James Zafiri
12/13/23
CSC6302 Module 7

Problem Set: Execution Plans

1.) What activities cannot be performed at Bear Brook, Pawtuckaway or Bradley Palmer parks

a.)

```
147  SELECT DISTINCT ActivityName
148  FROM Activity
149  WHERE ActivityName NOT IN (
150      SELECT DISTINCT ActivityName
151      FROM SummerPlan
152      WHERE ParkName IN ('Bear Brook', 'Pawtuckaway', 'Bradley Palmer')
153  );
154
```

100% 3:153

Tabular Explain

id	select_type	table	partitions	type	possible_keys	key	key_l...	ref
1	PRIMARY	Activity		index	PRIMARY	PRIMARY	402	
2	SUBQUERY	SummerPlan		ALL	ActivityName,ParkName			

rows	filtered	Extra
21	100.00	Using where; Using index
70	45.71	Using where

```
155  SELECT DISTINCT a.ActivityName
156  FROM Activity as a LEFT JOIN SummerPlan as sp ON a.ActivityName = sp.ActivityName
157  WHERE sp.ActivityName IS NULL OR sp.ParkName NOT IN ('Bear Brook', 'Pawtuckaway', 'Bradley Palmer');
158
```

100% 101:157

Tabular Explain

id	select_type	table	partitions	type	possible_keys	key	key_l...
1	SIMPLE	a		index	PRIMARY	PRIMARY	402
1	SIMPLE	sp		ref	ActivityName	ActivityName	403

ref	rows	filtered	Extra
	21	100.00	Using index; Using temporary
summervacation.a.ActivityName	3	100.00	Using where; Distinct

```
159   SELECT DISTINCT a.ActivityName
160   FROM Activity as a
161   WHERE NOT EXISTS (
162       SELECT *
163       FROM SummerPlan as sp
164       WHERE sp.ActivityName = a.ActivityName AND sp.ParkName IN ('Bear Brook', 'Pawtuckaway', 'Bradley Palmer')
165   );
166
```

100% 3:165

Tabular Explain

id	select_type	table	partitions	type	possible_keys	key	key_l...
1	SIMPLE	a		index	PRIMARY	PRIMARY	402
1	SIMPLE	<subquery2>		eq_ref	<auto_distinct_key>	<auto_distinct_key>	403
2	MATERIALIZED	sp		ALL	ActivityName,ParkName		

ref	rows	filtered	Extra
	21	100.00	Using index; Using temporary
summervacation.a.ActivityName	1	100.00	Using where; Not exists; Distin...
	70	100.00	Using where

b.)

```
-> Filter: <in_optimizer>(activity.ActivityName,activity.ActivityName in (select #2) is false)  (cost=2.35 rows=21) (actual time=0.27..0.34 rows=6 loops=1)
  -> Covering index scan on Activity using PRIMARY  (cost=2.35 rows=21) (actual time=0.0492..0.0581 rows=21 loops=1)
  -> Select #2 (subquery in condition; run only once)
    -> Filter: ((activity.ActivityName = `<materialized_subquery>`.ActivityName))  (cost=10.6..10.6 rows=1) (actual time=0.0115..0.0115 rows=0.682 loops=22)
      -> Limit: 1 row(s)  (cost=10.5..10.5 rows=1) (actual time=0.0109..0.0109 rows=0.682 loops=22)
        -> Index lookup on <materialized_subquery> using <auto_distinct_key> (ActivityName=activity.ActivityName)  (actual time=0.0106..0.0106 rows=0.682 loops=22)
          -> Materialize with deduplication  (cost=10.5..10.5 rows=32) (actual time=0.203..0.203 rows=15 loops=1)
            -> Filter: (summerplan.ParkName in ('Bear Brook','Pawtuckaway','Bradley Palmer'))  (cost=7.25 rows=32) (actual time=0.0716..0.15 rows=32 loops=1)
              -> Table scan on SummerPlan  (cost=7.25 rows=70) (actual time=0.0686..0.0983 rows=70 loops=1)
```

```
-> Table scan on <temporary>  (cost=32.1..35.5 rows=70) (actual time=1.62..1.62 rows=21 loops=1)
  -> Temporary table with deduplication  (cost=32.1..32.1 rows=70) (actual time=1.61..1.61 rows=21 loops=1)
    -> Filter: ((sp.ActivityName is null) or (sp.ParkName not in ('Bear Brook','Pawtuckaway','Bradley Palmer')))  (cost=25.1 rows=70) (actual time=0.689..1.2 rows=38 loops=1)
      -> Nested loop left join  (cost=25.1 rows=70) (actual time=0.686..1.17 rows=70 loops=1)
        -> Covering index scan on a using PRIMARY  (cost=2.35 rows=21) (actual time=0.129..0.135 rows=21 loops=1)
        -> Index lookup on sp using ActivityName (ActivityName=a.ActivityName)  (cost=0.766 rows=3.33) (actual time=0.0476..0.0489 rows=3.33 loops=21)
```

```
-> Table scan on <temporary>  (cost=298..319 rows=1470) (actual time=0.433..0.434 rows=6 loops=1)
  -> Temporary table with deduplication  (cost=298..298 rows=1470) (actual time=0.432..0.432 rows=6 loops=1)
    -> Nested loop antijoin  (cost=151 rows=1470) (actual time=0.342..0.383 rows=6 loops=1)
      -> Covering index scan on a using PRIMARY  (cost=2.35 rows=21) (actual time=0.0838..0.0938 rows=21 loops=1)
      -> Single-row index lookup on <subquery2> using <auto_distinct_key> (ActivityName=a.ActivityName)  (cost=14.3..14.3 rows=1) (actual time=0.0134..0.0134 rows=0.714
loops=21)
        -> Materialize with deduplication  (cost=14.2..14.2 rows=70) (actual time=0.248..0.248 rows=15 loops=1)
          -> Filter: (sp.ActivityName is not null)  (cost=7.25 rows=70) (actual time=0.121..0.207 rows=32 loops=1)
            -> Filter: (sp.ParkName in ('Bear Brook','Pawtuckaway','Bradley Palmer'))  (cost=7.25 rows=70) (actual time=0.12..0.201 rows=32 loops=1)
              -> Table scan on sp  (cost=7.25 rows=70) (actual time=0.116..0.148 rows=70 loops=1)
```

c.) For my first query, I see that it went through 21 rows in the activity table using an index, and then 70 rows in the SummerPlan table, which is the subquery. The first index scan was a cost of 2.35 and actual time of 0.0492. The second scan was a cost of 7.25 and actual time of 0.0686. I think this is solid optimization since even though it goes through a lot of rows on the subquery which takes up time, indexes are being used properly.

For my second query, I see that it went through 21 rows in the activity table using an index/temporary, and then 3 rows in the SummerPlan table using a reference(as a and sp). The first scan was a cost of 32.1 and actual time of 1.62. The second scan was a cost of 2.35 and an actual time of 0.129. It is important to note the nested loop for the left join, with a cost of 25.1 and actual time of 0.686. While at first it seemed like this was more optimized since there was less rows in the temporary table, it costs a lot to join and there is even more time spent than the first query.

For my third query, I see that it went through 21 rows in the Activity table as a with an index. It went through 1 row with the subquery as an extra that used multiple conditions. Also, it went through 70 rows of the SummerPlan table as sp using the where clause. The first scan had a cost of 298.319 and actual time of 0.433. A nested loop costing 151 with an actual time of 0.342 and an additional index scan with a cost of 2.35 and actual time of 0.0838. Lastly a table scan costing 7.25 with an actual time of 0.116. This query is extra complicated and is basically a combination of the first two. There are more operations, and it costs a lot without even being much faster. This doesn't seem very optimized, and it also makes me believe that "NOT IN" is more efficient than "NOT EXISTS."

d.) I think that my first query has the best performance. There are not many big operations, and while the second query might go through less rows, the first query costs a lot less and is faster. It seems the most efficient of the 3.

2.)

a.)

```
EXPLAIN ANALYZE
SELECT DISTINCT a.ActivityName
FROM Activity as a
JOIN SummerPlan as sp ON a.ActivityName = sp.ActivityName JOIN Park as p ON sp.ParkName = p.ParkName
WHERE p.State = 'MA'
GROUP BY a.ActivityName
HAVING COUNT(sp.ParkName) >= 4;
```

1:181

Explain data not available for statement

```
-> Filter: (count(sp.ParkName) >= 4)  (actual time=0.255..0.256 rows=1 loops=1)
   -> Table scan on <temporary>  (actual time=0.25..0.252 rows=13 loops=1)
      -> Aggregate using temporary table  (actual time=0.249..0.249 rows=13 loops=1)
         -> Nested loop inner join  (cost=5.4 rows=7.78) (actual time=0.125..0.21 rows=27 loops=1)
```

```
-> Nested loop inner join  (cost=2.68 rows=7.78) (actual time=0.116..0.164 rows=27 loops=1)
    -> Filter: (p.State = 'MA')  (cost=1.15 rows=1) (actual time=0.0638..0.0669 rows=5 loops=1)
        -> Table scan on p  (cost=1.15 rows=9) (actual time=0.0581..0.0608 rows=9 loops=1)
    -> Filter: (sp.ActivityName is not null)  (cost=1.53 rows=7.78) (actual time=0.0167..0.0189 rows=5.4 loops=5)
        -> Index lookup on sp using ParkName (ParkName=p.ParkName)  (cost=1.53 rows=7.78) (actual time=0.0165..0.0184 rows=5.4 loops=5)
    -> Single-row covering index lookup on a using PRIMARY (ActivityName=sp.ActivityName)  (cost=0.263 rows=1) (actual time=0.00149..0.00152 rows=1 loops=27)
```

b.) Index added in the sql code which is attached.

c.)

```
-> Filter: (count(sp.ParkName) >= 4)  (actual time=0.15..0.151 rows=1 loops=1)
    -> Table scan on <temporary>  (actual time=0.147..0.148 rows=13 loops=1)
        -> Aggregate using temporary table  (actual time=0.146..0.146 rows=13 loops=1)
            -> Nested loop inner join  (cost=22.1 rows=38.9) (actual time=0.0462..0.117 rows=27 loops=1)
                -> Nested loop inner join  (cost=8.48 rows=38.9) (actual time=0.0417..0.0829 rows=27 loops=1)
                    -> Covering index lookup on p using idx_state (State='MA')  (cost=0.841 rows=5) (actual time=0.0169..0.0183 rows=5 loops=1)
                    -> Filter: (sp.ActivityName is not null)  (cost=0.906 rows=7.78) (actual time=0.0102..0.0124 rows=5.4 loops=5)
                        -> Index lookup on sp using ParkName (ParkName=p.ParkName)  (cost=0.906 rows=7.78) (actual time=0.01..0.0118 rows=5.4 loops=5)
                -> Single-row covering index lookup on a using PRIMARY (ActivityName=sp.ActivityName)  (cost=0.253 rows=1) (actual time=0.00105..0.00109 rows=1 loops=27)
```

While I was not able to generate an execution plan, I can see the difference in the explain analyze. After the second nest loop inner join, instead of using a filter and then another table scan like how the query did before the index, it went straight to using a covering index lookup on the parks using the state name ‘MA”. This helps the performance since there are less operations, and I can see less time and cost was used.

3.) Queries are written in attached sql code.

a.)

```
-> Table scan on <temporary>  (cost=3.28..3.28 rows=0.957) (actual time=0.126..0.126 rows=0 loops=1)
    -> Temporary table with deduplication  (cost=0.776..0.776 rows=0.957) (actual time=0.124..0.124 rows=0 loops=1)
        -> Nested loop inner join  (cost=0.681 rows=0.957) (actual time=0.067..0.067 rows=0 loops=1)
            -> Filter: ((sp.ActivityName <> 'Camping') and (sp.ActivityName is not null))  (cost=0.346 rows=0.957) (actual time=0.066..0.066 rows=0 loops=1)
                -> Index lookup on sp using ParkName (ParkName='YourSelectedPark')  (cost=0.346 rows=1) (actual time=0.065..0.065 rows=0 loops=1)
            -> Single-row covering index lookup on a using PRIMARY (ActivityName=sp.ActivityName)  (cost=0.354 rows=1) (never executed)
```

b.)

```
-> Sort: a.ActivityName, p.State, sp.ParkName  (actual time=0.636..0.639 rows=12 loops=1)
    -> Table scan on <temporary>  (cost=15.3..17.9 rows=16.7) (actual time=0.235..0.241 rows=12 loops=1)
        -> Temporary table with deduplication  (cost=15.2..15.2 rows=16.7) (actual time=0.234..0.234 rows=12 loops=1)
            -> Nested loop inner join  (cost=13.5 rows=16.7) (actual time=0.118..0.207 rows=12 loops=1)
                -> Nested loop inner join  (cost=7.68 rows=16.7) (actual time=0.101..0.172 rows=12 loops=1)
                    -> Filter: (a.ActivityName in ('Swimming','Kayaking','Paddle Boarding','Surfing','Boogeyboarding'))  (cost=2.26 rows=5) (actual time=0.0753..0.0982 rows=5 loops=1)
                        -> Covering index range scan on a using PRIMARY over (ActivityName = 'Boogeyboarding') OR (ActivityName = 'Kayaking') OR (3 more)  (cost=2.26 rows=5) (actual time=0.073..0.0936 rows=5 loops=1)
                -> Filter: (sp.ParkName is not null)  (cost=0.817 rows=3.33) (actual time=0.013..0.0143 rows=2.4 loops=5)
                    -> Index lookup on sp using ActivityName (ActivityName=a.ActivityName)  (cost=0.817 rows=3.33) (actual time=0.0128..0.0139 rows=2.4 loops=5)
                -> Single-row index lookup on p using PRIMARY (ParkName=sp.ParkName)  (cost=0.256 rows=1) (actual time=0.00275..0.00278 rows=1 loops=12)
```

c.)

```
-> Table scan on <temporary>  (cost=5..6.91 rows=4) (actual time=0.0908..0.0914 rows=4 loops=1)
    -> Temporary table with deduplication  (cost=4.36..4.36 rows=4) (actual time=0.0898..0.0898 rows=4 loops=1)
        -> Nested loop inner join  (cost=3.96 rows=4) (actual time=0.0493..0.0712 rows=4 loops=1)
            -> Filter: (sp.ParkName is not null)  (cost=2.56 rows=4) (actual time=0.0412..0.0564 rows=4 loops=1)
                -> Index range scan on sp using ActivityName over (ActivityName = 'Baseball') OR (ActivityName = 'Soccer') OR (ActivityName = 'Tennis'), with index condition: (sp.ActivityName in ('Soccer','Baseball','Tennis'))  (cost=2.56 rows=4) (actual time=0.0403..0.0552 rows=4 loops=1)
            -> Single-row index lookup on p using PRIMARY (ParkName=sp.ParkName)  (cost=0.275 rows=1) (actual time=0.00328..0.0033 rows=1 loops=4)
```

4.)

```
-> Table scan on Activity  (cost=2.35 rows=21) (actual time=0.0655..0.0705 rows=21 loops=1)
```

5.)

```
-> Index lookup on Park using idx_state (State='MA')  (cost=1 rows=5) (actual time=0.0647..0.0672 rows=5 loops=1)
```

6.)

```
-> Index lookup on SummerPlan using ParkName (ParkName='Bear Brook')  (cost=1.85 rows=11) (actual time=0.0928..0.096 rows=11 loops=1)
```

7.)

```
-> Index lookup on SummerPlan using ActivityName (ActivityName='Hiking')  (cost=1.25 rows=5) (actual time=0.0316..0.0337 rows=5 loops=1)
```