

HY-460

Database Management Systems

Assignment 02

AM: 3848

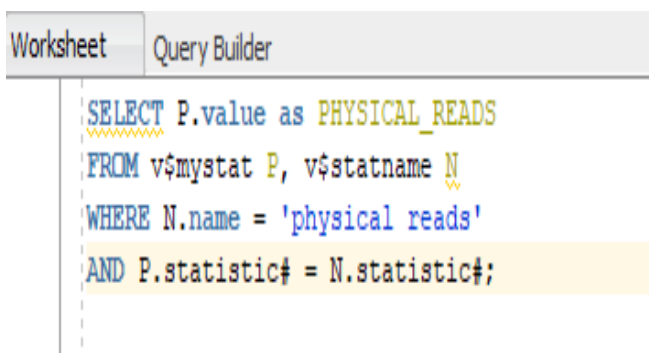
DUE: 29/11/2020

ANTONIOS SYKOUTRIS

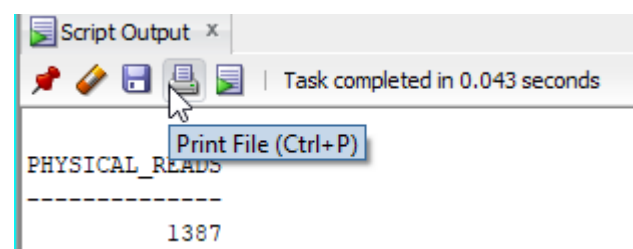
Task 1. Explore Cache Memory

A.1) In this question we are asked to count the physical reads of some queries. Due to the installation dummy scripts I run in order to check if everything is alright **we are starting with 1387 physical reads.** That means that any answer will be according to the following equation:

True Physical Reads = Output Physical Reads – 1.387;



```
SELECT P.value as PHYSICAL_READS
FROM v$mystat P, v$statname N
WHERE N.name = 'physical reads'
AND P.statistic# = N.statistic#;
```

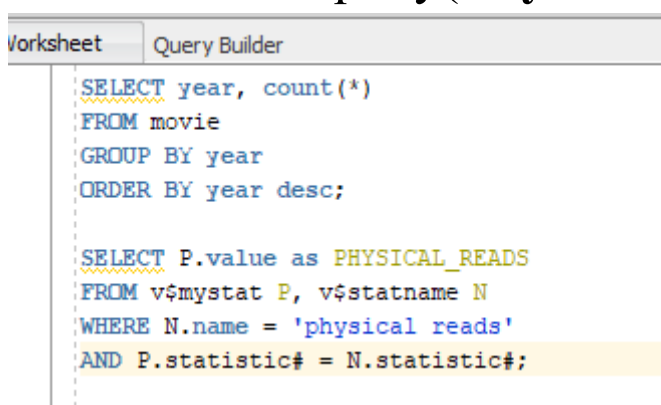


Script Output x | Task completed in 0.043 seconds

PHYSICAL_READS

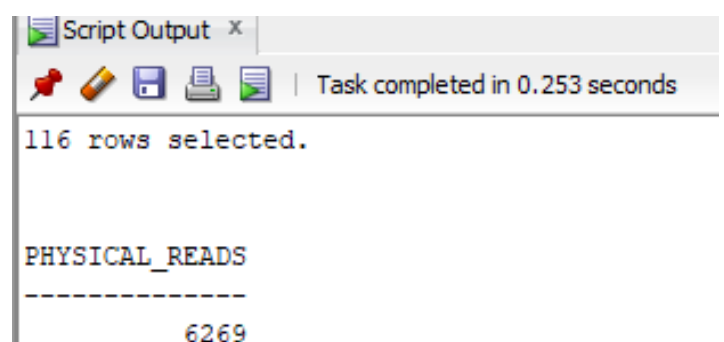
1387

So with that out of the way we are starting with the first query(**Physical Disk Reads(Aq1).sql**)



```
SELECT year, count(*)
FROM movie
GROUP BY year
ORDER BY year desc;

SELECT P.value as PHYSICAL_READS
FROM v$mystat P, v$statname N
WHERE N.name = 'physical reads'
AND P.statistic# = N.statistic#;
```



Script Output x | Task completed in 0.253 seconds

116 rows selected.

PHYSICAL_READS

6269

So True Physical Reads = $6.269 - 1.387 = 4.882$

A.2)

Now its time to do the same exact thing to the second query(**Physical Disk Reads(Aq2).sql**)

```
SELECT companyID, count(*)  
FROM distributedBy  
GROUP BY companyID;
```

```
SELECT P.value as PHYSICAL_READS  
FROM v$mystat P, v$statname N  
WHERE N.name = 'physical reads'  
AND P.statistic# = N.statistic#;
```

PHYSICAL_READS

12237

So True Physical Reads = $12.237 - 6.269 = 5.968$

B.1)

Now its time to find the hit cache ratio of the first query(**Cache_Hit_Ratio(Bq1).sql**)

By using the equation you gave us and by putting the **True_Physical_Reads** we find that:

BEFORE

DB_BLOCK_GETS	CONSISTENT_GETS	PHYSICAL_READS
68	39238	12237

AFTER

DB_BLOCK_GETS	CONSISTENT_GETS	PHYSICAL_READS
68	49238	17115

DB_BLOCK_GETS = **68**

TRUE_CONSISTENT_GETS = 49.238 – 39.238 = **10.000**

True_Physical_Reads = 17.115 – 12.237 = **4.878**

$$cache_hit_ratio = 1 - \frac{dPHYSICAL_READS}{dDB_BLOCK_GETS + dCONSISTENT_GETS}$$

Cache_Hit_Ratio = 51.5% Which is lower than **80%** so we need tuning.

B.2)

I will do the same thing for the second query.

BEFORE

DB_BLOCK_GETS	CONSISTENT_GETS	PHYSICAL_READS
68	55051	18156

AFTER

DB_BLOCK_GETS	CONSISTENT_GETS	PHYSICAL_READS
68	67088	24123

DB_BLOCK_GETS = **68**

TRUE_CONSISTENT_GETS = 67.088 – 55.051 = **12.037**

True_Physical_Reads = 24.123 – 18.151 = **5.972**

$$cache_hit_ratio = 1 - \frac{dPHYSICAL_READS}{dDB_BLOCK_GETS + dCONSISTENT_GETS}$$

Cache_Hit_Ratio = 49.3% Which is lower than **80%** so we need tuning.

C)

Now we are asked to calculate the size of the tables in the database (in blocks). To do this I will run the **Table_Size_In_Database(C).sql**. Below we can see the output of the file I previously mentioned.

<div>BLOCKS-EMPTY_BLOCKS</div> <div>-----</div> <div>5967</div>	table_name = 'DISTRIBUTEDBY';
<div>BLOCKS-EMPTY_BLOCKS</div> <div>-----</div> <div>4878</div>	table_name = 'MOVIE';
<div>BLOCKS-EMPTY_BLOCKS</div> <div>-----</div> <div>4451</div>	table_name = 'PEOPLE';
<div>BLOCKS-EMPTY_BLOCKS</div> <div>-----</div> <div>6813</div>	table_name = 'PLAYS';
<div>BLOCKS-EMPTY_BLOCKS</div> <div>-----</div> <div>8456</div>	table_name = 'PRODUCEDBY';

So If you put large tables into the cache and use them a lot they will stay there and you will have a more efficient system because it will not ask all the time the disk to bring data. That's why in order to improve DBMS performance I think the best choice is the PRODUCEDBY table.

D)

First Query

```
Statistics
-----
0 recursive calls
0 db block gets
0 consistent gets
0 physical reads
0 redo size
```

Second Query

```
Statistics
-----
0 recursive calls
0 db block gets
0 consistent gets
0 physical reads
```

As we can see I have 0 db block(pages) gets because I am reading from the cache(SGA) so it normal for db not to go to ask the disk but get the data from the cache.

Now we should drop the table movie and then proceed to the next task.

Task 2. Explore Indexing Techniques

A)

Below we can see the execution plan for the first query that was given alongside the execution time of it.

			QUERY PLAN
1	0		SELECT STATEMENT
2	1	0	HASH JOIN
3	2	1	TABLE ACCESSHASH MOVIE
4	3	1	TABLE ACCESSFULL PLAYS

```
SQL> set termout on;
SQL> timing stop;
timing for: query1
Elapsed: 00:00:00.61
```

As we can see from the execution time where a taking look a query with a small running time so the differences we can except are negligible. Most of the time the Distinct keyword performs the same and without it but they are few exceptions of running faster or slower.

Now the same thing will be running again but now we will put the DISTINCT statement in the query.

QUERY PLAN		
1	0	SELECT STATEMENT
2	1	0 HASHUNIQUE
3	2	1 HASH JOIN
4	3	2 TABLE ACCESSHASH MOVIE
5	4	2 TABLE ACCESSFULL PLAYS

```
SQL> set termout on;
SQL> timing stop;
timing for: query2
Elapsed: 00:00:00.36
SQL> /
```

This result is showing us that using Distinct we are putting in explain plan HASHUNIQUE. To do that(to remove duplicates first the optimizer has to track them. Using Google to find how it does that, I came to the conclusion that it uses very advanced algorithms in order to make it as fast as possible so it does not slow down that much the query. This can be verified because the difference in running times of the above queries it is small. So without having to deal with the duplicates in this case we are faster using Distinct.

B)

Now we were asked to create a cluster at first and then drop and create again the Movie table. After that we should insert again the values and then identify again the execution plan and the execution time. Below there are the 2 screenshots.

PLAN TABLE

STATEMENT_ID	TIMESTAMP	REMARKS	OPERATION	OPTIONS	OBJECT_NODE	OBJECT_OWNER	OBJECT_NAME	OBJECT_INSTANCE	OBJECT_TYPE	OPTIMIZER	SEARCH_COLUMNS	ID	PARENT_ID	POSITION	OTHER
1 2B1	12-NOV-20	(null)	HASH JOIN	(null)	(null)	(null)	(null)	(null)	(null)	(null)	(null)	1	0	1	(null)
2 2B1	12-NOV-20	(null)	TABLE ACCESS	FULL	(null)	SYSTEM	PLAYS	2 TABLE	ANALYZED	(null)	(null)	2	1	1	(null)
3 2B1	12-NOV-20	(null)	TABLE ACCESS	HASH	(null)	SYSTEM	MOVIE	1 CLUSTER (HASH)	(null)	(null)	(null)	1	3	1	2 (null)
4 2A1	11-NOV-20	(null)	SELECT STATEMENT	(null)	(null)	(null)	(null)	(null)	(null)	ALL_ROWS	(null)	0	(null)	3193	(null)
5 2A1	11-NOV-20	(null)	HASH JOIN	(null)	(null)	(null)	(null)	(null)	(null)	(null)	(null)	1	0	1	(null)
6 2A1	11-NOV-20	(null)	TABLE ACCESS	FULL	(null)	SYSTEM	MOVIE	1 TABLE	ANALYZED	(null)	(null)	2	1	1	(null)
7 2A1	11-NOV-20	(null)	TABLE ACCESS	FULL	(null)	SYSTEM	PLAYS	2 TABLE	ANALYZED	(null)	(null)	3	1	2	(null)
8 2A2	11-NOV-20	(null)	SELECT STATEMENT	(null)	(null)	(null)	(null)	(null)	(null)	ALL_ROWS	(null)	0	(null)	3193	(null)
9 2A2	11-NOV-20	(null)	HASH JOIN	(null)	(null)	(null)	(null)	(null)	(null)	(null)	(null)	1	0	1	(null)
10 2A2	11-NOV-20	(null)	TABLE ACCESS	FULL	(null)	SYSTEM	MOVIE	1 TABLE	ANALYZED	(null)	(null)	2	1	1	(null)
11 2A2	11-NOV-20	(null)	TABLE ACCESS	FULL	(null)	SYSTEM	PLAYS	2 TABLE	ANALYZED	(null)	(null)	3	1	2	(null)
12 2B1	12-NOV-20	(null)	SELECT STATEMENT	(null)	(null)	(null)	(null)	(null)	(null)	ALL_ROWS	(null)	0	(null)	1866	(null)

QUERY PLAN	
1 0	SELECT STATEMENT
2 1 0	HASH JOIN
3 2 1	TABLE ACCESS HASH MOVIE
4 3 1	TABLE ACCESS FULL PLAYS

After

EXECUTION TIME

```
timing for: query1
Elapsed: 00:00:00.25
```

Which is lower than the previous execution(without the cluster)

EXEC_IMPROVEMENT = OLD_EXEC_TIME - NEW_EXEC_TIME

EXEC_IMPROVEMENT = 00:00:00.36

So by using hash cluster AND by using the correct queries we can speed up a lot the execution time of a query because we can easily find the data of a search using hash clusters.

C)

Now we had to drop the PLAYS table then relate the field *movieID* with the cluster generated in **B** and after that show the execution plan and the execution time.

PLAN TABLE

STATEMENT_ID	TIMESTAMP	REMARKS	OPERATION	OPTIONS	OBJECT_NODE	OBJECT_OWNER	OBJECT_NAME	OBJECT_INSTANCE	OBJECT_TYPE	OPTIMIZER	SEARCH_COLUMNS	ID	PARENT_ID	POSITION	OTHER
1 2B1	12-NOV-20	(null)	HASH JOIN	(null)	(null)	(null)	(null)	(null)	(null)	(null)	(null)	1	0	1	(null)
2 2B1	12-NOV-20	(null)	TABLE ACCESS	FULL	(null)	SYSTEM	PLAYS	2	TABLE	ANALYZED	(null)	2	1	1	(null)
3 2B1	12-NOV-20	(null)	TABLE ACCESS	HASH	(null)	SYSTEM	MOVIE	1	CLUSTER (HASH)	(null)	(null)	3	1	2	(null)
4 2C2	12-NOV-20	(null)	SELECT STATEMENT	(null)	(null)	(null)	(null)	(null)	(null)	ALL_ROWS	(null)	0	(null)	2	(null)
5 2C2	12-NOV-20	(null)	HASH JOIN	(null)	(null)	(null)	(null)	(null)	(null)	(null)	(null)	1	0	1	(null)
6 2C2	12-NOV-20	(null)	TABLE ACCESS	HASH	(null)	SYSTEM	MOVIE	1	CLUSTER (HASH)	(null)	(null)	2	1	1	(null)
7 2C2	12-NOV-20	(null)	TABLE ACCESS	HASH	(null)	SYSTEM	PLAYS	2	CLUSTER (HASH)	(null)	(null)	3	1	2	(null)
8 2A1	11-NOV-20	(null)	SELECT STATEMENT	(null)	(null)	(null)	(null)	(null)	(null)	ALL_ROWS	(null)	0	(null)	3193	(null)
9 2A1	11-NOV-20	(null)	HASH JOIN	(null)	(null)	(null)	(null)	(null)	(null)	(null)	(null)	1	0	1	(null)
10 2A1	11-NOV-20	(null)	TABLE ACCESS	FULL	(null)	SYSTEM	MOVIE	1	TABLE	ANALYZED	(null)	2	1	1	(null)
11 2A1	11-NOV-20	(null)	TABLE ACCESS	FULL	(null)	SYSTEM	PLAYS	2	TABLE	ANALYZED	(null)	3	1	2	(null)
12 2A2	11-NOV-20	(null)	SELECT STATEMENT	(null)	(null)	(null)	(null)	(null)	(null)	ALL_ROWS	(null)	0	(null)	3193	(null)
13 2A2	11-NOV-20	(null)	HASH JOIN	(null)	(null)	(null)	(null)	(null)	(null)	(null)	(null)	1	0	1	(null)
14 2A2	11-NOV-20	(null)	TABLE ACCESS	FULL	(null)	SYSTEM	MOVIE	1	TABLE	ANALYZED	(null)	2	1	1	(null)
15 2A2	11-NOV-20	(null)	TABLE ACCESS	FULL	(null)	SYSTEM	PLAYS	2	TABLE	ANALYZED	(null)	3	1	2	(null)
16 2B1	12-NOV-20	(null)	SELECT STATEMENT	(null)	(null)	(null)	(null)	(null)	(null)	ALL_ROWS	(null)	0	(null)	1866	(null)

EXECUTION TIME

```
timing for: query1
Elapsed: 00:00:00.11
SQL> /_
```

Again the execution is faster for the reasons I mentioned in B

D)

Now we had to find the execution plans for the given queries(**Task_2(D.1).sql** , **Task_2(D.2).sql**).

For the first Query

		⚡ QUERY PLAN	
1	0	SELECT STATEMENT	
2	1 0	TABLE ACCESSFULL MOVIE	

For the second Query

		QUERY PLAN	
1	0	SELECT STATEMENT	
2	1 0	TABLE ACCESSFULL PEOPLE	
3	1 0	TABLE ACCESSFULL PEOPLE	

Observation: Both of those queries did a full table scanning to give the appropriate answers. That is not necessary a bad thing if the query is very fast but if it takes a long time to execute then there is a problem with that.

Now we have to create a **unclustered B+-Tree** for the *year* field of the table *MOVIE*. We expect the newly created explain table not to do a full scan but it might also do one if the query optimizer thinks that is easier to just read the whole table rather than looking for the relevant rows, then picking them off the disk.

```

1 0    TABLE ACCESS
FULL MOVIE

0    SELECT STATEMENT

1 0    TABLE ACCESS
FULL MOVIE

```

It is analyzed but still the optimizer prefer to ignore the

index and perform a full table scan even tho I have analyzed it and correctly made the index.

Now we have to create a **unclustered B+-Tree** for the *birthYear* field of the table *PEOPLE*.

	QUERY PLAN		
1	1	0	TABLE ACCESSFULL PEOPLE
2	0		SELECT STATEMENT
3	1	0	TABLE ACCESSFULL PEOPLE

The same thing as above happened. It completely ignored the index and due to reading the lectures and searching online this actually can happen quite often and only if it is a must the full table scan will be replaced by the index.

To prove my point I did a select query using a hint in order to force the index activation. Below there are the results of my action.

Worksheet

Query Builder

```

SELECT /*+ INDEX(a, idx_birthYear) */ a.* FROM PEOPLE a
WHERE birthYear > 1945

```

Query Result x

Autotrace x

SQL HotSpot

0.437 seconds

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST	LAST_CR_BUFFER_GETS	LAST_ELAPSED_TIME
SELECT STATEMENT					136965	198
TABLE ACCESS	PEOPLE	BY INDEX ROWID BATCHED		170641	136965	198
INDEX	IDX_BIRTHYEAR	RANGE SCAN		170641	362	51
Access Predicates						
BIRTHYEAR > 1945						

Because it made a RANGE scan, it means that the Cost Based Optimizer doesn't think the index is very useful. So that's why it makes a full scan instead.(I also put a Dracula theme on SQL Developer because I hate light themes).

E)

For this question I used(Task_2(E.1).sql , Task_2(E.2).sql).

First Query

QUERY PLAN		
1	0	SELECT STATEMENT
2	1 0	TABLE ACCESSFULL MOVIE

Second

QUERY PLAN		
1	0	SELECT STATEMENT
2	1 0	HASH JOIN
3	2 1	TABLE ACCESSHASH MOVIE
4	3 1	TABLE ACCESSFULL PLAYS

As we can see it shows a full table access in the first query but in the second one it uses Hash not full so it uses the index of B+-Tree because it is performing a range scan and for range scan B+-Tree is very beneficial.

F)

In this question we should make a new index for the movieTitle field (idx_movieTitle). To do this I used the file(Task_2(F).sql).

			QUERY PLAN
1	0		SELECT STATEMENT
2	1	0	NESTED LOOPS
3	2	1	TABLE ACCESS BY INDEX ROWID MOVIE
4	3	2	INDEX UNIQUE SCAN IDX_MOVIE_TITLE
5	4	1	TABLE ACCESS FULL PRODUCED BY

As it can be seen it used the idx _ movieTitle (so the non cluster was being used) because we put 2 tables data inside the same block (MOVIE, PRODUCED BY) and if it is under heavy data retrieval unclustered indexes can speed up a lot the searching process (we are not scanning with pk so the search would take a lot of time without index).

G)

Now we are asked to run the query that was given and after that to produce the explain plan and also calculate the execution time.

```
QUERY PLAN
-----
0  SELECT STATEMENT

1 0  HASH JOIN

2 1  TABLE ACCESS
FULL PEOPLE

3 1  TABLE ACCESS
FULL PLAYS
```

```
SQL> set termout on;
SQL> timing stop;
timing for: query1
Elapsed: 00:00:27.17
SQL> /
```

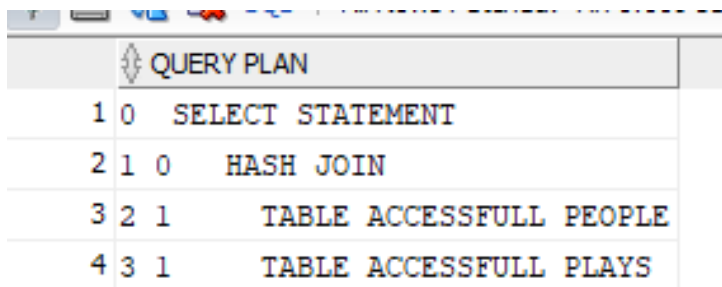
(i) Now I drop the idx_birthYear and repeat the above actions

	QUERY PLAN
1 0	SELECT STATEMENT
2 1 0	HASH JOIN
3 2 1	TABLE ACCESSFULL PEOPLE
4 3 1	TABLE ACCESSFULL PLAYS

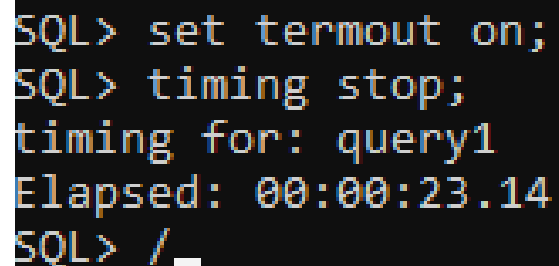
```
SQL> set termout on;
SQL> timing stop;
timing for: query1
Elapsed: 00:00:22.29
SQL> /
```

As we can see from the screenshots above we were having 5 seconds less when we didn't have the idx_birthYear.

(ii) Now I have to create a bitmap index on the birthYear field and repeat the above actions.



QUERY PLAN			
1	0	SELECT STATEMENT	
2	1	0	HASH JOIN
3	2	1	TABLE ACCESSFULL PEOPLE
4	3	1	TABLE ACCESSFULL PLAYS



```
SQL> set termout on;
SQL> timing stop;
timing for: query1
Elapsed: 00:00:23.14
SQL> /
```

As we can see (ii) is slightly slower than (I) but faster than the first(G). The reason for that is because bitmap uses a lot of resources to be able to function so that's why we see the small performance loss. On the other hand one huge advantage of using a bitmap index over a b+tree index is storage (it is less storage hungry).

H)

In the last question we are asked to define a clustered B+-Tree on the *birthYear* field. Afterwards we should drop the People table and then execute the query of G.

		QUERY PLAN
1	0	SELECT STATEMENT
2	1 0	HASH JOIN
3	2 1	TABLE ACCESSCLUSTER PEOPLE
4	3 2	INDEXRANGE SCAN IDX_CBIRTHYEAR
5	4 1	TABLE ACCESSFULL PLAYS

```
SQL> set termout on;  
SQL> timing stop;  
Timing for: query1  
Elapsed: 00:00:26.87  
SQL> /
```

As we can see if we compare the 2 explain plans we can see(row 3 was added) that because we are performing a range scan on a non primary key value clustered B+-Tree index was used in order to improve performance. We can also see that it is faster than the unclustered B+-Tree index because the unclustered one must duplicate values in order to map them with pointers(known as overhead cost) and that cost is responsible for the difference of speed we are experiencing. If we had made an insert here instead of search unclustered would be faster because we would just need to map the new insertion with a pointer rather than having rearrange the data.

Thanks a lot for reading my report and stay safe.