## Task1

### -- Task1.1

ALTER TABLE actor ADD CONSTRAINT PK\_ACTORID PRIMARY KEY (actor\_id);

/\*

Question: Which constraints in Table 1 have been created on these six tables?

\*/

primary key and foreign key

### -- Task1.2

ALTER TABLE actor ADD CONSTRAINT PK\_ACTORID PRIMARY KEY (actor\_id);

ALTER TABLE film ADD CONSTRAINT PK\_FILMID PRIMARY KEY (film\_id);

ALTER TABLE film\_actor ADD CONSTRAINT FK\_FILMID1 FOREIGN KEY (film\_id) REFERENCES film;

ALTER TABLE category ADD CONSTRAINT PK\_CATEGORYID PRIMARY KEY (category\_id );

ALTER TABLE language ADD CONSTRAINT PK\_LANGUAGEID PRIMARY KEY (language\_id );

ALTER TABLE film ADD CONSTRAINT UN\_DESCRIPTION UNIQUE(description);

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 category ADD CONSTRAINT CK\_CATNAME CHECK (name IS NOT NULL);

ALTER TABLE language ADD CONSTRAINT CK\_LANNAME CHECK (name IS NOT NULL);

ALTER TABLE film ADD CONSTRAINT CK\_TITLE CHECK (title IS NOT NULL);

ALTER TABLE film ADD CONSTRAINT CK\_RELEASEYR CHECK (release\_year<=2020);

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\_category ADD CONSTRAINT FK\_CATEGORYID FOREIGN KEY (category\_id ) REFERENCES category(category\_id);

ALTER TABLE film\_category ADD CONSTRAINT FK\_FILMID2 FOREIGN KEY (film\_id ) REFERENCES film(film\_id);

## Task2

### -- 1

create sequence FILM\_ID\_SEQ

start with 20010

increment by 10;

### -- 2

create trigger BI\_FILM\_ID

before insert on film for each row

begin

select FILM\_ID\_SEQ.nextval into :new.film\_id from dual;

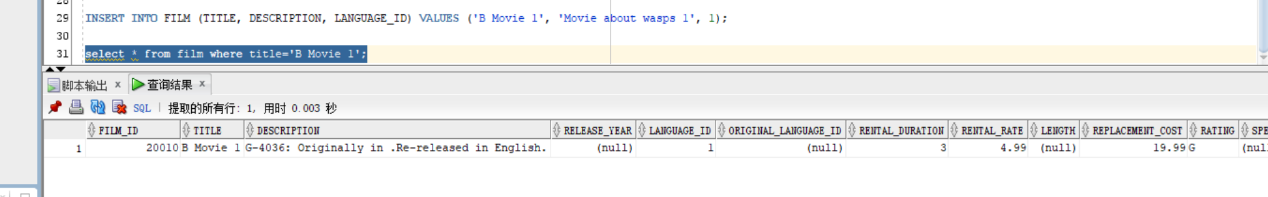
end;

/\*

INSERT INTO FILM (TITLE, DESCRIPTION, LANGUAGE\_ID) VALUES ('B Movie 1', 'Movie about wasps 1', 1);

select \* from film where title='B Movie 1';

\*/



### -- 3

-- drop trigger BI\_FILM\_DESP;

create trigger BI\_FILM\_DESP

before insert on film for each row

begin

select :new.rating||'-'||(select count(\*)+1 from film where rating=:new.rating)||': '||'Originally in '|| (select name

from language where language\_id=:new.original\_language\_id)

||'.'||'Re-released in '||(select name

from language where language\_id=:New.language\_id) ||'.'

into :new.description from dual;

end;

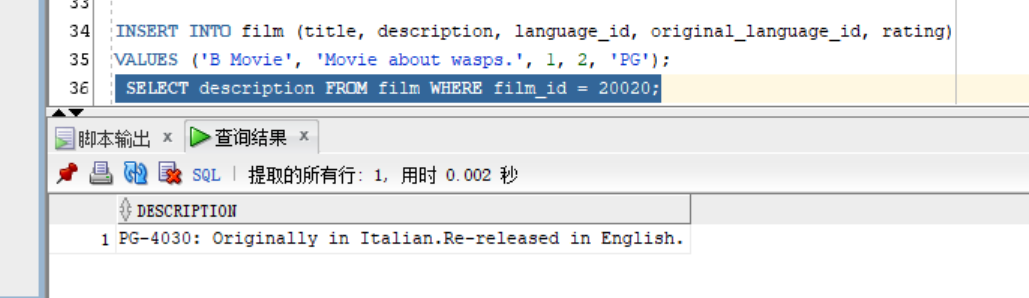
/\*

INSERT INTO film (title, description, language\_id, original\_language\_id, rating)

VALUES ('B Movie', 'Movie about wasps.', 1, 2, 'PG');

SELECT description FROM film WHERE film\_id = 20020;

\*/



## Task3

### -- 1

select f.title, f.length

from film f

join

(

select min(length) as length from film f join film\_category fc on f.film\_id=fc.film\_id

join category c on c.category\_id=fc.category\_id

where c.name='Action'

) t on f.length=t.length;

### -- 2

create or replace view MIN\_ACTION\_ACTORS

as

select distinct a.actor\_id, a.first\_name, a.last\_name

from actor a join film\_actor fa on a.actor\_id=fa.actor\_id

join film f on f.film\_id=fa.film\_id

join (

select f.title, f.length

from film f

join

(

select min(length) as length from film f join film\_category fc on f.film\_id=fc.film\_id

join category c on c.category\_id=fc.category\_id

where c.name='Action'

) t on f.length=t.length

) v1 on v1.title=f.title;

SELECT \* FROM MIN\_ACTION\_ACTORS;

### -- 3

create or replace view V\_ACTION\_ACTORS\_2012

as

select distinct a.actor\_id, a.first\_name, a.last\_name

from actor a join film\_actor fa on a.actor\_id=fa.actor\_id

join film f on f.film\_id=fa.film\_id

join film\_category fc on fc.film\_id=f.film\_id

join category c on c.category\_id=fc.category\_id

where c.name='Action' and f.release\_year=2012;

SELECT \* FROM V\_ACTION\_ACTORS\_2012;

### -- 4

create materialized view MV\_ACTION\_ACTORS\_2012

as

select distinct a.actor\_id, a.first\_name, a.last\_name

from actor a join film\_actor fa on a.actor\_id=fa.actor\_id

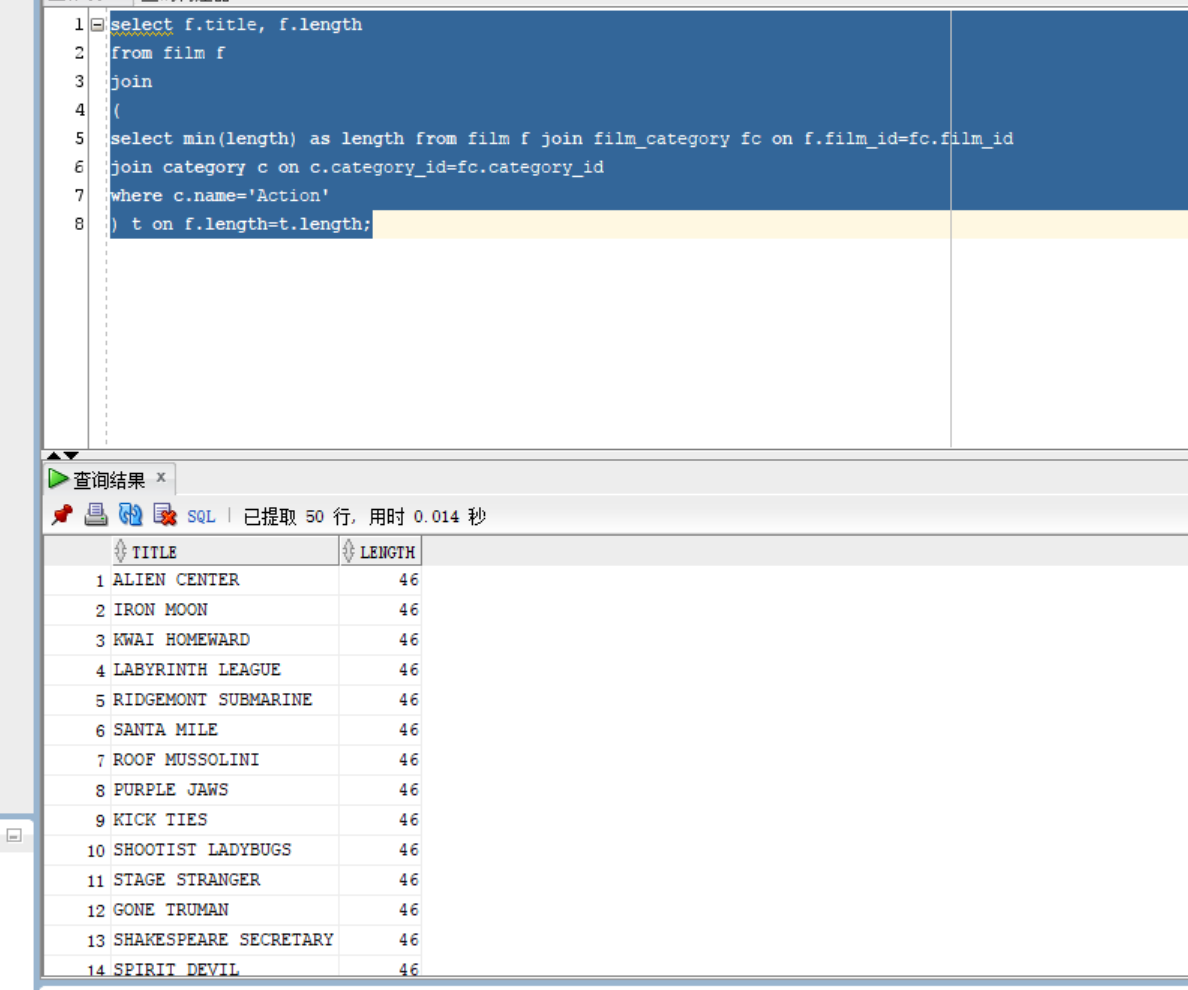
join film f on f.film\_id=fa.film\_id

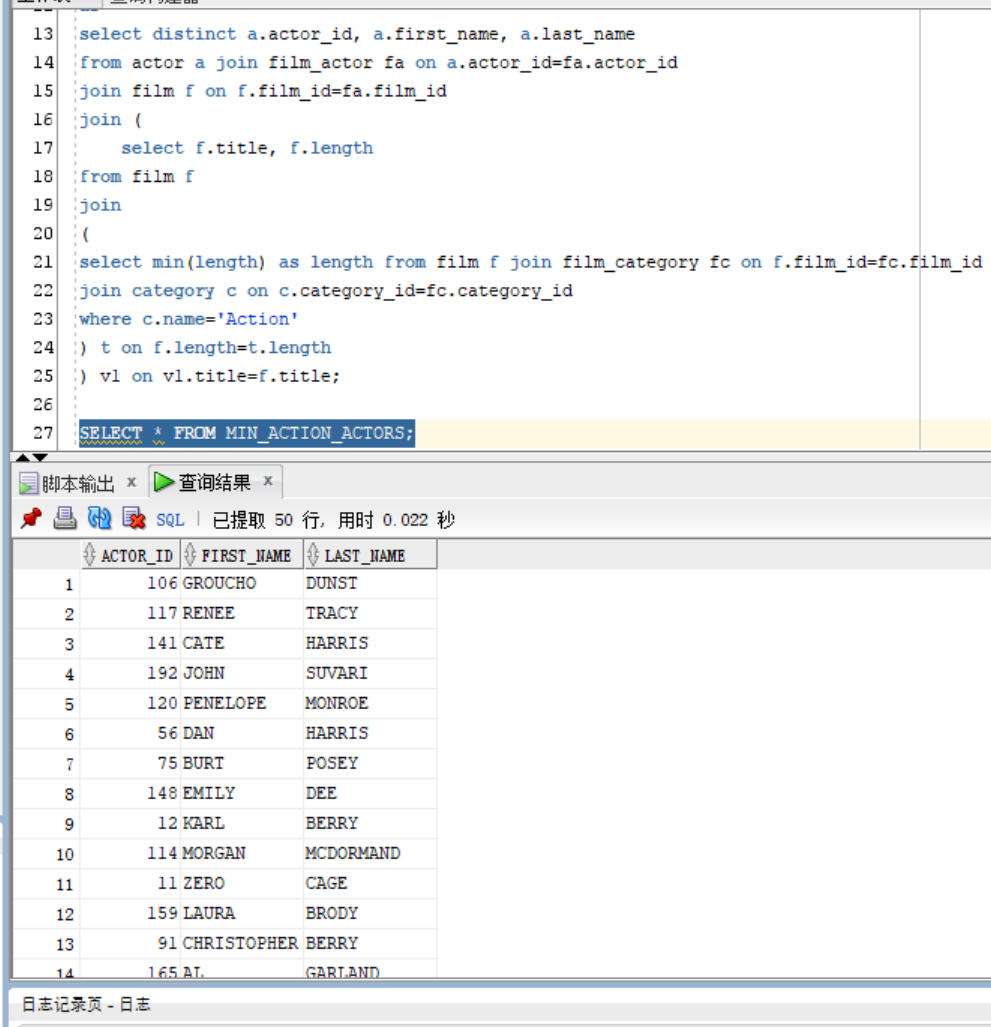
join film\_category fc on fc.film\_id=f.film\_id

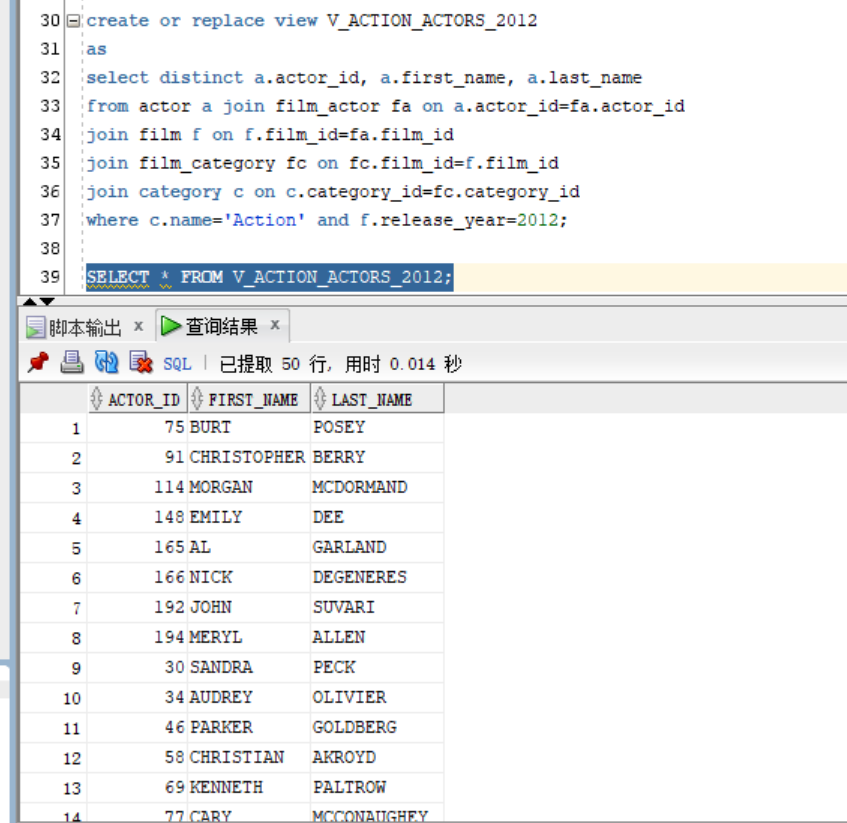
join category c on c.category\_id=fc.category\_id

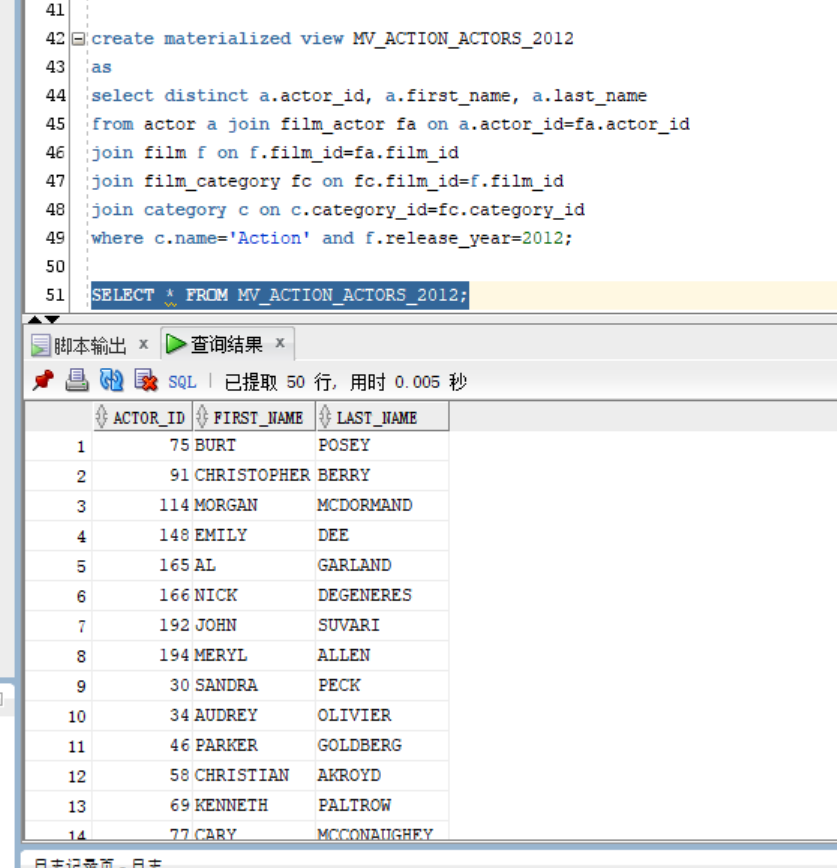
where c.name='Action' and f.release\_year=2012;

SELECT \* FROM MV\_ACTION\_ACTORS\_2012;









### -- 5

SQL> explain plan for SELECT \* FROM V\_ACTION\_ACTORS\_2012;

Explained.

SQL> select \* from table(dbms\_xplan.display());

PLAN\_TABLE\_OUTPUT

--------------------------------------------------------------------------------

Plan hash value: 1938178838

--------------------------------------------------------------------------------

------------------

| Id | Operation | Name | Rows | Bytes | Cost

(%CPU)| Time |

--------------------------------------------------------------------------------

------------------

PLAN\_TABLE\_OUTPUT

--------------------------------------------------------------------------------

| 0 | SELECT STATEMENT | | 121 | 7381 | 223

(1)| 00:00:01 |

| 1 | VIEW | V\_ACTION\_ACTORS\_2012 | 121 | 7381 | 223

(1)| 00:00:01 |

| 2 | HASH UNIQUE | | 121 | 5929 | 223

(1)| 00:00:01 |

|\* 3 | HASH JOIN | | 121 | 5929 | 222

(1)| 00:00:01 |

PLAN\_TABLE\_OUTPUT

--------------------------------------------------------------------------------

|\* 4 | HASH JOIN | | 121 | 3993 | 219

(1)| 00:00:01 |

|\* 5 | HASH JOIN SEMI | | 19 | 494 | 150

(0)| 00:00:01 |

| 6 | MERGE JOIN CARTESIAN| | 303 | 5757 | 139

(0)| 00:00:01 |

|\* 7 | TABLE ACCESS FULL | CATEGORY | 1 | 10 | 3

PLAN\_TABLE\_OUTPUT

--------------------------------------------------------------------------------

(0)| 00:00:01 |

| 8 | BUFFER SORT | | 303 | 2727 | 136

(0)| 00:00:01 |

|\* 9 | TABLE ACCESS FULL | FILM | 303 | 2727 | 136

(0)| 00:00:01 |

| 10 | TABLE ACCESS FULL | FILM\_CATEGORY | 20000 | 136K| 11

(0)| 00:00:01 |

PLAN\_TABLE\_OUTPUT

--------------------------------------------------------------------------------

| 11 | TABLE ACCESS FULL | FILM\_ACTOR | 128K| 879K| 69

(2)| 00:00:01 |

| 12 | TABLE ACCESS FULL | ACTOR | 200 | 3200 | 3

(0)| 00:00:01 |

--------------------------------------------------------------------------------

------------------

Predicate Information (identified by operation id):

PLAN\_TABLE\_OUTPUT

--------------------------------------------------------------------------------

---------------------------------------------------

3 - access("A"."ACTOR\_ID"="FA"."ACTOR\_ID")

4 - access("F"."FILM\_ID"="FA"."FILM\_ID")

5 - access("C"."CATEGORY\_ID"="FC"."CATEGORY\_ID" AND "FC"."FILM\_ID"="F"."FILM\_

ID")

7 - filter("C"."NAME"='Action')

9 - filter("F"."RELEASE\_YEAR"=2012)

28 rows selected.

SQL> explain plan for SELECT \* FROM MV\_ACTION\_ACTORS\_2012;

Explained.

SQL> select \* from table(dbms\_xplan.display());

PLAN\_TABLE\_OUTPUT

--------------------------------------------------------------------------------

Plan hash value: 1015139828

--------------------------------------------------------------------------------

--------------

| Id | Operation | Name | Rows | Bytes | Cost (%CP

U)| Time |

--------------------------------------------------------------------------------

--------------

PLAN\_TABLE\_OUTPUT

--------------------------------------------------------------------------------

| 0 | SELECT STATEMENT | | 109 | 1744 | 3 (

0)| 00:00:01 |

| 1 | MAT\_VIEW ACCESS FULL| MV\_ACTION\_ACTORS\_2012 | 109 | 1744 | 3 (

0)| 00:00:01 |

--------------------------------------------------------------------------------

--------------

8 rows selected.

based on the execution plans above, the MV\_ACTION\_ACTORS\_2012 indeed speed up the query processing.

## Task4

### -- 1

select \*

from film

where instr(description,'Boat')>0 and rownum<=100

order by title;



### -- 2

create index IDX\_BOAT on film (instr(description,'Boat'));

### -- 3

SQL> explain plan for select \*

from film

where instr(description,'Boat')>0 and rownum<=100

order by title;

Explained.

SQL> select \* from table(dbms\_xplan.display());

PLAN\_TABLE\_OUTPUT

--------------------------------------------------------------------------------

Plan hash value: 2306941308

----------------------------------------------------------------------------

| Id | Operation | Name | Rows | Bytes | Cost (%CPU)| Time |

----------------------------------------------------------------------------

| 0 | SELECT STATEMENT | | 100 | 15200 | 16 (7)| 00:00:01 |

| 1 | SORT ORDER BY | | 100 | 15200 | 16 (7)| 00:00:01 |

|\* 2 | COUNT STOPKEY | | | | | |

|\* 3 | TABLE ACCESS FULL| FILM | 100 | 15200 | 15 (0)| 00:00:01 |

----------------------------------------------------------------------------

PLAN\_TABLE\_OUTPUT

--------------------------------------------------------------------------------

Predicate Information (identified by operation id):

---------------------------------------------------

2 - filter(ROWNUM<=100)

3 - filter(INSTR("DESCRIPTION",'Boat')>0)

16 rows selected.

Execution Plan

----------------------------------------------------------

Plan hash value: 2137789089

--------------------------------------------------------------------------------

-------------

| Id | Operation | Name | Rows | Bytes | Cost (%CPU

)| Time |

--------------------------------------------------------------------------------

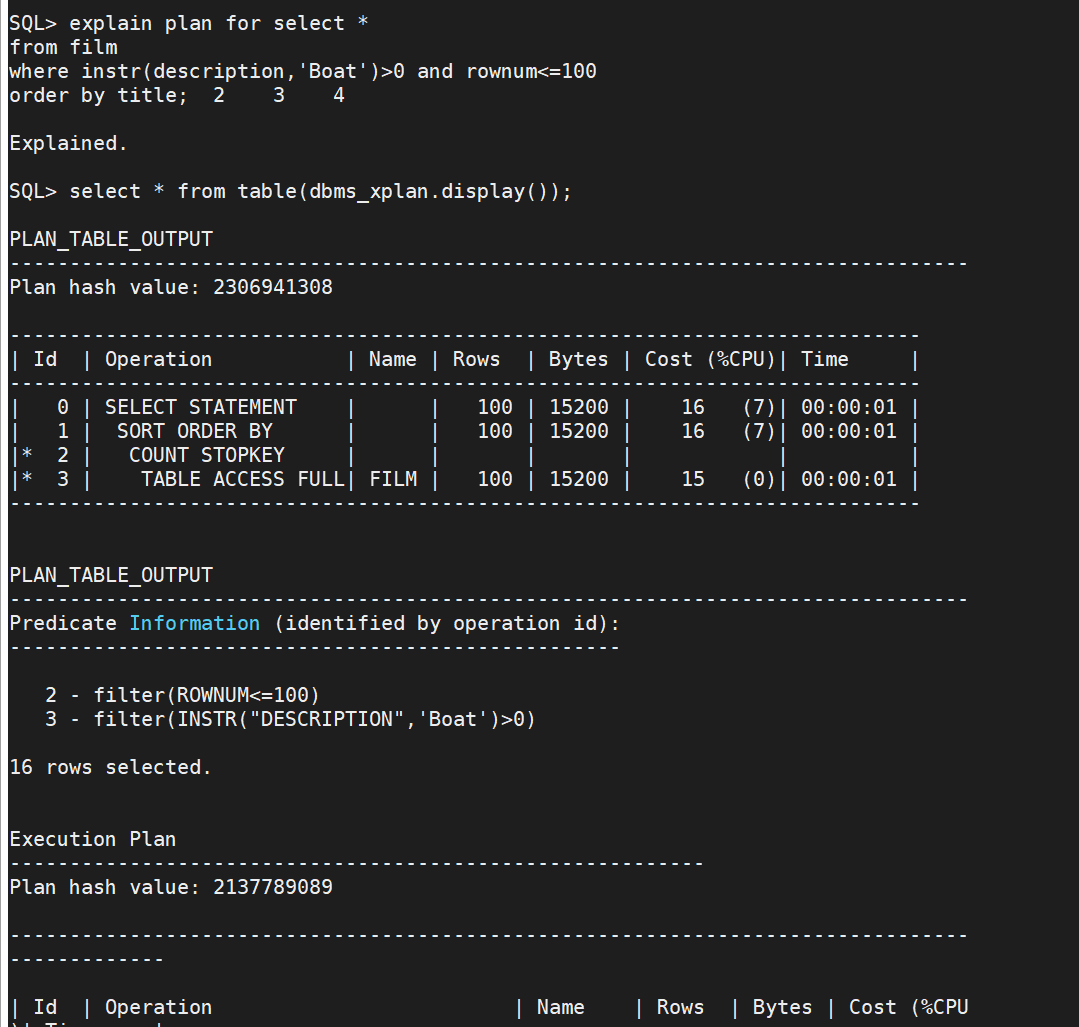
-------------

| 0 | SELECT STATEMENT | | 8168 | 16336 | 29 (0

)| 00:00:01 |

| 1 | COLLECTION ITERATOR PICKLER FETCH| DISPLAY | 8168 | 16336 | 29 (0

)| 00:00:01 |

--------------------------------------------------------------------- 

SQL> explain plan for select \*

from film

where instr(description,'Boat')>0 and rownum<=100

order by title; 2 3 4

Explained.

SQL> select \* from table(dbms\_xplan.display());

PLAN\_TABLE\_OUTPUT

--------------------------------------------------------------------------------

Plan hash value: 2306941308

----------------------------------------------------------------------------

| Id | Operation | Name | Rows | Bytes | Cost (%CPU)| Time |

----------------------------------------------------------------------------

| 0 | SELECT STATEMENT | | 100 | 15200 | 16 (7)| 00:00:01 |

| 1 | SORT ORDER BY | | 100 | 15200 | 16 (7)| 00:00:01 |

|\* 2 | COUNT STOPKEY | | | | | |

|\* 3 | TABLE ACCESS FULL| FILM | 100 | 15200 | 15 (0)| 00:00:01 |

----------------------------------------------------------------------------

PLAN\_TABLE\_OUTPUT

--------------------------------------------------------------------------------

Predicate Information (identified by operation id):

---------------------------------------------------

2 - filter(ROWNUM<=100)

3 - filter(INSTR("DESCRIPTION",'Boat')>0)

16 rows selected.

Execution Plan

----------------------------------------------------------

Plan hash value: 2137789089

--------------------------------------------------------------------------------

-------------

| Id | Operation | Name | Rows | Bytes | Cost (%CPU

)| Time |

--------------------------------------------------------------------------------

-------------

| 0 | SELECT STATEMENT | | 8168 | 16336 | 29 (0

)| 00:00:01 |

| 1 | COLLECTION ITERATOR PICKLER FETCH| DISPLAY | 8168 | 16336 | 29 (0

)| 00:00:01 |

--------------------------------------------------------------------------------

-------------

SQL> create index IDX\_BOAT on film (instr(description,'Boat'));

Index created.

SQL> explain plan for select \*

from film

where instr(description,'Boat')>0 and rownum<=100

order by title; 2 3 4

Explained.

SQL> select \* from table(dbms\_xplan.display());

PLAN\_TABLE\_OUTPUT

--------------------------------------------------------------------------------

Plan hash value: 1934079693

--------------------------------------------------------------------------------

------------------

| Id | Operation | Name | Rows | Bytes | Cost

(%CPU)| Time |

--------------------------------------------------------------------------------

------------------

PLAN\_TABLE\_OUTPUT

--------------------------------------------------------------------------------

| 0 | SELECT STATEMENT | | 100 | 15200 | 5

(20)| 00:00:01 |

| 1 | SORT ORDER BY | | 100 | 15200 | 5

(20)| 00:00:01 |

|\* 2 | COUNT STOPKEY | | | |

| |

| 3 | TABLE ACCESS BY INDEX ROWID BATCHED| FILM | 101 | 15352 | 4

(0)| 00:00:01 |

PLAN\_TABLE\_OUTPUT

--------------------------------------------------------------------------------

|\* 4 | INDEX RANGE SCAN | IDX\_BOAT | 180 | | 2

(0)| 00:00:01 |

--------------------------------------------------------------------------------

------------------

Predicate Information (identified by operation id):

---------------------------------------------------

PLAN\_TABLE\_OUTPUT

--------------------------------------------------------------------------------

2 - filter(ROWNUM<=100)

4 - access(INSTR("DESCRIPTION",'Boat')>0)

17 rows selected.

Execution Plan

----------------------------------------------------------

Plan hash value: 2137789089

--------------------------------------------------------------------------------

-------------

| Id | Operation | Name | Rows | Bytes | Cost (%CPU

)| Time |

--------------------------------------------------------------------------------

-------------

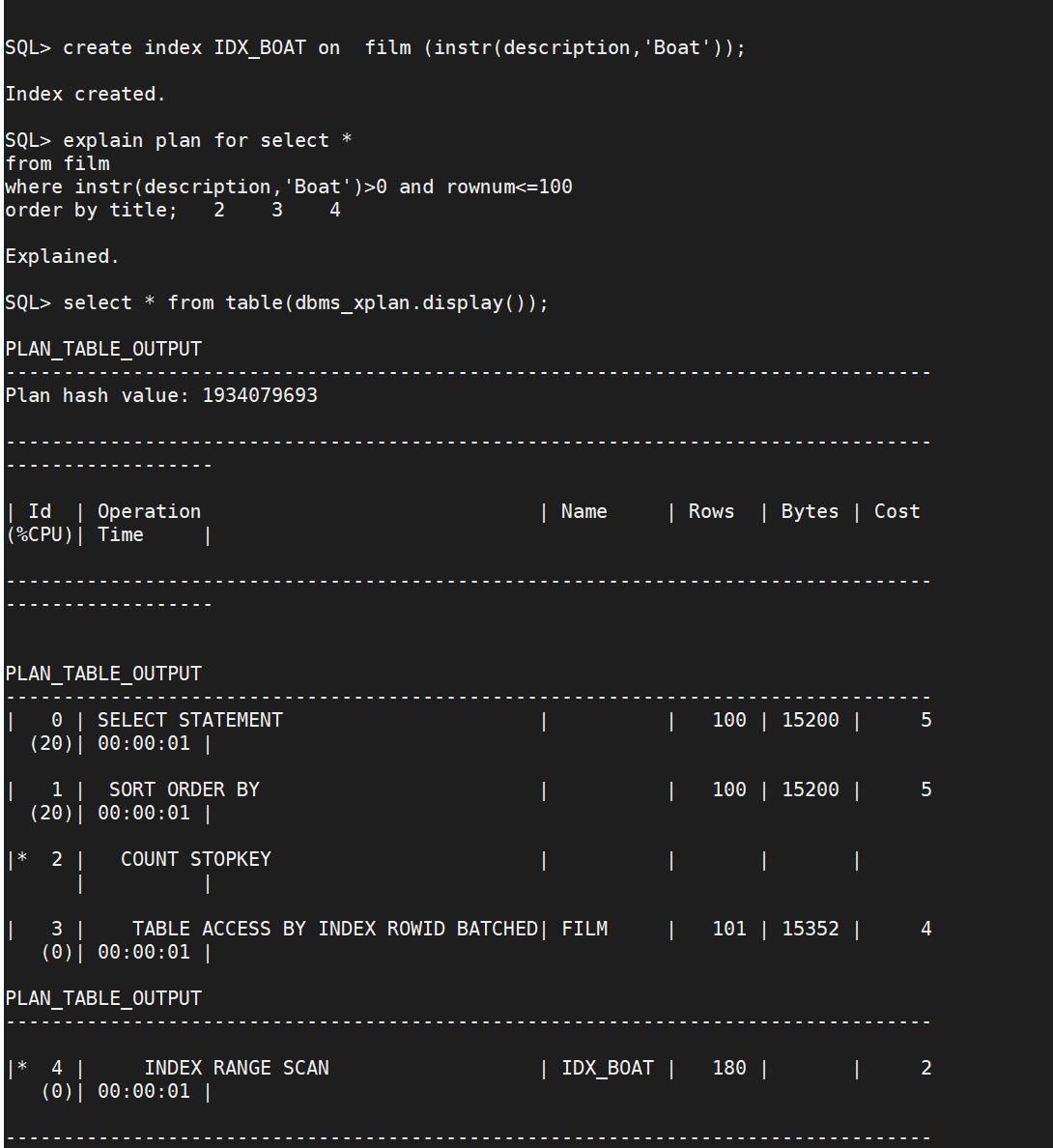
| 0 | SELECT STATEMENT | | 8168 | 16336 | 29 (0

)| 00:00:01 |

| 1 | COLLECTION ITERATOR PICKLER FETCH| DISPLAY | 8168 | 16336 | 29 (0

)| 00:00:01 |

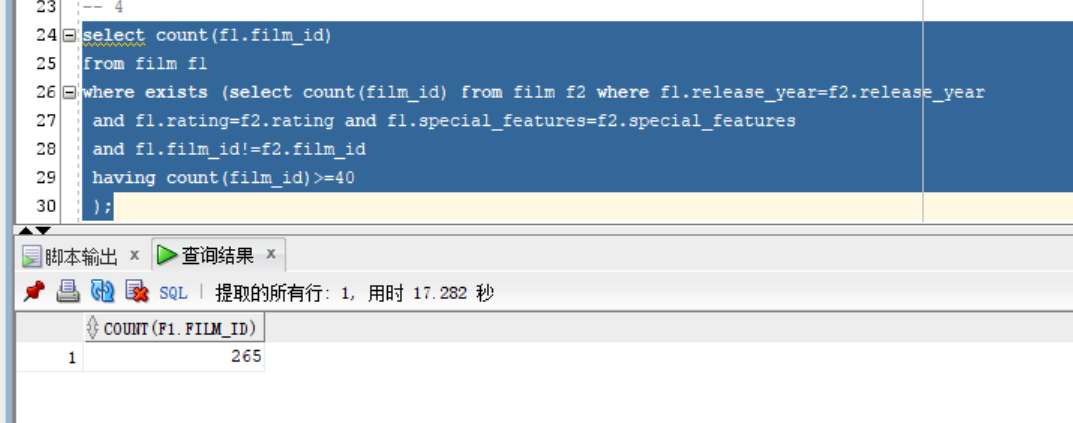
---------------------------------------------------------------------



conclusion: the index indeed speed up the query processing.

1. the first one which doesn’t have an index, use “TABLE ACCESS FULL” plan
2. the second one which created an index on column description, use “TABLE ACCESS BY INDEX ROWID BATCHED’ plan
3. the second one consumes time less than first one.

### -- 4



### -- 5

I think in the three columns release\_year, rating, and special\_features， the column rating and special\_features are suitable for using Hash indexes, and the release\_year is suitable for B+ tree index.

because the rating column, it only has 'G', 'PG', 'PG-13', 'R', 'NC-17' five values and usually use “=” to query records.

and the special\_features column, it only has null, 'Trailers',

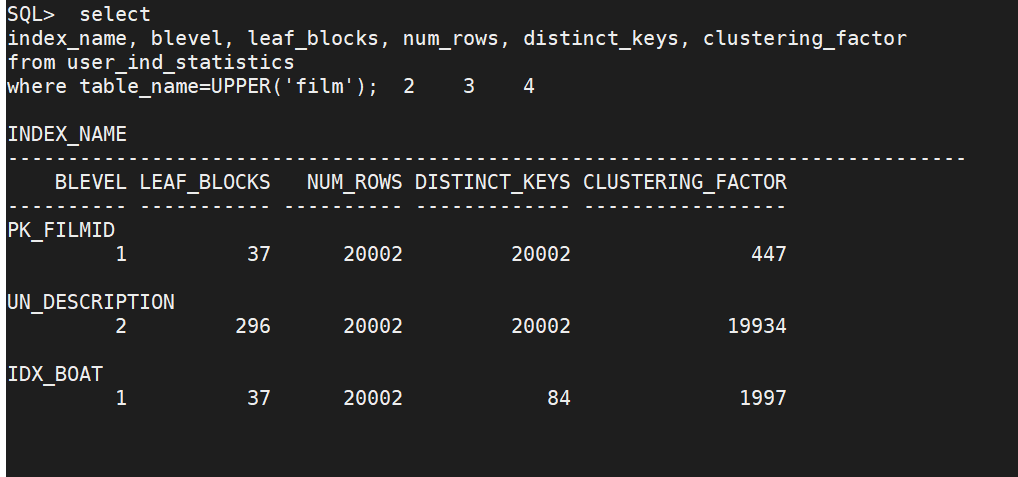
'Commentaries', 'Deleted

Scenes', 'Behind the Scenes' values, and also use “=” to find records.

release\_year column can use “>”,”<” and “>=”, “<=” to query records. so use B+ tree is more suitable.

## Task5

### -- 1



### -- 2

SQL> explain plan for SELECT \* FROM FILM WHERE FILM\_ID > 100;

Explained.

SQL> select \* from table(dbms\_xplan.display());

PLAN\_TABLE\_OUTPUT

--------------------------------------------------------------------------------

Plan hash value: 1232367652

--------------------------------------------------------------------------

| Id | Operation | Name | Rows | Bytes | Cost (%CPU)| Time |

--------------------------------------------------------------------------

| 0 | SELECT STATEMENT | | 19903 | 2993K| 136 (0)| 00:00:01 |

|\* 1 | TABLE ACCESS FULL| FILM | 19903 | 2993K| 136 (0)| 00:00:01 |

--------------------------------------------------------------------------

Predicate Information (identified by operation id):

---------------------------------------------------

PLAN\_TABLE\_OUTPUT

--------------------------------------------------------------------------------

1 - filter("FILM\_ID">100)

13 rows selected.

Execution Plan

----------------------------------------------------------

Plan hash value: 2137789089

--------------------------------------------------------------------------------

-------------

| Id | Operation | Name | Rows | Bytes | Cost (%CPU

)| Time |

--------------------------------------------------------------------------------

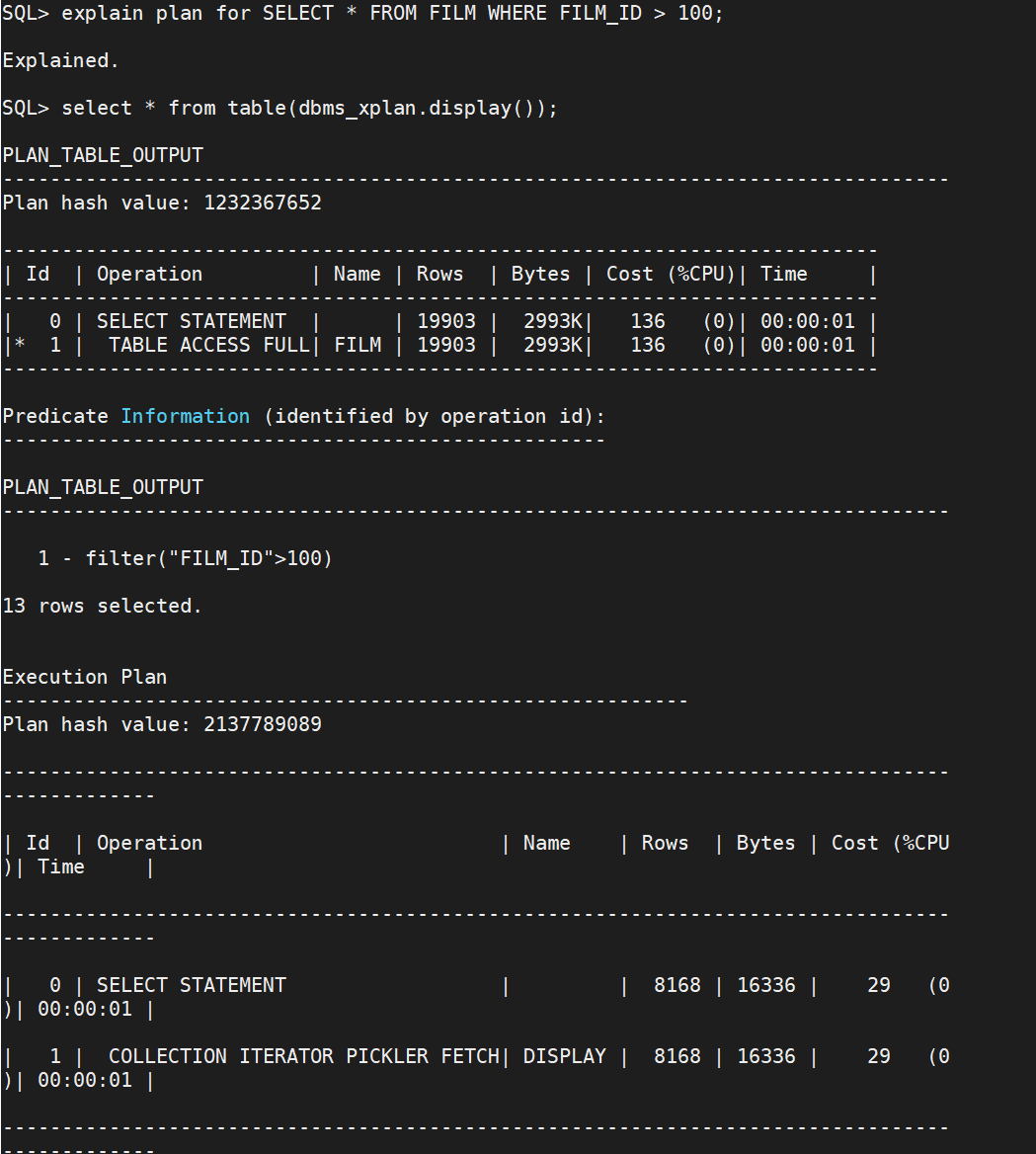
-------------

| 0 | SELECT STATEMENT | | 8168 | 16336 | 29 (0

)| 00:00:01 |

| 1 | COLLECTION ITERATOR PICKLER FETCH| DISPLAY | 8168 | 16336 | 29 (0

)| 00:00:01 |



use full access plan

### -- 3

rows number decreased to 8168 compared to the full table records.

### -- 4

SQL> explain plan for SELECT \* FROM FILM WHERE FILM\_ID > 19990;

Explained.

SQL> select \* from table(dbms\_xplan.display());

PLAN\_TABLE\_OUTPUT

--------------------------------------------------------------------------------

Plan hash value: 1620599584

--------------------------------------------------------------------------------

-----------------

| Id | Operation | Name | Rows | Bytes | Cost (

%CPU)| Time |

--------------------------------------------------------------------------------

-----------------

PLAN\_TABLE\_OUTPUT

--------------------------------------------------------------------------------

| 0 | SELECT STATEMENT | | 30 | 4620 | 3

(0)| 00:00:01 |

| 1 | TABLE ACCESS BY INDEX ROWID BATCHED| FILM | 30 | 4620 | 3

(0)| 00:00:01 |

|\* 2 | INDEX RANGE SCAN | PK\_FILMID | 30 | | 2

(0)| 00:00:01 |

--------------------------------------------------------------------------------

-----------------

PLAN\_TABLE\_OUTPUT

--------------------------------------------------------------------------------

Predicate Information (identified by operation id):

---------------------------------------------------

2 - access("FILM\_ID">19990)

14 rows selected.

Execution Plan

----------------------------------------------------------

Plan hash value: 2137789089

--------------------------------------------------------------------------------

-------------

| Id | Operation | Name | Rows | Bytes | Cost (%CPU

)| Time |

--------------------------------------------------------------------------------

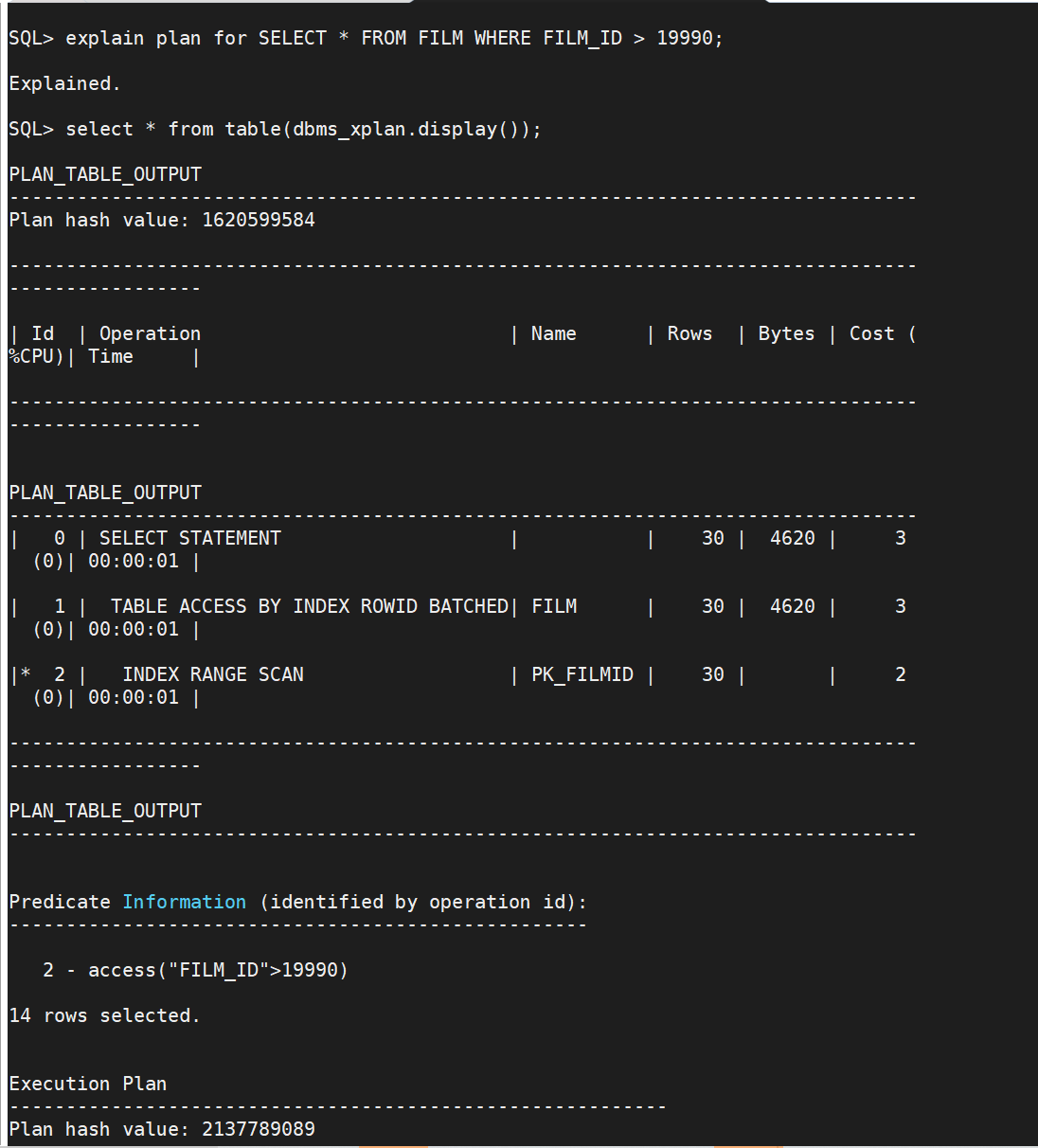
-------------

| 0 | SELECT STATEMENT | | 8168 | 16336 | 29 (0

)| 00:00:01 |

| 1 | COLLECTION ITERATOR PICKLER FETCH| DISPLAY | 8168 | 16336 | 29 (0

)| 00:00:01 |



use index range scan. row number is smaller than the task 5.3.

### -- 5

SQL> explain plan for SELECT \* FROM FILM WHERE FILM\_ID = 100;

Explained.

SQL> select \* from table(dbms\_xplan.display());

PLAN\_TABLE\_OUTPUT

--------------------------------------------------------------------------------

Plan hash value: 2104374699

--------------------------------------------------------------------------------

---------

| Id | Operation | Name | Rows | Bytes | Cost (%CPU)| T

ime |

--------------------------------------------------------------------------------

---------

PLAN\_TABLE\_OUTPUT

--------------------------------------------------------------------------------

| 0 | SELECT STATEMENT | | 1 | 154 | 2 (0)| 0

0:00:01 |

| 1 | TABLE ACCESS BY INDEX ROWID| FILM | 1 | 154 | 2 (0)| 0

0:00:01 |

|\* 2 | INDEX UNIQUE SCAN | PK\_FILMID | 1 | | 1 (0)| 0

0:00:01 |

--------------------------------------------------------------------------------

---------

PLAN\_TABLE\_OUTPUT

--------------------------------------------------------------------------------

Predicate Information (identified by operation id):

---------------------------------------------------

2 - access("FILM\_ID"=100)

14 rows selected.

Execution Plan

----------------------------------------------------------

Plan hash value: 2137789089

--------------------------------------------------------------------------------

-------------

| Id | Operation | Name | Rows | Bytes | Cost (%CPU

)| Time |

--------------------------------------------------------------------------------

-------------

| 0 | SELECT STATEMENT | | 8168 | 16336 | 29 (0

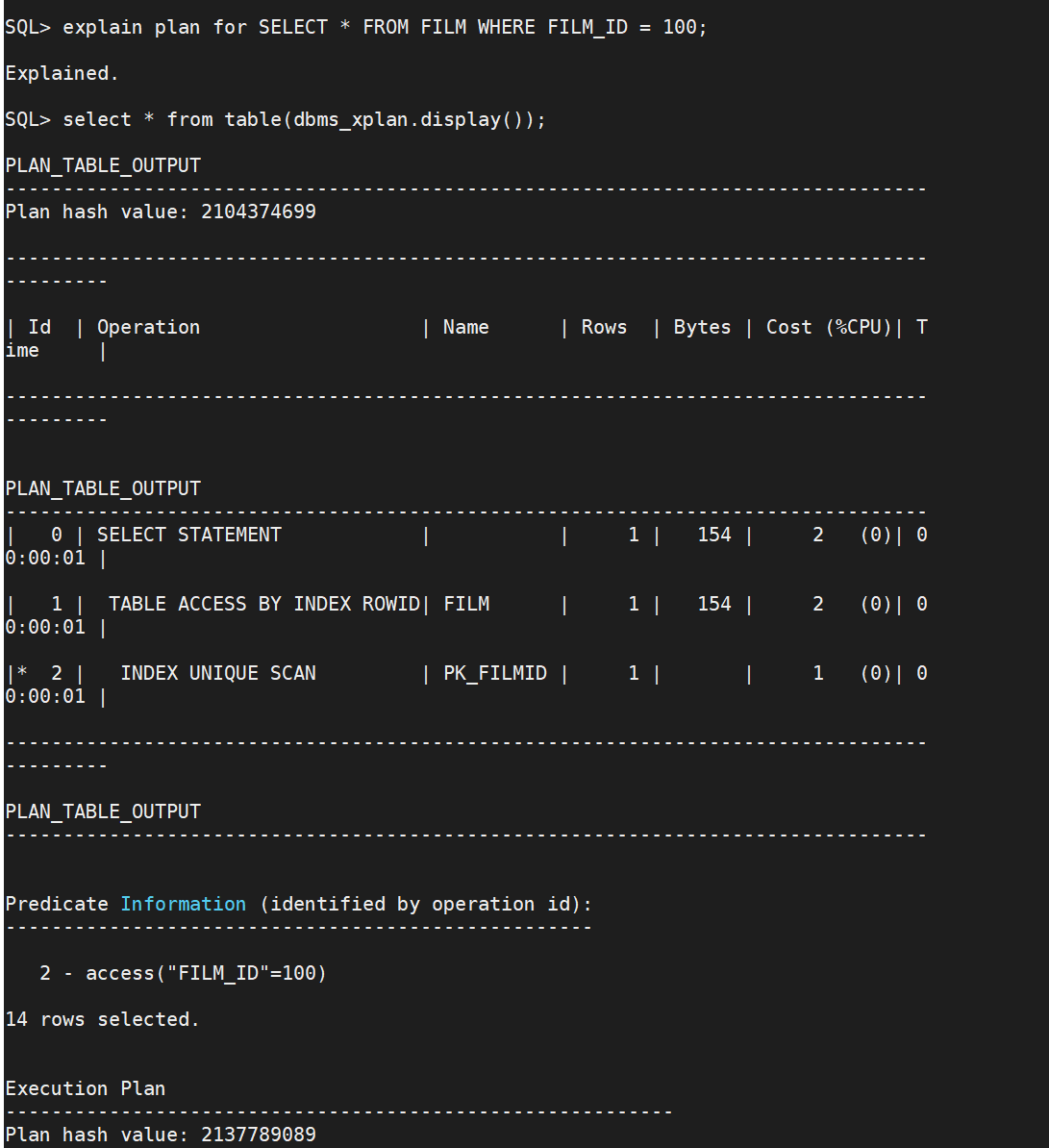
)| 00:00:01 |

| 1 | COLLECTION ITERATOR PICKLER FETCH| DISPLAY | 8168 | 16336 | 29 (0

)| 00:00:01 |

--------------------------------------------------------------------------------

-------------



use index unique scan, the row number is only 1.