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 screenshot of a computer

Description automatically generated

A screenshot of a phone

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a black screen

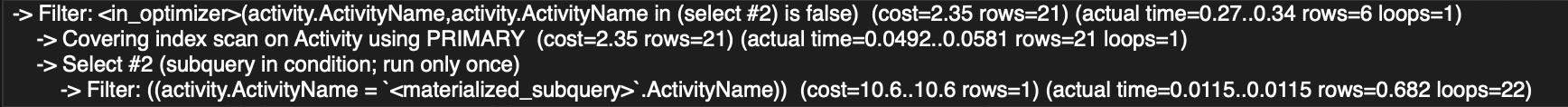
Description automatically generated

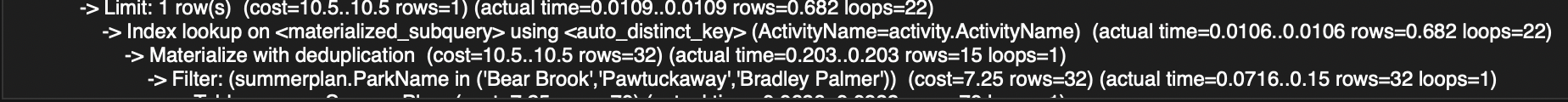
A screenshot of a computer

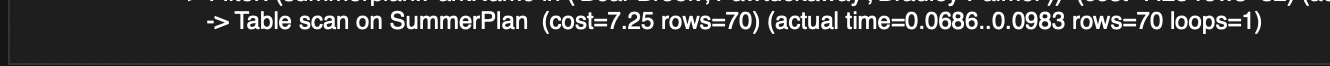
Description automatically generated

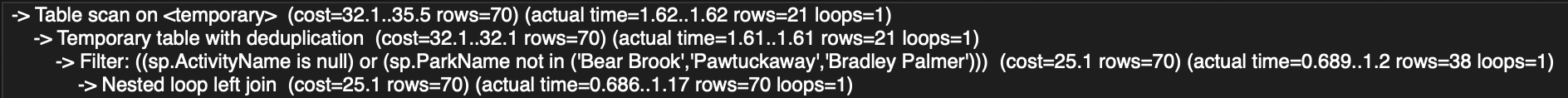
A screenshot of a black screen

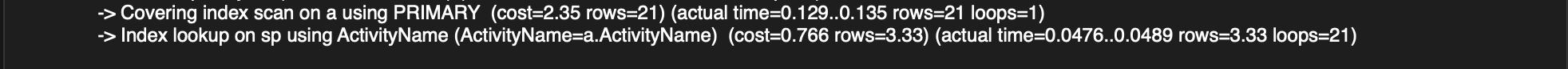
Description automatically generated

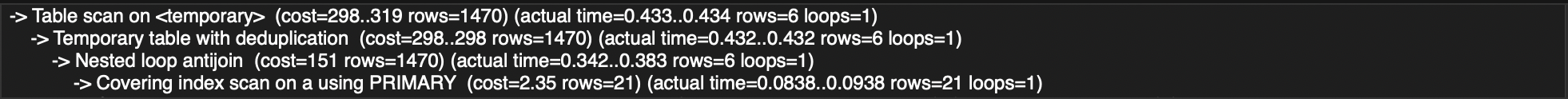




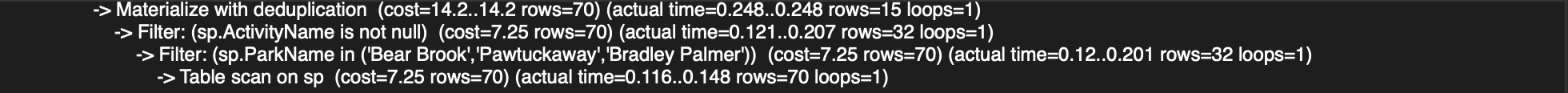












1. For my first query, I see that it went through 21 rows in the activity table using an index, and then 70 rows in the SummerPlan table, which is the subquery. The first index scan was a cost of 2.35 and actual time of 0.0492. The second scan was a cost of 7.25 and actual time of 0.0686. I think this 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.

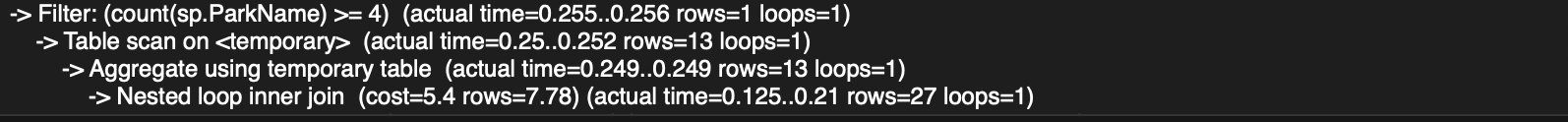
For my second query, I see that it went through 21 rows in the activity table using an index/temporary, and then 3 rows in the SummerPlan table using a reference(as a and sp). The first scan was a cost of 32.1 and actual time of 1.62. The second scan was a cost of 2.35 and an actual time of 0.129. It 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.

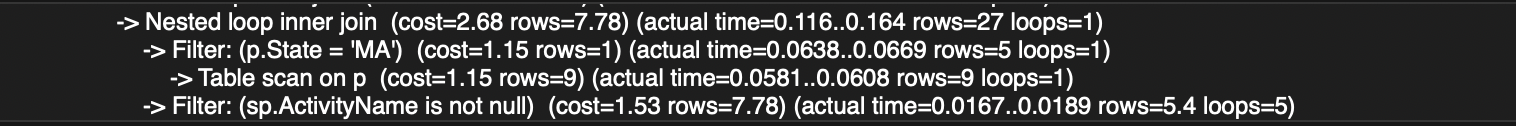
For my third query, I see that it went through 21 rows in the Activity table as a with an index. It went through 1 row with the subquery as an extra that used multiple conditions. Also, it went through 70 rows of the SummerPlan table as sp using the where clause. The first scan had a cost of 298.319 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.”

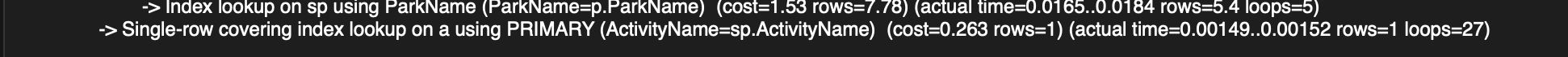
1. I think that my first query has the best performance. There are not many big operations, and while the second query might go through less rows, the first query costs a lot less and is faster. It seems the most efficient of the 3.

A screenshot of a computer

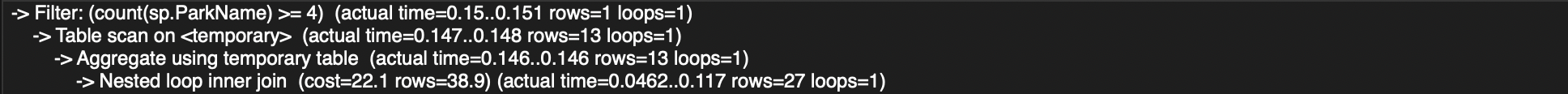
Description automatically generated

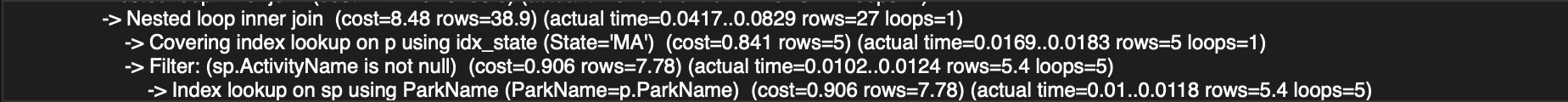






1. Index added in the sql code which is attached.







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.

1. Queries are written in attached sql code.

