TASK 1

Table Definition:

1st Table : Cinema

```
--Task 1a)--
-- CINEMA Table
CREATE TABLE Cinema (
    Cinema_ID NUMBER PRIMARY KEY,
    Cinema_Name VARCHAR2(100),
    Cinema_Address VARCHAR2(200),
    Cinema_Phone VARCHAR2(15)
);
```

```
Table created.
```

2nd Table: Theatre

```
-- THEATER Table
CREATE TABLE Theater (
    Theater_ID NUMBER PRIMARY KEY,
    Theater_Capacity NUMBER,
    Cinema_ID NUMBER,
    FOREIGN KEY (Cinema_ID) REFERENCES Cinema(Cinema_ID)
);

```

```
Table created.
```

3rd Table: Movie

```
-- MOVIE Table
CREATE TABLE Movie (
    Movie_ID NUMBER PRIMARY KEY,
    Movie_Name VARCHAR2(100),
    Movie_Director VARCHAR2(100),
    Movie_Rating NUMBER(2, 1)
);
```

4th Table: Showing

```
25  -- SHOWING Table
26  CREATE TABLE Showing (
27      Showing_Time TIMESTAMP,
28      Showing_Date DATE,
29      Theater_ID NUMBER,
30      Movie_ID NUMBER,
31      Showing_Attendance NUMBER,
32      PRIMARY KEY (Showing_Time, Showing_Date, Theater_ID, Movie_ID),
33      FOREIGN KEY (Theater_ID) REFERENCES Theater(Theater_ID),
34      FOREIGN KEY (Movie_ID) REFERENCES Movie(Movie_ID)
35  );
```

b) Adding a record

1st Table: Cinema

```
38  --TASK 1b)--
39  -- Insert records into CINEMA table
40  INSERT INTO Cinema (Cinema_ID, Cinema_Name, Cinema_Address, Cinema_Phone)
41  VALUES (001, 'KGALE CINEMAS', 'Gaborone, Botswana', '267-391-0808');
42  INSERT INTO Cinema (Cinema_ID, Cinema_Name, Cinema_Address, Cinema_Phone)
43  VALUES (002, 'NEW CAPITOL CINEMA', 'Riverwalk Shopping Mall, Gaborone, Botswana', '267-370-0110');
44  INSERT INTO Cinema (Cinema_ID, Cinema_Name, Cinema_Address, Cinema_Phone)
45  VALUES (003, 'The Magicstar Assembly', 'Shop 101, Riverwalk, Gaborone, Botswana', '267-370-0110');
46  INSERT INTO Cinema (Cinema_ID, Cinema_Name, Cinema_Address, Cinema_Phone)
47  VALUES (004, 'STARDUST CINEMA', 'Mogoditshane, Gaborone, Botswana', '267-395-9271');
48
```

```
1 row(s) inserted.

1 row(s) inserted.

1 row(s) inserted.

1 row(s) inserted.
```

2nd Table: Theatre

```sql
-- Insert records into THEATER table
INSERT INTO Theater (Theater_ID, Theater_Capacity, Cinema_ID)
VALUES (101, 300, 001);
INSERT INTO Theater (Theater_ID, Theater_Capacity, Cinema_ID)
VALUES (102, 250, 002);
INSERT INTO Theater (Theater_ID, Theater_Capacity, Cinema_ID)
VALUES (103, 350, 003);
INSERT INTO Theater (Theater_ID, Theater_Capacity, Cinema_ID)
VALUES (104, 200, 004);
```

```
1 row(s) inserted.

1 row(s) inserted.

1 row(s) inserted.

1 row(s) inserted.
```

3rd Table: Movie

```sql
-- Insert records into MOVIE table
INSERT INTO Movie (Movie_ID, Movie_Name, Movie_Director, Movie_Rating)
VALUES (1, 'Interstellar', 'Christopher Nolan', 8.6);
INSERT INTO Movie (Movie_ID, Movie_Name, Movie_Director, Movie_Rating)
VALUES (2, 'Fight Club ', 'David Fincher', 8.8);
INSERT INTO Movie (Movie_ID, Movie_Name, Movie_Director, Movie_Rating)
VALUES (3, 'The Lives of Others', 'Florian Henckel von Donnersmarck', 8.4);
INSERT INTO Movie (Movie_ID, Movie_Name, Movie_Director, Movie_Rating)
VALUES (4, 'The Godfather', 'Francis Ford Coppola', 9.2);
```

```
1 row(s) inserted.

1 row(s) inserted.

1 row(s) inserted.

1 row(s) inserted.
```

4th Table: Showing

```sql
-- Insert records into SHOWING table
INSERT INTO Showing (Showing_Time, Showing_Date, Theater_ID, Movie_ID, Showing_Attendance)
VALUES (TIMESTAMP '2024-11-05 14:00:00', DATE '2024-11-05', 101, 1, 104);
INSERT INTO Showing (Showing_Time, Showing_Date, Theater_ID, Movie_ID, Showing_Attendance)
VALUES (TIMESTAMP '2024-11-05 18:00:00', DATE '2024-11-05', 102, 2, 210);
INSERT INTO Showing (Showing_Time, Showing_Date, Theater_ID, Movie_ID, Showing_Attendance)
VALUES (TIMESTAMP '2024-11-06 16:00:00', DATE '2024-11-06', 103, 3, 350);
INSERT INTO Showing (Showing_Time, Showing_Date, Theater_ID, Movie_ID, Showing_Attendance)
VALUES (TIMESTAMP '2024-11-06 20:00:00', DATE '2024-11-06', 104, 4,203 );
```

```
1 row(s) inserted.

1 row(s) inserted.

1 row(s) inserted.

1 row(s) inserted.
```

c) A SQL query to remove a particular data row

```sql
--  Delete the specific row from the Cinema table
DELETE FROM Cinema
WHERE Cinema_ID = 3;
```

```
1 row(s) deleted.
```

d) Modify a certain row

```sql
--Updating a specific row
UPDATE Cinema
SET Cinema_Phone = '267-123-4567'
WHERE Cinema_ID = 2;
```

```
1 row(s) updated.
```

TASK 2:

a) List the names of the films that are showing on the day that the client has chosen

```
100 SELECT M.Movie_Name
101 FROM Movie M
102 JOIN Showing S ON M.Movie_ID = S.Movie_ID
103 WHERE S.Showing_Date = TO_DATE('2024-11-05', 'YYYY-MM-DD');
104
```

MOVIE_NAME
Interstellar
Fight Club

Download CSV

2 rows selected.

b) Get the names of all theatres, films, and directors depending on a particular film that the customer has chosen

```
105 --To get all cinema names, movie names, and movie directors based on a specific movie chosen by the customer
106 SELECT C.Cinema_Name, M.Movie_Name, M.Movie_Director
107 FROM Cinema C
108 JOIN Theater T ON C.Cinema_ID = T.Cinema_ID
109 JOIN Showing S ON T.Theater_ID = S.Theater_ID
110 JOIN Movie M ON S.Movie_ID = M.Movie_ID
111 WHERE M.Movie_Name = 'The Godfather';
```

CINEMA_NAME	MOVIE_NAME	MOVIE_DIRECTOR
STARDUST CINEMA	The Godfather	Francis Ford Coppola

c) Set Cinema_Phone to seven numbers

```
113 --Adjust Cinema_Phone to 7 digits
114 UPDATE Cinema
115 SET Cinema_Phone = SUBSTR(Cinema_Phone, -7)
116 WHERE LENGTH(Cinema_Phone) > 7;
117
```

3 row(s) updated.

d) Modify the ratings for the films

```
118 --Adjusting movie ratings
119 ALTER TABLE Movie
120 ADD Movie_Rating_Text VARCHAR2(100);
121 UPDATE Movie
122 SET Movie_Rating_Text =
123     CASE
124         WHEN Movie_Rating = 8.6 THEN 'G: General audiences – All ages admitted'
125         WHEN Movie_Rating = 8.8 THEN 'PG: Parental guidance suggested – Some material may not be suitable for children'
126         WHEN Movie_Rating = 8.4 THEN 'PG-13: Parents strongly cautioned – Some material may be inappropriate for children under 13'
127         WHEN Movie_Rating = 9.2 THEN 'R: Restricted – Under 17 requires accompanying parent or adult guardian'
128         ELSE TO_CHAR(Movie_Rating)
129     END;
130 ALTER TABLE Movie
```

```
Table altered.

4 row(s) updated.
```

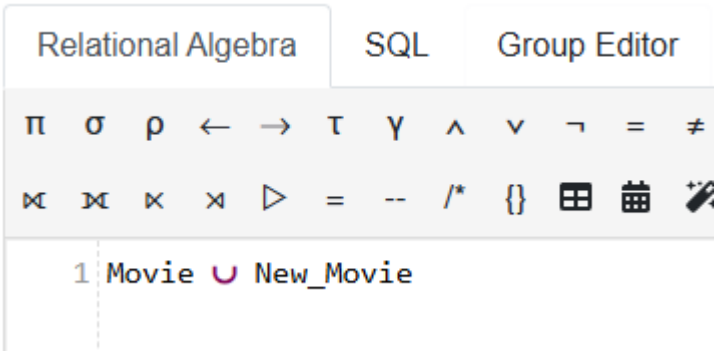
e) Show the films that have been rated G

```
135 --Display movies with G rating
136
137 SELECT Movie_Name, Movie_Rating
138 FROM Movie
139 WHERE Movie_Rating LIKE 'G%';
140
```

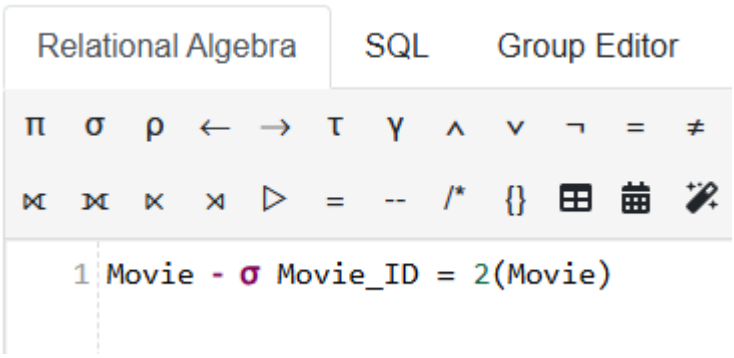
MOVIE_NAME	MOVIE_RATING
Interstellar	G: General audiences – All ages admitted

TASK 3

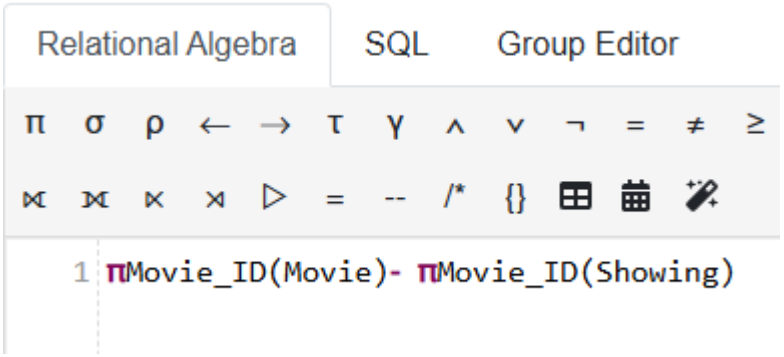
a) Adding a new tuple to the movie relation using a relational algebra command



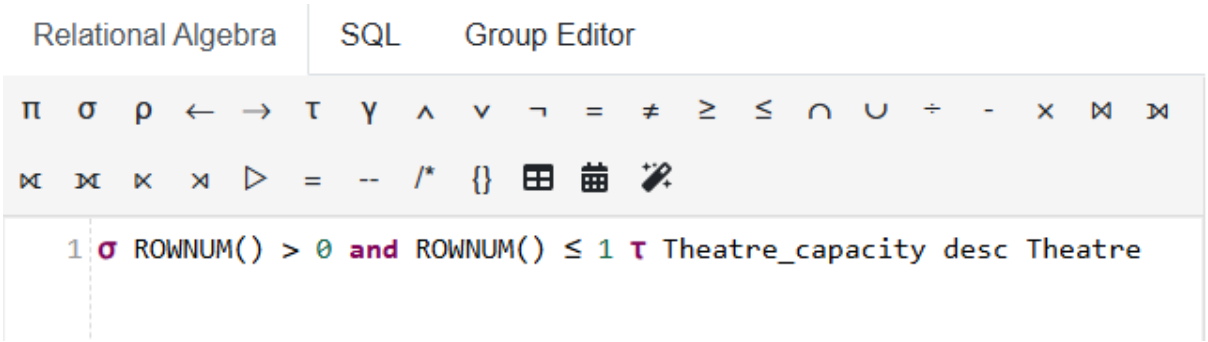
b) Relational algebra command to delete a tuple of your choice in the movie relation



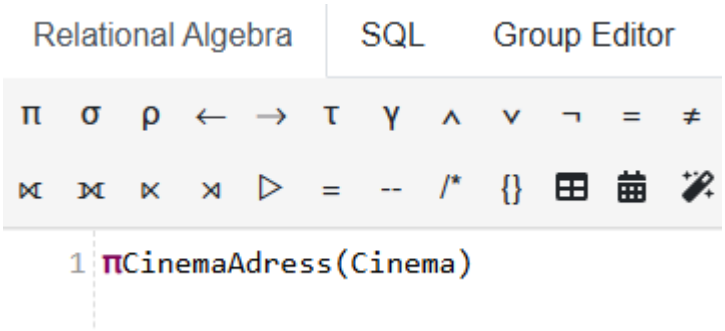
c) Relational algebra command to display the movie identifier that has not been programmed for projection



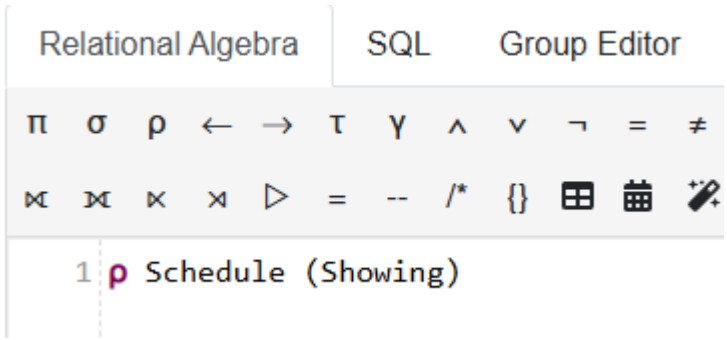
d) Relational algebra command to show the theater with the largest capacity



e) Relational algebra command to display all theater addresses



f) Relational algebra command to rename the projection relation to program



TASK 4

a) Trigger to prevent insertion and update capacity of 0 or negative for theatre

```sql
--Task 4
--trigger that prevents inserts and update of zero or negative capacity for the theatre
CREATE OR REPLACE TRIGGER trg_check_capacity
BEFORE INSERT OR UPDATE ON Theater
FOR EACH ROW
BEGIN
  IF :NEW.Capacity <= 0 THEN
    RAISE_APPLICATION_ERROR(-20001, 'Capacity must be greater than zero.');
  END IF;
END trg_check_capacity;
/
```

SQL Statement Output

b) Function to store a set of data in movie table using provided parameters of inserted data

```sql
CREATE OR REPLACE FUNCTION insert_movie(
  p_movie_id IN NUMBER,
  p_movie_name IN VARCHAR2,
  p_director IN VARCHAR2,
  p_genre IN VARCHAR2,
  p_rating IN VARCHAR2
)
RETURN VARCHAR2
IS
BEGIN
  INSERT INTO Movie (Movie_ID, Movie_Name, Movie_Director, Movie_Genre, Movie_Rating)
  VALUES (p_movie_id, p_movie_name, p_director, p_genre, p_rating);
  -- Commit the transaction to save the changes
  COMMIT;
  RETURN 'Movie inserted successfully';
EXCEPTION
  WHEN DUP_VAL_ON_INDEX THEN
    RETURN 'Error: Movie ID already exists';
```

Function created.

```
  p_genre IN VARCHAR2,
  p_rating IN VARCHAR2
)
RETURN VARCHAR2
IS
BEGIN
  INSERT INTO Movie (Movie_ID, Movie_Name, Movie_Director, Movie_Genre, Movie_Rating)
  VALUES (p_movie_id, p_movie_name, p_director, p_genre, p_rating);
  -- Commit the transaction to save the changes
  COMMIT;
  RETURN 'Movie inserted successfully';
EXCEPTION
  WHEN DUP_VAL_ON_INDEX THEN
    RETURN 'Error: Movie ID already exists';
  WHEN OTHERS THEN
    RETURN 'Error: ' || SQLERRM;
END insert_movie;
```

```
Function created.
```

c) Procedure to use cursor to display all details of movie

```
--procedure that uses a cursor to display all movie details
CREATE OR REPLACE PROCEDURE display_all_movies
IS
  -- Define a cursor to retrieve all movie details
  CURSOR movie_cursor IS
    SELECT Movie_ID, Movie_Name, Movie_Director, Movie_Genre, Movie_Rating
    FROM Movie;

  -- Variables to store individual movie details
  v_movie_id Movie.Movie_ID%TYPE;
  v_movie_name Movie.Movie_Name%TYPE;
  v_movie_director Movie.Movie_Director%TYPE;
  v_movie_genre Movie.Movie_Genre%TYPE;
  v_movie_rating Movie.Movie_Rating%TYPE;
BEGIN
  -- Open the cursor and fetch details
  OPEN movie_cursor;
  LOOP
```

```
Procedure created.
```

```
197     FETCH movie_cursor INTO v_movie_id, v_movie_name, v_movie_director, v_movie_genre, v_movie_rating;
198
199     -- Exit loop when no more rows are returned
200     EXIT WHEN movie_cursor%NOTFOUND;
201
202     -- Display the movie details
203     DBMS_OUTPUT.PUT_LINE('Movie ID: ' || v_movie_id);
204     DBMS_OUTPUT.PUT_LINE('Movie Name: ' || v_movie_name);
205     DBMS_OUTPUT.PUT_LINE('Director: ' || v_movie_director);
206     DBMS_OUTPUT.PUT_LINE('Genre: ' || v_movie_genre);
207     DBMS_OUTPUT.PUT_LINE('Rating: ' || v_movie_rating);
208     DBMS_OUTPUT.PUT_LINE('------------------------------');
209   END LOOP;
210
211   -- Close the cursor
212   CLOSE movie_cursor;
213 END display_all_movies;
214 /
```

```
Procedure created.
```

TASK 5

Memo
To: Management Team
From: Junior Mosimanenkwe
Subject: Justification for Adopting a Database Management System (DBMS) over a File System
Date: 18/11/2024

Introduction

Effective information management is essential to maintaining effectiveness, ensuring information intelligence, and promoting critical decision-making as the business grows as stated by (şık, et al., 2013). The limitations of the traditional file-based method are addressed by switching from a record framework to a Database Administration Framework (DBMS). I present a point-by-point comparison and the reasons why adopting a DBMS might be an essential first step for the company below.

1. Improved Organization and Management of Information
• **Centralized Information Capacity:** Unlike dispersed records, a database management system (DBMS) keeps information in an orderly, centralized manner that is easier to monitor and access as stated by Narang, (2018).

• **Information Connections:** Unlike record frameworks, a database management

system (DBMS) may manage intricate relationships between information substances, advancing the organization and relevance of information.

2. Knowledge and Accuracy of Information

• **Less Repetition**: According to Eessaar (2016), A DBMS reduces capacity requirements and irregularities by using normalizing techniques to get rid of extra information.

• **Information Approval:** A DBMS's rules and requirements ensure that significant information is entered accurately and with fewer errors.

3. Advancements in Information Security

• **Take Charge:**

Strong tools are provided by a DBMS to limit unwanted access through parts, consents, and verification.

Examine the trails:

It keeps track of modifications, ensures accountability, and authorizes evaluations for compliance as indicated by Force, et al., 2013.

4. Recovery of Productive Information

• **Inquiry Dialect:** A DBMS uses SQL (Organized Inquiry Dialect) to efficiently retrieve data as stated by Silva, et al., 2016, enabling quicker access to data than searching through physical records.

• **Ordering**: Advanced DBMS ordering strategies maximize look operations, saving time and achieving significant efficiency gains.

5. Flexibility and Performance

• **Handling Massive Volumes**: A DBMS is designed to manage large datasets without causing corruption in execution.

• **Concurrency management:** Unlike record frameworks that are prone to information degradation during concurrent use according to Zahabi, et al., 2015, it supports many clients accessing the information simultaneously without conflicts.

6. Information Recovery and Reinforcement

• **Robotized Reinforcements:** According to Omrany, et al., 2023, to ensure data integrity and reduce manual mediation, a database management system integrates mechanized reinforcement highlights.

• **Recuperation Tools:** A DBMS can use exchange logs and recovery techniques to restore data to a steady state in the event of a failure.

7. Long-Term Cost-Effectiveness

• Although the initial cost of implementing a DBMS may be greater, its productivity, reduced information risk, and advanced decision-making skills translate into significant long-term reserve funds as indicated by Niu, et al., (2021).

8. Support for Advanced Analytics

• Abdrabo, et al., (2016) stated that, A database management system (DBMS) provides tidbits of information for crucial organization and supports interaction with expository tools and innovations like information warehousing and machine learning.

The constraints of a file system

• **Manual Preparation:** Information recovery is repetitious since record frameworks require sophisticated inquiry skills.

• **Irregularity and repetition:** Records usually include conflicting information and duplicates, which leads to wasteful features.

• **Inadequate Security**: Record systems require strict access control, posing risks to sensitive data.

In conclusion

Adopting a DBMS would enable the company to manage its data more effectively, securely, and efficiently, paving the way for better operational and decision-making skills. In order to overcome the limitations of the record architecture and adapt to our long-term critical goals, I categorically recommend switching to a DBMS.

Sincerely,
Junior Mosimanenkwe
Student

REFERENCE

Eessaar, E., 2016. The Database Normalization Theory and the Theory of Normalized Systems: Finding a Common Ground. *Baltic Journal of Modern Computing*, *4*(1).

Force, J.T. and Initiative, T., 2013. Security and privacy controls for federal information systems and organizations. *NIST Special Publication*, *800*(53), pp.8-13.

Işık, Ö., Jones, M.C. and Sidorova, A., 2013. Business intelligence success: The roles of BI capabilities and decision environments. *Information & management*, *50*(1), pp.13-23.

Narang, R., 2018. *Database management systems*. PHI Learning Pvt. Ltd.

Omrany, H., Al-Obaidi, K.M., Husain, A. and Ghaffarianhoseini, A., 2023. Digital twins in the construction industry: a comprehensive review of current implementations, enabling technologies, and future directions. *Sustainability*, *15*(14), p.10908.

Silva, Y.N., Almeida, I. and Queiroz, M., 2016, February. SQL: From traditional databases to big data. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education* (pp. 413-418).

Zahabi, M., Kaber, D.B. and Swangnetr, M., 2015. Usability and safety in electronic medical records interface design: a review of recent literature and guideline formulation. *Human factors*, *57*(5), pp.805-834.

Niu, Y., Ying, L., Yang, J., Bao, M. and Sivaparthipan, C.B., 2021. Organizational business intelligence and decision making using big data analytics. *Information Processing & Management*, *58*(6), p.102725

Abdrabo, M., Elmogy, M., Eltaweel, G. and Barakat, S., 2016. Enhancing big data value using knowledge discovery techniques. *IJ Information Technology and Computer Science*, *8*(1-12), pp.1-4.