CS301 Project Report Phase-B

Course_id : CS301

Department: Computer Science and Engineering

Under the Guidance of: Dr. Viswanath Gunturi

Submitted by: Muskan Gupta entry no.: 2019csb1100

&

Praveen Kumar entry no.: 2019csb1108

PART-A

1)

QUERIES

Q(1):
```
SELECT name FROM tablemovie WHERE imdbscore < 2;
```

Q(2):
```
SELECT name FROM tablemovie WHERE imdbscore between 1.5 and 4.5;
```

Q(3):
```
SELECT name FROM tablemovie WHERE year between 1900 and 1990;
```

Q(4):
```
SELECT name FROM tablemovie WHERE year between 1990 and 1995;
```

Q(5):
```
SELECT * FROM tablemovie WHERE prod_company < 50;
```

Q(6):
```
SELECT * FROM tablemovie WHERE prod_company > 20000;
```

2)

Q(1):

OUTPUT:

```
 Bitmap Heap Scan on tablemovie  (cost=2254.18..12132.59 rows=202033 width=11) (actual time=34.205..124.471 rows=199905 loops=1)
   Recheck Cond: (imdbscore < 2)
   Heap Blocks: exact=7353
   ->  Bitmap Index Scan on idx_movie_imdbscore  (cost=0.00..2203.67 rows=202033 width=0) (actual time=33.273..33.273 rows=199905 loops=1)
         Index Cond: (imdbscore < 2)
 Planning Time: 2.343 ms
 Execution Time: 128.382 ms
(7 rows)
```

EXPLANATION:

The query optimizer has chosen to use a two-step plan here. The lower node for planning visits an index to find out about the locations of rows where it

matches the Index Cond: (imdbscore <2) and upper node for planning fetches the rows from the table. It could have used a sequential scan also as it is more expensive to fetch rows separately but in this case we do not have to visit all the pages of the table which costs less than the sequential scan.Two plan levels are used because the upper node for planning sorts the row locations which are identified by the index into physical order before it is reading them.It is done to minimize the cost of separate fetches. Here, the "bitmap" mentioned in the node names is the mechanism that is responsible for the work of sorting.
As the optimiser predicts that the cardinality of this query will be less so the query optimizer has skipped using the B+ trees.

Q(2):

OUTPUT:

```
Gather  (cost=1000.00..17186.33 rows=5000 width=11) (actual time=13.849..296.522
rows=600338 loops=1)
   Workers Planned: 2
   Workers Launched: 2
   ->  Parallel Seq Scan on tablemovie  (cost=0.00..15686.33 rows=2083 width=11)
(actual time=0.030..108.774 rows=200113 loops=3)
         Filter: (((imdbscore)::numeric >= 1.5) AND ((imdbscore)::numeric <= 4.5))
         Rows Removed by Filter: 133221
 Planning Time: 0.574 ms
 Execution Time: 313.954 ms
(8 rows)
```

EXPLANATION:

First of all the master divides the work into some appropriate workers. Then, the workers work on sequential scan parallely known as parallel seq scan and use the filter to remove the tuples which do not satisfy the condition. And then the master gathers the results and outputs the required tuples. Here we have to solve a range query so we need to use the parallel seq scan. Also it uses only the suitable number of workers i.e increase in workers may not mean better performance .

Q(3):

OUTPUT:

```
Seq Scan on tablemovie  (cost=0.00..22353.00 rows=890262 width=11) (actual
time=0.020..138.389 rows=901011 loops=1)
   Filter: ((year >= 1900) AND (year <= 1990))
   Rows Removed by Filter: 98989
 Planning Time: 62.898 ms
 Execution Time: 162.256 ms
(5 rows)
```

EXPLANATION:

Here, the query optimiser predicts the total rows to be very large and also the actual rows are =901011 which is approx 90% of total rows present in the tablemovie. So, it chose the full seq scan on the tablemovie where it uses a filter and removes the rows which do not satisfy the condition. When we have to access a large fraction of blocks it preferably chooses seq scan over other index range scans. This is because full table scans use larger I/O calls, and making fewer large I/O calls is less costlier than making many smaller calls.

Q(4):

OUTPUT:

```
Bitmap Heap Scan on tablemovie  (cost=820.92..9070.26 rows=59756 width=11) (actual
time=128.313..145.441 rows=59459 loops=1)
   Recheck Cond: ((year >= 1990) AND (year <= 1995))
   Heap Blocks: exact=7351
   ->  Bitmap Index Scan on idx_movie_year  (cost=0.00..805.99 rows=59756 width=0)
(actual time=126.024..126.024 rows=59459 loops=1)
         Index Cond: ((year >= 1990) AND (year <= 1995))
 Planning Time: 0.225 ms
 Execution Time: 147.107 ms
(7 rows)
```

EXPLANATION:

The query optimizer has chosen to use a two-step plan here. The lower node for planning visits an index to find out about the locations of rows where it matches the Index Cond: ((year >= 1990) AND (year <= 1995))and upper node for planning fetches the rows from the table. It could have used a

sequential scan also as it is more expensive to fetch rows separately but in this case we do not have to visit all the pages of the table which costs less than the sequential scan.Two plan levels are used because the upper node for planning sorts the row locations which are identified by the index into physical order before it is reading them.It is done to minimize the cost of separate fetches. Here, the "bitmap" mentioned in the node names is the mechanism that is responsible for the work of sorting.
As the cardinality of this query will be less so the query optimizer has skipped using the B+ trees.

Q(5):

OUTPUT:

```
 Bitmap Heap Scan on tablemovie  (cost=9.44..1957.42 rows=647 width=27) (actual
time=10.200..11.111 rows=586 loops=1)
   Recheck Cond: (prod_company < 50)
   Heap Blocks: exact=562
   ->  Bitmap Index Scan on idx_movie_pcid  (cost=0.00..9.28 rows=647 width=0) (actual
time=0.159..0.160 rows=586 loops=1)
         Index Cond: (prod_company < 50)
 Planning Time: 158.980 ms
 Execution Time: 11.223 ms
(7 rows)
```

EXPLANATION:

The query optimizer has chosen to use a two-step plan here. The lower node for planning visits an index to find out about the locations of rows where it matches the Index Cond: (prod_company < 50) and the upper node for planning fetches the rows from the table. It could have used a sequential scan also as it is more expensive to fetch rows separately but in this case we do not have to visit all the pages of the table which costs less than the sequential scan.Two plan levels are used because the upper node for planning sorts the row locations which are identified by the index into physical order before it is reading them.It is done to minimize the cost of separate fetches. Here, the "bitmap" mentioned in the node names is the mechanism that is responsible for the work of sorting.
As the cardinality of this query will be less so the query optimizer has skipped using the B+ trees.

Q6):

OUTPUT:

```
Seq Scan on tablemovie  (cost=0.00..19853.00 rows=748095 width=27) (actual
time=2.040..738.983 rows=750653 loops=1)
   Filter: (prod_company > 20000)
   Rows Removed by Filter: 249347
 Planning Time: 83.993 ms
 Execution Time: 765.775 ms
(5 rows)
```

EXPLANATION:

Here, the query optimiser predicts the total rows to be very large and also the actual rows are =750653 which is approx 75% of total rows present in the tablemovie. So, it chose the full seq scan on the tablemovie where it uses a filter and removes the rows which do not satisfy the condition. When we have to access a large fraction of blocks it preferably chooses seq scan over other index range scans. This is because full table scans use larger I/O calls, and making fewer large I/O calls is less costlier than making many smaller calls.

3)

- In **Q1** of **2)** it used the index on imdbscore. The optimiser uses the statistics to determine the selectivity of the predicates. As the optimiser estimates the number of rows to be approx 20% of the total rows. So the cardinality is low for this query.It chooses to use the index as it predicts it will need not to visit all the pages. So the reason behind choosing index is the low cardinality and it does not need to visit all the pages of the table.

- In **Q2** of **2)** it does not use the index because in it we have to find a range of imdbscore tuples where the parallel seq scan works the best using the filters, so due to that reason it chooses the parallel

sequential scan and finally gathers them. Also the cardinality was high, so better to avoid the use of index.

4)

- In **Q3** of **2)** it doesn't use the index on YR because it predicts high cardinality of the query. The cardinality is approx 90%. So, use of index would not be a better choice, also we are accessing a huge fraction of the table, so it is better to use the seq scan as used by the optimiser.When we have to access a large fraction of blocks it preferably chooses seq scan over other index range scans. This is because full table scans use larger I/O calls, and making fewer large I/O calls is less costlier than making many smaller calls.

- In **Q4** of **2)** it used index on YR because it predicts the low cardinality of the query. The cardinality is approx 6%. So, it does not have to visit all the pages of the table, so it chooses to use the index. A seq scan would not be a good choice to search for only 6% tuples.

5)

- In **Q5** of **2)** it used the index. The reason is low cardinality and less number of required access to the pages of the table. The cardinality is approx 0.06 which means very less fraction of the disk pages required. So here it uses the index.

- In **Q6** of **2)** it doesn't use the index. It used the seq scan because it estimates the cardinality quite very high, also the cardinality is approx 75%. Due to high cardinality a seq scan is better than to use an index.When we have to access a large fraction of blocks it preferably chooses seq scan over other index range scans. This is because full table scans use larger I/O calls, and making fewer large I/O calls is less costlier than making many smaller calls.

Part B

(2)

Q1:

QUERY:

```
select tableactor.name,tablemovie.name from
tableactor,tablemovie,tablecasting where tableactor.a_id<50 and
tableactor.a_id=tablecasting.a_id and
tablemovie.m_id=tablecasting.m_id;
```

OUTPUT:

```
Nested Loop  (cost=1.28..3355.27 rows=640 width=27) (actual time=40.334..10053.458
rows=682 loops=1)
  ->  Nested Loop  (cost=0.85..3065.61 rows=640 width=20) (actual
time=25.607..6560.531 rows=682 loops=1)
        ->  Index Scan using idx_actor_aid on tableactor  (cost=0.42..9.26 rows=48
width=20) (actual time=0.476..0.538 rows=49 loops=1)
              Index Cond: (a_id < 50)
        ->  Index Scan using idx_casting_aid on tablecasting  (cost=0.43..63.52 rows=15
width=8) (actual time=14.686..133.837 rows=14 loops=49)
              Index Cond: (a_id = tableactor.a_id)
  ->  Index Scan using idx_movie_mid on tablemovie  (cost=0.42..0.45 rows=1
width=15) (actual time=5.110..5.110 rows=1 loops=682)
        Index Cond: (m_id = tablecasting.m_id)
 Planning Time: 467.538 ms
 Execution Time: 10055.011 ms
(10 rows)
```

EXPLANATION:

Here two nested loops are being used. First of all an index scan runs on the tableactor with the index condition (a_id<50). Then an index scan is used on tablecasting with index condition (a_id = tableactor.a_id). And then they are joined through an inner nested loop. Then an index scan on tablemovie gets executed with index condition (m_id = tablecasting.m_id). And then this result is joined with the previous result of the inner loop with the help of a nested loop. Index scans have been used because the optimizer estimated the less cardinality and it will not require to visit all pages of the table.

Q2:

QUERY:

```
select tableactor.name from tableactor,tablecasting where
tablecasting.m_id<50 and tableactor.a_id=tablecasting.a_id;
```

OUTPUT:

```
 Nested Loop  (cost=0.85..1473.53 rows=186 width=16) (actual time=4.039..58.963
rows=196 loops=1)
   ->  Index Only Scan using tablecasting_pkey on tablecasting  (cost=0.43..7.68
rows=186 width=4) (actual time=1.651..1.875 rows=196 loops=1)
         Index Cond: (m_id < 50)
         Heap Fetches: 0
   ->  Index Scan using idx_actor_aid on tableactor  (cost=0.42..7.88 rows=1 width=20)
(actual time=0.281..0.281 rows=1 loops=196)
         Index Cond: (a_id = tablecasting.a_id)
 Planning Time: 185.110 ms
 Execution Time: 61.553 ms
```

EXPLANATION:

Here, the optimiser uses a nested loop. First it performs an index scan on tablecasting with the index condition (m_id < 50). Then an index scan gets executed on tableactor with the index condition (a_id = tablecasting.a_id). Then, these results are joined with the help of a nested loop and outputs the results. Index scans are used because the optimiser estimated the low cardinality of the query.

Q3:

QUERY:

```
select tablemovie.name,TableProductionCompany.name from
tablemovie,TableProductionCompany where tablemovie.imdbScore < 1.5
and
tablemovie.prod_company=TableProductionCompany.pc_id;
```

OUTPUT:

```
Hash Join  (cost=2939.00..29814.04 rows=333333 width=22) (actual
time=87.912..357.401 rows=199905 loops=1)
  Hash Cond: (tablemovie.prod_company = tableproductioncompany.pc_id)
  ->  Seq Scan on tablemovie  (cost=0.00..22353.00 rows=333333 width=15) (actual
time=0.059..168.874 rows=199905 loops=1)
        Filter: ((imdbscore)::numeric < 1.5)
        Rows Removed by Filter: 800095
  ->  Hash  (cost=1548.00..1548.00 rows=80000 width=15) (actual time=70.848..70.849
rows=80000 loops=1)
        Buckets: 131072  Batches: 2  Memory Usage: 2900kB
        ->  Seq Scan on tableproductioncompany  (cost=0.00..1548.00 rows=80000
width=15) (actual time=0.572..18.143 rows=80000 loops=1)
 Planning Time: 194.020 ms
 Execution Time: 362.436 ms
(10 rows)
```

EXPLANATION:

First, it processes a sequential scan on the tableproductioncompany and hash it. Then it performs a sequential scan on tablemovie with the filter (imdbscore<1.5). At the end it executes a hash join with the hash condition of tablemovie.prod_company=tableproductioncompany.pc_id and outputs the results. Also the hash is done on the tableproductioncompany because the optimiser estimated that the cardinality of this would be less than the cardinality of tablemovie part. Also the execution of seq scan on tablemovie is due to expected high cardinality from the optimiser that is nearly 33%.

Q4:

QUERY:

```
select tablemovie.name,TableProductionCompany.name from
tablemovie,TableProductionCompany where (tablemovie.year between
1950 and 2000) and
tablemovie.prod_company=TableProductionCompany.pc_id;
```

OUTPUT:

```
Hash Join  (cost=9893.45..31571.09 rows=508685 width=22) (actual
time=36.889..244.258 rows=504831 loops=1)
  Hash Cond: (tablemovie.prod_company = tableproductioncompany.pc_id)
  ->  Bitmap Heap Scan on tablemovie  (cost=6954.45..21937.72 rows=508685
width=15) (actual time=16.619..79.960 rows=504831 loops=1)
        Recheck Cond: ((year >= 1950) AND (year <= 2000))
        Heap Blocks: exact=7353
        ->  Bitmap Index Scan on idx_movie_year  (cost=0.00..6827.28 rows=508685
width=0) (actual time=15.789..15.789 rows=504831 loops=1)
              Index Cond: ((year >= 1950) AND (year <= 2000))
  ->  Hash  (cost=1548.00..1548.00 rows=80000 width=15) (actual time=19.787..19.788
rows=80000 loops=1)
        Buckets: 131072  Batches: 2  Memory Usage: 2900kB
        ->  Seq Scan on tableproductioncompany  (cost=0.00..1548.00 rows=80000
width=15) (actual time=0.014..6.317 rows=80000 loops=1)
 Planning Time: 0.267 ms
 Execution Time: 254.268 ms
(12 rows)
```

EXPLANATION:
First of all, it performs a seq scan on tableproductioncompany and hash it. Then it performs a BitMap Index scan on idx_movie_year with index condition year>=1950 & year<=2000 and BitMap Heap scan fetches the required rows .Finally it makes a hash join with a condition (tablemovie.prod_company = tableproductioncompany.pc_id) and returns the result. The reason for using hash on tableproductioncompany is its less cardinality.

(3)

1. In Q1, index scans have been used due to the optimiser's prediction of less cardinality and we need not to visit all the pages of the tables. And then two nested loops are used to join the intermediate results. For the cost model first of all it takes the size of the tables. Then add the cost for indexing on required tables. And then it adds the total required cost for the index scans on the required tables (tableactor, tablecasting, tablemovie). Finally adds the cost of the nested loops used to join the required intermediate outputs to get the final output. Also note that the upper loop will have less cardinality in order to that of the inner loop so that the inner loop executes less times.

Planning Time: 467.538 ms
Execution Time: 10055.011 ms

2. In Q2, index scans have been used due to the optimiser's prediction of less cardinality and we need not to visit all the pages of the tables. And a nested loop is used to join the intermediate results. For the cost model first of all it takes the size of the tables. Then it adds the cost for indexing on required tables. And then it adds the cost of two index scans on tableactor and tablecasting.And then it takes the cost for joining the results of index scans with the help of a nested loop.
   Planning Time: 185.110 ms
   Execution Time: 61.553 ms

3. In Q3, first it choose a seq scan on tableproductioncompany and hash it because total rows in it are less than that of compared other tables. Then a seq scan occurs on tablemovie due to the high cardinality according to the optimiser's estimation approx 33%. Then these two are joined through a hash join with the required conditions. For the cost model first of all it takes the size of the tables. Then it adds the cost for the seq scan on tableprodcutioncompany(scans all the pages of the tables), then adds the cost of hashing it. Then it adds the cost for seq scan on the tablemovie. Finally it takes the cost for the hash join with the required condition in the two intermediate results.
   Planning Time: 194.020 ms
   Execution Time: 362.436 ms

4. In Q4, first it chooses a seq scan on tableproductioncompany and hash it because total rows in it are less than that of compared other tables.Then a BitMap Index scan is performed on idx_movie_year with the required index condition. And then a BitMap Heap scan occurs to fetch all the desired tuples. Finally the intermediate tables are joined using the required hash condition and get the required condition. For the cost model first of all it takes the size of the tables. Then it adds the cost of scanning tableproductioncompany(scanning the whole table) and hashing it. Then it adds the cost for indexing and then performing the BitMap index scan and BitMap Heap scan. And finally it adds the cost for joining the intermediate results through a hash join with the required condition.
   Planning Time: 0.267 ms
   Execution Time: 254.268 ms

--------x--------