upGrad ENTERPRISE

Microsoft.NET Fullstack Bootcamp Training

Day23 (8th August, 2025)

upGrad ENTERPRISE

Execution Plans & Performance Tuning in SQL Server

By
Narasimha Rao T
Microsoft.Net FSD Trainer
Professional Development Trainer
tnrao.trainer@gmail.com

1. What is an Execution Plan in SQL Server?

An **execution plan** is the sequence of steps used by the SQL Server query optimizer to retrieve the requested data.

- It shows **how SQL Server processes a query internally**, including operations like scans, seeks, joins, and sorts.
- Helps in **identifying performance bottlenecks**.

Example:

```
SELECT * FROM Employees WHERE DepartmentID = 3;
```

- SQL Server decides whether to scan the entire table or seek using an index.

How to View:

- Use **Ctrl + M** in SSMS (Include Actual Execution Plan).
- OR prefix the query with:

```
SET SHOWPLAN_ALL ON;
```

2. Advantages of Using Execution Plans

Performance Insight: See how SQL Server executes queries.

Bottleneck Detection: Find slow operations like table scans.

Index Optimization: See if indexes are used.

Better Query Tuning: Helps optimize joins, filters, and aggregations.

Types of Execution Plans

Types of Execution Plans

1. **Estimated Execution Plan**: Generated without running the query, based on statistics and assumptions. Displayed using SET SHOWPLAN_ALL ON or via SSMS (Ctrl+L).
2. **Actual Execution Plan**: Generated after the query runs, showing real-time metrics like row counts and resource usage. Enabled with SET STATISTICS PROFILE ON or via SSMS (Ctrl+M).

3. Estimated vs Actual Execution Plans

Feature	Estimated Plan	Actual Plan
Based on	Statistics	Actual Execution
Shows Execution?	No	Yes
Uses resources?	No execution	Executes the query
When to use?	Early design phase	When tuning real performance

Example:

```
-- Estimated Plan
SET SHOWPLAN_XML ON;
GO
SELECT * FROM Products WHERE CategoryID = 2;
GO
SET SHOWPLAN_XML OFF;
```

```
-- Actual Plan
-- Enable via SSMS: Query → Include Actual Execution Plan
SELECT * FROM Products WHERE CategoryID = 2;
```

Clustered Index Scan (Clustered)

Scanning a clustered index, entirely or only a range.

Physical Operation	Clustered Index Scan
Logical Operation	Clustered Index Scan
Estimated Execution Mode	Row
Storage	RowStore
Estimated Operator Cost	0.003304 (100%)
Estimated I/O Cost	0.003125
Estimated Subtree Cost	0.003304
Estimated CPU Cost	0.000179
Estimated Number of Executions	1
Estimated Number of Rows to be Read	20
Estimated Number of Rows for All Executions	5
Estimated Number of Rows Per Execution	5
Estimated Row Size	80 B
Ordered	False
Node ID	0

Predicate

[EmployeeDb].[dbo].[Emps].[Salary]>CONVERT_IMPLICIT(decimal(10,2),[@1],0)

Object

[EmployeeDb].[dbo].[Emps].[PK__Emps__AF226A2B849EC6B2]

Output List

[EmployeeDb].[dbo].[Emps].Empno, [EmployeeDb].[dbo].[Emps].Ename, [EmployeeDb].[dbo].[Emps].Job, [EmployeeDb].[dbo].[Emps].Salary, [EmployeeDb].[dbo].[Emps].Deptno

Clustered Index Scan (Clustered)

Scanning a clustered index, entirely or only a range.

Physical Operation	Clustered Index Scan
Logical Operation	Clustered Index Scan
Actual Execution Mode	Row
Estimated Execution Mode	Row
Storage	RowStore
Actual Number of Rows Read	20
Actual Number of Rows for All Executions	7
Actual Number of Batches	0
Estimated I/O Cost	0.003125
Estimated Operator Cost	0.003304 (100%)
Estimated Subtree Cost	0.003304
Estimated CPU Cost	0.000179
Estimated Number of Executions	1
Number of Executions	1
Estimated Number of Rows for All Executions	5
Estimated Number of Rows Per Execution	5
Estimated Number of Rows to be Read	20
Estimated Row Size	80 B
Actual Rebinds	0
Actual Rewinds	0
Ordered	False
Node ID	0

Predicate

[EmployeeDb].[dbo].[Emps].[Salary]>CONVERT_IMPLICIT(decimal(10,2),[@1],0)

Object

[EmployeeDb].[dbo].[Emps].[PK__Emps__AF226A2B849EC6B2]

Output List

[EmployeeDb].[dbo].[Emps].Empno, [EmployeeDb].[dbo].[Emps].Ename, [EmployeeDb].[dbo].[Emps].Job, [EmployeeDb].[dbo].[Emps].Salary, [EmployeeDb].[dbo].[Emps].Deptno

Execution Plan Operators

4. Key Execution Plan Operators

Table Scan:

- Reads entire table.
- Bad for large tables.

```
SELECT * FROM Orders;
```

Index Seek:

- Efficiently finds rows using an index.
- Best performance.

```
SELECT * FROM Orders WHERE OrderID = 10248;
```

Index Scan:

- Scans all index entries.
- Acceptable for small result sets.

Nested Loop Join:

- Best for small input sets.
- Executes outer loop for each row of inner.

Merge Join:

- Good for sorted inputs.
- Fast and efficient for large datasets.

Hash Match:

- Uses hash table, good for large unindexed data.

Sort:

- Expensive if not already sorted.
- Appears when using `ORDER BY`.

SQL Server Profiler

5. SQL Server Profiler for Tracing Queries

SQL Server Profiler is a GUI tool that monitors SQL Server events.

Features:

- Trace query execution in real-time.
- View duration, reads, writes, and CPU usage.
- Identify **long-running queries** and bottlenecks.

Real-time Use Case:

- Monitoring production server performance.
- Debugging deadlocks and timeouts.

How to Use:

- Open Profiler → File → New Trace.
- Select template: **TSQL_Replay** or **Tuning**.
- Start and monitor live activity.

6. Query Hints in SQL Server

Hints guide the optimizer to follow specific strategies.

JOIN Hints:

Force join types (LOOP, MERGE, HASH).

```
SELECT * FROM A
INNER HASH JOIN B ON A.ID = B.ID;
```

INDEX Hint:

Force SQL Server to use a specific index.

```
SELECT * FROM Products WITH (INDEX(Index_ProductName))
WHERE ProductName = 'Chai';
```

TABLE Hint (NOLOCK):

Read data without locks — dirty reads possible.

```
SELECT * FROM Orders WITH (NOLOCK);
```

Use with caution — may read uncommitted data.

7. Performance Tuning with Execution Plans

Key Techniques:

- **Add Indexes** for frequently used WHERE columns.
- **Avoid SELECT *** — return only needed columns.
- **Use SARGable Queries**: Avoid wrapping columns in functions.

Bad Example:

```
SELECT * FROM Customers WHERE YEAR(JoinDate) = 2022;
```

Prevents index usage.

Good Example:

```
SELECT * FROM Customers
WHERE JoinDate >= '2022-01-01' AND JoinDate < '2023-01-01';
```

8. Real-time Usages of Profiler and Execution Plans

Real-world Scenarios:

- Analyzing slow stored procedures.
- Finding missing indexes.
- Detecting expensive queries in peak hours.
- Profiling application-generated SQL.

Tools:

- SQL Profiler
- Extended Events (modern alternative to Profiler)
- Database Engine Tuning Advisor

upGrad ENTERPRISE

Q & A

Narasimha Rao T
tnrao.trainer@gmail.com