Object-Relational Database 31075/42901

Spring 2017

Database Design and Implementation

Online Movies Database (OMDB)

Prepared by:

Tianpeng GOU, 12680373

Manchun CHENG, 12646269

Submitted on:

Wed 18/10/2017

Oracle Username:

12680373

Table of Contents

1. Introduction

This report will propose an object – relational design solution for an Online Movies Database (OMDB), along with implementing design schema into Oracle database by using SQL DDL and test certain search queries to validate the feasibility of the design.

Firstly, the design process will transform the Relational Model from requirements into Object-Relational (Data Type) Model by maximizing the utilization of object relational features. And it will also focus on discussing the design alternatives and how those alternatives progressively reflect and lead to the final design decision. Trade-offs and balances need to be considered to ensure the conciseness and elegance of this design. The critical design difficulty is to sort out the relations between Artist type and Movie type, where Artist can be divide into subtypes: Actor, Director, Writer and Crew. Those subtypes of Artist need to be act as data type and embedded into Movie as either a single type, a reference or a collection. The proposed solution is to populate basic artists' information into Artist table, establish reference link and insert the reference of Artist into the four subtypes (In this scenario, reference relations rather than inherit relations) and finally establish embedded structure link between collections of four types and Movie.

Secondly, the data types will be written as DDL into script, the creation sequence is critical to the design flow. Also, this part will emphasis on the design of member functions which will be used in the latter search queries. This part will be documented in Appendix A.

Finally, simple data will be inserted and updated into created tables to test the validity of queries which are given by requirements. This part will be documented in Appendix B.

Additionally, a modification has been implemented to comply with the requirement. This part will be documented in Appendix C.

Appendix D will illustrate the screenshot of query outputs to prove the authenticity of query results.

(Both team members fully participated in preparing the design and contributed equally.)

2. OMDB Database Design

2.1 Design Solution – Data Type Model

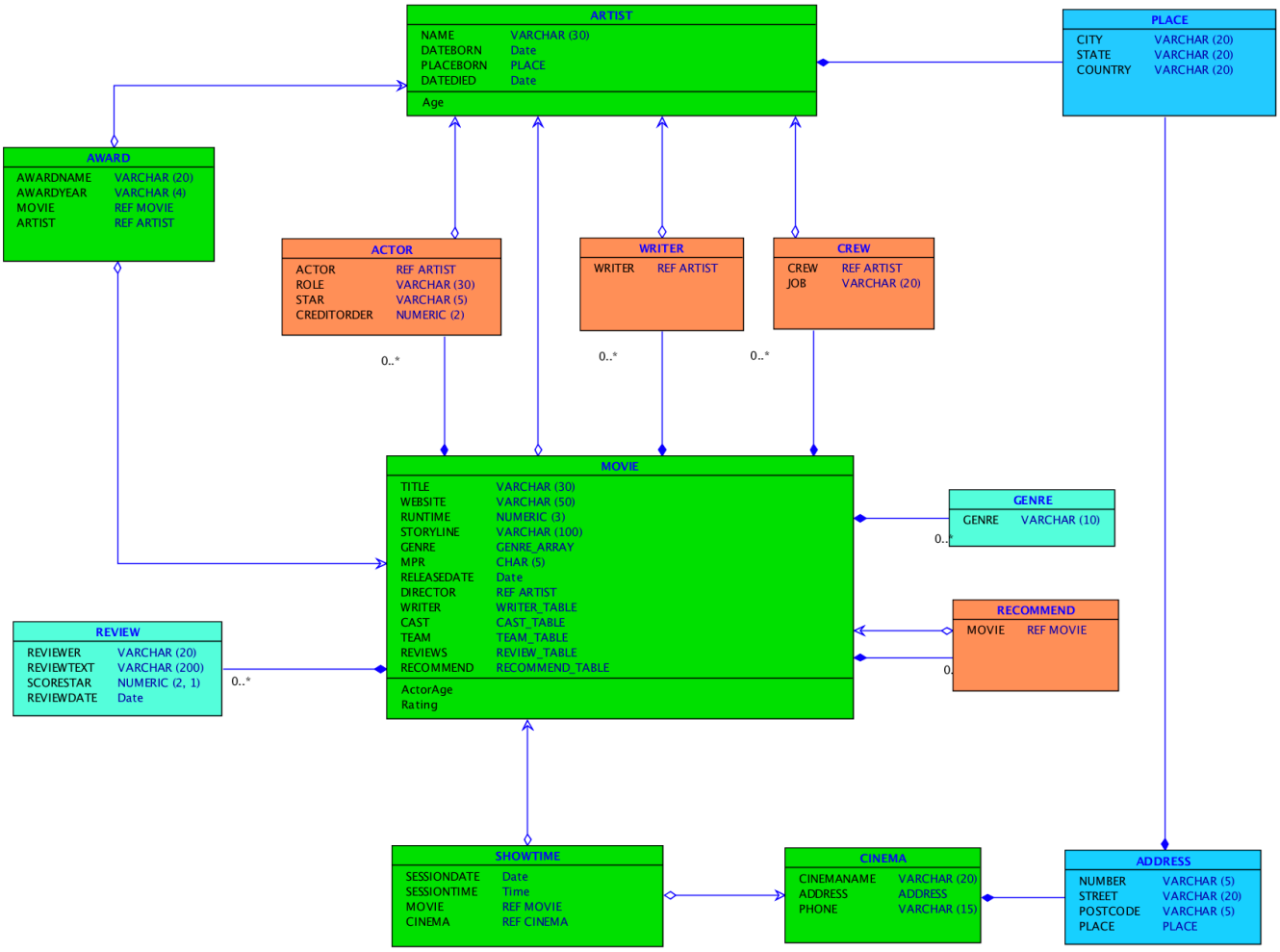

Figure 2.1 – OMDB Data Type Model

In Figure 2.1, it illustrates the complete data type model which complies with design requirement. The design starts with defining 'PLACE' type where it contains 'CITY', 'STATE' and 'COUNTRY' as attributes. Then, the 'ARTIST' type is created with 'PLACE' as an embedded type for 'PLACEBORN' attribute, along with 'NAME', 'DATEBORN' and 'DATEDIED'. Instead using supertype/subtype hierarchy approach, this design intends to populate all the artists in the 'ARTIST' table, so that the 'ACTOR', 'WRITER' and 'CREW' type are created. No actual tables will be created base on those specific artist types. However, these types will all refer an 'ARTIST' object from 'ARTIST' table plus the essential attributes, like 'ROLE', 'STAR' and 'CREDITEORDER' in 'ACTOR' type and 'JOB' in 'CREW' type. The reason for doing this will be discussed later. For accessories, the 'REVIEW' type is created to store the reviews for a specific movie, the 'GENRE' type is to store genre code. In addition, 'CINEMA' and 'ADDRESS' types are essential types. With all basic types created, the collection types can be defined and potentially will act as nested tables. Those collection types are: 'CAST_TABLE' of 'ACTOR' type, 'WRITER_TABLE' of 'WRITER' type, 'TEAM_TABLE' of 'CREW' type, 'REVIEW_TABLE' of 'REVIEW' type and 'GENRE_ARRAY' of 'GENRE' type with at most 3

elements. At this point, the main type, ‘MOVIE’ type is able to be presented as long as all the basic types and collection types are in place. In the ‘MOVIE’ type, only ‘DIRECTOR’ can be directly referred from ‘ARTIST’ table, and all other artist types have embedded structural link as collection with indirect reference from ‘ARTIST’ table. Finally, both ‘SHOWTIME’ and ‘AWARD’ have ‘MOVIE’ as reference from ‘MOVIE’ table. To be noted, all the types with ‘green’ background will have actual table with data inserted.

2.2 Design Alternative 1

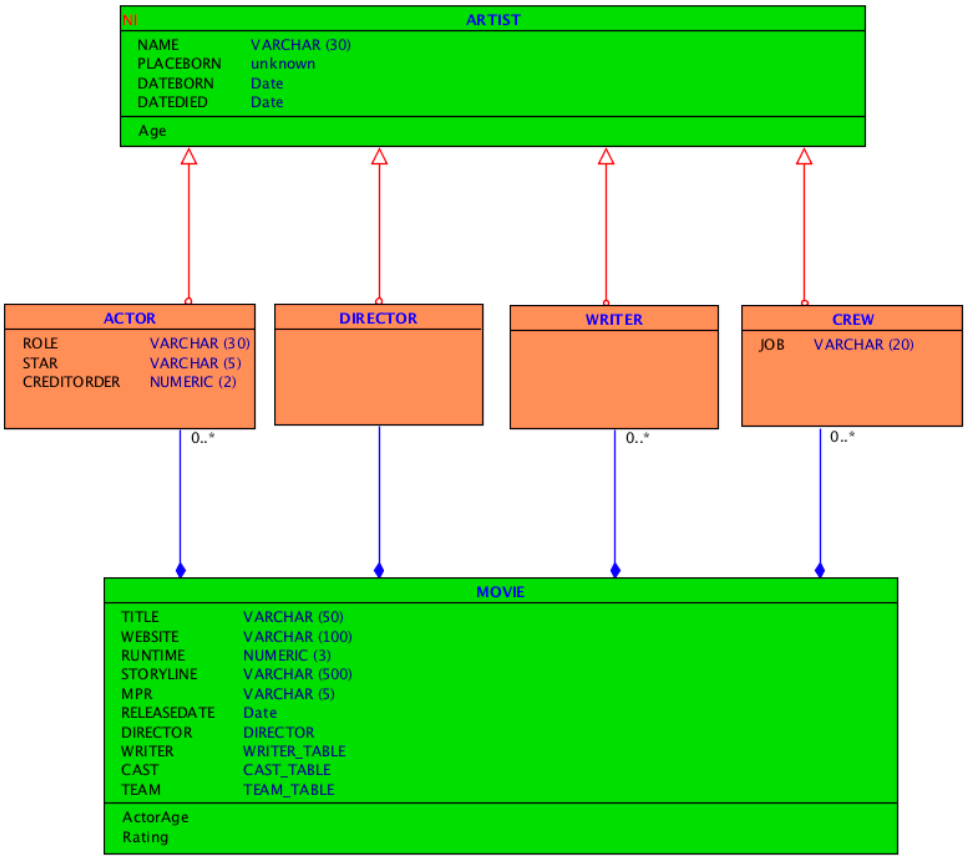

Figure 2.2 – OMDB Artist Design Alternative 1

The Figure 2.2 demonstrates the first thought of this design. By creating the non-instantiable super type ‘ARTIST’ and its subtypes, the subtypes with special attributes needed for movie properties are directly used to created collection type and embedded into ‘MOVIE’ type. The approach can reduce the need to create artist tables. Artists for each movie can be directly inserted into nested table of movie.

However, this approach has a major drawback, which is the normalization issue, those artist types are transitively dependent on ‘MOVIE’, where deleting a movie from ‘MOVIE’ type may permanently delete that related artist/s from the database. This also violates the independence of ‘ARTIST’ which ‘ARTIST’ is also to be referred in the ‘AWARD’ table. To remedy this issue, a second thought is proposed.

2.3 Design Alternative 2

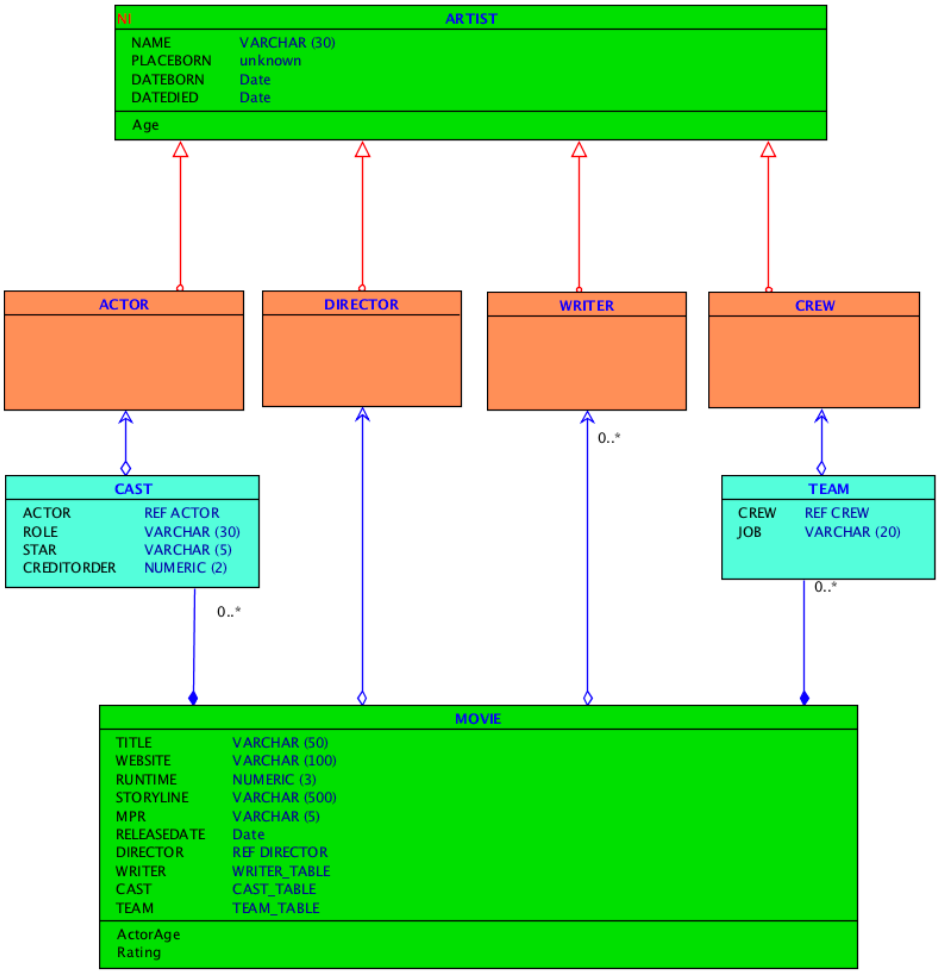

Figure 2.3 – OMDB Artist Design Alternative 2

In Figure 2.3, it shows the remedy to correct the normalization issues and maintain independence of artist types. 'ACTOR', 'DIRECTOR', 'WRITER' and 'CREW' will all inherit basic attributes from 'ARTIST' type and generate four tables base on those types. That means 'ARTIST' table will not be created and user need to insert artist data into corresponding artist tables. In this way, the artist belongs to certain specific type will be stored accordingly and make the structure clear. Then, the critical design part is to insert artists as nested table into each movie without violating the normalization constraint, and the proposed solution is to implement 'extra layer' between artist types and 'MOVIE' type. Such like 'CAST' type refers 'ACTOR' from 'ACTOR' table with other movie-oriented features, and 'CAST' type is acting as an element of 'CAST_TABLE' embedded in 'MOIVE' table. This approach can perfectly solve the update anomalies such like deleting a movie will only delete the reference link stored in the artist section without erasing corresponding artists from the database.

However, this design also has two drawbacks. First one is 'DIRECTOR' and 'WRITER' are lack of middle layers, so their basic information will be stored in the nested tables instead of reference link. Adding those missing layers certainly will solve this problem, but the design seems 'crowded' with plenty of types. Also, the second drawback is back to reality, one artist can have multiple role in one movie or different movies. For example, James Cameron, the director of movie Titanic, is also the writer

and producer of this movie. Therefore, this design may cause repeated data insertion, and become redundant if data volume is large.

Hence, this concern leads to the final design decision.

2.4 Design Alternative – Final Design Decision

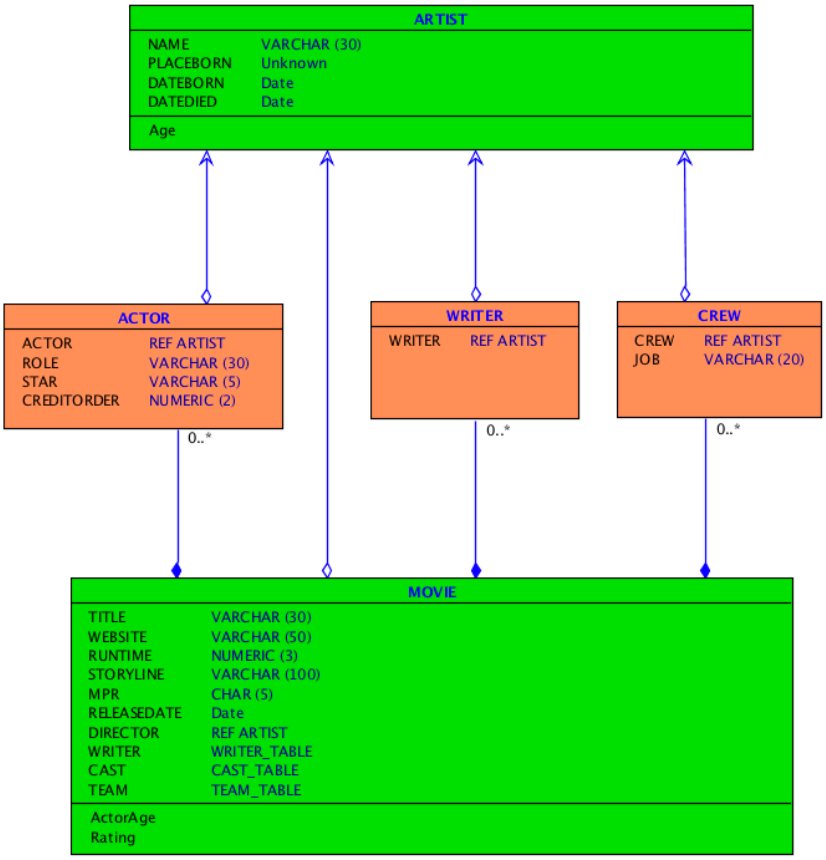

Figure 2.4 – OMDB Artist Final Design Decision

The final design decision is shown in Figure 2.4, where 'ARTIST' type is instantiable and 'ARTIST' table is the only table to store all relevant artist information. And 'ACTOR', 'WRTIER' and 'CREW' will act as 'middle layer' to refer artist from 'ARTIST' table along with their own movie-oriented features. Like the design 'Alternative 2', those middle layers will be the base of collection type and stored as nested table in each movie. This design also eliminates update or delete anomalies along with obeying the normalization of relational design constraints. In addition, this design removes supertype/subtype hierarchy which makes the schema more concise and easy to follow. In general, the final design complies the rule of simplicity and elegance where necessary references are largely used. Certainly, drawbacks can be found, suck like identifying artist's special role in different movies would sometime cause pain, but one can always query about whether an artist is already there before insertion.

3. OMDB Database Implementation

The full implementation of all DDL and INSERT statements can be found in Appendix A & B.

This section will mainly focus on the member function implementations.

3.1 Member Function – AGE ()

```
---- Create ARTIST type
CREATE OR REPLACE TYPE ARTIST_TYPE AS OBJECT(
        NAME      VARCHAR(30),
        PLACEBORN PLACE_TYPE,
        DATEBORN  DATE,
        DATEDIED  DATE,

    MEMBER FUNCTION AGE RETURN CHAR
);
/
-- AGE function
CREATE OR REPLACE TYPE BODY ARTIST_TYPE AS
MEMBER FUNCTION AGE
RETURN CHAR IS
    BEGIN
        IF DATEDIED IS NOT NULL THEN
            RETURN CONCAT(TO_CHAR(TRUNC((DATEDIED - DATEBORN)/365)), ' Passed');
        ELSE
            RETURN TO_CHAR(TRUNC((SYSDATE - DATEBORN)/365));
        END IF;
    END AGE;
END;
/
```

AGE () function is used to calculate the age of an artist from his/her date of born to today. First declare the member function in the 'ARTIST' type, followed by creating the AGE () function. The condition is if the artist is passed away, then output the age at death. So, in the data insertion, if artist is alive, then the 'DATEDIED' attribute is 'NOT NULL', otherwise it will be null. In Oracle database, two dates minus each other will give the number of days in difference, so divide that by 365 days, which is number of days of one year, will give the number in year, which is age. A minor problem is how to let user know if an artist is passed away. So, the solution is to let the return type to be CHAR, and if that artist is passed away, then its result of number of years will be converted into CHAR and concated with 'Passed'. In this way, the user can be well informed. The normal age calculation will use (SYSDATE – DATEBORN)/365 to find out artist's age until today. (**Appendix A – p.20**)

3.2 Member Function – ACTORAGE () & RATING ()

```
---- Create MOVIE_TYPE type
CREATE OR REPLACE TYPE MOVIE_TYPE AS OBJECT(
   ---- Omit other attributes
    MEMBER FUNCTION ACTORAGE(actor_name CHAR) RETURN NUMBER,
    MEMBER FUNCTION RATING RETURN NUMBER
);
/
CREATE OR REPLACE TYPE BODY MOVIE_TYPE AS
MEMBER FUNCTION ACTORAGE(actor_name CHAR)
RETURN NUMBER IS ACTOR_AGE NUMBER(2);
    BEGIN
        SELECT TRUNC((RELEASEDATE - DEREF(C.ACTOR).DATEBORN)/365)
        INTO ACTOR_AGE
        FROM TABLE(CAST) C
        WHERE DEREF(C.ACTOR).NAME = actor_name;
        RETURN ACTOR_AGE;
    END ACTORAGE;

MEMBER FUNCTION RATING
RETURN NUMBER IS RATING_S NUMBER(2,1);
    BEGIN
        SELECT AVG(SCOREPOINT)
        INTO RATING_S
        FROM TABLE(REVIEW);
        RETURN RATING_S;
    END RATING;
END;
/
```

The method behind ACTORAGE (actor_name) function is similar to AGE (), but it is a member function of 'MOVIE', and the only thing to be noticed is actor's name needs to be specified to avoid multiple outputs. The RATING () method is very straight forward, just calculates the average of SCOREPOINT of the nested review table for selected movie. (**Appendix A – p.22-23**)

3.3 Function – GENRE (movie_title) & STARACTOR(movie_title)

```
-- Genre(s) function to combine genres in one column
CREATE OR REPLACE FUNCTION GENRE(movie_title IN MOVIE.TITLE%TYPE)
RETURN CHAR IS GENRE_S CHAR(30);
    BEGIN
        SELECT LISTAGG(GENRE, ', ') WITHIN GROUP (ORDER BY GENRE) INTO GENRE_S
        FROM TABLE(SELECT M.GENRE FROM MOVIE M WHERE M.TITLE = movie_title);
        RETURN GENRE_S;
    END GENRE;
/

-- StarActor to combine star actors in one column
CREATE OR REPLACE FUNCTION STARACTOR(movie_title IN MOVIE.TITLE%TYPE)
RETURN CHAR IS STAR_ACTOR CHAR(100);
    BEGIN
        SELECT LISTAGG(DEREF(ACTOR).NAME, ', ') WITHIN GROUP (ORDER BY CREDITORDER) INTO STAR_ACTOR
        FROM TABLE(SELECT M.CAST FROM MOVIE M WHERE M.TITLE = movie_title)
        WHERE STAR = 'Star';
        RETURN STAR_ACTOR;
    END STARACTOR;
/
```

Two addition functions are added to aid the query, both GENRE () and STARACTOR () will aggregates the genres and names of star actors into one column to make the output clear and concise.

(**Appendix A – p.24**)

4. OMDB Queries

In this section, queries will be tested. For some of the required queries, they will break into two sections to progressively match the requirements in order to demonstrate the building process of query and compare the results to identify the differences. All Query can be copy-and-paste into database username '12680373' to test the correctness of Output. Also, Appendix D shows the screenshots for each query output.

4.1 Query

```
SELECT DEREF(MC.ACTOR).NAME "Actor Name", M.ACTORAGE(DEREF(MC.ACTOR).NAME) "Actor Age",
MC.ROLE "Role", MC.STAR "Star", MC.CREDITORDER "Credit Orders"
FROM MOVIE M, TABLE(M.CAST) MC
WHERE M.TITLE = 'Titanic'
ORDER BY MC.CREDITORDER;
```

4.1 Output

Actor Name	Actor Age	Role	Star	Credit Orders
Leonardo DiCaprio	23	Jack Dawson	Star	1
Kate Winslet	22	Rose Dewitt Bukater	Star	2
Billy Zane	31	Cal Hockley	Star	3
Kathy Bates	49	Molly Brown		4

Query 4.1 outputs the cast of 'Titanic', with ACTORAGE () function calculates the age of that actor at the release year of 'Titanic', order by CREDITORDER. If the actor is non-star actor, then his/her 'STAR' column will be empty (null).

4.2 – 1 Query

```
SELECT M.TITLE "Title", DEREF(M.DIRECTOR).NAME "Director", GENRE(M.TITLE) "Genre(s)"
FROM MOVIE M, TABLE(M.CAST) MC
WHERE DEREF(MC.ACTOR).NAME = 'Cate Blanchett';
```

4.2 – 1 Output

Title	Director	Genre(s)
The Lord of the Rings: The Return of the King	Peter Jackson	Adventure, Drama, Fantasy
Elizabeth: The Golden Age	Shekhar Kapur	Biography, Drama, History
The Aviator	Martin Scorsese	Biography, Drama, History
The Curious Case of Benjamin Button	David Fincher	Drama, Fantasy, Romance

4.2 – 2 Query

```
SELECT M.TITLE "Title", DEREF(M.DIRECTOR).NAME "Director", GENRE(M.TITLE) "Genre(s)"
FROM MOVIE M, TABLE(M.CAST) MC
WHERE DEREF(MC.ACTOR).NAME = 'Cate Blanchett'
AND MC.STAR = 'Star';
```

4.2 – 2 Output

```
Title                                Director              Genre(s)
------------------------------------ --------------------- --------------------------
Elizabeth: The Golden Age            Shekhar Kapur         Biography, Drama, History
The Aviator                          Martin Scorsese       Biography, Drama, History
The Curious Case of Benjamin Button  David Fincher         Drama, Fantasy, Romance
```

Query 4.2 – 1 outputs the movies that Cate Blanchett has acted in, and Query 4.2 – 2 outputs not only acting in it, but also she must be the 'Star' actress.

4.3 Query

```
SELECT TO_CHAR(S.SESSIONDATE, 'DD-MON-YYYY') "Date", TO_CHAR(S.SESSIONDATE, 'Day') "Day",
S.SESSIONTIME "Time", DEREF(S.MOVIE).TITLE "Title",
DEREF(S.MOVIE.DIRECTOR).NAME "Director",
S.MOVIE.RATING() "Rating", STARACTOR(DEREF(MOVIE).TITLE) "Star Actors"
FROM SHOWTIME S
WHERE S.SESSIONDATE = TO_DATE('14/10/2017','DD/MM/YYYY');
```

4.3 Output

```
Date        Day       Time   Title       Director      Rating Star Actors
----------  --------- -----  ----------  ------------  ------ ---------------------------------------------
14-OCT-2017 Saturday  18:00  Titanic     James Cameron    9.7 Leonardo DiCaprio,Kate Winslet,Billy Zane
14-OCT-2017 Saturday  20:00  The Aviator Martin Scorsese    9 Leonardo DiCaprio,Cate Blanchett,Kate Beckinsale
14-OCT-2017 Saturday  14:00  Elizabeth   Shekhar Kapur    7.2 Cate Blanchett,CliveOwen,Geoffrey Rush
```

Query 4.3 outputs the movies on show in Palace Verona this Saturday (14/10/2017), function STARACTOR () aggregates all the star actors of corresponding movie in one column. (The rating is based on the average of a few random selected reviews inserted in the database, it does not reflect the actual rating of that movie in real life.)

4.4 Query

```
SELECT DEREF(S.CINEMA).CINEMANAME "Cinema", DEREF(S.MOVIE).TITLE "Title",
DEREF(S.MOVIE.DIRECTOR).NAME "Director",
TO_CHAR(S.SESSIONDATE, 'DD-MON-YYYY') "Date", S.SESSIONTIME "Time"
FROM SHOWTIME S
WHERE DEREF(S.MOVIE).TITLE = 'Wind River';
```

4.4 Output

```
Cinema             Title        Director           Date           Time
------------------ -----------  ----------------   ------------   --------
Palace Verona      Wind River   Taylor Sheridan    25-AUG-2017    21:00
Reading Rhodes     Wind River   Taylor Sheridan    19-AUG-2017    16:00
Hoyts Chatswood    Wind River   Taylor Sheridan    26-AUG-2017    10:00
```

Query 4.4 outputs the cinemas that showing movie ‘Wind River’ from SHOWTIME table.

4.5 Query

```
SELECT DEREF(DIRECTOR).NAME "Director", DEREF(DIRECTOR).AGE() "Age"
FROM MOVIE M
WHERE DEREF(DIRECTOR).NAME IN
(SELECT DEREF(MC.ACTOR).NAME FROM MOVIE M, TABLE(M.CAST) MC);
```

4.5 Output

```
Director                      Age
---------------------------   -----
Jon Favreau                   51
Stephen Chow                  55
```

Query 4.5 outputs the directors who are also actors, and not necessarily in his own movie. Jon Favreau is the director of movie ‘Iron Man’, he also plays a minor role in that movie, ‘Happy’. But, Stephen Chow is the director of ‘The Mermaid’ and plays no role in it, and he also plays the first star actor in movie ‘A Chinese Odyssey Part One: Pandora"s Box’ with anyone else as director. So, the rule for the query is to implement sub-query to match all the directors with all the actors in every movie’s cast (nested table).

4.6 Query

```
SELECT DEREF(AW.MOVIE.DIRECTOR).NAME "Director", DEREF(AW.MOVIE).TITLE "Title",
TO_CHAR(DEREF(AW.MOVIE).RELEASEDATE,'DD-MON-YYYY') "Release Date"
FROM AWARD AW
WHERE AW.AWARDNAME = 'Academy Award for Best Director';
```

4.6 Output

Director	Title	Release Date
Peter Jackson	The Lord of the Rings: The Return of the King	26-DEC-2003
James Cameron	Titanic	18-DEC-1997
Robert Zemeckis	Forrest Gump	17-NOV-1994

Query 4.6 outputs the directors who received the 'Academy Award for Best Director' from AWARD table.

4.7 – 1 Query

```
SELECT AW.AWARDNAME "Award", AW.AWARDYEAR "Year",
DEREF(AW.MOVIE).TITLE "Title", DEREF(AW.ARTIST).NAME "Name"
FROM AWARD AW;
```

4.7 – 1 Output

Award	Year	Title	Name
Academy Award for Best Director	1997	Titanic	James Cameron
Academy Award for Best Picture	1997	Titanic	James Cameron
Academy Award for Best Director	2003	The Lord of the Rings: The Return of the King	Peter Jackson
Academy Award for Best Director	1994	Forrest Gump	Robert Zemeckis
Academy Award for Best Actor	1994	Forrest Gump	Tom Hanks

4.7 – 2 Query

```
SELECT DISTINCT FIRST_VALUE(DEREF(AW.MOVIE).TITLE)OVER(PARTITION BY AW.MOVIE) "Title",
DEREF(AW.MOVIE.DIRECTOR).NAME "Director", AW.MOVIE.RATING() "Rating"
FROM AWARD AW
WHERE AW.MOVIE IN
(SELECT AW.MOVIE FROM AWARD AW GROUP BY AW.MOVIE HAVING COUNT(*) > 1);
```

4.7 – 2 Output

```
Title                                Director                          Rating
---------------------------------- ------------------------------ ----------
Titanic                            James Cameron                        9.7
Forrest Gump                       Robert Zemeckis                      9.2
```

Query 4.7 – 1 outputs all the award in the AWARD table, only movie Titanic and Forrest Gump received more than one reward (HAVING COUNT(*) > 1). So, Query 4.7 – 2 shows the result. And 'DISTINCT FIRST_VALUE' is used to eliminate duplicate row.

4.8 Query

```
SELECT DEREF(S.MOVIE).TITLE "Title", DEREF(S.MOVIE).RATING() "Rating",
DEREF(S.CINEMA).CINEMANAME "Cinema",
S.SESSIONDATE "Date", S.SESSIONTIME "Time"
FROM SHOWTIME S, TABLE(S.MOVIE.GENRE) SMG
WHERE SMG.GENRE = 'Comedy'
AND S.MOVIE.RATING() > 4;
```

4.8 Output

```
Title              Rating    Cinema              Date        Time
-----------------  --------  ------------------ ----------  ------
Forrest Gump        9.2      Reading Rhodes      21/OCT/17   18:00
The Mermaid         5.6      Hoyts Chatswood     20/OCT/17   20:00
Forrest Gump        9.2      Reading Rhodes      22/OCT/17   14:00
The Mermaid         5.6      Palace Verona       22/OCT/17   20:00
```

Query 4.8 outputs the showing movies which genre is 'Comedy' and rating is greater than 4.

(Some of the movies in database has not been implemented review ratings.)

4.9 - 1 Query

```
SELECT M.TITLE, DEREF(M.DIRECTOR).NAME "Director"
FROM MOVIE M
WHERE M.STORYLINE LIKE '%satire%';
```

4.9 - 1 Output

```
TITLE                                              Director
-------------------------------------------------- ------------------------------
Man of the Year                                    Barry Levinson
The Country Bears                                  Peter Hastings
Drop Squad                                         David C. Johnson
The Fool                                           Christine Edzard
```

4.9 - 2 Query

```
SELECT M.TITLE, DEREF(M.DIRECTOR).NAME "Director"
FROM MOVIE M
WHERE M.STORYLINE LIKE '%satire%'
AND M.TITLE NOT IN
(SELECT M.TITLE FROM MOVIE M, TABLE(M.GENRE) MG
 WHERE MG.GENRE = 'Comedy');
```

4.9 - 2 Output

```
TITLE                                              Director
-------------------------------------------------- ------------------------------
The Fool                                           Christine Edzard
Drop Squad                                         David C. Johnson
```

Usually, movies with ‘satire’ in their description is always a ‘Comedy’ movie. Query 4.9 – 1 outputs the movie with ‘satire’ in their storyline regardless of its genre. Query 4.9 – 2 requires sub-query to eliminates the movie with ‘Comedy’ in their genre, and only ‘The Fool’ and ‘Drop Squad’ satisfies and their genre is both ‘Drama’.

4.10 Query

```
SELECT TO_CHAR(S.SESSIONDATE, 'DD-MON-YYYY') "Date", TO_CHAR(S.SESSIONDATE, 'Day') "Day",
DEREF(S.MOVIE).TITLE "Title", DEREF(S.MOVIE.DIRECTOR).NAME "Director",
S.MOVIE.RATING() "Highest Rating"
FROM SHOWTIME S
WHERE S.SESSIONDATE = TO_DATE('22/10/2017','DD/MM/YYYY')
AND S.MOVIE.RATING() >= ALL
(SELECT S.MOVIE.RATING() "Highest Rating"
 FROM SHOWTIME S
 WHERE S.SESSIONDATE = TO_DATE('22/10/2017','DD/MM/YYYY'));
```

4.10 Output

```
Date            Day         Title      Director          Highest Rating
--------------  ----------  ---------  --------------    -----------------
22-OCT-2017     Sunday      Titanic    James Cameron     9.7
```

To find the highest rating, sub query and '>= ALL' operator is used to compare all the ratings with others in SHOWTIME table, and '>=' ensures the one with highest rating won't rule out itself.

5. Design Modification

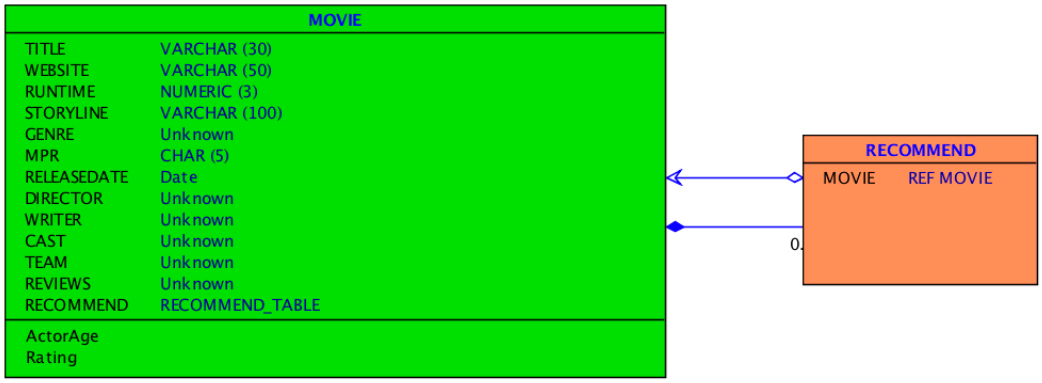

Figure 5.1 – OMDB Design Modification: Recommendation

In this section, it is required to implement modifications to accommodate recommendations of a movie. The remedy is to create a new type called 'RECOMMEND' and the only attribute it will have is 'MOVIE', and it is also a reference from 'MOVIE' table. Then, create a collection type of 'RECOMMEND' type and alter the attribute of 'MOVIE' type and add attribute RECOMMEND into it with data type of 'RECOMMEND_TABLE'. The tricky part is the use of 'ALTER', since RECOMMEND attribute needs to have the 'MOVIE' type reference, so it is impossible to insert 'RECOMMEND' in 'MOVIE' type in the first place, therefore, only 'ALTER' later is reasonable. The detail of implementation of DDL and INSERT statements are documented in **Appendix C – p.44 - 46**.

The required Query and its output are:

4.10 Query

```
SELECT DEREF(MR.MOVIE).TITLE "Title", DEREF(MR.MOVIE.DIRECTOR).NAME "Director",
DEREF(MR.MOVIE).WEBSITE "Website URL"
FROM MOVIE M, TABLE(M.RECOMMEND) MR
WHERE M.TITLE = 'Fever';
```

4.10 Output

```
Title      Director          Website URL
---------- ----------------  ------------------------------------------------------------
Judwa2     David Dhawan      http://www.imdb.com/title/tt5456546/?ref_=india_t_hifull
Dangal     Nitesh Tiwari     http://www.imdb.com/title/tt5074352/?ref_=india_t_hifull
Newton     Amit Masurkar     http://www.imdb.com/title/tt6484982/?ref_=india_t_hifull
```

6. Conclusion

During the implementation of artists into the nested table of MOVIE, it appears often it is unsure the role of certain artist because all of them are mixed in on ARTIST table. Therefore, design alternative 2 in Section 2 can be tried out. Even with data redundancy, it would somehow minimize the workload of database admin due to the complexity of concept integration. So, it is a trade-off between the simplicity of design and reduction of workload. From a designer's prospective, if the volume of data is large with least search query request, then the design proposed in this report is adequate. Otherwise, if the design wishes to quickly adapt into deployment, and shall handle large number of queries, then solution in alternative 2 is more appropriate since the artist references are evenly distributed in four tables, and this schema certainly will accelerate the query speed. Therefore, for commercial or public use, alternative 2 more attractive than the simplicity design proposed in this report.

Appendix A – DDL Script

```
-- University of Technology, Sydney
-- Faculty of Engineering and Information Technology
-- Object-Relational Databases 31075/42901
-- Spring, 2017

-- Database Design and Implementation Assignment
-- Online Movie Database

-- Team:
-- Tianpeng GOU, 12680373 & Manchun CHENG, 12646269

-----------------------------------------------------------------
---- Drops ----
DROP TABLE AWARD;
DROP TABLE SHOWTIME;
DROP TABLE MOVIE;
DROP TABLE ARTIST;
DROP TABLE CINEMA;

---------------------------------------
-- Drop Functions --
DROP FUNCTION RATING;
DROP FUNCTION ACTORAGE;
DROP FUNCTION GENRE;
DROP FUNCTION STARACTOR;
-- Drop Main Types --
DROP TYPE AWARD_TYPE       FORCE;
DROP TYPE SHOWTIME_TYPE    FORCE;
DROP TYPE MOVIE_TYPE       FORCE;
-- Drop Table Types --
DROP TYPE TABLE_RECOMMEND  FORCE;
DROP TYPE TABLE_CAST       FORCE;
DROP TYPE TABLE_TEAM       FORCE;
DROP TYPE TABLE_WRITER     FORCE;
DROP TYPE TABLE_REVIEW     FORCE;
```

```
-- Drop Arrays --
DROP TYPE GENRE_ARRAY      FORCE;
-- Drop Basic Types --
DROP TYPE RECOMMEND_TYPE   FORCE;
DROP TYPE ACTOR_TYPE       FORCE;
DROP TYPE CREW_TYPE        FORCE;
DROP TYPE WRITER_TYPE      FORCE;
DROP TYPE ARTIST_TYPE      FORCE;
DROP TYPE REVIEW_TYPE      FORCE;
DROP TYPE CINEMA_TYPE      FORCE;
DROP TYPE GENRE_TYPE       FORCE;
DROP TYPE PLACE_TYPE       FORCE;
DROP TYPE ADDRESS_TYPE     FORCE;


-------- Type Definition --------

---- Create PLACE type
CREATE OR REPLACE TYPE PLACE_TYPE AS OBJECT(
       CITY      VARCHAR(20),
       STATE     VARCHAR(20),
       COUNTRY   VARCHAR(20)
);
/
---- Create ADDRESS type
CREATE OR REPLACE TYPE ADDRESS_TYPE AS OBJECT(
       STREETNUMBER     VARCHAR(5),
       STREETNAME       VARCHAR(20),
       POSTCODE         VARCHAR(5),
       PLACE            PLACE_TYPE
);
/
---- Create GENRE type
CREATE OR REPLACE TYPE GENRE_TYPE AS OBJECT(
    GENRE    VARCHAR(10)
);
/
---- Create ARTIST type
```

```
CREATE OR REPLACE TYPE ARTIST_TYPE AS OBJECT(
       NAME      VARCHAR(30),
       PLACEBORN PLACE_TYPE,
       DATEBORN  DATE,
       DATEDIED  DATE,

    MEMBER FUNCTION AGE RETURN CHAR
);
/
-- AGE function
CREATE OR REPLACE TYPE BODY ARTIST_TYPE AS
MEMBER FUNCTION AGE
RETURN CHAR IS
    BEGIN
        IF DATEDIED IS NOT NULL THEN
            RETURN CONCAT(TO_CHAR(TRUNC((DATEDIED - DATEBORN)/365)), ' Passed');
        ELSE
            RETURN TO_CHAR(TRUNC((SYSDATE - DATEBORN)/365));
        END IF;
    END AGE;
END;
/

---- Create ACTOR, CREW, and  WRITER types refer to ARTIST_TYPE
CREATE OR REPLACE TYPE ACTOR_TYPE AS OBJECT(
       ACTOR        REF ARTIST_TYPE,
       ROLE         VARCHAR(30),
       STAR         VARCHAR(5),
       CREDITORDER  NUMBER(2)
);
/

CREATE OR REPLACE TYPE CREW_TYPE AS OBJECT(
       CREW    REF ARTIST_TYPE,
       JOB     VARCHAR(20)
);
/
CREATE OR REPLACE TYPE WRITER_TYPE AS OBJECT(
```

```
       WRITER  REF ARTIST_TYPE
);

/
---- Create REVIEW type
CREATE OR REPLACE TYPE REVIEW_TYPE AS OBJECT(
       REVIEWER      VARCHAR(20),
       REVIEWTEXT    VARCHAR(200),
       SCOREPOINT    NUMBER(2,1),  -- 0.0 - 9.9
       REVIEWDATE    DATE

);
/
---- Create CINEMA type
CREATE OR REPLACE TYPE CINEMA_TYPE AS OBJECT(
       CINEMANAME    VARCHAR(20),
       ADDRESS       ADDRESS_TYPE,
       PHONE         VARCHAR(15)
);
/

-------- ARRAY & COLLECTION DEFINITION --------

---- Create GENRE_ARRAY array
CREATE OR REPLACE TYPE ARRAY_GENRE  AS VARRAY(3) OF GENRE_TYPE;
/
---- Create WRITER_TABLE collection
CREATE OR REPLACE TYPE TABLE_WRITER AS TABLE OF WRITER_TYPE;
/
---- Create CAST_TABLE collection
CREATE OR REPLACE TYPE TABLE_CAST   AS TABLE OF ACTOR_TYPE;
/
---- Create TEAM_TABLE collection
CREATE OR REPLACE TYPE TABLE_TEAM   AS TABLE OF CREW_TYPE;
/
---- Create REVIEW_TABLE collection
CREATE OR REPLACE TYPE TABLE_REVIEW AS TABLE OF REVIEW_TYPE;
/
```

```
-------- Main Types --------

---- Create MOVIE_TYPE type
CREATE OR REPLACE TYPE MOVIE_TYPE AS OBJECT(
       TITLE          VARCHAR(50),
       WEBSITE        VARCHAR(100),
       RUNTIME        NUMBER(3),
       STORYLINE      VARCHAR(500),
       GENRE          ARRAY_GENRE,
       MPR            VARCHAR(5),
       RELEASEDATE    DATE,
       DIRECTOR       REF ARTIST_TYPE,
       WRITER         TABLE_WRITER,
       CAST           TABLE_CAST,
       TEAM           TABLE_TEAM,
    REVIEW         TABLE_REVIEW,

    MEMBER FUNCTION ACTORAGE(actor_name CHAR) RETURN NUMBER,
    MEMBER FUNCTION RATING RETURN NUMBER
);
/
CREATE OR REPLACE TYPE BODY MOVIE_TYPE AS
MEMBER FUNCTION ACTORAGE(actor_name CHAR)
RETURN NUMBER IS ACTOR_AGE NUMBER(2);
    BEGIN
        SELECT TRUNC((RELEASEDATE - DEREF(C.ACTOR).DATEBORN)/365)
        INTO ACTOR_AGE
        FROM TABLE(CAST) C
        WHERE DEREF(C.ACTOR).NAME = actor_name;
        RETURN ACTOR_AGE;
    END ACTORAGE;

MEMBER FUNCTION RATING
RETURN NUMBER IS RATING_S NUMBER(2,1);
    BEGIN
        SELECT AVG(SCOREPOINT)
```

```
        INTO RATING_S
        FROM TABLE(REVIEW);
        RETURN RATING_S;
    END RATING;
END;
/

---- Create SHOWTIME_TYPE
CREATE OR REPLACE TYPE SHOWTIME_TYPE AS OBJECT(
       SESSIONDATE     DATE,
       SESSIONTIME     VARCHAR(20),
       MOVIE           REF MOVIE_TYPE,
       CINEMA          REF CINEMA_TYPE
);
/
---- Create AWARD_TYPE
CREATE OR REPLACE TYPE AWARD_TYPE AS OBJECT(
       AWARDNAME    VARCHAR(50),
       AWARDYEAR    VARCHAR(4),
       MOVIE        REF MOVIE_TYPE,
    ARTIST       REF ARTIST_TYPE
);
/

-------------------------------------------------------------------------------------
-------- Create Tables --------

---- Create ARTIST table
CREATE TABLE ARTIST OF ARTIST_TYPE OBJECT IDENTIFIER IS SYSTEM GENERATED;
/

---- Create CINEMA table
CREATE TABLE CINEMA OF CINEMA_TYPE OBJECT IDENTIFIER IS SYSTEM GENERATED;
/

---- Create MOVIE table
CREATE TABLE MOVIE OF MOVIE_TYPE OBJECT IDENTIFIER IS SYSTEM GENERATED
NESTED TABLE WRITER  STORE AS NT_WRITER
```

```
NESTED TABLE CAST    STORE AS NT_CAST
NESTED TABLE TEAM    STORE AS NT_TEAM
NESTED TABLE REVIEW  STORE AS NT_REVIEW;
/

-- Alter MOVIE's Scope

ALTER TABLE MOVIE ADD (SCOPE FOR (DIRECTOR) IS ARTIST);
/
ALTER TABLE NT_WRITER ADD (SCOPE FOR (WRITER) IS ARTIST);
/
ALTER TABLE NT_CAST   ADD (SCOPE FOR (ACTOR)  IS ARTIST);
/
ALTER TABLE NT_TEAM   ADD (SCOPE FOR (CREW)   IS ARTIST);
/



-- Genre(s) function to combine genres in one column
CREATE OR REPLACE FUNCTION GENRE(movie_title IN MOVIE.TITLE%TYPE)
RETURN CHAR IS GENRE_S CHAR(30);
    BEGIN
        SELECT LISTAGG(GENRE, ', ') WITHIN GROUP (ORDER BY GENRE) INTO GENRE_S
        FROM TABLE(SELECT M.GENRE FROM MOVIE M WHERE M.TITLE = movie_title);
        RETURN GENRE_S;
    END GENRE;
/

-- StarActor to combin star actors in one column
CREATE OR REPLACE FUNCTION STARACTOR(movie_title IN MOVIE.TITLE%TYPE)
RETURN CHAR IS STAR_ACTOR CHAR(100);
    BEGIN
        SELECT LISTAGG(DEREF(ACTOR).NAME, ', ') WITHIN GROUP (ORDER BY CREDITORDER) INTO
STAR_ACTOR
        FROM TABLE(SELECT M.CAST FROM MOVIE M WHERE M.TITLE = movie_title)
        WHERE STAR = 'Star';
        RETURN STAR_ACTOR;
    END STARACTOR;
/
```

```
---- Create SHOWTIME table
CREATE TABLE SHOWTIME OF SHOWTIME_TYPE OBJECT IDENTIFIER IS SYSTEM GENERATED;
/
-- Alter SHOWTIME's Scope
ALTER TABLE SHOWTIME ADD (SCOPE FOR (MOVIE)  IS MOVIE);
/
ALTER TABLE SHOWTIME ADD (SCOPE FOR (CINEMA) IS CINEMA);
/


---- Create AWARD table
CREATE TABLE AWARD OF AWARD_TYPE OBJECT IDENTIFIER IS SYSTEM GENERATED;
/
-- Alter AWARD's Scope
ALTER TABLE AWARD ADD (SCOPE FOR (MOVIE)  IS MOVIE);
/
ALTER TABLE AWARD ADD (SCOPE FOR (ARTIST) IS ARTIST);
/
```

Appendix B – Insert & Update Script

```
-------- 5.1 Start --------


---- Insert Artist for Titanic

INSERT INTO ARTIST VALUES('James Cameron', PLACE_TYPE('Kapuskasing','Ontario','Canada'),
TO_DATE('16/08/1954', 'DD/MM/YYYY'), NULL);/ -- director,writer,producer

INSERT INTO ARTIST VALUES('Leonardo DiCaprio', PLACE_TYPE('Los Angeles','California','USA'),
TO_DATE('11/11/1974', 'DD/MM/YYYY'), NULL);/ -- star

INSERT INTO ARTIST VALUES('Kate Winslet', PLACE_TYPE('Berkshire','England','UK'), TO_DATE('05/10/1975',
'DD/MM/YYYY'), NULL);/  -- star

INSERT INTO ARTIST VALUES('Billy Zane', PLACE_TYPE('Chicago','Illinois','USA'), TO_DATE('24/02/1966',
'DD/MM/YYYY'), NULL);/ -- star

INSERT INTO ARTIST VALUES('Kathy Bates', PLACE_TYPE('Memphis','Tennessee','USA'), TO_DATE('28/06/1948',
'DD/MM/YYYY'), NULL);/ -- non-star


--Test Passed

--INSERT INTO ARTIST VALUES('Ha Ha', PLACE_TYPE('Chicago', 'Illinois', 'USA'), TO_DATE('24/02/1966',
'DD/MM/YYYY'), TO_DATE('24/02/1986', 'DD/MM/YYYY'));

--SELECT A.NAME, A.AGE() FROM ARTIST A;/

--SELECT A.NAME, A.PLACEBORN.CITY, A.PLACEBORN.STATE, A.PLACEBORN.COUNTRY FROM ARTIST A WHERE A.NAME =
'James Cameron';/


---- Insert Movie for Titanic

INSERT INTO MOVIE(TITLE,WEBSITE,RUNTIME,STORYLINE,GENRE,MPR,RELEASEDATE)
VALUES('Titanic','http://www.imdb.com/title/tt0120338/?ref_=fn_al_tt_1',

            194,'A seventeen-year-old aristocrat falls in love with a kind but poor artist aboard the
luxurious, ill-fated R.M.S. Titanic.',

            ARRAY_GENRE(GENRE_TYPE('Drama'), GENRE_TYPE('Romance')),'M',TO_DATE('18/12/1997',
'DD/MM/YYYY'));/

-- Test Genre

--SELECT M.TITLE, GENRE(M.TITLE) "Genre(s)" FROM MOVIE M WHERE M.TITLE = 'Titanic';


--- Update Director

UPDATE MOVIE SET DIRECTOR = (SELECT REF(A) FROM ARTIST A WHERE A.NAME = 'James Cameron')

WHERE MOVIE.TITLE = 'Titanic';/

-- Test Director

--SELECT M.TITLE, DEREF(M.DIRECTOR).NAME FROM MOVIE M WHERE M.TITLE = 'Titanic';


--- Update Writer

UPDATE MOVIE SET WRITER = TABLE_WRITER() WHERE MOVIE.TITLE = 'Titanic';/

INSERT INTO TABLE(SELECT M.WRITER FROM MOVIE M WHERE M.TITLE = 'Titanic')

VALUE((SELECT REF(A) FROM ARTIST A WHERE A.NAME = 'James Cameron'));/

-- Test Writer
```

```
--SELECT DEREF(MW.WRITER).NAME FROM MOVIE M, TABLE(M.WRITER)MW WHERE M.TITLE = 'Titanic';


--- Update Cast
UPDATE MOVIE SET CAST = TABLE_CAST() WHERE MOVIE.TITLE = 'Titanic';/

INSERT INTO TABLE(SELECT M.CAST FROM MOVIE M WHERE M.TITLE = 'Titanic')
VALUES((SELECT REF(A) FROM ARTIST A WHERE A.NAME = 'Leonardo DiCaprio'),'Jack Dawson','Star',1);
/
INSERT INTO TABLE(SELECT M.CAST FROM MOVIE M WHERE M.TITLE = 'Titanic')
VALUES((SELECT REF(A) FROM ARTIST A WHERE A.NAME = 'Kate Winslet'),'Rose Dewitt Bukater','Star',2);
/
INSERT INTO TABLE(SELECT M.CAST FROM MOVIE M WHERE M.TITLE = 'Titanic')
VALUES((SELECT REF(A) FROM ARTIST A WHERE A.NAME = 'Billy Zane'),'Cal Hockley','Star',3);
/
INSERT INTO TABLE(SELECT M.CAST FROM MOVIE M WHERE M.TITLE = 'Titanic')
VALUES((SELECT REF(A) FROM ARTIST A WHERE A.NAME = 'Kathy Bates'),'Molly Brown',NULL,4);
/


--## 5.1 Query
SELECT DEREF(MC.ACTOR).NAME "Actor Name", M.ACTORAGE(DEREF(MC.ACTOR).NAME) "Actor Age",
MC.ROLE "Role", MC.STAR "Star", MC.CREDITORDER "Credit Orders"
FROM MOVIE M, TABLE(M.CAST) MC
WHERE M.TITLE = 'Titanic'
ORDER BY MC.CREDITORDER;


-------- 5.1 End --------




-------- 5.2 Start -------


---- Insert Artist for The Curious Case of Benjamin Button
INSERT INTO ARTIST VALUES('David Fincher', PLACE_TYPE('Denver','Colorado','USA'), TO_DATE('28/08/1962',
'DD/MM/YYYY'), NULL);/ -- director
INSERT INTO ARTIST VALUES('Eric Roth', PLACE_TYPE('New York City','New York','USA'),
TO_DATE('22/03/1945', 'DD/MM/YYYY'), NULL);/ -- writer
INSERT INTO ARTIST VALUES('Robin Swicord', PLACE_TYPE('Columbia','South Carolina','USA'),
TO_DATE('23/10/1945', 'DD/MM/YYYY'), NULL);/ -- writer
INSERT INTO ARTIST VALUES('Scott Fitzgerald', PLACE_TYPE('St. Paul','Minnesota','USA'),
TO_DATE('24/09/1896', 'DD/MM/YYYY'), TO_DATE('21/12/1940', 'DD/MM/YYYY'));/ -- writer
INSERT INTO ARTIST VALUES('Brad Pitt', PLACE_TYPE('Shawnee','Oklahoma','USA'), TO_DATE('18/12/1963',
'DD/MM/YYYY'), NULL);/ -- star
```

```
INSERT INTO ARTIST VALUES('Cate Blanchett', PLACE_TYPE('Melbourne','Victoria','Australia'),
TO_DATE('14/05/1969', 'DD/MM/YYYY'), NULL);/ -- star

INSERT INTO ARTIST VALUES('Tilda Swinton', PLACE_TYPE('London','England','UK'), TO_DATE('05/11/1960',
'DD/MM/YYYY'), NULL);/ -- star

INSERT INTO ARTIST VALUES('Taraji Henson', PLACE_TYPE( 'Washington','Columbia','USA'),
TO_DATE('11/09/1970', 'DD/MM/YYYY'), NULL);/ -- non-star


--SELECT A.NAME,  A.PLACEBORN.CITY, A.PLACEBORN.STATE, A.PLACEBORN.COUNTRY FROM ARTIST A;


---- Insert Movie for The Curious Case of Benjamin Button

INSERT INTO MOVIE(TITLE,WEBSITE,RUNTIME,STORYLINE,GENRE,MPR,RELEASEDATE) VALUES('The Curious Case of
Benjamin Button','http://www.imdb.com/title/tt0421715/?ref_=ttfc_fc_tt',

            166,'Tells the story of Benjamin Button, a man who starts aging backwards with bizarre
consequences.',

            ARRAY_GENRE(GENRE_TYPE('Drama'), GENRE_TYPE('Fantasy'),
GENRE_TYPE('Romance')),'M',TO_DATE('26/12/2008', 'DD/MM/YYYY'));/


--- Update Director

UPDATE MOVIE SET DIRECTOR = (SELECT REF(A) FROM ARTIST A WHERE A.NAME = 'David Fincher')

WHERE MOVIE.TITLE = 'The Curious Case of Benjamin Button';/


--- Update Writer

UPDATE MOVIE SET WRITER = TABLE_WRITER() WHERE MOVIE.TITLE = 'The Curious Case of Benjamin Button';/


INSERT INTO TABLE(SELECT M.WRITER FROM MOVIE M WHERE M.TITLE = 'The Curious Case of Benjamin Button')

VALUE((SELECT REF(A) FROM ARTIST A WHERE A.NAME = 'Eric Roth'));

/

INSERT INTO TABLE(SELECT M.WRITER FROM MOVIE M WHERE M.TITLE = 'The Curious Case of Benjamin Button')

VALUE((SELECT REF(A) FROM ARTIST A WHERE A.NAME = 'Robin Swicord'));

/

INSERT INTO TABLE(SELECT M.WRITER FROM MOVIE M WHERE M.TITLE = 'The Curious Case of Benjamin Button')

VALUE((SELECT REF(A) FROM ARTIST A WHERE A.NAME = 'Scott Fitzgerald'));

/


--- Update Cast

UPDATE MOVIE SET CAST = TABLE_CAST() WHERE MOVIE.TITLE = 'The Curious Case of Benjamin Button';/


INSERT INTO TABLE(SELECT M.CAST FROM MOVIE M WHERE M.TITLE = 'The Curious Case of Benjamin Button')

VALUES((SELECT REF(A) FROM ARTIST A WHERE A.NAME = 'Brad Pitt'),'Benjamin Button','Star',1);

/

INSERT INTO TABLE(SELECT M.CAST FROM MOVIE M WHERE M.TITLE = 'The Curious Case of Benjamin Button')

VALUES((SELECT REF(A) FROM ARTIST A WHERE A.NAME = 'Cate Blanchett'),'Daisy','Star',2);

/
```

```
INSERT INTO TABLE(SELECT M.CAST FROM MOVIE M WHERE M.TITLE = 'The Curious Case of Benjamin Button')

VALUES((SELECT REF(A) FROM ARTIST A WHERE A.NAME = 'Tilda Swinton'),'Elizabeth Abbott','Star',3);

/

INSERT INTO TABLE(SELECT M.CAST FROM MOVIE M WHERE M.TITLE = 'The Curious Case of Benjamin Button')

VALUES((SELECT REF(A) FROM ARTIST A WHERE A.NAME = 'Taraji Henson'),'Queenie',NULL,4);

/


-- Test Cast

--SELECT DEREF(MC.ACTOR).NAME "Actor Name", ACTORAGE(M.TITLE, DEREF(MC.ACTOR).NAME) "Actor Age",
MC.ROLE, MC.STAR, MC.CREDITORDER

--FROM MOVIE M, TABLE(M.CAST) MC

--WHERE M.TITLE = 'The Curious Case of Benjamin Button';



---- Insert Artist for The Aviator

INSERT INTO ARTIST VALUES('Martin Scorsese', PLACE_TYPE('New York City','New York','USA'),
TO_DATE('17/11/1942', 'DD/MM/YYYY'), NULL);/ -- director

-- John Logan (writer)

-- Leonardo DiCaprio -- star

-- Cate Blanchett -- star

INSERT INTO ARTIST VALUES('Kate Beckinsale', PLACE_TYPE('London','England','UK'), TO_DATE('26/07/1973',
'DD/MM/YYYY'), NULL);/ -- star


---- Insert Movie for The Aviator

INSERT INTO MOVIE(TITLE,WEBSITE,RUNTIME,STORYLINE,GENRE,MPR,RELEASEDATE) VALUES('The
Aviator','http://www.imdb.com/title/tt0338751/?ref_=nm_flmg_act_41',

            170,'A biopic depicting the early years of legendary Director and aviator Howard Hughes ''
career from the late 1920s to the mid 1940s.',

            ARRAY_GENRE(GENRE_TYPE('Drama'), GENRE_TYPE('Biography'),
GENRE_TYPE('History')),'M',TO_DATE('10/02/2005', 'DD/MM/YYYY'));/


--- Update Director

UPDATE MOVIE SET DIRECTOR = (SELECT REF(A) FROM ARTIST A WHERE A.NAME = 'Martin Scorsese')

WHERE MOVIE.TITLE = 'The Aviator';/


--- Update Writer

UPDATE MOVIE SET WRITER = TABLE_WRITER() WHERE MOVIE.TITLE = 'The Aviator';/

--


--- Update Cast

UPDATE MOVIE SET CAST = TABLE_CAST() WHERE MOVIE.TITLE = 'The Aviator';/


INSERT INTO TABLE(SELECT M.CAST FROM MOVIE M WHERE M.TITLE = 'The Aviator')
```

```
VALUES((SELECT REF(A) FROM ARTIST A WHERE A.NAME = 'Leonardo DiCaprio'),'Howard Hughes','Star',1);

/

INSERT INTO TABLE(SELECT M.CAST FROM MOVIE M WHERE M.TITLE = 'The Aviator')

VALUES((SELECT REF(A) FROM ARTIST A WHERE A.NAME = 'Cate Blanchett'),'Katharine Hepburn','Star',2);

/

INSERT INTO TABLE(SELECT M.CAST FROM MOVIE M WHERE M.TITLE = 'The Aviator')

VALUES((SELECT REF(A) FROM ARTIST A WHERE A.NAME = 'Kate Beckinsale'),'Ava Gardner','Star',3);

/


-- Test Cast

--SELECT DEREF(MC.ACTOR).NAME "Actor Name", ACTORAGE(M.TITLE, DEREF(MC.ACTOR).NAME) "Actor Age",
MC.ROLE, MC.STAR, MC.CREDITORDER

--FROM MOVIE M, TABLE(M.CAST) MC

--WHERE M.TITLE = 'The Aviator';


---- Insert Artist for Elizabeth: The Golden Age

INSERT INTO ARTIST VALUES('Shekhar Kapur', PLACE_TYPE('Lahore','Punjab','British India'),
TO_DATE('06/12/1945', 'DD/MM/YYYY'), NULL);/ -- director

-- Cate Blanchett -- star

INSERT INTO ARTIST VALUES('Clive Owen', PLACE_TYPE('Warwickshire','England','UK'),
TO_DATE('03/10/1964', 'DD/MM/YYYY'), NULL);/ -- star

INSERT INTO ARTIST VALUES('Geoffrey Rush', PLACE_TYPE('Toowoomba','Queensland','Australia'),
TO_DATE('06/07/1951', 'DD/MM/YYYY'), NULL);/ -- star


---- Insert Movie for Elizabeth: The Golden Age

INSERT INTO MOVIE(TITLE,WEBSITE,RUNTIME,STORYLINE,GENRE,MPR,RELEASEDATE) VALUES('Elizabeth: The Golden
Age','http://www.imdb.com/title/tt0414055/?ref_=nm_flmg_act_34',

            114,'A mature Queen Elizabeth endures multiple crises late in her reign including court
intrigues,

            an assassination plot, the Spanish Armada, and romantic disappointments.',

            ARRAY_GENRE(GENRE_TYPE('Drama'), GENRE_TYPE('Biography'),
GENRE_TYPE('History')),'M',TO_DATE('15/11/2007', 'DD/MM/YYYY'));/


--- Update Director

UPDATE MOVIE SET DIRECTOR = (SELECT REF(A) FROM ARTIST A WHERE A.NAME = 'Shekhar Kapur')

WHERE MOVIE.TITLE = 'Elizabeth: The Golden Age';/


--- Update Writer

UPDATE MOVIE SET WRITER = TABLE_WRITER() WHERE MOVIE.TITLE = 'Elizabeth: The Golden Age';/

--


--- Update Cast

UPDATE MOVIE SET CAST = TABLE_CAST() WHERE MOVIE.TITLE = 'Elizabeth: The Golden Age';/
```

```
INSERT INTO TABLE(SELECT M.CAST FROM MOVIE M WHERE M.TITLE = 'Elizabeth: The Golden Age')

VALUES((SELECT REF(A) FROM ARTIST A WHERE A.NAME = 'Cate Blanchett'),'Queen Elizabeth I','Star',1);

/

INSERT INTO TABLE(SELECT M.CAST FROM MOVIE M WHERE M.TITLE = 'Elizabeth: The Golden Age')

VALUES((SELECT REF(A) FROM ARTIST A WHERE A.NAME = 'Clive Owen'),'Sir Walter Raleigh','Star',2);

/

INSERT INTO TABLE(SELECT M.CAST FROM MOVIE M WHERE M.TITLE = 'Elizabeth: The Golden Age')

VALUES((SELECT REF(A) FROM ARTIST A WHERE A.NAME = 'Geoffrey Rush'),'Sir Francis Walsingham','Star',3);


-- Test Cast

--SELECT DEREF(MC.ACTOR).NAME "Actor Name", ACTORAGE(M.TITLE, DEREF(MC.ACTOR).NAME) "Actor Age",
MC.ROLE, MC.STAR, MC.CREDITORDER

--FROM MOVIE M, TABLE(M.CAST) MC

--WHERE M.TITLE = 'Elizabeth: The Golden Age';


----- Insert Artist for The Lord of the Rings: The Return of the King

INSERT INTO ARTIST VALUES('Peter Jackson', PLACE_TYPE('Pukerua Bay','North Island','New Zealand'),
TO_DATE('31/10/1961', 'DD/MM/YYYY'), NULL);/ -- director


---- Insert Movie for The Lord of the Rings: The Return of the King

INSERT INTO MOVIE(TITLE,WEBSITE,RUNTIME,STORYLINE,GENRE,MPR,RELEASEDATE) VALUES('The Lord of the Rings:
The Return of the King',

            'http://www.imdb.com/title/tt0167260/?ref_=fn_al_tt_1',

            201,'Gandalf and Aragorn lead the World of Men against Sauron''s

            army to draw his gaze from Frodo and Sam as they approach Mount Doom with the One Ring.',

            ARRAY_GENRE(GENRE_TYPE('Drama'), GENRE_TYPE('Adventure'),
GENRE_TYPE('Fantasy')),'M',TO_DATE('26/12/2003', 'DD/MM/YYYY'));/


--- Update Director

UPDATE MOVIE SET DIRECTOR = (SELECT REF(A) FROM ARTIST A WHERE A.NAME = 'Peter Jackson')

WHERE MOVIE.TITLE = 'The Lord of the Rings: The Return of the King';/



--- Update Cast

UPDATE MOVIE SET CAST = TABLE_CAST() WHERE MOVIE.TITLE = 'The Lord of the Rings: The Return of the
King';/


INSERT INTO TABLE(SELECT M.CAST FROM MOVIE M WHERE M.TITLE = 'The Lord of the Rings: The Return of the
King')

VALUES((SELECT REF(A) FROM ARTIST A WHERE A.NAME = 'Cate Blanchett'),'Galadriel',NULL,7);

/
```

```
--## 5.2Query

SELECT M.TITLE "Title", DEREF(M.DIRECTOR).NAME "Director", GENRE(M.TITLE) "Genre(s)"

FROM MOVIE M, TABLE(M.CAST) MC

WHERE DEREF(MC.ACTOR).NAME = 'Cate Blanchett'

AND MC.STAR = 'Star';


-------- 5.2 End -------



-------- 5.3 Start -------


--- Insert Cinema - Verona

INSERT INTO CINEMA VALUES('Palace Verona', ADDRESS_TYPE('17','Oxford
St','2021',PLACE_TYPE('Paddington','NSW','Australia')),'02-93606099');/


--- Insert Review Ratings for 'Titanic' and 'The Aviator'

UPDATE MOVIE SET REVIEW = TABLE_REVIEW() WHERE TITLE = 'Titanic';/


INSERT INTO TABLE(SELECT M.REVIEW FROM MOVIE M WHERE M.TITLE = 'Titanic')(REVIEWER,SCOREPOINT)

VALUES('sddavis63', 9.0);

/

INSERT INTO TABLE(SELECT M.REVIEW FROM MOVIE M WHERE M.TITLE = 'Titanic')(REVIEWER,SCOREPOINT)

VALUES('Boyo-2', 9.9);

/

INSERT INTO TABLE(SELECT M.REVIEW FROM MOVIE M WHERE M.TITLE = 'Titanic')(REVIEWER,SCOREPOINT)

VALUES('Kristine', 9.9);

/

INSERT INTO TABLE(SELECT M.REVIEW FROM MOVIE M WHERE M.TITLE = 'Titanic')(REVIEWER,SCOREPOINT)

VALUES('crystalc1020', 9.9);

/

INSERT INTO TABLE(SELECT M.REVIEW FROM MOVIE M WHERE M.TITLE = 'Titanic')(REVIEWER,SCOREPOINT)

VALUES('cyndymarks', 9.9);

/

----

UPDATE MOVIE SET REVIEW = TABLE_REVIEW() WHERE TITLE = 'The Aviator';/


INSERT INTO TABLE(SELECT M.REVIEW FROM MOVIE M WHERE M.TITLE = 'The Aviator')(REVIEWER,SCOREPOINT)

VALUES('Rathko', 8.0);

/

INSERT INTO TABLE(SELECT M.REVIEW FROM MOVIE M WHERE M.TITLE = 'The Aviator')(REVIEWER,SCOREPOINT)

VALUES('Mister1045', 9.9);
```

```
/
INSERT INTO TABLE(SELECT M.REVIEW FROM MOVIE M WHERE M.TITLE = 'The Aviator')(REVIEWER,SCOREPOINT)
VALUES('gmorgan-4', 8.0);
/
INSERT INTO TABLE(SELECT M.REVIEW FROM MOVIE M WHERE M.TITLE = 'The Aviator')(REVIEWER,SCOREPOINT)
VALUES('colonel_green', 9.0);
/
INSERT INTO TABLE(SELECT M.REVIEW FROM MOVIE M WHERE M.TITLE = 'The Aviator')(REVIEWER,SCOREPOINT)
VALUES('drplw', 9.9);
/
----
UPDATE MOVIE SET REVIEW = TABLE_REVIEW() WHERE TITLE = 'Elizabeth: The Golden Age';/


INSERT INTO TABLE(SELECT M.REVIEW FROM MOVIE M WHERE M.TITLE = 'Elizabeth: The Golden
Age')(REVIEWER,SCOREPOINT)
VALUES('eastbergholt2002', 8.0);
/
INSERT INTO TABLE(SELECT M.REVIEW FROM MOVIE M WHERE M.TITLE = 'Elizabeth: The Golden
Age')(REVIEWER,SCOREPOINT)
VALUES('Brent Trafton', 7.0);
/
INSERT INTO TABLE(SELECT M.REVIEW FROM MOVIE M WHERE M.TITLE = 'Elizabeth: The Golden
Age')(REVIEWER,SCOREPOINT)
VALUES('MistinParadise', 8.0);
/
INSERT INTO TABLE(SELECT M.REVIEW FROM MOVIE M WHERE M.TITLE = 'Elizabeth: The Golden
Age')(REVIEWER,SCOREPOINT)
VALUES('Harker207', 5.0);
/
INSERT INTO TABLE(SELECT M.REVIEW FROM MOVIE M WHERE M.TITLE = 'Elizabeth: The Golden
Age')(REVIEWER,SCOREPOINT)
VALUES('Righty-Sock', 8.0);
/



---Insert Showtime
INSERT INTO SHOWTIME VALUES(TO_DATE('14/10/2017','DD/MM/YYYY'),'18:00',
(SELECT REF(M) FROM MOVIE M WHERE M.TITLE = 'Titanic'),
(SELECT REF(C) FROM CINEMA C WHERE C.CINEMANAME = 'Palace Verona'));
/
INSERT INTO SHOWTIME VALUES(TO_DATE('14/10/2017','DD/MM/YYYY'),'20:00',
(SELECT REF(M) FROM MOVIE M WHERE M.TITLE = 'The Aviator'),
(SELECT REF(C) FROM CINEMA C WHERE C.CINEMANAME = 'Palace Verona'));
```

```
/

INSERT INTO SHOWTIME VALUES(TO_DATE('14/10/2017','DD/MM/YYYY'),'14:00',

(SELECT REF(M) FROM MOVIE M WHERE M.TITLE = 'Elizabeth: The Golden Age'),

(SELECT REF(C) FROM CINEMA C WHERE C.CINEMANAME = 'Palace Verona'));

/


--## 5.3 Query

SELECT TO_CHAR(S.SESSIONDATE, 'DD-MON-YYYY') "Date", TO_CHAR(S.SESSIONDATE, 'Day') "Day",

S.SESSIONTIME "Time", DEREF(S.MOVIE).TITLE "Title",

DEREF(S.MOVIE.DIRECTOR).NAME "Director",

S.MOVIE.RATING() "Rating", STARACTOR(DEREF(MOVIE).TITLE) "Star Actors"

FROM SHOWTIME S

WHERE S.SESSIONDATE = TO_DATE('14/10/2017','DD/MM/YYYY');


-------- 5.3 End --------



-------- 5.4 Start --------


---- Insert Artist for Wind River

INSERT INTO ARTIST VALUES('Taylor Sheridan', PLACE_TYPE('Cranfills Gap','Texas','United States'),
TO_DATE('21/05/1970', 'DD/MM/YYYY'), NULL);/ -- director


---- Insert Movie for Wind River

INSERT INTO MOVIE(TITLE,WEBSITE,RUNTIME,STORYLINE,GENRE,MPR,RELEASEDATE) VALUES('Wind
River','http://www.imdb.com/title/tt5362988/?ref_=nv_sr_1',

            107,'A veteran tracker with the Fish and Wildlife Service helps to investigate the murder
of a young Native American woman,

            and uses the case as a means of seeking redemption for an earlier act of irresponsibility
which ended in tragedy.',

            ARRAY_GENRE(GENRE_TYPE('Drama'), GENRE_TYPE('Crime'),
GENRE_TYPE('Mystery')),'MA15+',TO_DATE('10/08/2017', 'DD/MM/YYYY'));/


---- Update Director

UPDATE MOVIE SET DIRECTOR = (SELECT REF(A) FROM ARTIST A WHERE A.NAME = 'Taylor Sheridan')

WHERE MOVIE.TITLE = 'Wind River';/



---- Insert Cinemas

INSERT INTO CINEMA VALUES('Reading Rhodes', ADDRESS_TYPE('1','Rider
Boulevard','2138',PLACE_TYPE('Rhodes','NSW','Australia')),'02-97367900');/

INSERT INTO CINEMA VALUES('Hoyts Chatswood', ADDRESS_TYPE('1','Anderson
St','2067',PLACE_TYPE('Chatswood','NSW','Australia')),'02-90033840');/
```

```
---- Insert Showtime for Wind River

INSERT INTO SHOWTIME VALUES(TO_DATE('19/08/2017','DD/MM/YYYY'),'16:00',

(SELECT REF(M) FROM MOVIE M WHERE M.TITLE = 'Wind River'),

(SELECT REF(C) FROM CINEMA C WHERE C.CINEMANAME = 'Reading Rhodes'));

/

INSERT INTO SHOWTIME VALUES(TO_DATE('25/08/2017','DD/MM/YYYY'),'21:00',

(SELECT REF(M) FROM MOVIE M WHERE M.TITLE = 'Wind River'),

(SELECT REF(C) FROM CINEMA C WHERE C.CINEMANAME = 'Palace Verona'));

/

INSERT INTO SHOWTIME VALUES(TO_DATE('26/08/2017','DD/MM/YYYY'),'10:00',

(SELECT REF(M) FROM MOVIE M WHERE M.TITLE = 'Wind River'),

(SELECT REF(C) FROM CINEMA C WHERE C.CINEMANAME = 'Hoyts Chatswood'));

/


--## 5.4 Query

SELECT DEREF(S.CINEMA).CINEMANAME "Cinema", DEREF(S.MOVIE).TITLE "Title",

DEREF(S.MOVIE.DIRECTOR).NAME "Director",

TO_CHAR(S.SESSIONDATE, 'DD-MON-YYYY') "Date", S.SESSIONTIME "Time"

FROM SHOWTIME S

WHERE DEREF(S.MOVIE).TITLE = 'Wind River';


-------- 5.4 End --------


-------- 5.5 Start -------


---- Insert Artist for Iron Man

INSERT INTO ARTIST VALUES('Jon Favreau', PLACE_TYPE('New York City','New York','USA'),
TO_DATE('19/10/1966', 'DD/MM/YYYY'), NULL);/ -- director


---- Insert Movie Iron Man

INSERT INTO MOVIE(TITLE,WEBSITE,RUNTIME,STORYLINE,GENRE,MPR,RELEASEDATE) VALUES('Iron
Man','http://www.imdb.com/title/tt0371746/?ref_=fn_al_tt_1',

            126,'After being held captive in an Afghan cave, billionaire engineer Tony Stark creates a
unique weaponized suit of armor to fight evil.',

            ARRAY_GENRE(GENRE_TYPE('Action'), GENRE_TYPE('Adventure'), GENRE_TYPE('Sci-
Fi')),'M',TO_DATE('01/05/2008', 'DD/MM/YYYY'));/


-- Update Director

UPDATE MOVIE SET DIRECTOR = (SELECT REF(A) FROM ARTIST A WHERE A.NAME = 'Jon Favreau')

WHERE MOVIE.TITLE = 'Iron Man';/
```

```
--- Update Cast

UPDATE MOVIE SET CAST = TABLE_CAST() WHERE MOVIE.TITLE = 'Iron Man';/


INSERT INTO TABLE(SELECT M.CAST FROM MOVIE M WHERE M.TITLE = 'Iron Man')

VALUES((SELECT REF(A) FROM ARTIST A WHERE A.NAME = 'Jon Favreau'),'Happy',NULL,4);

/


--DELETE FROM TABLE(SELECT CAST FROM MOVIE WHERE TITLE = 'Iron Man') MC WHERE DEREF(MC.ACTOR).NAME =
'James Cameron';

--


---- Insert Artist for A Chinese Odyssey Part One: Pandora's Box

INSERT INTO ARTIST VALUES('Jeffrey Lau', PLACE_TYPE('Hong Kong','Hong Kong','China'),
TO_DATE('02/08/1952', 'DD/MM/YYYY'), NULL);/ -- director

INSERT INTO ARTIST VALUES('Stephen Chow', PLACE_TYPE('Hong Kong','Hong Kong','China'),
TO_DATE('22/06/1962', 'DD/MM/YYYY'), NULL);/ -- star


---- Insert Movie for A Chinese Odyssey Part One: Pandora's Box

INSERT INTO MOVIE(TITLE,WEBSITE,RUNTIME,STORYLINE,GENRE,MPR,RELEASEDATE) VALUES('A Chinese Odyssey Part
One: Pandora''s Box',

'http://www.imdb.com/title/tt0112778/?ref_=nm_flmg_act_17',87,'Fantasy adventure of Monkey King',

ARRAY_GENRE(GENRE_TYPE('Action'), GENRE_TYPE('Adventure'),
GENRE_TYPE('Comedy')),'M',TO_DATE('21/01/1995', 'DD/MM/YYYY'));/


-- Update Director

UPDATE MOVIE SET DIRECTOR = (SELECT REF(A) FROM ARTIST A WHERE A.NAME = 'Jeffrey Lau')

WHERE MOVIE.TITLE = 'A Chinese Odyssey Part One: Pandora''s Box';/


--- Update Cast

UPDATE MOVIE SET CAST = TABLE_CAST() WHERE MOVIE.TITLE = 'A Chinese Odyssey Part One: Pandora''s Box';/


INSERT INTO TABLE(SELECT M.CAST FROM MOVIE M WHERE M.TITLE = 'A Chinese Odyssey Part One: Pandora''s
Box')

VALUES((SELECT REF(A) FROM ARTIST A WHERE A.NAME = 'Stephen Chow'),'Monkey King','Star',1);

/


---- Insert Artist for The Mermaid

INSERT INTO ARTIST VALUES('Chao Deng', PLACE_TYPE('Nanchang','Jiangxi','China'), TO_DATE('08/02/1979',
'DD/MM/YYYY'), NULL);/ -- star

INSERT INTO ARTIST VALUES('Show Lo', PLACE_TYPE('Keelung','Taiwan','China'), TO_DATE('30/07/1979',
'DD/MM/YYYY'), NULL);/ -- star




---- Insert Movie for The Mermaid
```

```
INSERT INTO MOVIE(TITLE,WEBSITE,RUNTIME,STORYLINE,GENRE,MPR,RELEASEDATE) VALUES('The Mermaid',

'http://www.imdb.com/title/tt4701660/?ref_=nm_flmg_dr_1',94,'Shan, a mermaid, is sent to assassinate
Xuan,

a developer who threatens the ecosystem of her race, but ends up falling in love with him instead.',

ARRAY_GENRE(GENRE_TYPE('Fantasy'), GENRE_TYPE('Drama'), GENRE_TYPE('Comedy')),'M',TO_DATE('18/02/2016',
'DD/MM/YYYY'));/



-- Update Director

UPDATE MOVIE SET DIRECTOR = (SELECT REF(A) FROM ARTIST A WHERE A.NAME = 'Stephen Chow')

WHERE MOVIE.TITLE = 'The Mermaid';/


--- Update Cast

UPDATE MOVIE SET CAST = TABLE_CAST() WHERE MOVIE.TITLE = 'The Mermaid';/


INSERT INTO TABLE(SELECT M.CAST FROM MOVIE M WHERE M.TITLE = 'The Mermaid')

VALUES((SELECT REF(A) FROM ARTIST A WHERE A.NAME = 'Chao Deng'),'Liu Xuan','Star',1);

/

INSERT INTO TABLE(SELECT M.CAST FROM MOVIE M WHERE M.TITLE = 'The Mermaid')

VALUES((SELECT REF(A) FROM ARTIST A WHERE A.NAME = 'Show Lo'),'Octopus','Star',2);

/


--## 5.5 Query

SELECT DEREF(DIRECTOR).NAME "Director", DEREF(DIRECTOR).AGE() "Age"

FROM MOVIE M

WHERE DEREF(DIRECTOR).NAME IN

(SELECT DEREF(MC.ACTOR).NAME FROM MOVIE M, TABLE(M.CAST) MC);


COMMIT;


--DELETE FROM TABLE(SELECT M.CAST FROM MOVIE M WHERE M.TITLE = 'Titanic') MC WHERE DEREF(MC.ACTOR).NAME
= 'James Cameron';


-------- 5.5 End -------



-------- 5.6 Start -------

/

INSERT INTO AWARD VALUES('Academy Award for Best Director', '1997',

(SELECT REF(M) FROM MOVIE M WHERE M.TITLE = 'Titanic'),

(SELECT REF(A) FROM ARTIST A WHERE A.NAME = 'James Cameron'));

/
```

```
INSERT INTO AWARD VALUES('Academy Award for Best Director', '2003',

(SELECT REF(M) FROM MOVIE M WHERE M.TITLE = 'The Lord of the Rings: The Return of the King'),

(SELECT REF(A) FROM ARTIST A WHERE A.NAME = 'Peter Jackson'));

/


---- Insert Artist for Forrest Gump

INSERT INTO ARTIST VALUES('Robert Zemeckis', PLACE_TYPE('Chicago','Illinois','USA'),
TO_DATE('14/05/1952', 'DD/MM/YYYY'), NULL);/ -- director

INSERT INTO ARTIST VALUES('Tom Hanks', PLACE_TYPE('Concord','California','USA'), TO_DATE('09/07/1956',
'DD/MM/YYYY'), NULL);/ -- star


---- Insert Movie for Forrest Gump

INSERT INTO MOVIE(TITLE,WEBSITE,RUNTIME,STORYLINE,GENRE,MPR,RELEASEDATE) VALUES('Forrest Gump',

'http://www.imdb.com/title/tt0109830/?ref_=nv_sr_1',142,'JFK, LBJ, Vietnam,

Watergate, and other history unfold through the perspective of an Alabama man with an IQ of 75.',

ARRAY_GENRE(GENRE_TYPE('Comedy'), GENRE_TYPE('Drama'), GENRE_TYPE('Romance')),'M',TO_DATE('17/11/1994',
'DD/MM/YYYY'));/


-- Update Director

UPDATE MOVIE SET DIRECTOR = (SELECT REF(A) FROM ARTIST A WHERE A.NAME = 'Robert Zemeckis')

WHERE MOVIE.TITLE = 'Forrest Gump';/


--- Update Cast

UPDATE MOVIE SET CAST = TABLE_CAST() WHERE MOVIE.TITLE = 'Forrest Gump';/


INSERT INTO TABLE(SELECT M.CAST FROM MOVIE M WHERE M.TITLE = 'Forrest Gump')

VALUES((SELECT REF(A) FROM ARTIST A WHERE A.NAME = 'Tom Hanks'),'Forrest Gump','Star',1);

/


INSERT INTO AWARD VALUES('Academy Award for Best Director', '1994',

(SELECT REF(M) FROM MOVIE M WHERE M.TITLE = 'Forrest Gump'),

(SELECT REF(A) FROM ARTIST A WHERE A.NAME = 'Robert Zemeckis'));

/


--## 5.6 Query

SELECT DEREF(AW.MOVIE.DIRECTOR).NAME "Director", DEREF(AW.MOVIE).TITLE "Title",

TO_CHAR(DEREF(AW.MOVIE).RELEASEDATE,'DD-MON-YYYY') "Release Date"

FROM AWARD AW

WHERE AW.AWARDNAME = 'Academy Award for Best Director';


-------- 5.6 End -------
```

```
-------- 5.7 Start --------
/
INSERT INTO AWARD VALUES('Academy Award for Best Actor', '1994',
(SELECT REF(M) FROM MOVIE M WHERE M.TITLE = 'Forrest Gump'),
(SELECT REF(A) FROM ARTIST A WHERE A.NAME = 'Tom Hanks'));
/
INSERT INTO AWARD VALUES('Academy Award for Best Picture', '1997',
(SELECT REF(M) FROM MOVIE M WHERE M.TITLE = 'Titanic'),
(SELECT REF(A) FROM ARTIST A WHERE A.NAME = 'James Cameron'));
/

UPDATE MOVIE SET REVIEW = TABLE_REVIEW() WHERE TITLE = 'Forrest Gump';/

INSERT INTO TABLE(SELECT M.REVIEW FROM MOVIE M WHERE M.TITLE = 'Forrest Gump')(REVIEWER,SCOREPOINT)
VALUES('Zonieboy', 9.9);
/
INSERT INTO TABLE(SELECT M.REVIEW FROM MOVIE M WHERE M.TITLE = 'Forrest Gump')(REVIEWER,SCOREPOINT)
VALUES('inspectors71', 9.0);
/
INSERT INTO TABLE(SELECT M.REVIEW FROM MOVIE M WHERE M.TITLE = 'Forrest Gump')(REVIEWER,SCOREPOINT)
VALUES('kofi-62048', 9.0);
/
INSERT INTO TABLE(SELECT M.REVIEW FROM MOVIE M WHERE M.TITLE = 'Forrest Gump')(REVIEWER,SCOREPOINT)
VALUES('toccina', 9.0);
/
INSERT INTO TABLE(SELECT M.REVIEW FROM MOVIE M WHERE M.TITLE = 'Forrest Gump')(REVIEWER,SCOREPOINT)
VALUES('murphaus', 9.0);
/

--## 5.7 Query
SELECT AW.AWARDNAME "Award", AW.AWARDYEAR "Year",
DEREF(AW.MOVIE).TITLE "Title", DEREF(AW.ARTIST).NAME "Name"
FROM AWARD AW;

SELECT DISTINCT FIRST_VALUE(DEREF(AW.MOVIE).TITLE)OVER(PARTITION BY AW.MOVIE) "Title",
DEREF(AW.MOVIE.DIRECTOR).NAME "Director", AW.MOVIE.RATING() "Rating"
FROM AWARD AW
WHERE AW.MOVIE IN
(SELECT AW.MOVIE FROM AWARD AW GROUP BY AW.MOVIE HAVING COUNT(*) > 1);
```

```
-------- 5.7 End--------


-------- 5.8 Start -------
UPDATE MOVIE SET REVIEW = TABLE_REVIEW() WHERE TITLE = 'The Mermaid';/

INSERT INTO TABLE(SELECT M.REVIEW FROM MOVIE M WHERE M.TITLE = 'The Mermaid')(REVIEWER,SCOREPOINT)
VALUES('Tiger Heng', 8.0);
/
INSERT INTO TABLE(SELECT M.REVIEW FROM MOVIE M WHERE M.TITLE = 'The Mermaid')(REVIEWER,SCOREPOINT)
VALUES('Robb C.', 3.0);
/
INSERT INTO TABLE(SELECT M.REVIEW FROM MOVIE M WHERE M.TITLE = 'The Mermaid')(REVIEWER,SCOREPOINT)
VALUES('Phoebe C Lim', 7.0);
/
INSERT INTO TABLE(SELECT M.REVIEW FROM MOVIE M WHERE M.TITLE = 'The Mermaid')(REVIEWER,SCOREPOINT)
VALUES('Reno Rangan', 4.0);
/
INSERT INTO TABLE(SELECT M.REVIEW FROM MOVIE M WHERE M.TITLE = 'The Mermaid')(REVIEWER,SCOREPOINT)
VALUES('cherold', 6.0);

/
INSERT INTO SHOWTIME VALUES(TO_DATE('21/10/2017','DD/MM/YYYY'),'18:00',
(SELECT REF(M) FROM MOVIE M WHERE M.TITLE = 'Forrest Gump'),
(SELECT REF(C) FROM CINEMA C WHERE C.CINEMANAME = 'Reading Rhodes'));
/
INSERT INTO SHOWTIME VALUES(TO_DATE('20/10/2017','DD/MM/YYYY'),'20:00',
(SELECT REF(M) FROM MOVIE M WHERE M.TITLE = 'The Mermaid'),
(SELECT REF(C) FROM CINEMA C WHERE C.CINEMANAME = 'Hoyts Chatswood'));
/

--## 5.8 Query
SELECT DEREF(S.MOVIE).TITLE "Title", DEREF(S.MOVIE).RATING() "Rating",
DEREF(S.CINEMA).CINEMANAME "Cinema",
S.SESSIONDATE "Date", S.SESSIONTIME "Time"
FROM SHOWTIME S, TABLE(S.MOVIE.GENRE) SMG
WHERE SMG.GENRE = 'Comedy'
AND S.MOVIE.RATING() > 4;
-------- 5.8 End -------
```

```
-------- 5.9 Start -------


---- Insert for Movie The Country Bears

INSERT INTO ARTIST VALUES('Peter Hastings', PLACE_TYPE('Haverford','Pennsylvania','USA'),
TO_DATE('09/01/1960', 'DD/MM/YYYY'), NULL);/ -- director


INSERT INTO MOVIE(TITLE,WEBSITE,RUNTIME,STORYLINE,GENRE,MPR,RELEASEDATE) VALUES('The Country Bears',

'http://www.imdb.com/title/tt0276033/?ref_=kw_li_tt',88,'Based on an attraction at Disneyland, the
Country Bear Jamboree,

this movie is one in a long line of live action Disney family films. The movie is a satire of Behind
the Music rock and roll bands.',

ARRAY_GENRE(GENRE_TYPE('Comedy'), GENRE_TYPE('Family'), GENRE_TYPE('Music')),'G',TO_DATE('16/01/2003',
'DD/MM/YYYY'));/


UPDATE MOVIE SET DIRECTOR = (SELECT REF(A) FROM ARTIST A WHERE A.NAME = 'Peter Hastings')

WHERE MOVIE.TITLE = 'The Country Bears';/


---- Insert for Movie Man of the Year

INSERT INTO ARTIST VALUES('Barry Levinson', PLACE_TYPE('Baltimore','Maryland','USA'),
TO_DATE('06/04/1942', 'DD/MM/YYYY'), NULL);/ -- director


INSERT INTO MOVIE(TITLE,WEBSITE,RUNTIME,STORYLINE,GENRE,MPR,RELEASEDATE) VALUES('Man of the Year',

'http://www.imdb.com/title/tt0483726/?ref_=kw_li_tt',115,'A comedian who hosts a news satire program

decides to run for president, and a computerized voting machine malfunction gets him elected.',

ARRAY_GENRE(GENRE_TYPE('Comedy'), GENRE_TYPE('Drama'), GENRE_TYPE('Romance')),'M',TO_DATE('01/03/2007',
'DD/MM/YYYY'));/


UPDATE MOVIE SET DIRECTOR = (SELECT REF(A) FROM ARTIST A WHERE A.NAME = 'Barry Levinson')

WHERE MOVIE.TITLE = 'Man of the Year';/


---- Insert for Movie Drop Squad

INSERT INTO ARTIST VALUES('David C. Johnson', PLACE_TYPE('Baltimore','Maryland','USA'),
TO_DATE('23/03/1962', 'DD/MM/YYYY'), NULL);/ -- director


INSERT INTO MOVIE(TITLE,WEBSITE,RUNTIME,STORYLINE,GENRE,MPR,RELEASEDATE) VALUES('Drop Squad',

'http://www.imdb.com/title/tt0109675/?ref_=kw_li_tt',86,'Political satire about an

underground militant group that kidnaps African-Americans who have sold out their race.',

ARRAY_GENRE(GENRE_TYPE('Drama')),'R',TO_DATE('28/10/1994', 'DD/MM/YYYY'));/


UPDATE MOVIE SET DIRECTOR = (SELECT REF(A) FROM ARTIST A WHERE A.NAME = 'David C. Johnson')

WHERE MOVIE.TITLE = 'Drop Squad';/
```

```
---- Insert for Movie The Fool

INSERT INTO ARTIST VALUES('Christine Edzard', PLACE_TYPE('Paris','Paris Region','France'),
TO_DATE('15/02/1945', 'DD/MM/YYYY'), NULL);/ -- director


INSERT INTO MOVIE(TITLE,WEBSITE,RUNTIME,STORYLINE,GENRE,MPR,RELEASEDATE) VALUES('The Fool',

'http://www.imdb.com/title/tt0099593/?ref_=kw_li_tt',140,'A costume drama / satire about financial
skull-duggery,

and confidence tricksters in both the upper and lower classes in Victorian London.',

ARRAY_GENRE(GENRE_TYPE('Drama')),'U',TO_DATE('07/12/1990', 'DD/MM/YYYY'));/


UPDATE MOVIE SET DIRECTOR = (SELECT REF(A) FROM ARTIST A WHERE A.NAME = 'Christine Edzard')

WHERE MOVIE.TITLE = 'The Fool';/


--## 5.9 Query

SELECT M.TITLE, DEREF(M.DIRECTOR).NAME "Director"

FROM MOVIE M

WHERE M.STORYLINE LIKE '%satire%'

AND M.TITLE NOT IN

(SELECT M.TITLE FROM MOVIE M, TABLE(M.GENRE) MG

 WHERE MG.GENRE = 'Comedy');


-------- 5.9 End -------



-------- 5.10 Start -------

/

INSERT INTO SHOWTIME VALUES(TO_DATE('22/10/2017','DD/MM/YYYY'),'14:00',

(SELECT REF(M) FROM MOVIE M WHERE M.TITLE = 'Forrest Gump'),

(SELECT REF(C) FROM CINEMA C WHERE C.CINEMANAME = 'Reading Rhodes'));

/

INSERT INTO SHOWTIME VALUES(TO_DATE('22/10/2017','DD/MM/YYYY'),'16:00',

(SELECT REF(M) FROM MOVIE M WHERE M.TITLE = 'The Aviator'),

(SELECT REF(C) FROM CINEMA C WHERE C.CINEMANAME = 'Palace Verona'));

/

INSERT INTO SHOWTIME VALUES(TO_DATE('22/10/2017','DD/MM/YYYY'),'18:00',

(SELECT REF(M) FROM MOVIE M WHERE M.TITLE = 'Titanic'),

(SELECT REF(C) FROM CINEMA C WHERE C.CINEMANAME = 'Hoyts Chatswood'));

/

INSERT INTO SHOWTIME VALUES(TO_DATE('22/10/2017','DD/MM/YYYY'),'20:00',

(SELECT REF(M) FROM MOVIE M WHERE M.TITLE = 'The Mermaid'),
```

```
(SELECT REF(C) FROM CINEMA C WHERE C.CINEMANAME = 'Palace Verona'));
/

--## 5.10 Query
SELECT TO_CHAR(S.SESSIONDATE, 'DD-MON-YYYY') "Date", TO_CHAR(S.SESSIONDATE, 'Day') "Day",
DEREF(S.MOVIE).TITLE "Title", DEREF(S.MOVIE.DIRECTOR).NAME "Director",
S.MOVIE.RATING() "Highest Rating"
FROM SHOWTIME S
WHERE S.SESSIONDATE = TO_DATE('22/10/2017','DD/MM/YYYY')
AND S.MOVIE.RATING() >= ALL
(SELECT S.MOVIE.RATING() "Highest Rating"
 FROM SHOWTIME S
 WHERE S.SESSIONDATE = TO_DATE('22/10/2017','DD/MM/YYYY'));
-------- 5.10 End -------
```

Appendix C – Modification of Recommendation

```
--------- Design Modification --------


---- Create RECOMMEND_TYPE type

CREATE OR REPLACE TYPE RECOMMEND_TYPE AS OBJECT(

    MOVIE          REF MOVIE_TYPE

);

/

---- Create RECOMMEND_TABLE collection

CREATE OR REPLACE TYPE RECOMMEND_TABLE AS TABLE OF RECOMMEND_TYPE;

/

---- Alter MOVIE_TYPE and MOVIE

ALTER TYPE MOVIE_TYPE

ADD ATTRIBUTE (RECOMMEND RECOMMEND_TABLE)

CASCADE NOT INCLUDING TABLE DATA;

/

ALTER TABLE MOVIE UPGRADE INCLUDING DATA;

/


---- Recommendations

---- Insert for Movie Judwaa 2

INSERT INTO ARTIST VALUES('David Dhawan', PLACE_TYPE('Agartala','Chandigarh','India'),
TO_DATE('16/08/1955', 'DD/MM/YYYY'), NULL);/ -- director


INSERT INTO MOVIE(TITLE,WEBSITE,RUNTIME,STORYLINE,GENRE,MPR,RELEASEDATE) VALUES('Judwaa 2',

'http://www.imdb.com/title/tt5456546/?ref_=india_t_hifull',145,'Prem and Raja are twin
brothers

who are seperated at birth but are uniquely connected to eachother via their reflexes.',

ARRAY_GENRE(GENRE_TYPE('Action'),GENRE_TYPE('Comedy'),GENRE_TYPE('Romance')),'M',TO_DATE('29
/09/2017', 'DD/MM/YYYY'));/


UPDATE MOVIE SET DIRECTOR = (SELECT REF(A) FROM ARTIST A WHERE A.NAME = 'David Dhawan')

WHERE MOVIE.TITLE = 'Judwaa 2';/


---- Insert for Movie Dangal

INSERT INTO ARTIST VALUES('Nitesh Tiwari', PLACE_TYPE('Itarsi','Madhya Pradesh','India'),
NULL, NULL);/ -- director
```

```
INSERT INTO MOVIE(TITLE,WEBSITE,RUNTIME,STORYLINE,GENRE,MPR,RELEASEDATE) VALUES('Dangal',

'http://www.imdb.com/title/tt5074352/?ref_=india_t_hifull',161,'Former wrestler Mahavir
Singh Phogat

and his two wrestler daughters struggle towards glory at the Commonwealth Games in the face
of societal oppression.',

ARRAY_GENRE(GENRE_TYPE('Action'),GENRE_TYPE('Biography'),GENRE_TYPE('Drama')),'M',TO_DATE('2
3/12/2016', 'DD/MM/YYYY'));/


UPDATE MOVIE SET DIRECTOR = (SELECT REF(A) FROM ARTIST A WHERE A.NAME = 'Nitesh Tiwari')

WHERE MOVIE.TITLE = 'Dangal';/


---- Insert for Movie Newton

INSERT INTO ARTIST VALUES('Amit Masurkar', PLACE_TYPE(NULL,NULL,NULL), NULL, NULL);/ --
director


INSERT INTO MOVIE(TITLE,WEBSITE,RUNTIME,STORYLINE,GENRE,MPR,RELEASEDATE) VALUES('Newton',

'http://www.imdb.com/title/tt6484982/?ref_=india_t_hifull',106,'A government clerk on
election duty

in the conflict ridden jungle of Central India tries his best to conduct free and fair
voting despite

the apathy of security forces and the looming fear of guerrilla attacks by communist
rebels.',

ARRAY_GENRE(GENRE_TYPE('Comedy'),GENRE_TYPE('Drama')),'M',TO_DATE('22/09/2017',
'DD/MM/YYYY'));/


UPDATE MOVIE SET DIRECTOR = (SELECT REF(A) FROM ARTIST A WHERE A.NAME = 'Amit Masurkar')

WHERE MOVIE.TITLE = 'Newton';/



---- Insert int RECOMMEND

UPDATE MOVIE SET RECOMMEND = RECOMMEND_TABLE() WHERE MOVIE.TITLE = 'Fever';/


INSERT INTO TABLE(SELECT M.RECOMMEND FROM MOVIE M WHERE M.TITLE = 'Fever')

VALUE(SELECT REF(M) FROM MOVIE M WHERE M.TITLE = 'Judwaa 2');

/

INSERT INTO TABLE(SELECT M.RECOMMEND FROM MOVIE M WHERE M.TITLE = 'Fever')

VALUE(SELECT REF(M) FROM MOVIE M WHERE M.TITLE = 'Dangal');

/

INSERT INTO TABLE(SELECT M.RECOMMEND FROM MOVIE M WHERE M.TITLE = 'Fever')

VALUE(SELECT REF(M) FROM MOVIE M WHERE M.TITLE = 'Newton');

/
```

```
--## Recommend Query
SELECT DEREF(MR.MOVIE).TITLE "Title", DEREF(MR.MOVIE.DIRECTOR).NAME "Director",
DEREF(MR.MOVIE).WEBSITE "Website URL"
FROM MOVIE M, TABLE(M.RECOMMEND) MR
WHERE M.TITLE = 'Fever';
```

Appendix D – Query Output Screenshots

Query 1

```
354  --## 5.1 Query
355  SELECT DEREF(MC.ACTOR).NAME "Actor Name", M.ACTORAGE(DEREF(MC.ACTOR).NAME) "Actor Age",
356  MC.ROLE "Role", MC.STAR "Star", MC.CREDITORDER "Credit Orders"
357  FROM MOVIE M, TABLE(M.CAST) MC
358  WHERE M.TITLE = 'Titanic'
359  ORDER BY MC.CREDITORDER;
```

Script Output | Query Result

SQL | All Rows Fetched: 4 in 0.2 seconds

	Actor Name	Actor Age	Role	Star	Credit Orders
1	Leonardo DiCaprio	23	Jack Dawson	Star	1
2	Kate Winslet	22	Rose Dewitt Bukater	Star	2
3	Billy Zane	31	Cal Hockley	Star	3
4	Kathy Bates	49	Molly Brown	(null)	4

Query 2 - 1

```
520  --## 5.2Query
521  SELECT M.TITLE "Title", DEREF(M.DIRECTOR).NAME "Director", GENRE(M.TITLE) "Genre(s)"
522  FROM MOVIE M, TABLE(M.CAST) MC
523  WHERE DEREF(MC.ACTOR).NAME = 'Cate Blanchett'
```

Script Output | Query Result

SQL | All Rows Fetched: 4 in 0.208 seconds

	Title	Director	Genre(s)
1	The Lord of the Rings: The Return of the King	Peter Jackson	Adventure, Drama, Fantasy
2	Elizabeth: The Golden Age	Shekhar Kapur	Biography, Drama, History
3	The Aviator	Martin Scorsese	Biography, Drama, History
4	The Curious Case of Benjamin Button	David Fincher	Drama, Fantasy, Romance

Query 2 - 2

```
520  --## 5.2Query
521  SELECT M.TITLE "Title", DEREF(M.DIRECTOR).NAME "Director", GENRE(M.TITLE) "Genre(s)"
522  FROM MOVIE M, TABLE(M.CAST) MC
523  WHERE DEREF(MC.ACTOR).NAME = 'Cate Blanchett'
524  AND MC.STAR = 'Star';
```

Script Output | Query Result

SQL | All Rows Fetched: 3 in 0.176 seconds

	Title	Director	Genre(s)
1	Elizabeth: The Golden Age	Shekhar Kapur	Biography, Drama, History
2	The Aviator	Martin Scorsese	Biography, Drama, History
3	The Curious Case of Benjamin Button	David Fincher	Drama, Fantasy, Romance

Query 3

```
--## 5.3 Query
SELECT TO_CHAR(S.SESSIONDATE, 'DD-MON-YYYY') "Date", TO_CHAR(S.SESSIONDATE, 'Day') "Day",
S.SESSIONTIME "Time", DEREF(S.MOVIE).TITLE "Title",
DEREF(S.MOVIE.DIRECTOR).NAME "Director",
S.MOVIE.RATING() "Rating", STARACTOR(DEREF(MOVIE).TITLE) "Star Actors"
FROM SHOWTIME S
WHERE S.SESSIONDATE = TO_DATE('14/10/2017','DD/MM/YYYY');
```

Script Output | Query Result

SQL | All Rows Fetched: 3 in 0.135 seconds

	Date	Day	Time	Title	Director	Rating	Star Actors
1	14-OCT-2017	Saturday	18:00	Titanic	James Cameron	9.7	Leonardo DiCaprio, Kate Winslet, Billy Zane
2	14-OCT-2017	Saturday	20:00	The Aviator	Martin Scorsese	9	Leonardo DiCaprio, Cate Blanchett, Kate Beckinsale
3	14-OCT-2017	Saturday	14:00	Elizabeth: The Golden Age	Shekhar Kapur	7.2	Cate Blanchett, Clive Owen, Geoffrey Rush

Query 4

```
--## 5.4 Query
SELECT DEREF(S.CINEMA).CINEMANAME "Cinema", DEREF(S.MOVIE).TITLE "Title",
DEREF(S.MOVIE.DIRECTOR).NAME "Director",
TO_CHAR(S.SESSIONDATE, 'DD-MON-YYYY') "Date", S.SESSIONTIME "Time"
FROM SHOWTIME S
WHERE DEREF(S.MOVIE).TITLE = 'Wind River';
```

Script Output | Query Result

SQL | All Rows Fetched: 3 in 0.074 seconds

	Cinema	Title	Director	Date	Time
1	Palace Verona	Wind River	Taylor Sheridan	25-AUG-2017	21:00
2	Reading Rhodes	Wind River	Taylor Sheridan	19-AUG-2017	16:00
3	Hoyts Chatswood	Wind River	Taylor Sheridan	26-AUG-2017	10:00

Query 5

```
--## 5.5 Query
SELECT DEREF(DIRECTOR).NAME "Director", DEREF(DIRECTOR).AGE() "Age"
FROM MOVIE M
WHERE DEREF(DIRECTOR).NAME IN
(SELECT DEREF(MC.ACTOR).NAME FROM MOVIE M, TABLE(M.CAST) MC);
```

Script Output | Query Result

SQL | All Rows Fetched: 2 in 0.07 seconds

	Director	Age
1	Jon Favreau	51
2	Stephen Chow	55

Query 6

```
778  --## 5.6 Query
779  SELECT DEREF(AW.MOVIE.DIRECTOR).NAME "Director", DEREF(AW.MOVIE).TITLE "Title",
780  TO_CHAR(DEREF(AW.MOVIE).RELEASEDATE,'DD-MON-YYYY') "Release Date"
781  FROM AWARD AW
782  WHERE AW.AWARDNAME = 'Academy Award for Best Director';
```

Script Output | Query Result

SQL | All Rows Fetched: 3 in 0.096 seconds

	Director	Title	Release Date
1	Peter Jackson	The Lord of the Rings: The Return of the King	26-DEC-2003
2	James Cameron	Titanic	18-DEC-1997
3	Robert Zemeckis	Forrest Gump	17-NOV-1994

Query 7 - 1

```
1042  SELECT AW.AWARDNAME "Award", AW.AWARDYEAR "Year",
1043  DEREF(AW.MOVIE).TITLE "Title", DEREF(AW.ARTIST).NAME "Name"
1044  FROM AWARD AW;
```

Script Output | Query Result

SQL | All Rows Fetched: 5 in 0.102 seconds

	Award	Year	Title	Name
1	Academy Award for Best Director	1997	Titanic	James Cameron
2	Academy Award for Best Picture	1997	Titanic	James Cameron
3	Academy Award for Best Director	2003	The Lord of the Rings: The Return of the King	Peter Jackson
4	Academy Award for Best Director	1994	Forrest Gump	Robert Zemeckis
5	Academy Award for Best Actor	1994	Forrest Gump	Tom Hanks

Query 7 - 2

```
821  SELECT DISTINCT FIRST_VALUE(DEREF(AW.MOVIE).TITLE)OVER(PARTITION BY AW.MOVIE) "Title",
822  DEREF(AW.MOVIE.DIRECTOR).NAME "Director", AW.MOVIE.RATING() "Rating"
823  FROM AWARD AW
824  WHERE AW.MOVIE IN
825  (SELECT AW.MOVIE FROM AWARD AW GROUP BY AW.MOVIE HAVING COUNT(*) > 1);
```

Script Output | Query Result

SQL | All Rows Fetched: 2 in 0.072 seconds

	Title	Director	Rating
1	Titanic	James Cameron	9.7
2	Forrest Gump	Robert Zemeckis	9.2

Query 8

```
858  --## 5.8 Query
859  SELECT DEREF(S.MOVIE).TITLE "Title", DEREF(S.MOVIE).RATING() "Rating",
860  DEREF(S.CINEMA).CINEMANAME "Cinema",
861  S.SESSIONDATE "Date", S.SESSIONTIME "Time"
862  FROM SHOWTIME S, TABLE(S.MOVIE.GENRE) SMG
863  WHERE SMG.GENRE = 'Comedy'
864  AND S.MOVIE.RATING() > 4;
```

Script Output | Query Result

SQL | All Rows Fetched: 4 in 0.078 seconds

	Title	Rating	Cinema	Date	Time
1	Forrest Gump	9.2	Reading Rhodes	21/OCT/17	18:00
2	The Mermaid	5.6	Hoyts Chatswood	20/OCT/17	20:00
3	Forrest Gump	9.2	Reading Rhodes	22/OCT/17	14:00
4	The Mermaid	5.6	Palace Verona	22/OCT/17	20:00

Query 9 - 1

```
1039  SELECT M.TITLE, DEREF(M.DIRECTOR).NAME "Director"
1040  FROM MOVIE M
1041  WHERE M.STORYLINE LIKE '%satire%';
```

Script Output | Query Result

SQL | All Rows Fetched: 4 in 0.059 seconds

	TITLE	Director
1	Man of the Year	Barry Levinson
2	The Country Bears	Peter Hastings
3	Drop Squad	David C. Johnson
4	The Fool	Christine Edzard

Query 9 – 2

```
914  --## 5.9 Query
915  SELECT M.TITLE, DEREF(M.DIRECTOR).NAME "Director"
916  FROM MOVIE M
917  WHERE M.STORYLINE LIKE '%satire%'
918  AND M.TITLE NOT IN
919  (SELECT M.TITLE FROM MOVIE M, TABLE(M.GENRE) MG
920   WHERE MG.GENRE = 'Comedy');
```

Script Output | Query Result

SQL | All Rows Fetched: 2 in 0.061 seconds

	TITLE	Director
1	The Fool	Christine Edzard
2	Drop Squad	David C. Johnson

Query 10

```
944  --## 5.10 Query
945  SELECT TO_CHAR(S.SESSIONDATE, 'DD-MON-YYYY') "Date", TO_CHAR(S.SESSIONDATE, 'Day') "Day",
946  DEREF(S.MOVIE).TITLE "Title", DEREF(S.MOVIE.DIRECTOR).NAME "Director",
947  S.MOVIE.RATING() "Highest Rating"
948  FROM SHOWTIME S
949  WHERE S.SESSIONDATE = TO_DATE('22/10/2017','DD/MM/YYYY')
950  AND S.MOVIE.RATING() >= ALL
951  (SELECT S.MOVIE.RATING() "Highest Rating"
952   FROM SHOWTIME S
953   WHERE S.SESSIONDATE = TO_DATE('22/10/2017','DD/MM/YYYY'));
```

Script Output | Query Result

SQL | All Rows Fetched: 1 in 0.078 seconds

	Date	Day	Title	Director	Highest Rating
1	22-OCT-2017	Sunday	Titanic	James Cameron	9.7

Query Recommendation

```
1019  --## Recommend Query
1020  SELECT DEREF(MR.MOVIE).TITLE "Title", DEREF(MR.MOVIE.DIRECTOR).NAME "Director",
1021  DEREF(MR.MOVIE).WEBSITE "Website URL"
1022  FROM MOVIE M, TABLE(M.RECOMMEND) MR
1023  WHERE M.TITLE = 'Fever';
```

Script Output | Query Result

SQL | All Rows Fetched: 3 in 0.083 seconds

	Title	Director	Website URL
1	Judwaa 2	David Dhawan	http://www.imdb.com/title/tt5456546/?ref_=india_t_hifull
2	Dangal	Nitesh Tiwari	http://www.imdb.com/title/tt5074352/?ref_=india_t_hifull
3	Newton	Amit Masurkar	http://www.imdb.com/title/tt6484982/?ref_=india_t_hifull