OCTOBER 10, 2014

INFS2200 ASSIGNMENT

ELLIOT RANDALL

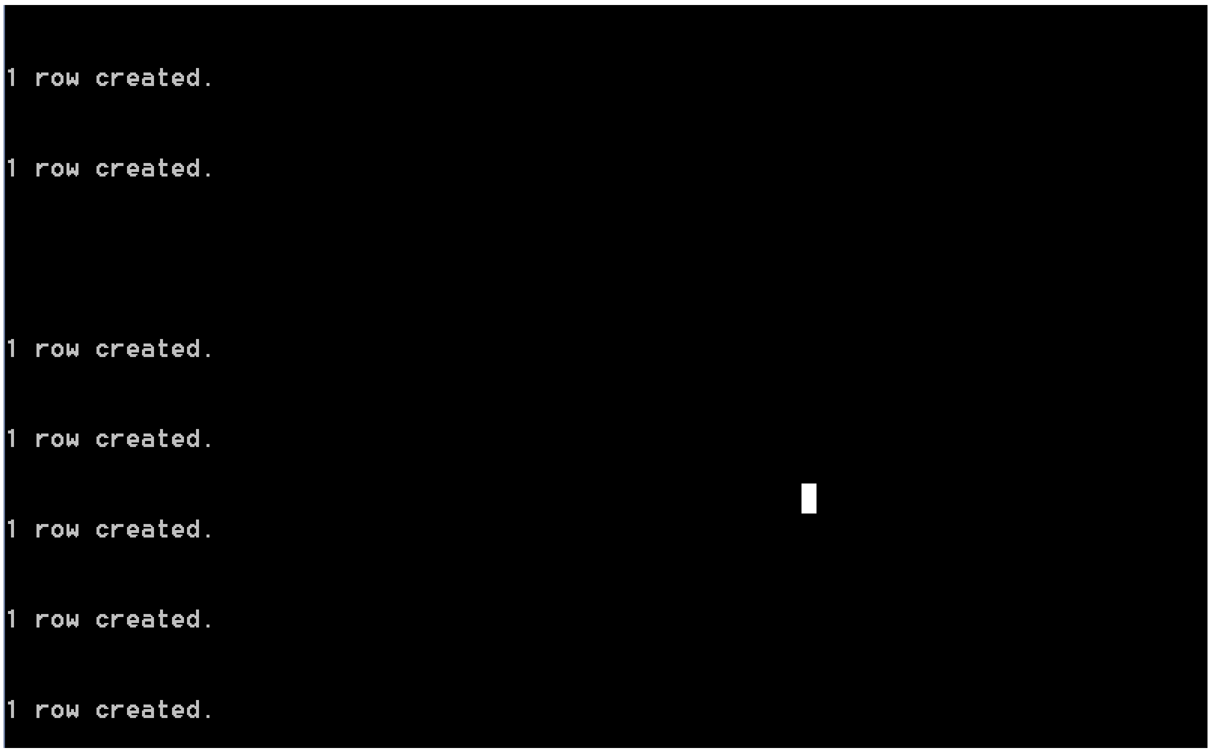
43569175
INFS2200
UQ

Task 0 - Database

To import the database into Oracle:

```
@C:\INFS2200/prjScript(1).sql
```

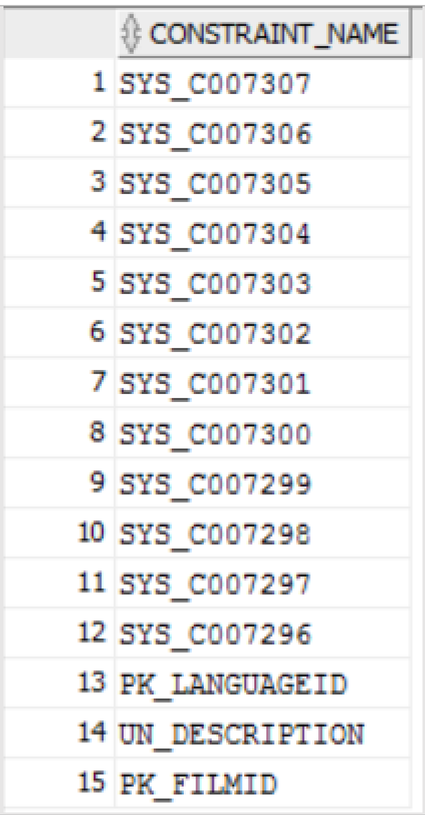
For the output, a constant stream of *'1 row created'* was displayed.



```
1 row created.

1 row created.



1 row created.

1 row created.

1 row created.

1 row created.

1 row created.
```

Task 1 - Constraints

A)



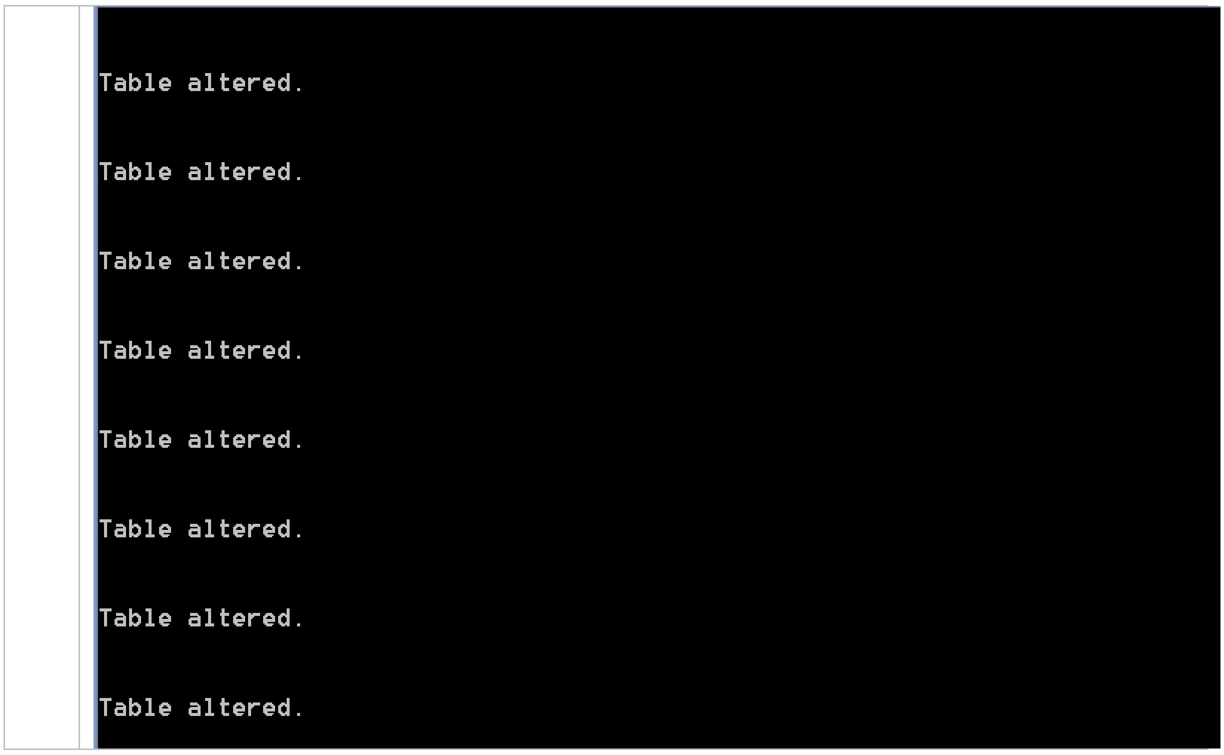
```
   CONSTRAINT_NAME
1  SYS_C007307
2  SYS_C007306
3  SYS_C007305
4  SYS_C007304
5  SYS_C007303
6  SYS_C007302
7  SYS_C007301
8  SYS_C007300
9  SYS_C007299
10 SYS_C007298
11 SYS_C007297
12 SYS_C007296
13 PK_LANGUAGEID
14 UN_DESCRIPTION
15 PK_FILMID
```

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

B)

```
ALTER TABLE actor
ADD CONSTRAINT PK_ACTORID PRIMARY KEY (actor_id);

ALTER TABLE category
ADD CONSTRAINT PK_CATEGORYID PRIMARY KEY (category_id);

ALTER TABLE actor
ADD CONSTRAINT CK_FNAME
CHECK(first_name IS NOT NULL);

ALTER TABLE actor
ADD CONSTRAINT CK_LNAME
CHECK(last_name IS NOT NULL);

ALTER TABLE film
ADD CONSTRAINT CK_TITLE
CHECK(title IS NOT NULL);

ALTER TABLE category
ADD CONSTRAINT CK_CATNAME
CHECK(category IS NOT NULL);

ALTER TABLE film
ADD CONSTRAINT CK_RENTALRATE
CHECK(rental_rate IS NOT NULL);
```

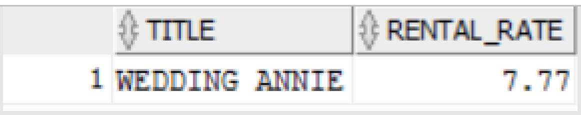
```
ALTER TABLE film
ADD CONSTRAINT CK_RATING
CHECK (rating IN ('G', 'PG', 'PG-13', 'R',
'NC-17'));

ALTER TABLE film
ADD CONSTRAINT CK_SPLFEATURES
CHECK (special_features IN (NULL,
'Trailers', 'Commentaries', 'Deleted
Scenes', 'Behind the Scenes'));

ALTER TABLE film
ADD CONSTRAINT FK_LANGUAGEID
FOREIGN KEY (language_id)
REFERENCES "language"(language_id);

ALTER TABLE film
ADD CONSTRAINT FK_ORLANGUAGEID
FOREIGN KEY (original_language_id)
REFERENCES "language"(language_id);

ALTER TABLE film_actor
ADD CONSTRAINT FK_ACTORID
FOREIGN KEY (actor_id) REFERENCES
actor(actor_id);

ALTER TABLE film
ADD CONSTRAINT CK_RELEASEYR
CHECK (release_year <= '2014');
```



```
Table altered.

Table altered.

Table altered.

Table altered.

Table altered.

Table altered.

Table altered.

Table altered.
```

Task 2 - Triggers

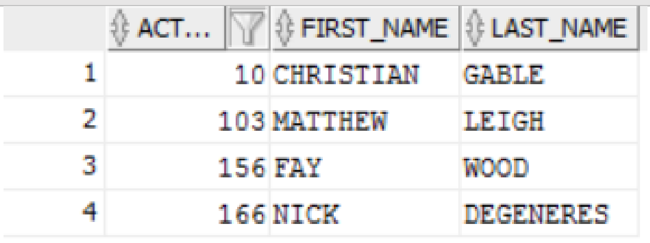
A)



	TITLE	RENTAL_RATE
1	WEDDING ANNIE	7.77

```
SELECT F.title, F.rental_rate
FROM film F, film_category FC,
Category C
WHERE F.rental_rate IN(
SELECT max(rental_rate)
FROM film)
AND F.film_id = FC.film_id
AND FC.category_id =
C.category_id
AND C.name = 'Documentary';
```

B)

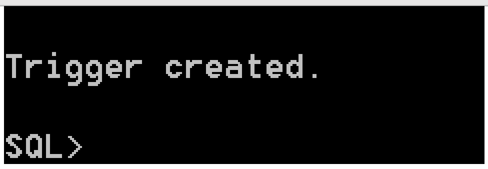


	ACT...	FIRST_NAME	LAST_NAME
1	10	CHRISTIAN	GABLE
2	103	MATTHEW	LEIGH
3	156	FAY	WOOD
4	166	NICK	DEGENERES

```
SELECT A.actor_id,
A.first_name, A.last_name
FROM actor A, film_actor FA,
film F, film_category FC,
category C
WHERE A.actor_id = FA.actor_id
AND FA.film_id = F.film_id
AND F.film_id = FC.film_id
AND FC.category_id =
C.category_id
AND C.name = 'Documentary'
AND F.rental_rate IN(
SELECT max(rental_rate)
FROM film);
```

C) Yes this is a possible action. Unlike checks, triggers are global, and work over the whole database. The trigger will be automatically fired when the user inserts a new actor into the database. Checks on the other hand would not be able to complete this as they are not global and only work on a tuple or attribute based level. Another way of going about this is to create an updateable view between the two tables and create a trigger in which fires when a new actor is inserted. The base table is then updated.

D)

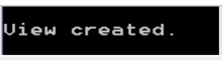


```
Trigger created.

SQL>
```

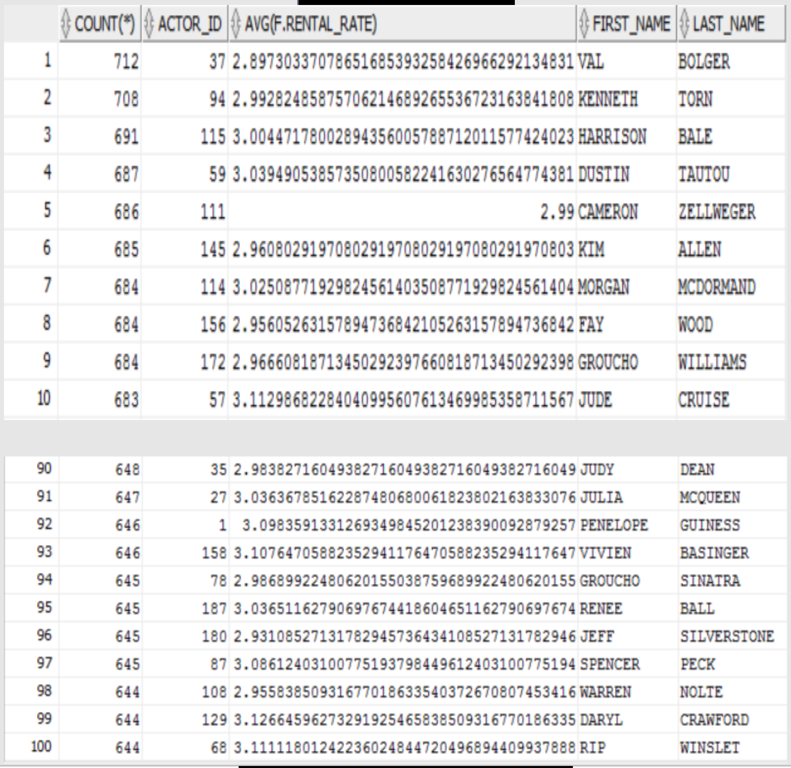
```
CREATE OR REPLACE TRIGGER CH_RENTALS
BEFORE INSERT
ON FILM
FOR EACH ROW
BEGIN
  IF :new.special_features = 'Commentaries' THEN
     :new.rental_rate := :new.rental_rate + 0.5;
  ELSIF :new.special_features = 'Deleted Scenes' THEN
     :new.rental_rate := :new.rental_rate + 0.2;
  ELSIF :new.special_features = 'Behind the Scenes' THEN
     :new.rental_rate := :new.rental_rate - 0.2;
  ELSE
     :new.rental_rate := :new.rental_rate + 0.1;
  END IF;
END;
```

Task 3 - Views

A)



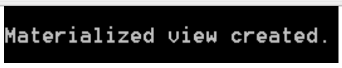
```
View created.
```



	COUNT(*)	ACTOR_ID	AVG(F.RENTAL_RATE)	FIRST_NAME	LAST_NAME
1	712	37	2.897303370786516853932584269662921348 31	VAL	BOLGER
2	708	94	2.99282485875706214689265536723163841808	KENNETH	TORN
3	691	115	3.00447178002894356005788712011577424023	HARRISON	BALE
4	687	59	3.03949053857350800582241630276564774381	DUSTIN	TAUTOU
5	686	111	2.99	CAMERON	ZELLWEGER
6	685	145	2.96080291970802919708029197080291970803	KIM	ALLEN
7	684	114	3.02508771929824561403508771929824561404	MORGAN	MCDORMAND
8	684	156	2.95605263157894736842105263157894736842	FAY	WOOD
9	684	172	2.96660818713450292397660818713450292398	GROUCHO	WILLIAMS
10	683	57	3.11298682284040995607613469985358711567	JUDE	CRUISE
90	648	35	2.98382716049382716049382716049382716049	JUDY	DEAN
91	647	27	3.03636785162287480680061823802163833076	JULIA	MCQUEEN
92	646	1	3.0983591331269349845201238390092879257	PENELOPE	GUINESS
93	646	158	3.10764705882352941176470588235294117647	VIVIEN	BASINGER
94	645	78	2.98689922480620155038759689922480620155	GROUCHO	SINATRA
95	645	187	3.03651162790697674418604651162790697674	RENEE	BALL
96	645	180	2.93108527131782945736434108527131782946	JEFF	SILVERSTONE
97	645	87	3.08612403100775193798449612403100775194	SPENCER	PECK
98	644	108	2.95583850931677018633540372670807453416	WARREN	NOLTE
99	644	129	3.12664596273291925465838509316770186335	DARYL	CRAWFORD
100	644	68	3.11111801242236024844720496894409937888	RIP	WINSLET

```
CREATE VIEW
V_DETAILS_BY_ACTOR
AS
SELECT *
FROM(
SELECT count(*), A.actor_id,
A.first_name, A.last_name,
avg(F.rental_rate)
FROM actor A, film F, film_actor
FA
WHERE A.actor_id = FA.actor_id
AND FA.film_id = F.film_id
GROUP BY A.actor_id,
A.first_name, A.last_name
ORDER BY count(*) DESC)
WHERE rownum <= 100;
```

B)



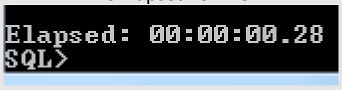
```
Materialized view created.
```

Data tables are the same as above.

```
CREATE MATERIALIZED VIEW
MV_DETAILS_BY_ACTOR
BUILD IMMEDIATE AS
SELECT *
FROM(
SELECT count(*), A.actor_id,
A.first_name, A.last_name,
avg(F.rental_rate)
FROM actor A, film F, film_actor
FA
WHERE A.actor_id = FA.actor_id
AND FA.film_id = F.film_id
GROUP BY A.actor_id,
A.first_name, A.last_name
ORDER BY count(*) DESC)
WHERE rownum <= 100;
```

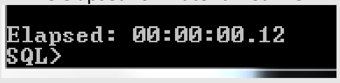
C)

Time Elapsed for View:



```
Elapsed: 00:00:00.28
SQL>
```

Time elapsed for Materialized View:

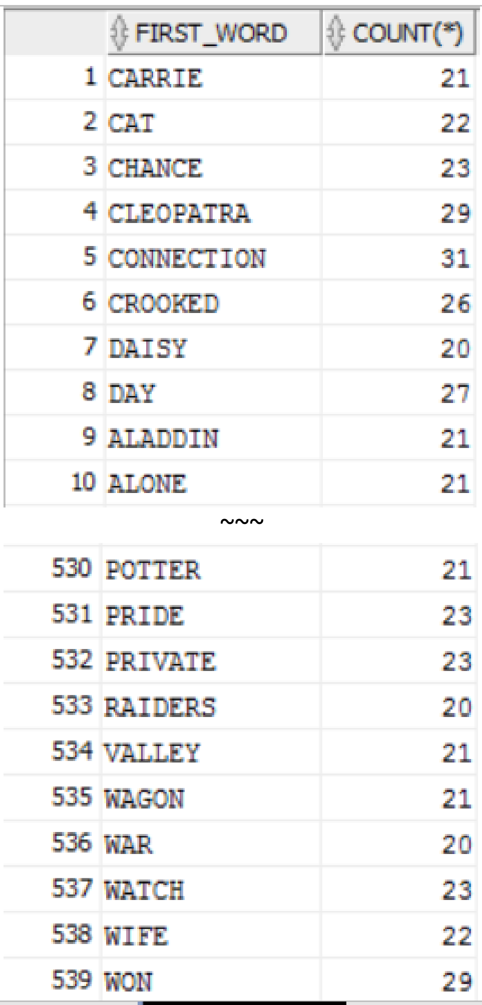


```
Elapsed: 00:00:00.12
SQL>
```

There is a large difference between the two times recorded. The materialised view is quicker to access because it is stored physically on disk, whereas the normal view has to be computed every time it is retrieved.

Task 4 - Indexes

A)



	FIRST_WORD	COUNT(*)
1	CARRIE	21
2	CAT	22
3	CHANCE	23
4	CLEOPATRA	29
5	CONNECTION	31
6	CROOKED	26
7	DAISY	20
8	DAY	27
9	ALADDIN	21
10	ALONE	21
	~~	
530	POTTER	21
531	PRIDE	23
532	PRIVATE	23
533	RAIDERS	20
534	VALLEY	21
535	WAGON	21
536	WAR	20
537	WATCH	23
538	WIFE	22
539	WON	29

```
SELECT DISTINCT(substr(title,1,instr(title,' ',1,1)-1)) AS first_word, count(*)
FROM film
GROUP BY (substr(title,1,instr(title,' ',1,1)-1))
HAVING count(*) >= 20;
```

B)

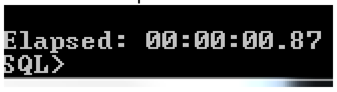
```
Index created.

SQL> _
```

```
CREATE INDEX TITLE_FIRSTWORD
ON film(substr(title,1,instr(title,' ',1,1)-1));
```

This was my choice of index as it improves the search efficiency of a function on the title. The function being, finding the first word of each film title. This is more efficient than creating an index on just title, as we are searching on *the first word of title.*

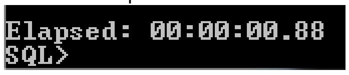
C)

Time elapsed with Index



```
Elapsed: 00:00:00.87
SQL>
```

Time elapsed without Index

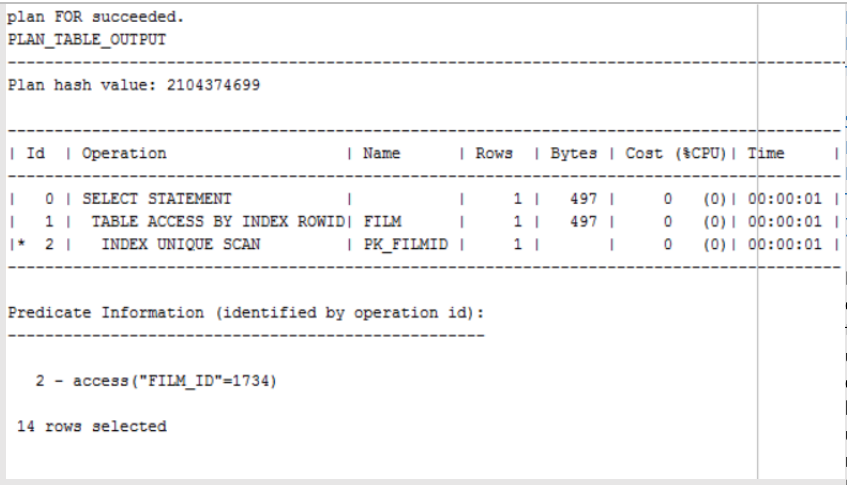


```
Elapsed: 00:00:00.88
SQL>
```

It was observed that the index had not made a substantial difference on the result of the query. The reason for this is that Oracle will try to run whatever is most efficient. At times, running an index may not increase the efficiency. In this case, oracle found it suitable to not implement the index. This can also be observed when running an execution plan. It is also dependant on the density and distribution of the data and how large/small the tables are. In this case, Oracle found it suitable not to use the index.

Task 5 – Execution Plan

A)



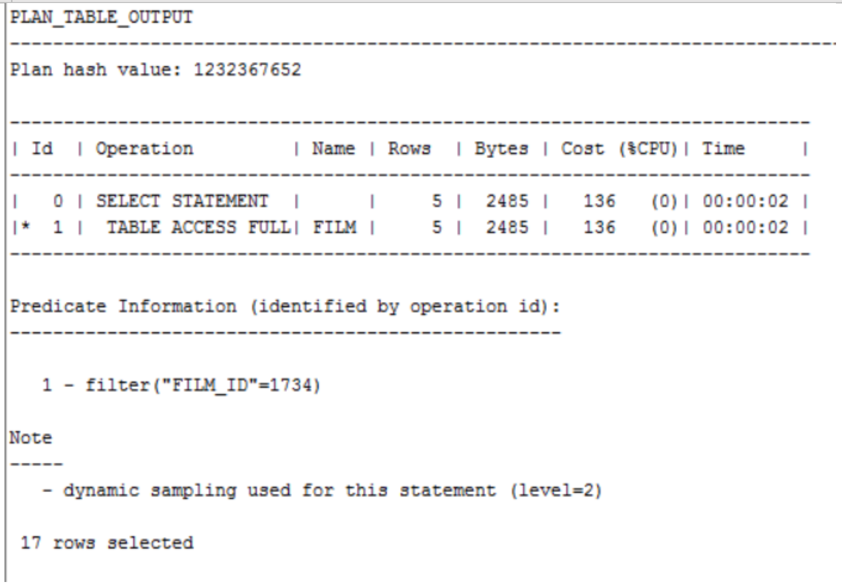
```
plan FOR succeeded.
PLAN_TABLE_OUTPUT
---------------------------------------------------------------------------------------
Plan hash value: 2104374699

---------------------------------------------------------------------------------------
| Id  | Operation                   | Name      | Rows  | Bytes | Cost (%CPU)| Time     |
---------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT            |           |     1 |   497 |     0   (0)| 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID| FILM      |     1 |   497 |     0   (0)| 00:00:01 |
|*  2 |   INDEX UNIQUE SCAN         | PK_FILMID |     1 |       |     0   (0)| 00:00:01 |
---------------------------------------------------------------------------------------

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

   2 - access("FILM_ID"=1734)

 14 rows selected
```

EXPLAIN PLAN FOR SELECT * FROM film WHERE film_id = 1734;

SELECT PLAN_TABLE_OUTPUT FROM TABLE(DBMS_XPLAN.DISPLAY);

In my opinion, this plan is efficient as it does not access the table in full, instead, it uses the primary key constraint. In my opinion, because the primary key is used to scan, the full table is not accessed, and because PK_FILMID is used, it will return only one row.

B)



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

--------------------------------------------------------------------------
| Id  | Operation         | Name | Rows  | Bytes | Cost (%CPU)| Time     |
--------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |      |     5 |  2485 |   136   (0)| 00:00:02 |
|*  1 |  TABLE ACCESS FULL| FILM |     5 |  2485 |   136   (0)| 00:00:02 |
--------------------------------------------------------------------------

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

   1 - filter("FILM_ID"=1734)

Note
-----
   - dynamic sampling used for this statement (level=2)

 17 rows selected
```

ALTER TABLE film DROP CONSTRAINT PK_FILMID;

EXPLAIN PLAN FOR SELECT * FROM film WHERE film_id = 1734;

SELECT PLAN_TABLE_OUTPUT FROM TABLE(DBMS_XPLAN.DISPLAY);

As compared to the plan above, when removing the primary key, oracle access the full table, which takes more time than using the primary key. Overall there is much more CPU cost. Every row is evaluated against the WHERE clause criteria, in my opinion being much more inefficient than using an index.

C)

The main difference between the two plans is that the first one uses a unique index scan and is accessed on ROWID because of the primary key constraint. The first plan accesses the unique index, accesses the table b index row and then feeds the output into the select statement. The second one accesses the full table and feeds it through to the select statement. This is inefficient as every row in film is accessed with the WHERE statement performed on it.
For the first plan with the PK constraint still enabled, it will UNIQUE SCAN the index to evaluate the WHERE clause and returns only one row from the index (That which satisfies the where clause). The table is then accessed by index rowed and the full table is not accessed. The select statement will then only return the rows which satisfy the WHERE clause.