Execution Plans:
The Secret to Query Tuning Success

MagicPASS
January 2015
What’s an execution plan?
Inside SQL Server...

The following steps are being taken:

Parsing
Compiling
Optimizing
In the optimizing phase

An execution plan is built

This is expensive!
- CPU!
- So SQL Server stores them in memory (most of the time)
Some are very simple
Some are very complex
Where or how do I get a plan?

If you have a specific query you’re running in SSMS, turn on estimated or actual graphical plan

Use SHOWPLAN to view text or XML plan

If you’re running a trace or Extended Events session, you can capture the plan

Looking in the plan cache
  • Not all plans are stored here
  • They don’t stay forever
  • They are estimated plans
How do I view the plan?

SQL Server Management Studio
- Built-in
- Not pretty
- Hard to find trouble spots in big, complicated plans

SQL Sentry Plan Explorer
- 3rd party tool
- Free download
- Much better visibility of trouble spots
Estimated vs actual

Estimated – “the query execution plan that SQL Server Database Engine would most probably use if the queries were actually executed”
  • Can’t generate with parameters or temp tables

Actual – “the actual query execution plan that the SQL Server Database Engine uses to execute the queries”
  • Stats can be updated when the query is run

Can there be differences? Heck yes!
Understanding execution plans
We read right to left

The SELECT is the final operation

We start here!
Hover over an operator for more information.
The final operator
How much does that cost?

“Cost” tells us how much work – relatively - SQL Server did to return the results of the query

Lower is better
Why are my lines fat?
Common operators
Scan

Reads all the rows in the object

If there’s a WHERE clause, only the matching rows are output

• Table
• Clustered index
• Nonclustered index
Seek

Only the rows that satisfy the predicate are searched

If there’s a WHERE clause, only the matching rows are output

- Clustered index
- Nonclustered index
Lookup

All the needed columns are not in the index used, so the base table has to be accessed

RID Lookup
  • Performed against a heap

Key Lookup
  • Performed against a clustered index
Join types

Nested loops
Merge
Hash
Nested loops

There’s an outer table
...and an inner table
Each row in the outer table
...is compared to each row in the inner table

Works best if data sets are small
...particularly the outer set
Think of it this way

You have a handful of Skittles

And a bag of M&Ms
“What colors do they have in common?”
Compare, piece by piece
Merge

First: both indexes need to be sorted
Then: both sides are compared
Are they equal? OK, good!
Think of it this way

You have a bag of Skittles

And a bag of M&Ms
“What colors do they have in common?”
You sort the Skittles by color
You sort the M&Ms by color
You compare them

3 matching yellow
3 matching orange
3 matching green
3 matching red
Hash

Each value in each index is converted to a hash

The hashes are put in buckets

The buckets are compared

Great for large data sets!
Think of it this way

You have a big bag of Skittles

And a big bag of M&Ms
“What colors do they have in common?”
You sort the Skittles by color, and put 100 in each bowl
You sort the M&Ms by color, and put 100 in each bowl.
Then we compare the bowls
Sort

Usually seen with ORDER BY

SQL Server will try to sort in memory first
Not enough room? Will spill over to tempdb
Look out for Sort Warnings!
Compute scalar

A computation is being performed

What is it?

In SSMS, click the ... in Properties
Parallelism

Not an operator in itself, but indicates operations that were able to run in parallel

Not all queries will be able to run in parallel

- Parallelism must be enabled
- The cost to run the query in serial must be above the cost threshold for parallelism
- There must be no parallelism-inhibiting components

INSERT
UPDATE
DELETE
Are they any different?
Nonclustered indexes exist on four fields:

- ProductID (PK, int, not null)
- Name (nvarchar(80), null)
- ProductNumber (nvarchar(56), null)
- MakeFlag (Flag(bit), not null)
- FinishedGoodsFlag (Flag(bit), not null)
- Color (nvarchar(15), null)
- SafetyStockLevel (smallint, not null)
- ReorderPoint (smallint, not null)
- StandardCost (money, not null)
- ListPrice (money, not null)
- Size (nvarchar(5), null)
- SizeUnitMeasureCode (nchar(3), null)
- WeightUnitMeasureCode (nchar(3), null)
- Weight (decimal(8,2), null)
- DaysToManufacture (int, not null)
- ProductLine (nchar(2), null)
- Class (nchar(2), null)
- Style (nchar(2), null)
- ProductSubcategoryID (int, null)
- ProductModelID (int, null)
- SellStartDate (datetime, not null)
- SellEndDate (datetime, null)
- DiscontinuedDate (datetime, null)
UPDATE dbo.bigProduct
SET ProductID = 51002
WHERE ProductID = 51001;
Properties for Clustered Index Update

<table>
<thead>
<tr>
<th>Number of Execution</th>
<th>Object</th>
<th>Output List</th>
</tr>
</thead>
</table>
Plan Explorer
Finding common problems
Key lookups
Scan

Reads all the rows in the object

If there’s a WHERE clause, only the matching rows are output

- Table
- Clustered index
- Nonclustered index
Seek

Only the rows that satisfy the predicate are searched

If there’s a WHERE clause, only the matching rows are output

- Clustered index
- Nonclustered index
Lookup

All the needed columns are not in the index used, so the base table has to be accessed.

RID Lookup
  • Performed against a heap

Key Lookup
  • Performed against a clustered index
Table
Query

SELECT SalesOrderID, SalesOrderDetailID, ProductID, OrderQty, UnitPrice
FROM Sales.SalesOrderDetail
WHERE CarrierTrackingNumber = 'AB1B-4CE2-BA'
Fixing it

Depending on how many columns are being retrieved, adding key or include columns to the index may relieve the problem

Or, reduce the number of columns returned in your query!
Predicates instead of seek predicates
<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ProductID (PK)</td>
<td>int, not null</td>
<td></td>
<td>Primary key</td>
</tr>
<tr>
<td>Name</td>
<td>nvarchar(80)</td>
<td>null</td>
<td></td>
</tr>
<tr>
<td>ProductNumber</td>
<td>nvarchar(56)</td>
<td>null</td>
<td></td>
</tr>
<tr>
<td>MakeFlag</td>
<td>Flag(bit)</td>
<td>not null</td>
<td></td>
</tr>
<tr>
<td>FinishedGoodsFlag</td>
<td>Flag(bit)</td>
<td>not null</td>
<td></td>
</tr>
<tr>
<td>Color</td>
<td>nvarchar(15)</td>
<td>null</td>
<td></td>
</tr>
<tr>
<td>SafetyStockLevel</td>
<td>smallint</td>
<td>not null</td>
<td></td>
</tr>
<tr>
<td>ReorderPoint</td>
<td>smallint</td>
<td>not null</td>
<td></td>
</tr>
<tr>
<td>StandardCost</td>
<td>money</td>
<td>not null</td>
<td></td>
</tr>
<tr>
<td>ListPrice</td>
<td>money</td>
<td>null</td>
<td></td>
</tr>
<tr>
<td>Size</td>
<td>nvarchar(5)</td>
<td>null</td>
<td></td>
</tr>
<tr>
<td>SizeUnitMeasureCode</td>
<td>nchar(3)</td>
<td>null</td>
<td></td>
</tr>
<tr>
<td>WeightUnitMeasureCode</td>
<td>nchar(3)</td>
<td>null</td>
<td></td>
</tr>
<tr>
<td>Weight</td>
<td>decimal(3,2)</td>
<td>null</td>
<td></td>
</tr>
<tr>
<td>DaysToManufacture</td>
<td>int</td>
<td>not null</td>
<td></td>
</tr>
<tr>
<td>ProductLine</td>
<td>nchar(2)</td>
<td>null</td>
<td></td>
</tr>
<tr>
<td>Class</td>
<td>nchar(2)</td>
<td>null</td>
<td></td>
</tr>
<tr>
<td>Style</td>
<td>nchar(2)</td>
<td>null</td>
<td></td>
</tr>
<tr>
<td>ProductSubcategoryId</td>
<td>int</td>
<td>null</td>
<td></td>
</tr>
<tr>
<td>ProductModelID</td>
<td>int</td>
<td>null</td>
<td></td>
</tr>
<tr>
<td>SellStartDate</td>
<td>datetime</td>
<td>not null</td>
<td></td>
</tr>
<tr>
<td>SellEndDate</td>
<td>datetime</td>
<td>null</td>
<td></td>
</tr>
<tr>
<td>DiscontinuedDate</td>
<td>datetime</td>
<td>null</td>
<td></td>
</tr>
</tbody>
</table>
Seek predicates vs predicates

```sql
SELECT ProductID, Name, Color, Size, Weight
FROM dbo.bigproduct
WHERE ProductID = 50345;
```

ProductID is a (Clustered) Index key column – can SEEK on it

```
SELECT ProductID, Name, Color, Size, Weight
FROM dbo.bigproduct
WHERE Color IS NOT NULL;
```

Color is not a key column – can’t SEEK on it
## Clustered Index Seek (Clustered)
Scanning a particular range of rows from a clustered index.

<table>
<thead>
<tr>
<th>Physical Operation</th>
<th>Clustered Index Seek</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logical Operation</td>
<td>Clustered Index Seek</td>
</tr>
<tr>
<td>Actual Execution Mode</td>
<td>Row</td>
</tr>
<tr>
<td>Estimated Execution Mode</td>
<td>Row</td>
</tr>
<tr>
<td>Storage</td>
<td>RowStore</td>
</tr>
<tr>
<td>Actual Number of Rows</td>
<td>1</td>
</tr>
<tr>
<td>Actual Number of Batches</td>
<td>0</td>
</tr>
<tr>
<td>Estimated Operator Cost</td>
<td>0.0032831 (100%)</td>
</tr>
<tr>
<td>Estimated I/O Cost</td>
<td>0.003125</td>
</tr>
<tr>
<td>Estimated CPU Cost</td>
<td>0.0001561</td>
</tr>
<tr>
<td>Estimated Subtree Cost</td>
<td>0.0032831</td>
</tr>
<tr>
<td>Number of Executions</td>
<td>1</td>
</tr>
<tr>
<td>Estimated Number of Executions</td>
<td>1</td>
</tr>
<tr>
<td>Estimated Number of Rows</td>
<td>1</td>
</tr>
<tr>
<td>Estimated Row Size</td>
<td>124 B</td>
</tr>
<tr>
<td>Actual Rebinds</td>
<td>0</td>
</tr>
<tr>
<td>Actual Rewinds</td>
<td>0</td>
</tr>
<tr>
<td>Ordered</td>
<td>True</td>
</tr>
<tr>
<td>Node ID</td>
<td>0</td>
</tr>
</tbody>
</table>

### Object
[AdventureWorks2012].[dbo].[bigProduct].[pk_bigProduct]

### Output List
[AdventureWorks2012].[dbo].[bigProduct].ProductID,
[AdventureWorks2012].[dbo].[bigProduct].Name,
[AdventureWorks2012].[dbo].[bigProduct].Color,
[AdventureWorks2012].[dbo].[bigProduct].Size,
[AdventureWorks2012].[dbo].[bigProduct].Weight

### Seek Predicates
Seek Keys[1]: Prefix [AdventureWorks2012].[dbo].[bigProduct].ProductID = Scalar Operator (@1)

---

## Clustered Index Scan (Clustered)
Scanning a particular index, entirely or only a range.

<table>
<thead>
<tr>
<th>Physical Operation</th>
<th>Clustered Index Scan</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logical Operation</td>
<td>Clustered Index Scan</td>
</tr>
<tr>
<td>Actual Execution Mode</td>
<td>Row</td>
</tr>
<tr>
<td>Estimated Execution Mode</td>
<td>Row</td>
</tr>
<tr>
<td>Actual Number of Rows</td>
<td>12799</td>
</tr>
<tr>
<td>Actual Number of Batches</td>
<td>0</td>
</tr>
<tr>
<td>Estimated I/O Cost</td>
<td>0.446829</td>
</tr>
<tr>
<td>Estimated Operator Cost</td>
<td>0.474706 (100%)</td>
</tr>
<tr>
<td>Estimated CPU Cost</td>
<td>0.027877</td>
</tr>
<tr>
<td>Estimated Subtree Cost</td>
<td>0.474706</td>
</tr>
<tr>
<td>Number of Executions</td>
<td>1</td>
</tr>
<tr>
<td>Estimated Number of Executions</td>
<td>1</td>
</tr>
<tr>
<td>Estimated Number of Rows</td>
<td>12800</td>
</tr>
<tr>
<td>Estimated Row Size</td>
<td>114 B</td>
</tr>
<tr>
<td>Actual Rebinds</td>
<td>0</td>
</tr>
<tr>
<td>Actual Rewinds</td>
<td>0</td>
</tr>
<tr>
<td>Ordered</td>
<td>False</td>
</tr>
<tr>
<td>Node ID</td>
<td>0</td>
</tr>
</tbody>
</table>

### Predicate
[AdventureWorks2012].[dbo].[bigProduct].[Color] IS NOT NULL

### Object
[AdventureWorks2012].[dbo].[bigProduct].[pk_bigProduct]

### Output List
[AdventureWorks2012].[dbo].[bigProduct].ProductID,
[AdventureWorks2012].[dbo].[bigProduct].Name,
[AdventureWorks2012].[dbo].[bigProduct].Color,
[AdventureWorks2012].[dbo].[bigProduct].Size,
[AdventureWorks2012].[dbo].[bigProduct].Weight
Seek predicates vs predicates

Seek predicate: values in the WHERE clause line up with the key columns of an index

Predicate: values in the WHERE clause don’t match key columns, and the index is scanned

Predicates lead to: Extra I/O and CPU (especially if it’s millions of rows)

Mitigations: Tune query or modify index
Fixing it

Depending on how many columns are being retrieved, adding key or include columns to the index may relieve the problem.

Rewrite the query to search on fields that are indexed.
Implicit conversion
It’s all about the data types

SQL Server can’t compare unlike data types

It compares fields when
  • They are used in a join
  • They are used in a where clause
Joins

CREATE TABLE TestInt
(ID INT IDENTITY PRIMARY KEY,
Name VARCHAR(25))

CREATE TABLE TestVarchar
(ID VARCHAR(25),
Name VARCHAR(50))

SELECT TestInt.id, TestInt.Name as Company, TestVarchar.Name as Employee
FROM TestInt
INNER JOIN TestVarchar ON TestVarchar.id = TestInt.id
Query 1: Query cost (relative to the batch): 100%

```
SELECT TestInt.id, TestInt.Name as Company, TestVarchar.Name as Employees
FROM TestInt INNER JOIN TestVarchar
ON TestVarchar.id = TestInt.id
```

**Execution Plan**

- **SELECT** Cost: 0 %
  - Nested Loops (Inner Join) Cost: 0 %
  - Compute Scalar Cost: 0 %
  - Table Scan [TestVarchar] Cost: 50 %

**Clustered Index Seek (Clustered)**

- [TestInt].{PK_TestInt_5214EC27090... Cost: 50 %

**Details**

- Coached plan size: 16 KB
- Degree of Parallelism: 1
- Estimated Operator Cost: 0.03%
- Estimated Subtree Cost: 0.0065705
- Estimated Number of Rows: 1

**Warnings**

- Type conversion in expression
  - `CONVERT_IMPLICIT(int, [master].[dbo]. [TestVarchar].[id])` may affect "CardinalityEstimate" in query plan choice.
- Type conversion in expression
  - `CONVERT_IMPLICIT(int, [master].[dbo]. [TestVarchar].[id])` may affect "SeekPlan" in query plan choice.
WHERE

CREATE TABLE TestInt
(ID INT IDENTITY PRIMARY KEY,
Name VARCHAR(25))

DECLARE @ID VARCHAR(25) = 1

SELECT *
FROM TestInt
WHERE ID = @ID
This causes problems

Extra CPU - every row that is compared must have a conversion applied
Fixing it

Finding it is the hardest part!
  • Review high-CPU queries in the plan cache
  • Query the XML of the plans in the cache (you can run screaming now)

Change data type of parameter

Change data type in table
Finding help
Measuring impact
Executing a query

```
SELECT TransactionID, ProductID, Quantity
FROM dbo.bigTransactionHistory
WHERE TransactionDate = '2007/06/30';
```

(13684 row(s) affected)
Turn on options

SET STATISTICS IO ON
SET STATISTICS TIME ON
SET STATISTICS IO ON

SET STATISTICS TIME ON

SELECT TransactionID,
       ProductID,
       Quantity
FROM dbo.bigTransactionHistory
WHERE TransactionDate = '2007/06/30';
Missing indexes
dbo.bigProduct

- ProductID (PK, int, not null)
- Name (nchar(80), null)
- ProductNumber (nvarchar(50), null)
- MakeFlag (Flag(bit), not null)
- FinishedGoodsFlag (Flag(bit), not null)
- Color (nvarchar(15), null)
- SafetyStockLevel (smalldatetime, not null)
- ReorderPoint (smalldatetime, not null)
- StandardCost (money, not null)
- ListPrice (money, not null)
- Size (nvarchar(5), null)
- SizeUnitMeasureCode (nchar(3), null)
- WeightUnitMeasureCode (nchar(3), null)
- Weight (decimal(3,2), null)
- DaysToManufacture (int, not null)
- ProductLine (nchar(2), null)
- Class (nchar(2), null)
- Style (nchar(2), null)
- ProductSubcategoryID (int, null)
- ProductModelID (int, null)
- SellStartDate (datetime, not null)
- SellEndDate (datetime, null)
- DiscontinuedDate (datetime, null)
Run a query

```sql
SELECT ProductID,
       Name,
       Color,
       Size,
       Weight,
       DaysToManufacture
FROM dbo.bigProduct
WHERE Size = 'M';
```
Help me!
Right-click for more info

```sql
/*
2 Missing Index Details from SQLQuery4.sql - JES-X230.AdventureWorks2012 (JES-X230\Jes (S7))
3 The Query Processor estimates that implementing the following index could improve the query cost by 95.5472%.
4 */
5
6 /*
7 USE [AdventureWorks2012]
8 GO
9 CREATE NONCLUSTERED INDEX [<Name of Missing Index, sysname,>]
10 ON [dbo].[bigProduct] ([Size])
11 INCLUDE ([ProductID],[Name],[Color],[Weight],[DaysToManufacture])
12 GO
13 */
14```
Careful!

Only the first missing index recommendation is shown here!
  • Parse the XML to get all recommendations in SSMS
  • Plan Explorer shows all

Get more info
  • Missing index DMVs
    • sys.dm_db_missing_index_group_stats
    • sys.dm_db_missing_index_groups
    • sys.dm_db_missing_index_details
    • sys.dm_db_missing_index_columns
  • sp_BlitzIndex®
    • http://www.brentozar.com/blitzindex/
Missing statistics
## Table

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Nullability</th>
</tr>
</thead>
<tbody>
<tr>
<td>SalesOrderID</td>
<td>int, not null</td>
<td></td>
</tr>
<tr>
<td>SalesOrderDetailID</td>
<td>int, not null</td>
<td></td>
</tr>
<tr>
<td>CarrierTrackingNumber</td>
<td>nvarchar(25), null</td>
<td></td>
</tr>
<tr>
<td>OrderQty</td>
<td>smallint, not null</td>
<td></td>
</tr>
<tr>
<td>ProductID</td>
<td>int, not null</td>
<td></td>
</tr>
<tr>
<td>SpecialOfferID</td>
<td>int, not null</td>
<td></td>
</tr>
<tr>
<td>UnitPrice</td>
<td>money, not null</td>
<td></td>
</tr>
<tr>
<td>UnitPriceDiscount</td>
<td>money, not null</td>
<td></td>
</tr>
<tr>
<td>LineTotal</td>
<td>numeric(38,9), not null</td>
<td></td>
</tr>
<tr>
<td>rowguid</td>
<td>uniqueidentifier, not null</td>
<td></td>
</tr>
<tr>
<td>ModifiedDate</td>
<td>datetime, not null</td>
<td></td>
</tr>
<tr>
<td>TotalPrice</td>
<td>money, null</td>
<td></td>
</tr>
<tr>
<td>TotalPricePersisted</td>
<td>money, null</td>
<td></td>
</tr>
</tbody>
</table>

**Clustered index key:** SalesOrderID  
**Nonclustered index:** ProductID, OrderQty
AUTO_CREATE_STATISTICS

ALTER DATABASE [AdventureWorks2012]
SET AUTO_CREATE_STATISTICS OFF
WITH NO_WAIT
Query

SELECT ProductID
FROM dbo.Copy_SalesOrderDetail
WHERE OrderQty = 5
Another query

```
SELECT *
FROM dbo.Copy_SalesOrderDetail
WHERE SalesOrderID = 53541
```
Indexed view
53  SELECT  CustomerID ,
      ProductID ,
      OrderQty ,
      LineTotal ,
      CountOrderLines
FROM  SalesByCustByProduct;
Add statistics

```
CREATE STATISTICS ST_SalesByCustByProduct_ProductID
ON dbo.SalesByCustByProduct(ProductID)
WITH FULLSCAN;
```
...and nothing changes

To avoid “Columns with no statistics” errors on indexed views, use the WITH NOEXPAND hint.
What we’ve learned
The basics

How to find the execution plan of a query

Reading right to left

Understanding seek vs. scan, lookups, nested loop vs. merge vs. hash joins, sorts, and computes

Being able to find the cost
  • And, as you iterate through either index tuning or query tuning, see if cost is being reduced
Finding where it hurts

Looking for the index seek/key lookup pattern
Watching out for seek predicates vs predicates
Being aware of implicit conversion
Listening when it asks for help

Using STATISTICS TIME, IO to measure your query impact

Paying attention to missing index requests

Paying attention to missing statistics requests
Resources
Books and articles


simple-talk.com

sqlperformance.com
Jes Schultz Borland

jes@brentozar.com

@grrl_geek