Kannika Armstrong
TCSS 564 Database Systems Internals
June 1, 2025

Database Systems Internals - Project
Query Optimization
Part 3: The optimizer makes the wrong decision

1. Database Information:

- ***My database name:*** "WeatherDB3" with 2 tables: Stations and WeatherAlerts
- Each table has a Clustered Index on the Primary Key
- The number of rows of the Stations table is 100.
- The number of rows of the WeatherAlerts table is 500,000.

Table Name	Column Names	Primary Key	Foreign Key	Number of rows
Stations	StationID, Name, Region, Latitude, and Longitude	StationID	-	100
WeatherAlerts	AlertID, StationID, AlertDate, AlertType, and Severity	AlertID	StationID	500,000

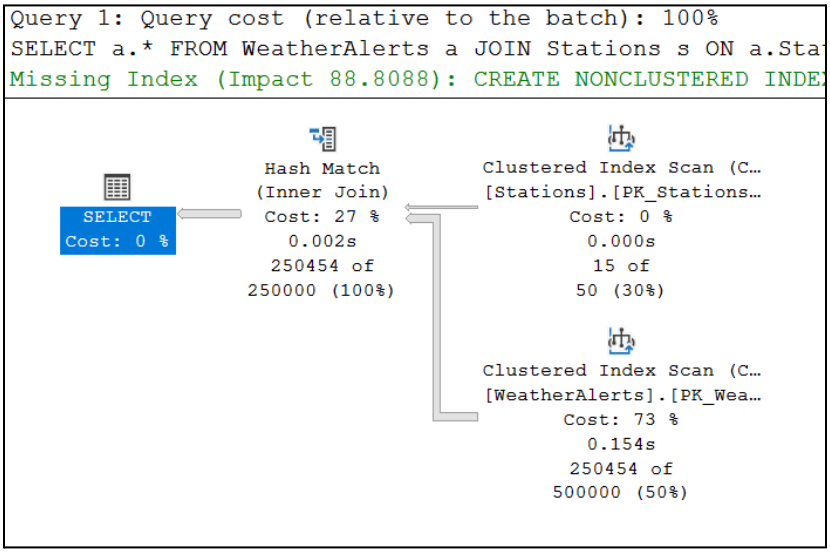
Query 1:

```sql
-- Query 1 without the hint
SELECT a.*
FROM WeatherAlerts a
JOIN Stations s ON a.StationID = s.StationID
WHERE s.Region = 'WA'
```

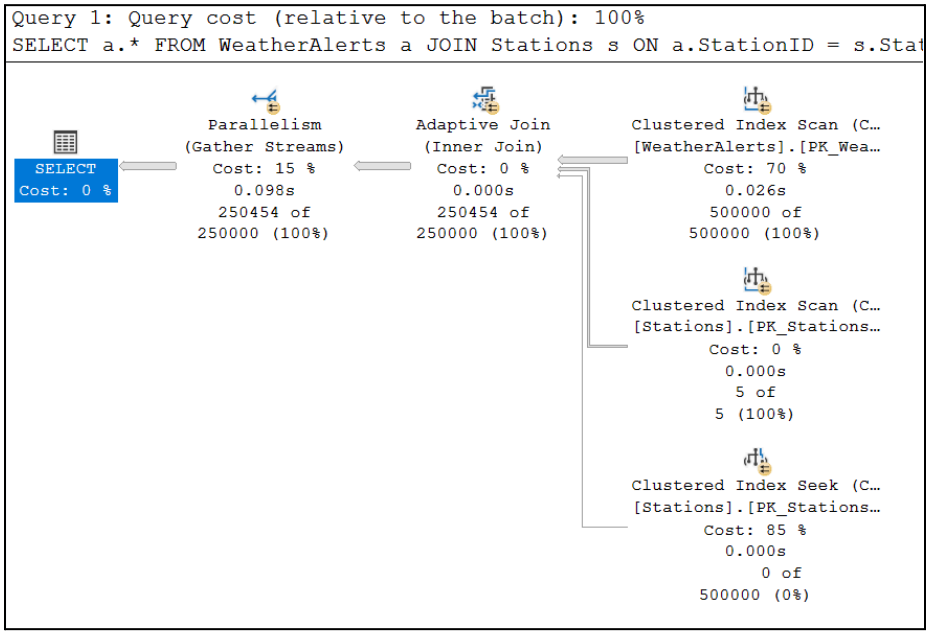
```sql
-- Query 1 with the hint
SELECT a.*
FROM WeatherAlerts a
JOIN Stations s ON a.StationID = s.StationID
WHERE s.Region = 'WA'
OPTION (FORCE ORDER)
```

**Add the `OPTION (FORCE ORDER)`: The hint forces the optimizer to follow the FROM clause join order.

- ***Screenshot: Query 1 without the hint***



- ***Screenshot: Query 1 with the hint***



- **Comparison table**

Query Plan	Operations	Actual runtime (s)	Estimated Subtree Cost
Without hint	- Clustered Index Scan on Stations and WeatherAlerts tables - Hash Join - No parallelism	1.176	4.62905
With hint	- Parallelism enabled - Clustered Index Scans on both tables - Adaptive Join	1.111	4.20606

Conclusion:

- Percent improvement = 100 * (1 - (1.111/1.176)) = 5.5 %
- The reason why the query optimizer makes the wrong decision:
  - The Stations table size is small (100 rows), so the query optimizer assumes that a full scan is cheap.
  - The query optimizer doesn't expect filtering with `WHERE s.Region = 'WA'` will significantly reduce the size of the results.
  - The query optimizer doesn't use the parallelism because it assumes that scanning 100 rows and joining with the large table is fast enough.
- The reason why the query plan runs faster with the hint:
  - The hint forces the optimizer to follow the FROM clause join order.
  - Using the hint forces the query optimizer to use parallelism.
  - Used Adaptive Join, which is better when input row counts are hard to estimate.
- Without the hint: choose a safe plan based on estimates (single-threaded hash join).
- With the hint: forced a better join order, enabling adaptive join and parallelism, leading to faster performance.

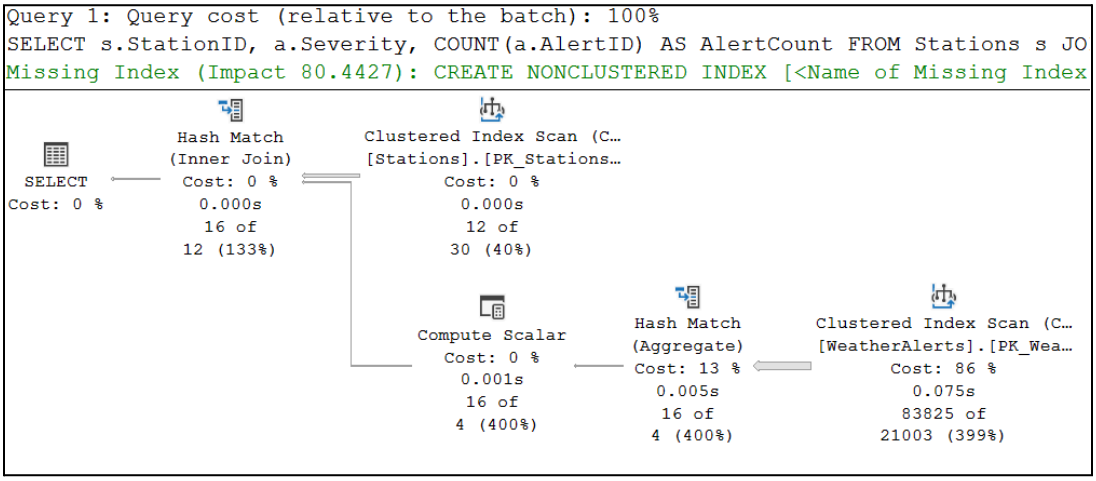
Query 2:

```sql
-- Query 2 without the hint
SELECT
s.StationID, a.Severity,
COUNT(a.AlertID) AS AlertCount
FROM Stations s
JOIN WeatherAlerts a ON
s.StationID = a.StationID
WHERE s.Latitude > 47 and
a.AlertDate BETWEEN '2023-05-01'
AND '2023-09-30'
GROUP BY s.StationID, a.Severity
```

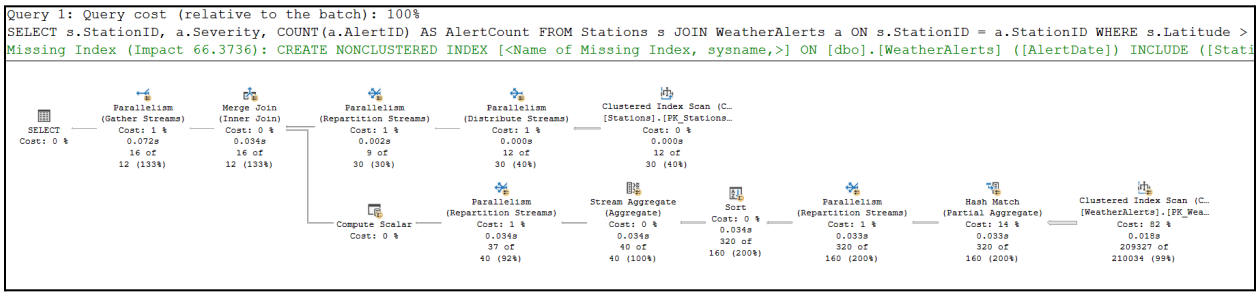
```sql
-- Query 1 with the hint
SELECT
s.StationID, a.Severity,
COUNT(a.AlertID) AS AlertCount
FROM Stations s
JOIN WeatherAlerts a ON
s.StationID = a.StationID
WHERE s.Latitude > 47 and
a.AlertDate BETWEEN '2023-05-01'
AND '2023-09-30'
GROUP BY s.StationID, a.Severity
OPTION (MERGE JOIN);
```

**Add the `OPTION (MERGE JOIN)`: The hint forces the optimizer to use Merge Join instead of the Hash Join, just for practicality, but the result was interesting because when forced to use the Merge Join, the query optimization chose to use the Parallelism Plan.

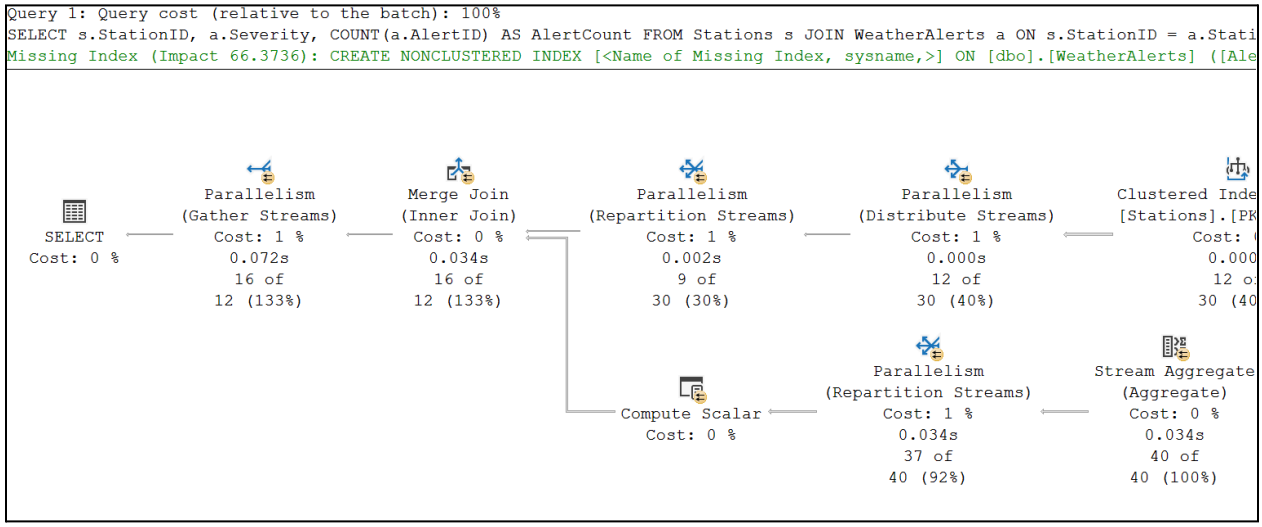
- ***Screenshot: Query 2 without the hint***



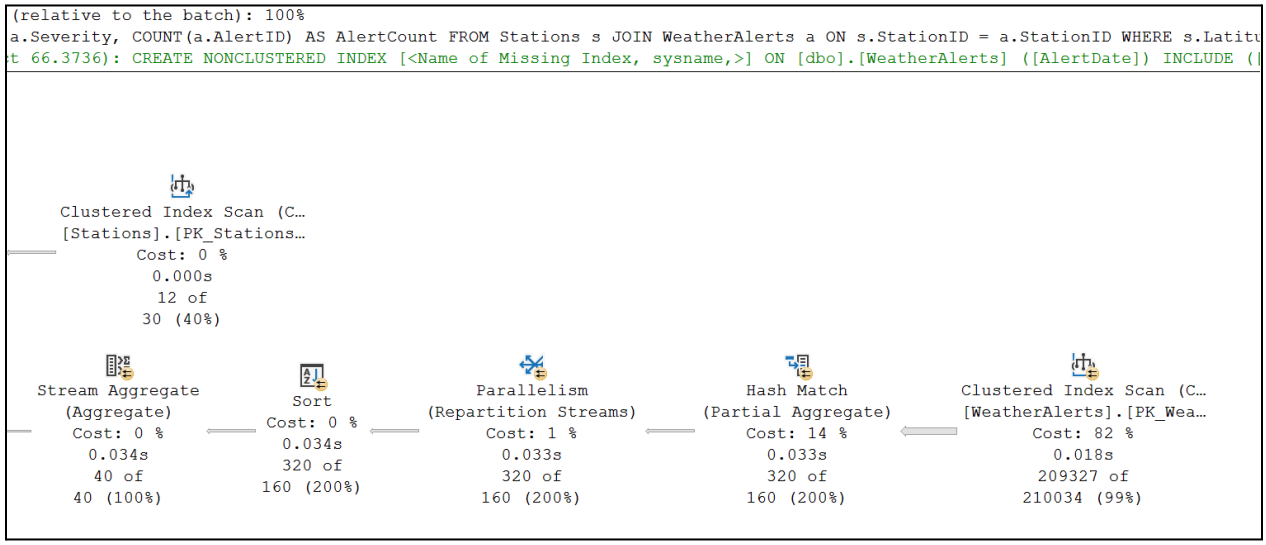
- ***Screenshot: Query 2 with the hint***



- ***Screenshot: Query 2 with the hint; Zoom in on the front.***



- ***Screenshot: Query 2 with the hint; Zoom in on the back.***



- **Comparison table**

Query Plan	**Operations**	**Actual runtime (s)**	**Estimated Subtree Cost**
Without hint	- Clustered Index Scan on Stations table - Clustered Index Scan on WeatherAlerts table, then do Hash Aggregation before doing the COUNT operation - Hash Join - No parallelism	0.247	3.90804
With hint	- Parallelism enabled - Clustered Index Scan on Stations table, then do Parallelism (Distributed Streams) before Parallelism (Repartition Streams) - Clustered Index Scan on WeatherAlerts table, then do Hash Aggregation before Parallelism (Repartition Streams). Then sort it before do the Stream Aggregation before the COUNT operation - Merge Join - Parallelism (Gather Streams)	0.156	3.61833

Conclusion:

- Percent improvement = 100 * (1 - (0.156/0.247)) = 36.84 %
- The reason why the query optimizer makes the wrong decision:
    - The query optimizer prefers Hash Joins by default because there is no useful index to help with the Merge Join
    - The query optimizer expects a large dataset.
    - Hash Join is good for unordered, large, and unindexed input.
    - No Parallelism because the estimated cost was too low.
- The reason why the query plan runs faster with the hint:
    - Enabling Parallelism is a huge benefit to make it faster.
    - The inputs are sorted on the join keys.
    - After doing Hash Aggregation on the WeatherAlerts data after scan, the data is smaller than the optimizer chooses to sort it before doing the Stream Aggregation.
- Forcing Merge Join encouraged a plan that used sorted, partitioned data with parallel streams, which made full use of CPU cores and improved performance.

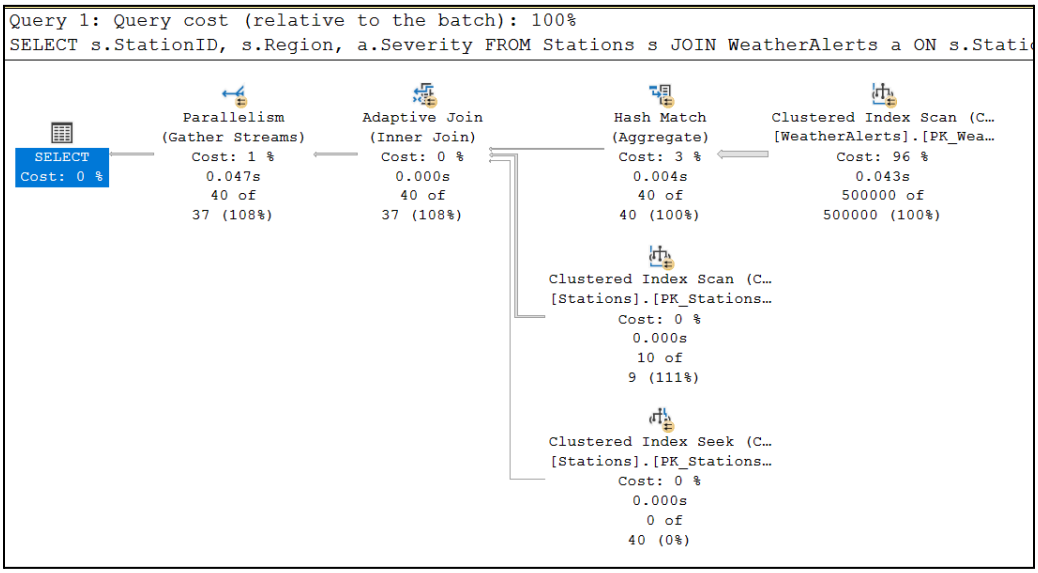
Query 3:

```sql
-- Query 3 without the hint
SELECT
s.StationID, s.Region, a.Severity
FROM Stations s
JOIN WeatherAlerts a ON
s.StationID = a.StationID
WHERE s.Latitude < 47 or
s.StationID < 90
GROUP BY s.StationID, s.Region,
a.Severity
```

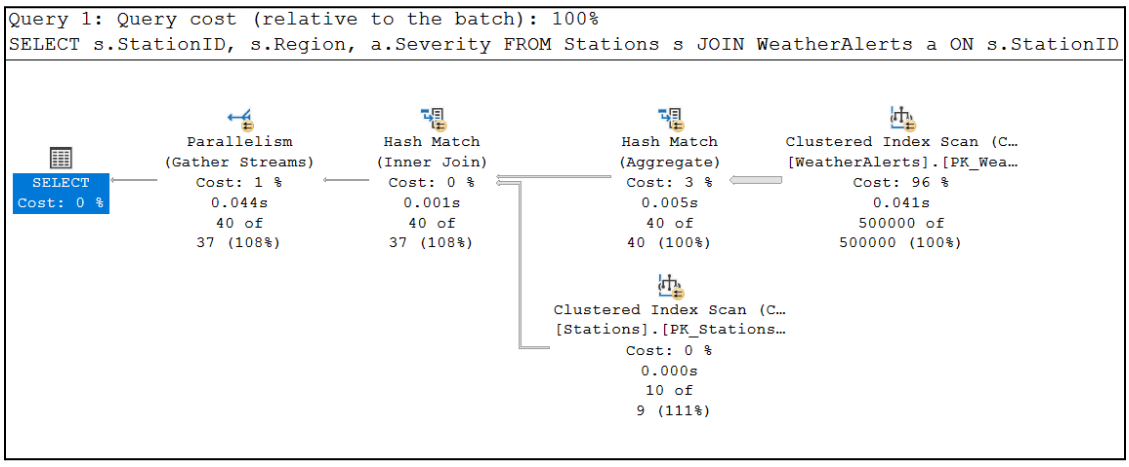
```sql
-- Query 3 with the hint
SELECT
s.StationID, s.Region, a.Severity
FROM Stations s
JOIN WeatherAlerts a ON
s.StationID = a.StationID
WHERE s.Latitude < 47 or
s.StationID < 90
GROUP BY s.StationID, s.Region,
a.Severity
OPTION (HASH JOIN);
```

**Add the `OPTION (HASH JOIN)`: Forcing a Hash Join was more efficient because it skipped the decision-making overhead.

- ***Screenshot: Query 3 without the hint***



- ***Screenshot: Query 3 with the hint***



- **Comparison table**

Query Plan	**Operations**	**Actual runtime (s)**	**Estimated Subtree Cost**
Without hint	- Parallelism enabled - Clustered Index Scan on WeatherAlerts table, then do Hash Aggregation - Clustered Index Scan and Clustered Index Seek on the Stations table - Adaptive Join - Parallelism (Gather Streams)	0.183	3.08146
With hint	- Parallelism enabled - Clustered Index Scan on WeatherAlerts table, then do Hash Aggregation - Clustered Index Scan on Stations table - Hash Join - Parallelism (Gather Streams)	0.127	3.08146

Conclusion:

- Percent improvement = 100 * (1 - (0.127/0.183)) = 30.60 %
- The reason why the query optimizer makes the wrong decision:
  - The query optimizer chooses to use the Adaptive Join because it tries to balance performance with safety, especially in uncertain cases. But the Adaptive Join has more runtime cost than the Hash Join.
  - `WHERE s.Latitude < 47 or s.StationID < 90` has non-trivial logic. SQL Server couldn't confidently estimate how many rows would match. So, it can return fewer or many rows than expected.
- The reason why the query plan runs faster with the hint:
  - Forcing a Hash Join was more efficient because it skipped the decision-making overhead.
  - For large inputs, Hash Join is often the best choice, so forcing it makes sense.
- The Adaptive Join has slight overhead: setup, decision-making, and buffering. That can be slower than a direct Hash Join, especially when the correct path is predictable.

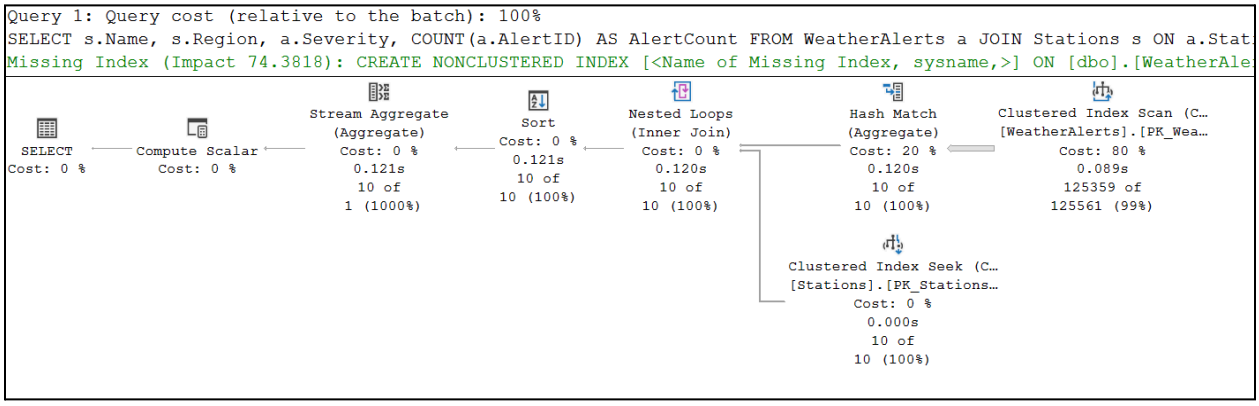
Query 4:

```sql
-- Query 4 without the hint
SELECT s.Name, s.Region,
a.Severity, COUNT(a.AlertID) AS
AlertCount
FROM WeatherAlerts a
JOIN Stations s ON a.StationID =
s.StationID
WHERE s.Latitude > 40 and
a.Severity = 'None'
GROUP BY s.Name, s.Region,
a.Severity
```

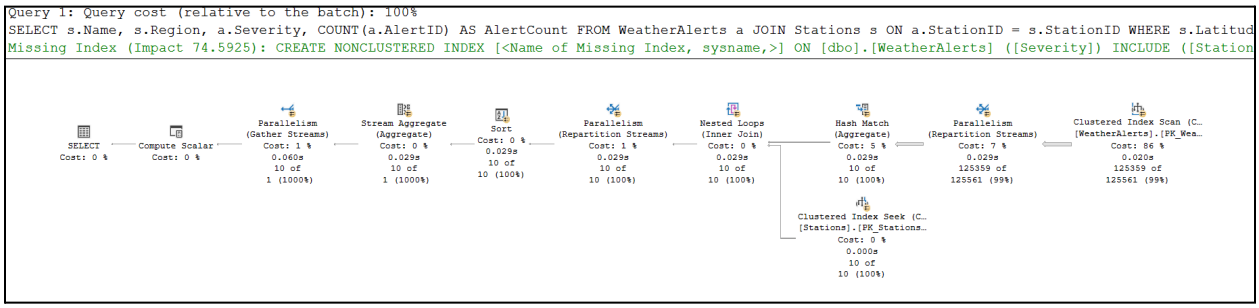
```sql
-- Query 4 with the hint
SELECT s.Name, s.Region,
a.Severity, COUNT(a.AlertID) AS
AlertCount
FROM WeatherAlerts a
JOIN Stations s ON a.StationID =
s.StationID
WHERE s.Latitude > 40 and
a.Severity = 'None'
GROUP BY s.Name, s.Region,
a.Severity
OPTION(USE
HINT('ENABLE_PARALLEL_PLAN_PREFER
ENCE'));
```

**Add the `OPTION(USE HINT('ENABLE_PARALLEL_PLAN_PREFERENCE'));:` It nudged the query optimizer to explore and choose a faster plan.

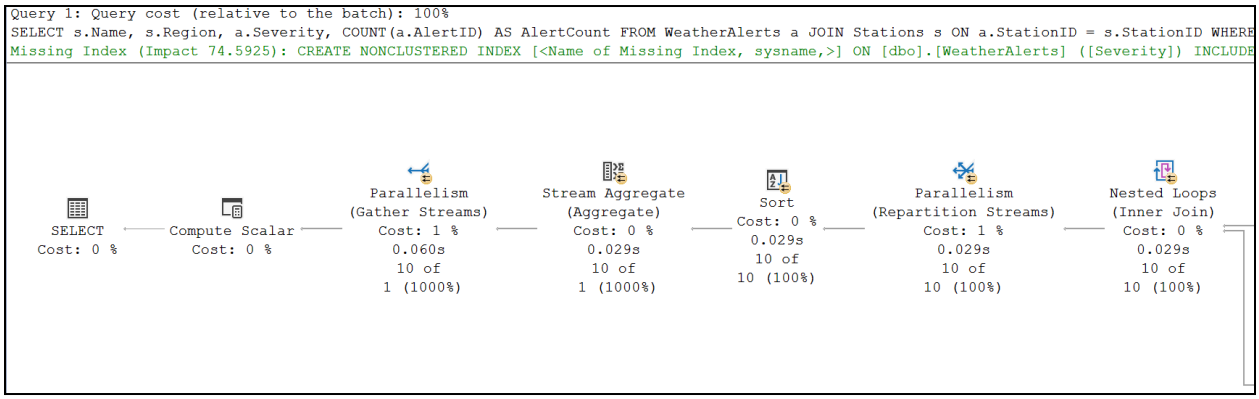
- ***Screenshot: Query 4 without the hint***



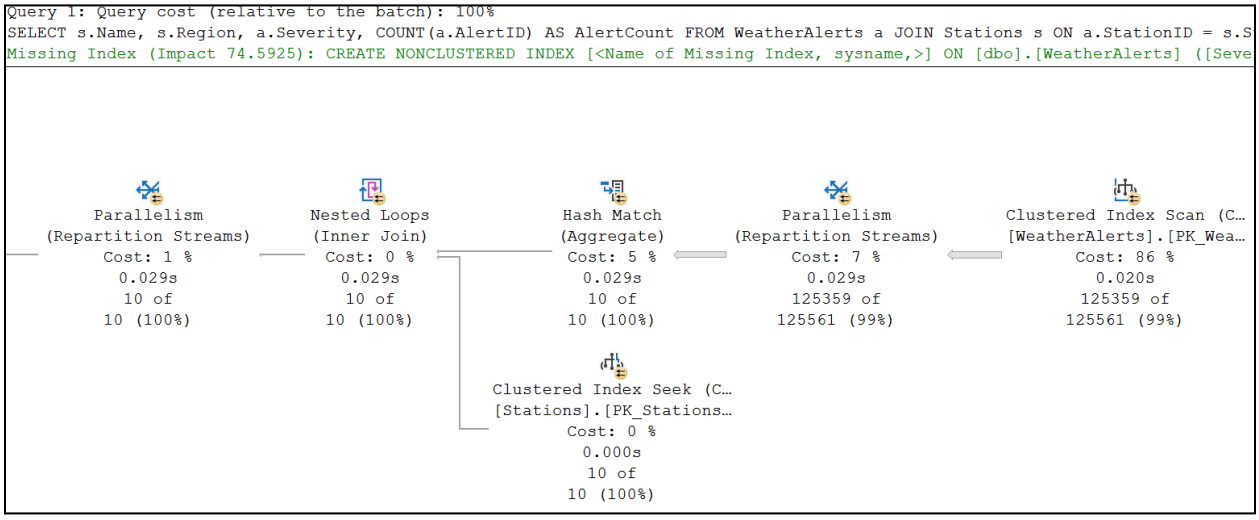
- ***Screenshot: Query 4 with the hint***



- ***Screenshot: Query 4 with the hint: Zoom in on the front.***



- ***Screenshot: Query 4 with the hint: Zoom in on the back.***



- **Comparison table**

Query Plan	Operations	Actual runtime (s)	Estimated Subtree Cost
Without hint	- Clustered Index Scan on WeatherAlerts table, then do Hash Aggregation - Clustered Index Seek on Stations table - Nested Loops Join - Sort - Stream Aggregation before the COUNT operation - No parallelism	0.189	4.2381
With hint	- Parallelism enabled - Clustered Index Scan on WeatherAlerts table, then do Parallelism (Repartition Streams) before Hash Aggregation - Clustered Index Seek on Stations table - Nested Loops Join then Parallelism (Repartition Streams) - Sort - Stream Aggregation - Parallelism (Gather Streams) before the COUNT operation	0.094	3.43166

Conclusion:

- Percent improvement = 100 * (1 - (0.094/0.189)) = 50.26 %
- The reason why the query optimizer makes the wrong decision:
  - The query optimizer chooses to avoid parallelism because the Stations table has a small data (100 rows) and a small join
  - The filtering on `a.Severity = 'None'`, possibly reducing WeatherAlerts row count substantially.
  - The `Group By` is on a low-cardinality combination of Region and Severity.
  - All the above cause the optimizer to underestimate the true cost.
- The reason why the query plan runs faster with the hint:
  - Use `OPTION(USE HINT('ENABLE_PARALLEL_PLAN_PREFERENCE'))` to tell the query optimizer to consider using parallelism.
- Query hint opened up better options for the query optimization, and it nudged the query optimizer to explore and choose a faster plan.

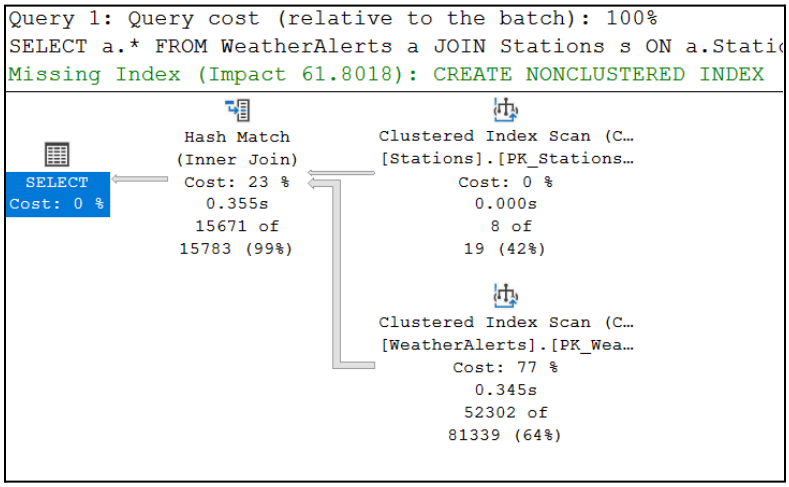
Query 5:

```sql
-- Query 5 without the hint
SELECT a.*
FROM WeatherAlerts a
JOIN Stations s ON a.StationID =
s.StationID
WHERE s.Region = 'WA'
  AND s.Latitude > 47 AND
s.Longitude < -120
  AND a.AlertDate BETWEEN
'2023-05-01' AND '2023-09-30'
  AND a.Severity = 'Extreme Heat
Alert'
```

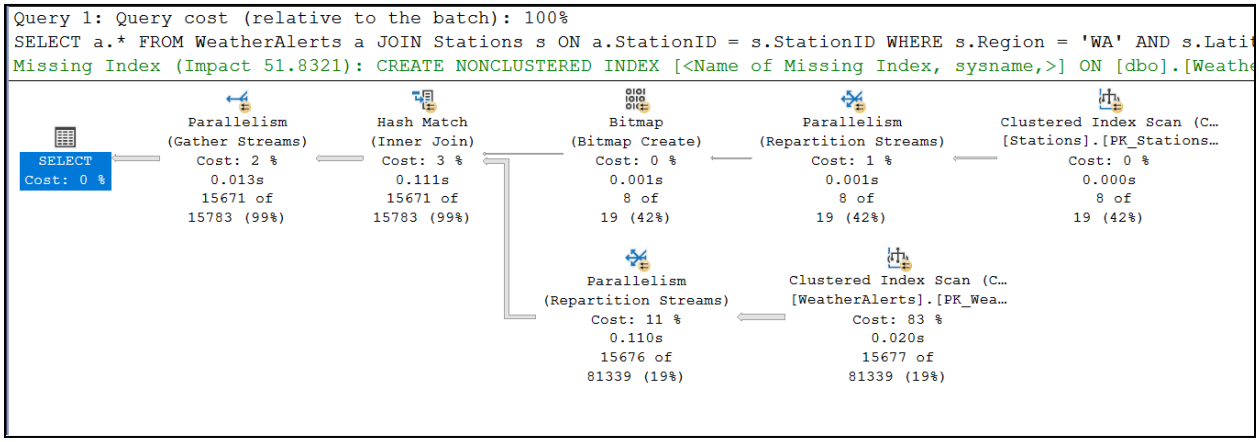
```sql
-- Query 5 with the hint
SELECT a.*
FROM WeatherAlerts a
JOIN Stations s ON a.StationID =
s.StationID
WHERE s.Region = 'WA'
  AND s.Latitude > 47 AND
s.Longitude < -120
  AND a.AlertDate BETWEEN
'2023-05-01' AND '2023-09-30'
  AND a.Severity = 'Extreme Heat
Alert'
OPTION(USE
HINT('ENABLE_PARALLEL_PLAN_PREFER
ENCE'));
```

**Add the `OPTION(USE HINT('ENABLE_PARALLEL_PLAN_PREFERENCE'));` It nudged the query optimizer to explore and choose a faster plan.

- ***Screenshot: Query 5 without the hint***



- ***Screenshot: Query 5 with the hint***



- **Comparison table**

Query Plan	**Operations**	**Actual runtime (s)**	**Estimated Subtree Cost**
Without hint	- Clustered Index Scan on both tables, Stations and WeatherAlerts - Hash Join - No parallelism	0.478	4.47265
With hint	- Parallelism enabled - Clustered Index Scan on Stations table, then Parallelism (Repartition Streams) before creating the Bitmap on the data - Clustered Index Scan on WeatherAlerts table, then do Parallelism (Repartition Streams) - Hash Join - Parallelism (Gather Streams)	0.174	3.56393

Conclusion:

- Percent improvement = 100 * (1 - (0.174/0.478)) = 63.60 %
- The reason why the query optimizer makes the wrong decision:
    - The query optimizer chooses to avoid parallelism because the small data in the Stations table, the WHERE clause on the WeatherAlerts seems selective, and the joining on the keys relationship makes the query seem to be cheap and doesn't need the parallelism.
- The reason why the query plan runs faster with the hint:
    - Use `OPTION(USE HINT('ENABLE_PARALLEL_PLAN_PREFERENCE'))` to tell the query optimizer to consider using parallelism.
    - The query optimizer allowed the Has Join and Scan to run in parallel.
    - The query optimizer chooses to use the Bitmap filtering to eliminate early rows.
- Query hint opened up better options for the query optimization and can help to cut the runtime by more than half (63.60%).