close to the size of one page can be shown by just including the first 10 lines and the last 10 lines. A report that includes multiple pages of a query output will lose presentation marks. You may find some helpful instructions for formatting query output in Practical 1 or the following Oracle documentation)*
  https://docs.oracle.com/cd/A57673_01/DOC/server/doc/SP33/ch4.htm

Your **script** file is in plain text format. You must make sure that your script file can be executed on the ITEE lab computers by the "@" command. The same SQL statements in your script file should also be copied and pasted into your report file (as explained above). Even though the script file does not introduce any new information compared to the report, it is intended to help the lecturer/tutors to quickly check the correctness of your SQL statements before checking the details in your report file.

Enjoy the project! Good luck!