0. @C:\seed2018.sql

1. 1

```
SELECT OWNER, CONSTRAINT_NAME, TABLE_NAME
FROM USER_CONSTRAINTS
WHERE TABLE_NAME = 'FILM' OR TABLE_NAME = 'FILM_CATEGORY' OR
TABLE_NAME = 'CATEGORY' OR TABLE_NAME ='FILM_ACTOR' OR TABLE_NAME =
'LANGUAGE' OR TABLE_NAME ='ACTOR';
```

```
SQL> SELECT OWNER, CONSTRAINT_NAME, TABLE_NAME
  2  FROM USER_CONSTRAINTS
  3  WHERE TABLE_NAME='FILM' OR TABLE_NAME='FILM_CATEGORY' OR TABLE_NAME='CATEGORY' OR TABLE_NAME ='FILM_ACTOR' OR TABLE_NAME='LANGUAGE' OR TABLE_NAME='ACTOR';

OWNER
------------------------------------------------------------------------------
CONSTRAINT_NAME
------------------------------------------------------------
TABLE_NAME
------------------------------------------------------------
SYS
SYS_C006997
ACTOR

SYS
SYS_C006998
CATEGORY

OWNER
------------------------------------------------------------------------------
CONSTRAINT_NAME
------------------------------------------------------------
TABLE_NAME
------------------------------------------------------------

SYS
SYS_C006999
FILM

SYS
SYS_C007000

OWNER
------------------------------------------------------------------------------
CONSTRAINT_NAME
------------------------------------------------------------
TABLE_NAME
------------------------------------------------------------
FILM

SYS
SYS_C007001
FILM

SYS
```

```
OWNER
------------------------------------------------------------------------------
CONSTRAINT_NAME
------------------------------------------------------------
TABLE_NAME
------------------------------------------------------------
FILM_CATEGORY

SYS
SYS_C007006
FILM_CATEGORY

SYS

OWNER
------------------------------------------------------------------------------
CONSTRAINT_NAME
------------------------------------------------------------
TABLE_NAME
------------------------------------------------------------
SYS_C007007
LANGUAGE

SYS
SYS_C007008
LANGUAGE


OWNER
------------------------------------------------------------------------------
CONSTRAINT_NAME
------------------------------------------------------------
TABLE_NAME
------------------------------------------------------------
SYS
PK_LANGUAGEID
LANGUAGE


15 rows selected.
```

1.2

A. ALTER TABLE ACTOR
ADD CONSTRAINT PK_ACTORID PRIMARY KEY (actor_id);

```
SQL> ALTER TABLE ACTOR
  2  ADD CONSTRAINT PK_ACTORID PRIMARY KEY (actor_id);

Table altered.
```

B. ALTER TABLE CATEGORY
ADD CONSTRAINT PK_CATEGORYID PRIMARY KEY (category_id);

```
SQL> ALTER TABLE CATEGORY
  2  ADD CONSTRAINT PK_CATEGORYID PRIMARY KEY (category_id);

Table altered.
```

C. ALTER TABLE FILM
ADD CONSTRAINT CK_TITLE

CHECK(TITLE IS NOT NULL);

```
SQL> ALTER TABLE FILM
  2  ADD CONSTRAINT CK_TITLE
  3  CHECK(TITLE IS NOT NULL);

Table altered.
```

D. ALTER TABLE ACTOR
ADD CONSTRAINT CK_FNAME
CHECK(first_name IS NOT NULL);

```
SQL> ALTER TABLE ACTOR
  2
SQL> ALTER TABLE ACTOR
  2  ADD CONSTRAINT CK_FNAME
  3  CHECK(first_name IS NOT NULL);

Table altered.
```

E. ALTER TABLE ACTOR
ADD CONSTRAINT CK_LNAME
CHECK(last_name IS NOT NULL);

```
SQL>  ALTER TABLE ACTOR
  2  ADD CONSTRAINT CK_LNAME
  3  CHECK(last_name IS NOT NULL);

Table altered.
```

F. ALTER TABLE CATEGORY
ADD CONSTRAINT CK_CATNAME
CHECK (NAME IS NOT NULL);

```
SQL> ALTER TABLE CATEGORY
  2  ADD CONSTRAINT CK_CATNAME
  3   CHECK (NAME IS NOT NULL);

Table altered.
```

G. ALTER TABLE FILM
ADD CONSTRAINT CK_RENTALRATE
CH ECK(RENTAL_RATE IS NOT NULL);

```
SQL> ALTER TABLE FILM
  2  ADD CONSTRAINT CK_RENTALRATE
  3  CHECK(RENTAL_RATE IS NOT NULL);

Table altered.
```

H. ALTER TABLE FILM
ADD CONSTRAINT CK_RATING
CHECK(RATING IN ('G','PG','PG-13','R','NC-17'));

```
SQL> ALTER TABLE FILM
  2  ADD CONSTRAINT CK_RATING
  3  CHECK(RATING IN ('G','PG','PG-13','R','NC-17'));

Table altered.
```

i. ALTER TABLE FILM
ADD CONSTRAINT CK_SPLFEATURES
CHECK(SPECIAL_FEATURES IN
(NULL,'TRAILERS','COMMENTARIES','DELETED SCENES','BEHIND THE SCENES'));

```
SQL> ALTER TABLE FILM
  2  ADD CONSTRAINT CK_SPLFEATURES
  3  CHECK(SPECIAL_FEATURES IN (NULL,'TRAILERS','COMMENTARIES','DELETED SCENES','BEHIND THE SCENES'));

Table altered.
```

J. ALTER TABLE FILM
ADD CONSTRAINT FK_LANGUAGEID

FOREIGN KEY(language_id)
REFERENCES LANGUAGE(language_id);

```
SQL>  ALTER TABLE FILM
  2   ADD CONSTRAINT FK_LANGUAGEID
  3   FOREIGN KEY(language_id)
  4   REFERENCES LANGUAGE(language_id);

Table altered.
```

K. ALTER TABLE FILM
ADD CONSTRAINT FK_ORLANGUAGEID
FOREIGN KEY(original_language_id)
REFERENCES LANGUAGE (language_id);

```
SQL> ALTER TABLE FILM
  2   ADD CONSTRAINT FK_ORLANGUAGEID
  3  FOREIGN KEY(original_language_id)
  4  REFERENCES LANGUAGE (language_id);

Table altered.
```

L. ALTER TABLE FILM_ACTOR
ADD CONSTRAINT FK_ACTORID
FOREIGN KEY (actor_id)
REFERENCES ACTOR (actor_id);

```
SQL>  ALTER TABLE FILM_ACTOR
  2   ADD CONSTRAINT FK_ACTORID
  3  FOREIGN KEY(ACTOR_ID)
  4  REFERENCES ACTOR(ACTOR_ID);

Table altered.
```

M. ALTER TABLE FILM
ADD CONSTRAINT CK_RELEASEYR
CHECK(RELEASE_YEAR<='2018');

```
SQL>  ALTER TABLE FILM
  2   ADD CONSTRAINT CK_RELEASEYR
  3  CHECK(RELEASE_YEAR<='2018');

Table altered.
```

2.1

A. CREATE SEQUENCE "FILM_ID_SEQ"
INCREMENT BY 2 START WITH 22000;

```
SQL> CREATE SEQUENCE "FILM_ID_SEQ"
  2  INCREMENT BY 2 START WITH 22000;

Sequence created.
```

B. CREATE OR REPLACE TRIGGER "BI_FILM_ID"
BEFORE INSERT ON "FILM"
FOR EACH ROW
BEGIN
SELECT "FILM_ID_SEQ".NEXTVAL INTO :NEW.FILM_ID FROM DUAL;
END;

/

```
SQL> CREATE OR REPLACE TRIGGER "BI_FILM_ID"
  2    BEFORE INSERT ON "FILM"
  3    FOR EACH ROW
  4  BEGIN
  5      SELECT "FILM_ID_SEQ".NEXTVAL INTO :NEW.FILM_ID FROM DUAL;
  6  END;
  7  /

Trigger created.
```

2.2

```
        CREATE OR REPLACE TRIGGER BI_FILM_LANG
   BEFORE INSERT ON FILM
   FOR EACH ROW
   DECLARE
   original_language varchar2(200);
   new_language varchar2(200);

   original_release varchar2(1000);
   new_release varchar2(100);

  new_release_stop varchar2(1000);

   full_new_section varchar2(1000);

  full_new varchar2(1000);

   BEGIN
   IF (:new.original_language_id IS NOT NULL AND :new.language_id IS NOT NULL) THEN

    SELECT name INTO original_language FROM language WHERE language_id =
:NEW.original_language_id;
    SELECT name INTO new_language FROM language WHERE language_id =
:NEW.language_id;

     SELECT CONCAT('Originally in ', original_language) INTO original_release FROM
DUAL;
     SELECT CONCAT('. Re-released in ', new_language) INTO new_release FROM DUAL;

      SELECT CONCAT(new_release, '.') INTO new_release_stop FROM DUAL;

     SELECT CONCAT(original_release, new_release_stop) INTO full_new_section FROM
DUAL;
      SELECT CONCAT(:new.description, full_new_section) INTO full_new FROM DUAL;

     SELECT full_new INTO :new.DESCRIPTION FROM DUAL;
    END IF;
 END;
   /
```

SHOW ERRORS;

```
SQL>   CREATE OR REPLACE TRIGGER BI_FILM_LANG
  2      BEFORE INSERT ON FILM
  3      FOR EACH ROW
  4      DECLARE
  5      original_language varchar2(200);
  6      new_language varchar2(200);
  7
  8      original_release varchar2(1000);
  9      new_release varchar2(100);
 10
 11    new_release_stop varchar2(1000);
 12
 13     full_new_section varchar2(1000);
 14
 15    full_new varchar2(1000);
 16
 17     BEGIN
 18     IF (:new.original_language_id IS NOT NULL AND :new.language_id IS NOT NULL) THEN
 19
 20      SELECT name INTO original_language FROM language WHERE language_id = :NEW.original_language_id;
 21      SELECT name INTO new_language FROM language WHERE language_id = :NEW.language_id;
 22
 23       SELECT CONCAT('Originally in ', original_language) INTO original_release FROM DUAL;
 24       SELECT CONCAT('. Re-released in ', new_language) INTO new_release FROM DUAL;
 25
 26        SELECT CONCAT(new_release, '.') INTO new_release_stop FROM DUAL;
 27
 28        SELECT CONCAT(original_release, new_release_stop) INTO full_new_section FROM DUAL;
 29        SELECT CONCAT(:new.description, full_new_section) INTO full_new FROM DUAL;
 30
 31       SELECT full_new INTO :new.DESCRIPTION FROM DUAL;
 32      END IF;
 33   END;
 34     /

Trigger created.

Elapsed: 00:00:00.06
SQL> SHOW ERRORS;
No errors.
```

3.1

```
SELECT CATEGORY.NAME, FILM.TITLE, FILM.LENGTH AS LONGEST_DURATION
   FROM FILM, CATEGORY, FILM_CATEGORY
   WHERE FILM.FILM_ID = FILM_CATEGORY.FILM_ID AND
FILM_CATEGORY.CATEGORY_ID = CATEGORY.CATEGORY_ID AND
CATEGORY.NAME = 'Comedy' AND FILM.LENGTH >=ALL(
   SELECT MAX(FILM.LENGTH) FROM FILM);
```

```
SQL> SELECT CATEGORY.NAME, FILM.TITLE, FILM.LENGTH AS LONGEST_DURATION
  2  FROM FILM, CATEGORY, FILM_CATEGORY
  3  WHERE FILM.FILM_ID = FILM_CATEGORY.FILM_ID AND FILM_CATEGORY.CATEGORY_ID = CATEGORY.CATEGORY_ID AND CATEGORY.NAME = 'Comedy' AND FILM.LENGTH >=ALL(
  4  SELECT MAX(FILM.LENGTH) FROM FILM);

NAME
----------------------------------------------------
TITLE
--------------------------------------------------------------------------------
LONGEST_DURATION
----------------
Comedy
REBEL DINOSAUR
             185

Comedy
FIDELITY DANCING
             185

NAME
----------------------------------------------------
TITLE
--------------------------------------------------------------------------------
LONGEST_DURATION
----------------

Comedy
DYING ANTITRUST
             185

Comedy
BANG MUMMY

NAME
----------------------------------------------------
TITLE
--------------------------------------------------------------------------------
LONGEST_DURATION
----------------
             185

Comedy
BROTHERHOOD SATISFACTION
             185
```

```
Comedy

NAME
--------------------------------------------------
TITLE
--------------------------------------------------------------------------------
LONGEST_DURATION
----------------
CONFESSIONS DESTINY
             185

Comedy
CONTROL ANTHEM
             185


NAME
--------------------------------------------------
TITLE
--------------------------------------------------------------------------------
LONGEST_DURATION
----------------
Comedy
CINCINATTI RUSH
             185


8 rows selected.
```

3.2

```
CREATE OR REPLACE VIEW MAX_COMEDY_ACTORS AS
 SELECT DISTINCT ACTOR.ACTOR_ID, ACTOR.FIRST_NAME, ACTOR.LAST_NAME
 FROM FILM,  ACTOR,  FILM_ACTOR
 WHERE  FILM_ACTOR.ACTOR_ID = ACTOR.ACTOR_ID AND FILM.FILM_ID =
FILM_ACTOR.FILM_ID AND FILM.FILM_ID IN (
SELECT FILM.FILM_ID
FROM FILM, FILM_CATEGORY,CATEGORY
WHERE FILM.FILM_ID = FILM_CATEGORY.FILM_ID AND
FILM_CATEGORY.CATEGORY_ID = CATEGORY.CATEGORY_ID AND
CATEGORY.NAME ='Comedy' AND FILM.LENGTH = (SELECT MAX (FILM.LENGTH)
FROM FILM));
```

```
SQL>  CREATE OR REPLACE VIEW MAX_COMEDY_ACTORS AS
  2   SELECT DISTINCT ACTOR.ACTOR_ID,  ACTOR.FIRST_NAME,  ACTOR.LAST_NAME
  3   FROM FILM,   ACTOR,   FILM_ACTOR
  4   WHERE  FILM_ACTOR.ACTOR_ID = ACTOR.ACTOR_ID AND FILM.FILM_ID = FILM_ACTOR.FILM_ID AND FILM.FILM_ID IN (
  5  SELECT FILM.FILM_ID
  6  FROM FILM,  FILM_CATEGORY, CATEGORY
  7  WHERE FILM.FILM_ID = FILM_CATEGORY.FILM_ID AND FILM_CATEGORY.CATEGORY_ID = CATEGORY.CATEGORY_ID AND CATEGORY.NAME ='Comedy' AND FILM.LENGTH = (SELECT MAX (FILM.LENGTH) FROM
 FILM));

View created.

Elapsed: 00:00:00.06
```

```
SQL> SELECT * FROM MAX_COMEDY_ACTORS;

  ACTOR_ID
----------
FIRST_NAME
--------------------------------------------------------------------------------
LAST_NAME
--------------------------------------------------------------------------------
        56
DAN
HARRIS

       114
MORGAN
MCDORMAND

  ACTOR_ID
----------
FIRST_NAME
--------------------------------------------------------------------------------
LAST_NAME
--------------------------------------------------------------------------------

       165
AL
GARLAND

       166
NICK

  ACTOR_ID
----------
FIRST_NAME
--------------------------------------------------------------------------------
LAST_NAME
--------------------------------------------------------------------------------
DEGENERES

       194
MERYL
ALLEN

        34
```

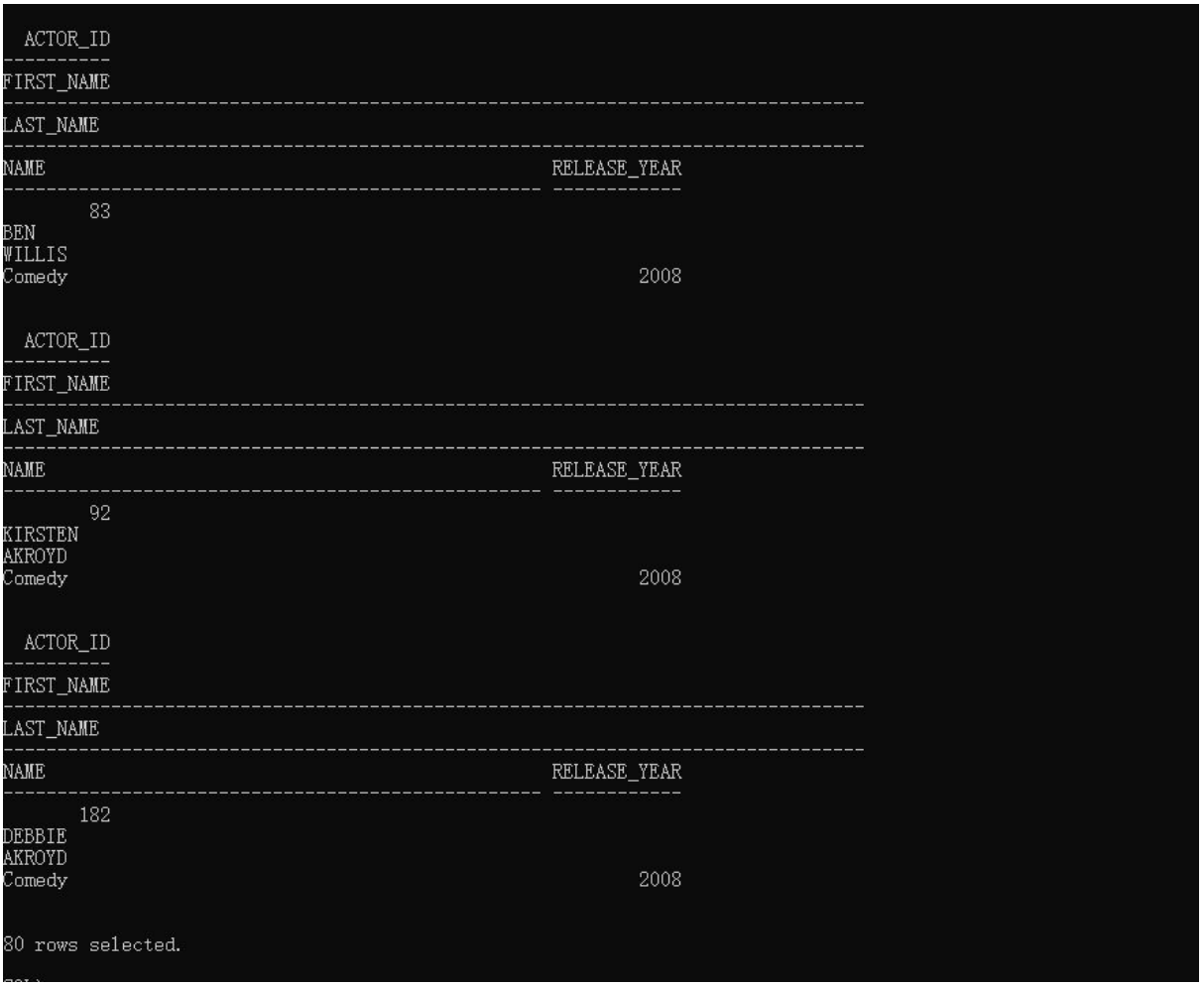
```
--------------------------------------------------------------------------------
LAST_NAME
--------------------------------------------------------------------------------
NEESON

         66
MARY
TANDY

       109

  ACTOR_ID
----------
FIRST_NAME
--------------------------------------------------------------------------------
LAST_NAME
--------------------------------------------------------------------------------
SYLVESTER
DERN

       118
CUBA
ALLEN


  ACTOR_ID
----------
FIRST_NAME
--------------------------------------------------------------------------------
LAST_NAME
--------------------------------------------------------------------------------
       131
JANE
JACKMAN

       184
HUMPHREY
GARLAND

  ACTOR_ID
----------
FIRST_NAME
--------------------------------------------------------------------------------
LAST_NAME
--------------------------------------------------------------------------------


51 rows selected.

Elapsed: 00:00:00.43
```

3.3

```
CREATE VIEW V_COMEDY_ACTORS_2008 AS
   SELECT DISTINCT ACTOR.ACTOR_ID, ACTOR.FIRST_NAME, ACTOR.LAST_NAME,
CATEGORY.NAME, FILM.RELEASE_YEAR
   FROM FILM, ACTOR, CATEGORY, FILM_CATEGORY, FILM_ACTOR
```

WHERE FILM.FILM_ID = FILM_CATEGORY.FILM_ID AND FILM_CATEGORY.CATEGORY_ID = CATEGORY.CATEGORY_ID AND FILM_ACTOR.ACTOR_ID = ACTOR.ACTOR_ID AND FILM_ACTOR.FILM_ID = FILM.FILM_ID AND CATEGORY.NAME = 'Comedy' AND FILM.RELEASE_YEAR = '2008';

```
SQL> CREATE VIEW V_COMEDY_ACTORS_2008 AS
  2  SELECT DISTINCT ACTOR.ACTOR_ID, ACTOR.FIRST_NAME, ACTOR.LAST_NAME, CATEGORY.NAME, FILM.RELEASE_YEAR
  3  FROM FILM, ACTOR, CATEGORY, FILM_CATEGORY, FILM_ACTOR
  4  WHERE FILM.FILM_ID = FILM_CATEGORY.FILM_ID AND FILM_CATEGORY.CATEGORY_ID = CATEGORY.CATEGORY_ID AND FILM_ACTOR.ACTOR_ID = ACTOR.ACTOR_ID AND FILM_ACTOR.FILM_ID = FILM.FILM_
ID AND CATEGORY.NAME = 'Comedy' AND FILM.RELEASE_YEAR = '2008';

View created.

SQL> SELECT * FROM V_COMEDY_ACTORS_2008;

  ACTOR_ID
----------
FIRST_NAME
--------------------------------------------------------------------------------
LAST_NAME
--------------------------------------------------------------------------------
NAME                                               RELEASE_YEAR
-------------------------------------------------- ------------
        20
LUCILLE
TRACY
Comedy                                                     2008


  ACTOR_ID
----------
FIRST_NAME
--------------------------------------------------------------------------------
LAST_NAME
--------------------------------------------------------------------------------
NAME                                               RELEASE_YEAR
-------------------------------------------------- ------------
        77
CARY
MCCONAUGHEY
Comedy                                                     2008
```



```
  ACTOR_ID
----------
FIRST_NAME
--------------------------------------------------------------------------------
LAST_NAME
--------------------------------------------------------------------------------
NAME                                               RELEASE_YEAR
-------------------------------------------------- ------------
        83
BEN
WILLIS
Comedy                                                     2008


  ACTOR_ID
----------
FIRST_NAME
--------------------------------------------------------------------------------
LAST_NAME
--------------------------------------------------------------------------------
NAME                                               RELEASE_YEAR
-------------------------------------------------- ------------
        92
KIRSTEN
AKROYD
Comedy                                                     2008


  ACTOR_ID
----------
FIRST_NAME
--------------------------------------------------------------------------------
LAST_NAME
--------------------------------------------------------------------------------
NAME                                               RELEASE_YEAR
-------------------------------------------------- ------------
       182
DEBBIE
AKROYD
Comedy                                                     2008


80 rows selected.
```

```
80 rows selected.

Elapsed: 00:00:01.48
```

3.4 CREATE MATERIALIZED VIEW MV_COMEDY_ACTORS_2008

```
 BUILD IMMEDIATE
AS
SELECT DISTINCT ACTOR.ACTOR_ID, ACTOR.FIRST_NAME, ACTOR.LAST_NAME,
CATEGORY.NAME, FILM.RELEASE_YEAR
FROM FILM, ACTOR, CATEGORY, FILM_CATEGORY, FILM_ACTOR
 WHERE FILM.FILM_ID = FILM_CATEGORY.FILM_ID AND
FILM_CATEGORY.CATEGORY_ID = CATEGORY.CATEGORY_ID
AND FILM_ACTOR.ACTOR_ID = ACTOR.ACTOR_ID AND FILM_ACTOR.FILM_ID =
FILM.FILM_ID AND CATEGORY.NAME = 'Comedy'
AND FILM.RELEASE_YEAR = '2008';
```

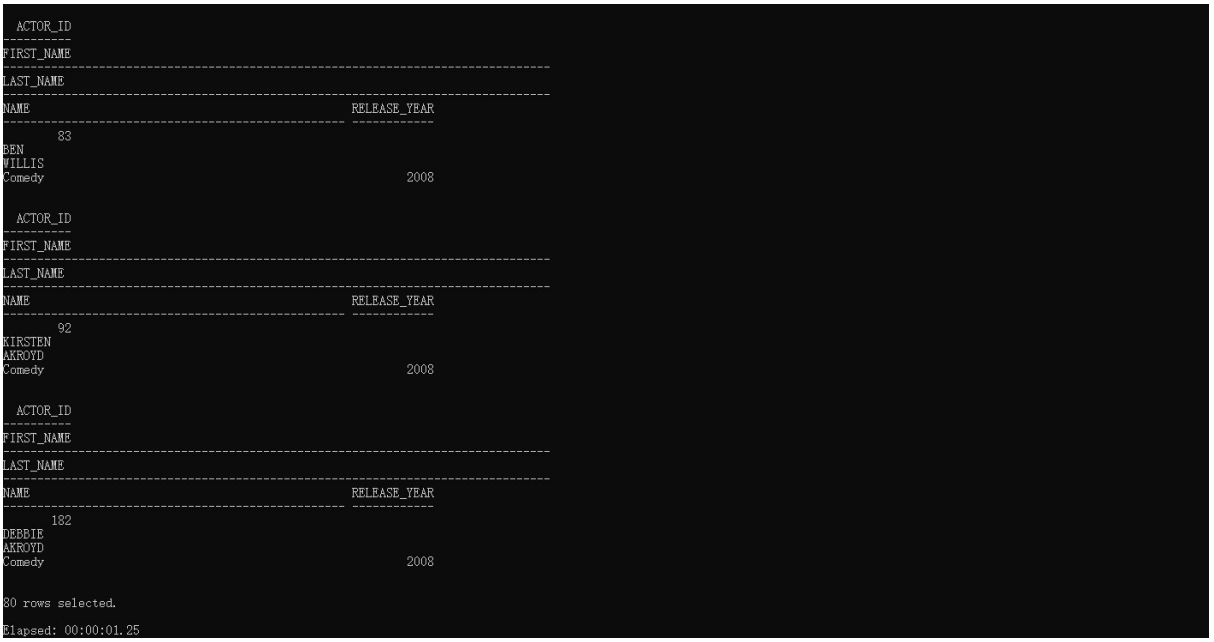
```
SQL> CREATE MATERIALIZED VIEW MV_COMEDY_ACTORS_2008
  2  BUILD IMMEDIATE
  3  AS
  4  SELECT DISTINCT ACTOR.ACTOR_ID, ACTOR.FIRST_NAME, ACTOR.LAST_NAME, CATEGORY.NAME, FILM.RELEASE_YEAR
  5  FROM FILM, ACTOR, CATEGORY, FILM_CATEGORY, FILM_ACTOR
  6   WHERE FILM.FILM_ID = FILM_CATEGORY.FILM_ID AND FILM_CATEGORY.CATEGORY_ID = CATEGORY.CATEGORY_ID AND FILM_ACTOR.ACTOR_ID = ACTOR.ACTOR_ID AND FILM_ACTOR.FILM_ID = FILM.FILM
_ID AND CATEGORY.NAME = 'Comedy' AND FILM.RELEASE_YEAR = '2008';

Materialized view created.

SQL> SELECT * FROM MV_COMEDY_ACTORS_2008;

  ACTOR_ID
----------
FIRST_NAME
--------------------------------------------------------------------------------
LAST_NAME
--------------------------------------------------------------------------------
NAME                                               RELEASE_YEAR
-------------------------------------------------- ------------
        20
LUCILLE
TRACY
Comedy                                                     2008


  ACTOR_ID
----------
FIRST_NAME
--------------------------------------------------------------------------------
LAST_NAME
--------------------------------------------------------------------------------
NAME                                               RELEASE_YEAR
-------------------------------------------------- ------------
        77
CARY
MCCONAUGHEY
Comedy                                                     2008
```

```
  ACTOR_ID
----------
FIRST_NAME
--------------------------------------------------------------------------------
LAST_NAME
--------------------------------------------------------------------------------
NAME                                               RELEASE_YEAR
-------------------------------------------------- ------------
        83
BEN
WILLIS
Comedy                                                     2008


  ACTOR_ID
----------
FIRST_NAME
--------------------------------------------------------------------------------
LAST_NAME
--------------------------------------------------------------------------------
NAME                                               RELEASE_YEAR
-------------------------------------------------- ------------
        92
KIRSTEN
AKROYD
Comedy                                                     2008


  ACTOR_ID
----------
FIRST_NAME
--------------------------------------------------------------------------------
LAST_NAME
--------------------------------------------------------------------------------
NAME                                               RELEASE_YEAR
-------------------------------------------------- ------------
       182
DEBBIE
AKROYD
Comedy                                                     2008


80 rows selected.
```

```
80 rows selected.

Elapsed: 00:00:01.08
```

3.5

For the V_COMEDY_ACTORS_2008 view, the execution time is 1 minute and 25 seconds. However, for the second view MV_COMEDY_ACTORS_2008, it only takes 1 minute and 6 seconds to execute. The normal views use a query to pull data from the underlying tables, but a materialized view is a table on disk that contains the result set of a query. Since the Materialized views are automatically updated as their base tables are updated so it needs lesser execution time than the view. Moreover, the query transformation does not happen with a materialized view and this improves query performance.



4.1

```
SELECT *
   FROM (SELECT TITLE FROM FILM
```

```
   WHERE INSTR(SUBSTR(film.description, INSTR(film.description,'in')),'Boat')>0
ORDER BY TITLE ASC)
WHERE ROWNUM <=200;
```

```
SQL> SELECT *
  2      FROM (SELECT TITLE FROM FILM
  3      WHERE INSTR(SUBSTR(film.description, INSTR(film.description,'in')),'Boat')>0
  4  ORDER BY TITLE ASC)
  5  WHERE ROWNUM <=200;

TITLE
--------------------------------------------------------------------------------
ACADEMY REDS
ADAPTATION JEOPARDY
AFFAIR PANIC
AFFAIR ROXANNE
AGENT SEATTLE
AIRPLANE ANONYMOUS
AIRPLANE IDENTITY
AIRPLANE SIERRA
AIRPORT GOODFELLAS
AIRPORT JAWS
ALABAMA DEVIL

TITLE
--------------------------------------------------------------------------------
ALABAMA GO
ALAMO TITANS
ALASKA DIVINE
ALI INSTINCT
ALI RESURRECTION
ALICE GRAIL
ALICE TOOTSIE
ALIEN ATTACKS
ALIEN WORKER
ALLEY MILE
ALLEY SINNERS

TITLE
--------------------------------------------------------------------------------
ALONE HUSTLER
ALTER FLINTSTONES
AMADEUS LABYRINTH
AMADEUS ROSES
AMELIE DYING
AMELIE FREEDOM
AMERICAN MOTIONS
AMERICAN PUNK
AMISTAD ROBBERS
AMISTAD SUPER
ANACONDA COMMAND
```

```
--------------------------------------------------------------------------------
CAMPUS REMEMBER
CAMPUS SUSPECTS
CANDIDATE ALIEN
CANDIDATE IDAHO
CANDIDATE TOMATOES
CANDLES RAGE
CANYON EVERYONE
CAPER ALIEN
CAPER SATURDAY
CAPER STORY
CARIBBEAN SHAKESPEARE

TITLE
--------------------------------------------------------------------------------
CAROL DARES
CARRIE ARIZONA

200 rows selected.

Elapsed: 00:00:00.41
```

4.2

```
CREATE INDEX  IDX_SEARCH_LOCATION ON
FILM(INSTR(SUBSTR(description,INSTR(description,'in')),'Boat'));
```

```
SQL> CREATE INDEX  IDX_SEARCH_LOCATION ON FILM(INSTR(SUBSTR(description,INSTR(description,'in')),'Boat'));

Index created.
```

```
TITLE
--------------------------------------------------------------------------------
TALENTED HOMICIDE
TEEN INTRIGUE
TELEGRAPH VOYAGE
THEORY RESURRECTION
TIES HUNGER
TIMBERLAND SKY
TOMORROW MUMMY
TOWERS RACER
TRAFFIC HOBBIT
UNCUT SUICIDES
UNFORGIVEN MINDS

TITLE
--------------------------------------------------------------------------------
VANISHED INTERVIEW
VANISHING BANG
VERTIGO AGENT
VIDEOTAPE TROOPERS
WEEKEND PERSONAL
WEST DOOM
WON DARES
WON LABYRINTH
WONDERFUL CABIN
WONDERFUL DROP
WORLD FOREVER

TITLE
--------------------------------------------------------------------------------
YOUTH DAY
YOUTH KICK

200 rows selected.

Elapsed: 00:00:00.34
```

After index execution time.
I created a CREATE INDEX query to add the existed where clause from task 4.1. So it will execute the same data in 4.1 but much more efficient.

4.3
The normal query needs 41 seconds to execute and the index only need 34 seconds. A function-based index, is an index that is created on the results of a function or expression. It can improve the performance of a query. Therefore the execution time after create function-based index a lot shorter.

5.1
EXPLAIN PLAN FOR SELECT /*+RULE*/ * FROM FILM WHERE FILM.FILM_ID >1000;

SELECT PLAN_TABLE_OUTPUT FROM TABLE (DBMS_XPLAN.DISPLAY);

```
SQL> EXPLAIN PLAN FOR SELECT /*+RULE*/ * FROM FILM WHERE FILM.FILM_ID >1000;

Explained.

Elapsed: 00:00:00.06
SQL>
```

```
SQL> EXPLAIN PLAN FOR SELECT /*+RULE*/ * FROM FILM WHERE FILM.FILM_ID >1000;

Explained.

Elapsed: 00:00:00.06
SQL> SELECT PLAN_TABLE_OUTPUT FROM TABLE (DBMS_XPLAN.DISPLAY);

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
Plan hash value: 657888726

-----------------------------------------------
| Id  | Operation                   | Name      |
-----------------------------------------------
|   0 | SELECT STATEMENT            |           |
|   1 |  TABLE ACCESS BY INDEX ROWID| FILM      |
|*  2 |   INDEX RANGE SCAN          | PK_FILMID |
-----------------------------------------------

Predicate Information (identified by operation id):

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
---------------------------------------------------

   2 - access("FILM"."FILM_ID">1000)

Note
-----
   - rule based optimizer used (consider using cbo)

18 rows selected.
```

The query processing is 1 -- 2 -- 0; For outer loop, it accesses to 'FILM.FILM_ID' > 1000 and the outer loop will retrieve all the rows of the FILM table (the outer object). Each row of the FILM will be fully accessed. To evaluate whether the rows can meet the conditions of the WHERE clause criteria, the index FILM.FILM_ID will be used to do the scan operation in a range. Then ID 0 will select the row that satisfy the condition.

5.2

EXPLAIN PLAN FOR SELECT/*+COST*/* FROM FILM WHERE FILM.FILM_ID >1000;

```
SQL> EXPLAIN PLAN FOR SELECT/*+COST*/* FROM FILM WHERE FILM.FILM_ID >1000;

Explained.

Elapsed: 00:00:00.10
SQL>  SELECT PLAN_TABLE_OUTPUT FROM TABLE (DBMS_XPLAN.DISPLAY);

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
Plan hash value: 1232367652

--------------------------------------------------------------------------
| Id  | Operation         | Name | Rows  | Bytes | Cost (%CPU)| Time     |
--------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |      | 19001 |  2709K|   136   (0)| 00:00:02 |
|*  1 |  TABLE ACCESS FULL| FILM | 19001 |  2709K|   136   (0)| 00:00:02 |
--------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------

   1 - filter("FILM"."FILM_ID">1000)

13 rows selected.

Elapsed: 00:00:00.23
```

The cost-based execution plan is more efficient than the rule=based execution plan. The execution time for cost one only takes 23 seconds and the rule one need 28 seconds. For the cost-based execution plan, the processing steps are 1 -- 2; The table will access to each row in FILM then select all the rows that satisfy the condition.

5.3

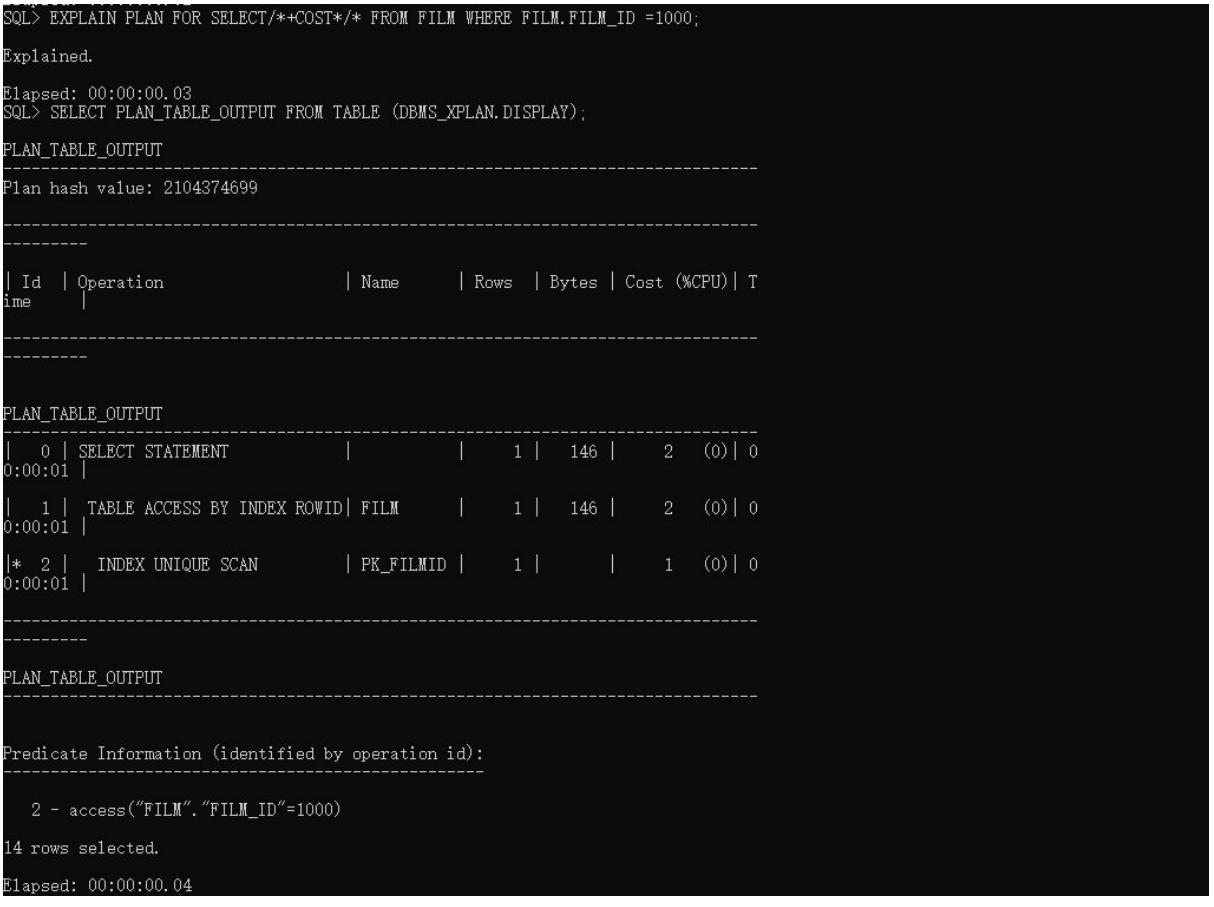
SELECT * FROM FILM

WHERE FILM.FILM_ID = ‘1000’;

```
SQL> SELECT * FROM FILM
  2  WHERE FILM.FILM_ID = '1000';

   FILM_ID
----------
TITLE
--------------------------------------------------------------------------------
DESCRIPTION
--------------------------------------------------------------------------------
RELEASE_YEAR LANGUAGE_ID ORIGINAL_LANGUAGE_ID RENTAL_DURATION RENTAL_RATE
------------ ----------- -------------------- --------------- -----------
    LENGTH REPLACEMENT_COST RATING
---------- ---------------- ----------
SPECIAL_FEATURES
--------------------------------------------------------------------------------
      1000

   FILM_ID
----------
TITLE
--------------------------------------------------------------------------------
DESCRIPTION
--------------------------------------------------------------------------------
RELEASE_YEAR LANGUAGE_ID ORIGINAL_LANGUAGE_ID RENTAL_DURATION RENTAL_RATE
------------ ----------- -------------------- --------------- -----------
    LENGTH REPLACEMENT_COST RATING
---------- ---------------- ----------
SPECIAL_FEATURES
--------------------------------------------------------------------------------
ZORRO ARK

   FILM_ID
----------
TITLE
--------------------------------------------------------------------------------
DESCRIPTION
--------------------------------------------------------------------------------
RELEASE_YEAR LANGUAGE_ID ORIGINAL_LANGUAGE_ID RENTAL_DURATION RENTAL_RATE
------------ ----------- -------------------- --------------- -----------
    LENGTH REPLACEMENT_COST RATING
---------- ---------------- ----------
SPECIAL_FEATURES
--------------------------------------------------------------------------------
A Intrepid Panorama of a Mad Scientist And a Boy who must Redeem a Boy in A Mona

   FILM_ID
----------
TITLE
--------------------------------------------------------------------------------
DESCRIPTION
```

```
TITLE
--------------------------------------------------------------------------------
DESCRIPTION
--------------------------------------------------------------------------------
RELEASE_YEAR LANGUAGE_ID ORIGINAL_LANGUAGE_ID RENTAL_DURATION RENTAL_RATE
------------ ----------- -------------------- --------------- -----------
    LENGTH REPLACEMENT_COST RATING
---------- ---------------- ----------
SPECIAL_FEATURES
--------------------------------------------------------------------------------
Behind the Scenes

   FILM_ID
----------
TITLE
--------------------------------------------------------------------------------
DESCRIPTION
--------------------------------------------------------------------------------
RELEASE_YEAR LANGUAGE_ID ORIGINAL_LANGUAGE_ID RENTAL_DURATION RENTAL_RATE
------------ ----------- -------------------- --------------- -----------
    LENGTH REPLACEMENT_COST RATING
---------- ---------------- ----------
SPECIAL_FEATURES
--------------------------------------------------------------------------------


Elapsed: 00:00:00.09
```

EXPLAIN PLAN FOR SELECT/*+COST*/* FROM FILM WHERE FILM.FILM_ID =1000;
SELECT PLAN_TABLE_OUTPUT FROM TABLE (DBMS_XPLAN.DISPLAY);



5.4

RBO follows a set of rules mostly based on indexes and types of indexes.CBO uses statistics and math to make an educated guess at the lowest cost. Therefore, the execution time of CBO is much lesser than RBO. CBO will partitioned queries look for the fastest

elapsed time and will use the least amount of resources to get the desired output. The RBO is updated technology for execution.

5.5

Task 5.2 and 5.3 are different queries although they all used the cost-based execution plan. The task 5.2 will display all the details of films that id is greater than 1000, but the task 5.3 only need to display film id that equal to 1000. The outcome of them is totally different . Moreover, the plan_table_output of task 5.3 is slightly different to 5.2. In 5.3, the processing steps are 1-- 2 -- 0;

5.6

ANALYZE INDEX PK_FILMID VALIDATE STRUCTURE;

a. SELECT HEIGHT FROM INDEX_STATS;

```
SQL>  ANALYZE INDEX PK_FILMID VALIDATE STRUCTURE;

Index analyzed.

Elapsed: 00:00:00.04
SQL> SELECT HEIGHT FROM INDEX_STATS;

    HEIGHT
----------
         2

Elapsed: 00:00:00.04
```

b. SELECT LF_BLKS FROM INDEX_STATS;

```
SQL> SELECT  LF_BLKS FROM INDEX_STATS;

   LF_BLKS
----------
        37

Elapsed: 00:00:00.03
```

c. SELECT TABLE_NAME, BLOCKS
FROM USER_TABLES
WHERE TABLE_NAME ='FILM';

```
SQL> SELECT TABLE_NAME, BLOCKS
  2  FROM  USER_TABLES
  3  WHERE TABLE_NAME ='FILM';

TABLE_NAME                                                       BLOCKS
------------------------------------------------------------ ----------
FILM                                                                496

Elapsed: 00:00:00.03
```