Kannika Armstrong
TCSS 564 Database Systems Internals
May 9, 2025

Database Systems Internals - Project
Query Optimization
Part 2: query plan change in response to a change in the cardinalities

1. Database Information:

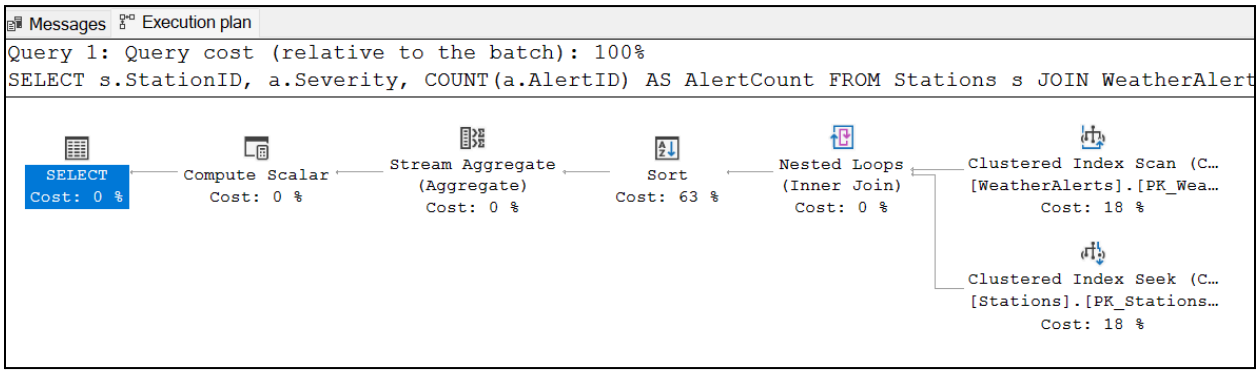
- ***My database name:*** "WeatherDB2" with 2 tables: Stations and WeatherAlerts
- Each table has a Clustered Index on the Primary Key
- I reduced the table into 2 tables for Part 2 of the project
- The number of rows of the Stations table is fixed to 10 rows
- The number of rows of the WeatherAlerts data will be changed

Table Name	Column Names	Primary Key	Foreign Key
Stations	StationID, Name, Region, Latitude, and Longitude	StationID	-
WeatherAlerts	AlertID, StationID, AlertDate, AlertType, and Severity	AlertID	StationID

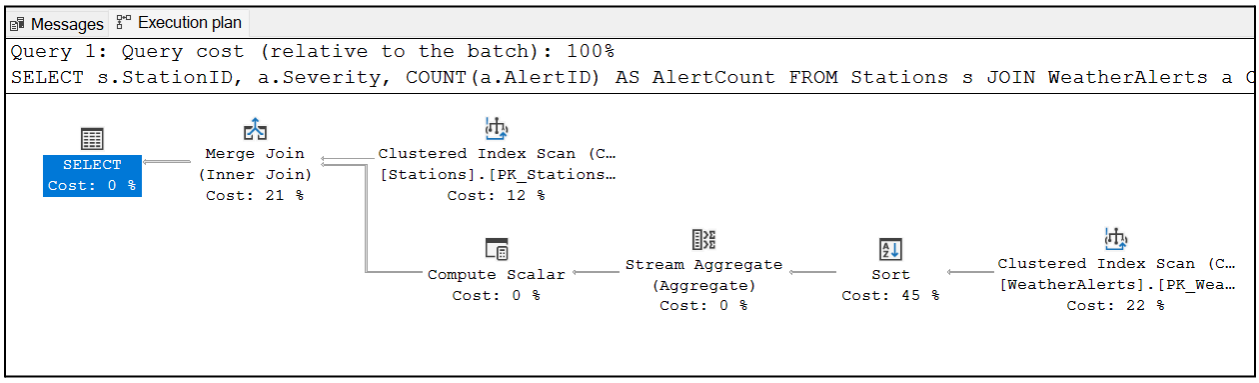
2. The base query used for Part 2:

```
SELECT  s.StationID, a.Severity, COUNT(a.AlertID) AS AlertCount
FROM Stations s
JOIN WeatherAlerts a ON s.StationID = a.StationID
WHERE a.AlertDate BETWEEN '2023-07-01' AND '2023-07-15'
GROUP BY s.StationID, a.Severity
ORDER BY s.StationID;
```

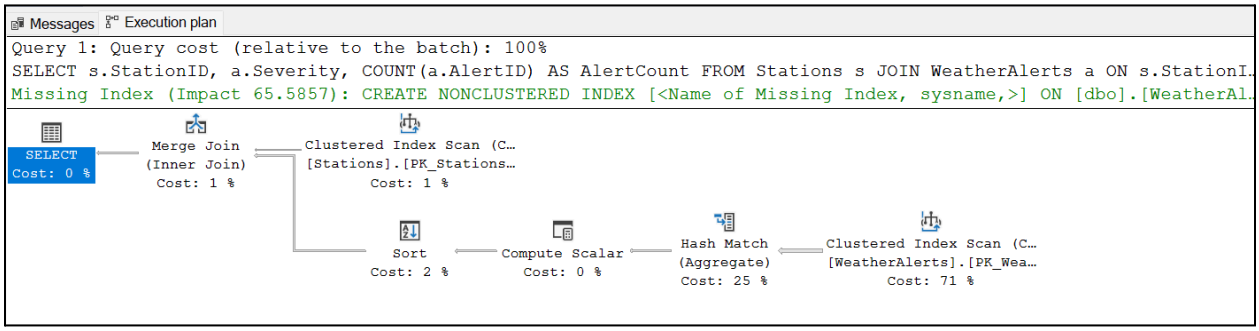
3. **Query Plan from the Query Optimizer**
    - ***Query Plan 1***
        - Number of rows in the WeatherAlerts Table: **0 rows**
        - Actual Execution time (Elapsed Time): **47 ms**
        - Explanation:
            - The query optimizer chooses to use the Clustered-Index Scan on the WeatherAlert because there is no row of data in there, but chooses the Clustered-Index Seek for the Stations table because there are 10 rows in the table that support the Seek operation.
            - The query optimizer chose to use the Nested Loop Join because there are only 10 rows of data in the Station table, and there are 0 rows. The Nested Loop Join then sorts, which should be cheaper for this case.
            - Then the optimizer chose to use the Stream Aggregate because of the GROUP BY clause.



- ***Query Plan 2***
    - Number of rows in the WeatherAlerts Table: **500 rows**
    - Actual Execution time (Elapsed Time): **59 ms**
    - Explanation:
        - The optimizer chose to use the Clustered-Index Scan on the WeatherAlerts table, then Sort and Stream Aggregate before Merge Join with the result from the Clustered-Index Scan on the Stations table because the Merge Join requires the sorted data. Because the aggregated inner input is now smaller and grouped, making the Merge Join cheaper.

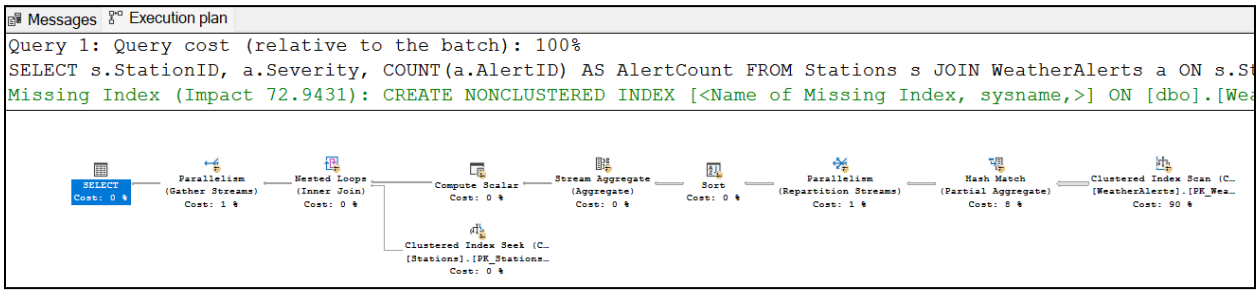


- ***Query Plan 3***
    - Number of rows in the WeatherAlerts Table: **50,000 rows**
    - Actual Execution time (Elapsed Time): **95 ms**
    - Explanation:
        - Since there are more rows in the WeatherAlerts table, the optimizer chooses to use the Clustered-Index Scan then Hash Match, rather than sorting the data because there is more data for this case, so sorting the data can cost.
        - The optimizer chooses to use the Hash Table to help with the Count operation, then sort to make the sort cost cheaper, and because the Merge Join requires the sorted data.
        - Finally, the optimizer chose to use the Merge Join because both datasets are already sorted, and it's cheaper to do the Merge Join.

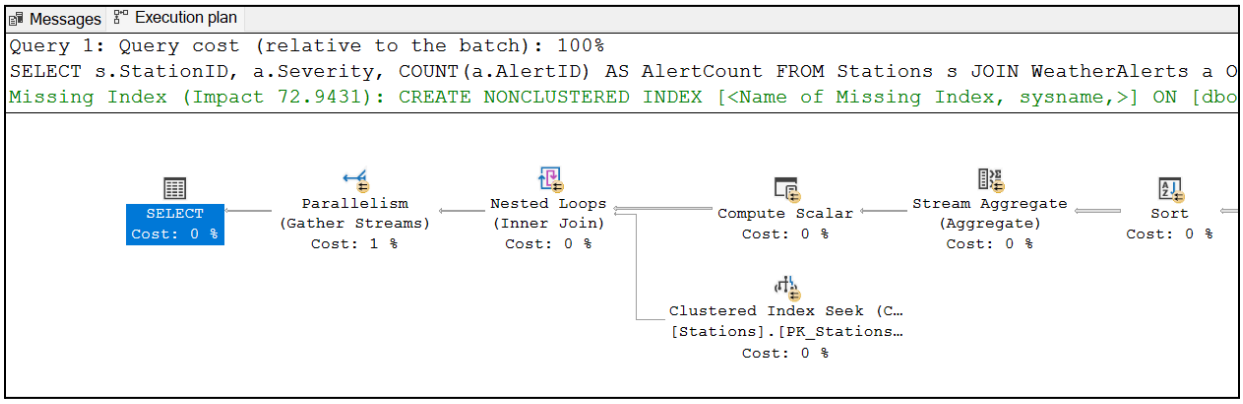


- ***Query Plan 4***
    - Number of rows in the WeatherAlerts Table: **600,000 rows**
    - Actual Execution time (Elapsed Time): **78 ms**
    - Explanation:
        - The optimizer chose to use the Parallelism operator because there is a lot of data, 600K rows here. So the SQL Server is trying to optimize, but it also implies higher data volume/complexity.
        - Then the optimizer chooses to use the Hash Aggregate because it's cheaper for the larger data before sorting it.
        - After computing the count operator after sorting them, the optimizer chooses to use the Merge Join because both datasets are already sorted, so it's cheaper to do the Merge Join.

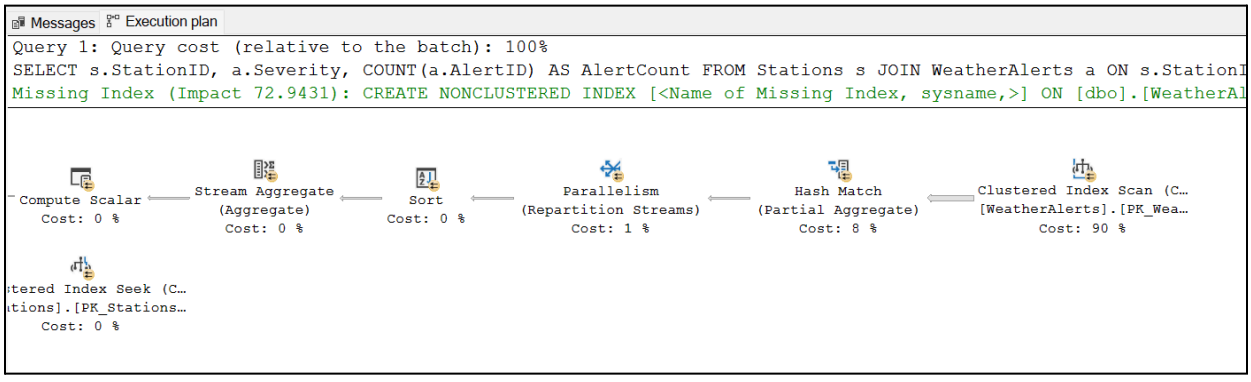
The full query plan:



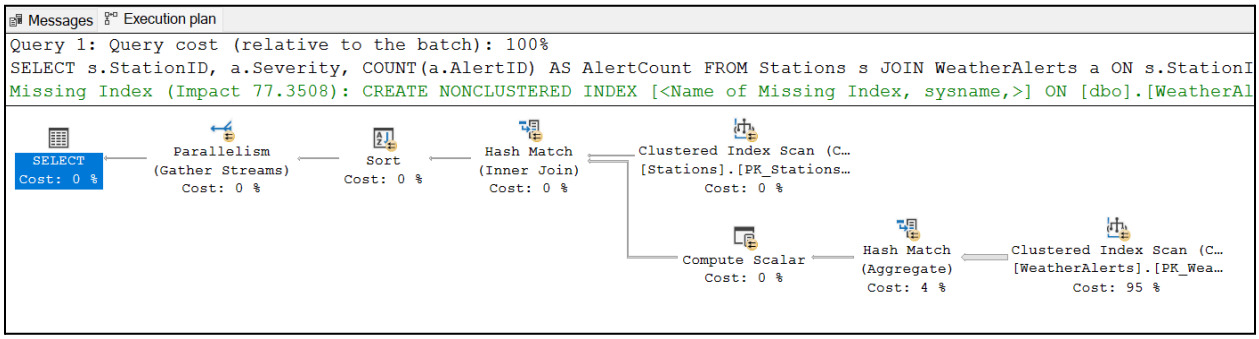
Zoom in the front:



Zoom in the back:



- ***Query Plan 5***
    - Number of rows in the WeatherAlerts Table: **1,000,000 rows**
    - Actual Execution time (Elapsed Time): **79 ms**
    - Explanation:
        - Since there are too many rows, 1M rows for this case, the optimizer chooses to use the Batch mode with Parallelism, which is good for the optimization.
        - The optimizer chose to use the Clustered-Index Scan in the WeatherAlerts table, then use the Hash Aggregate rather than sort them to use the Stream Aggregate here.
        - Since the results from both tables still have a large amount of data, the optimizer chooses to use the Hash aggregate again to avoid the cost of sorting them.
        - Then sort it after that for the cheaper cost.



4. **The execution times for 5 different datasets and 5 different query plans**

No. Of rows	Actual Execution Time (Elapsed Time) (ms)				
	QP1	QP2	QP3	QP4	QP5
0	47	79	97	78	87
500	143	59	78	95	63
50,000	317	111	95	63	130
600,000	728	413	236	78	186
1,000,000	899	396	350	158	79

Summary:

1. In most cases the cases the optimizer seems to make the right decision to choose the best query plan to execute the base query, except in case 3 with 50,000 rows of data.
2. For cases 4 and 5, the optimizer chooses to use the Batch mode with Parallelism because the large data involved and the aggregate operation, COUNT, are expensive. So, splitting the workload can help to reduce the execution time significantly because aggregations over many rows will take longer when executed serially
3. For case 3, the optimizer doesn't choose to use the parallelism because the WHERE clause can help to filter the dataset. Even though there's a JOIN and a GROUP BY, the optimizer might determine that a Sequential Scan (even with joins and aggregation) is more efficient in this case, as the filtered result set may not justify the overhead of parallel execution. The date range filter also makes the dataset smaller, so the query is potentially fast enough without the need for parallelism. Additionally, if SQL Server estimates that the cost of parallel execution is higher (due to factors like small result size or costs of managing parallel threads), it might decide not to use parallelism
4. However, after forcing the query plan to use the Parallelism, from the query plan 4, executing case 3 can help to reduce the execution time.