What Every Developer Should Know About SQL Server Performance

David Berry
@DavidCBerry13

About This Talk

Pluralsight Course	http://bit.ly/SqlPerformanceCourse
Sample Database	http://bit.ly/SampleSqlPerformanceDatabase
DMV Queries	http://bit.ly/PluralsightCourseDmvQueries

Why are we here?

- Almost every system built uses an RDBMS in some capacity

- Performance is important for every system we build

What Will You Learn?

- How to analyze a SQL statement
- How to create effective indexes
- How to find your worst performing SQL statements
- Profiling SQL Server
- Performance practices

How do I write SQL statements that run fast?

Think **less** about how long your SQL statement takes to run

Think **more** about making your statement more efficient by using **less CPU**, performing **less IO** and having a **lower cost**

Execution Plan Cost

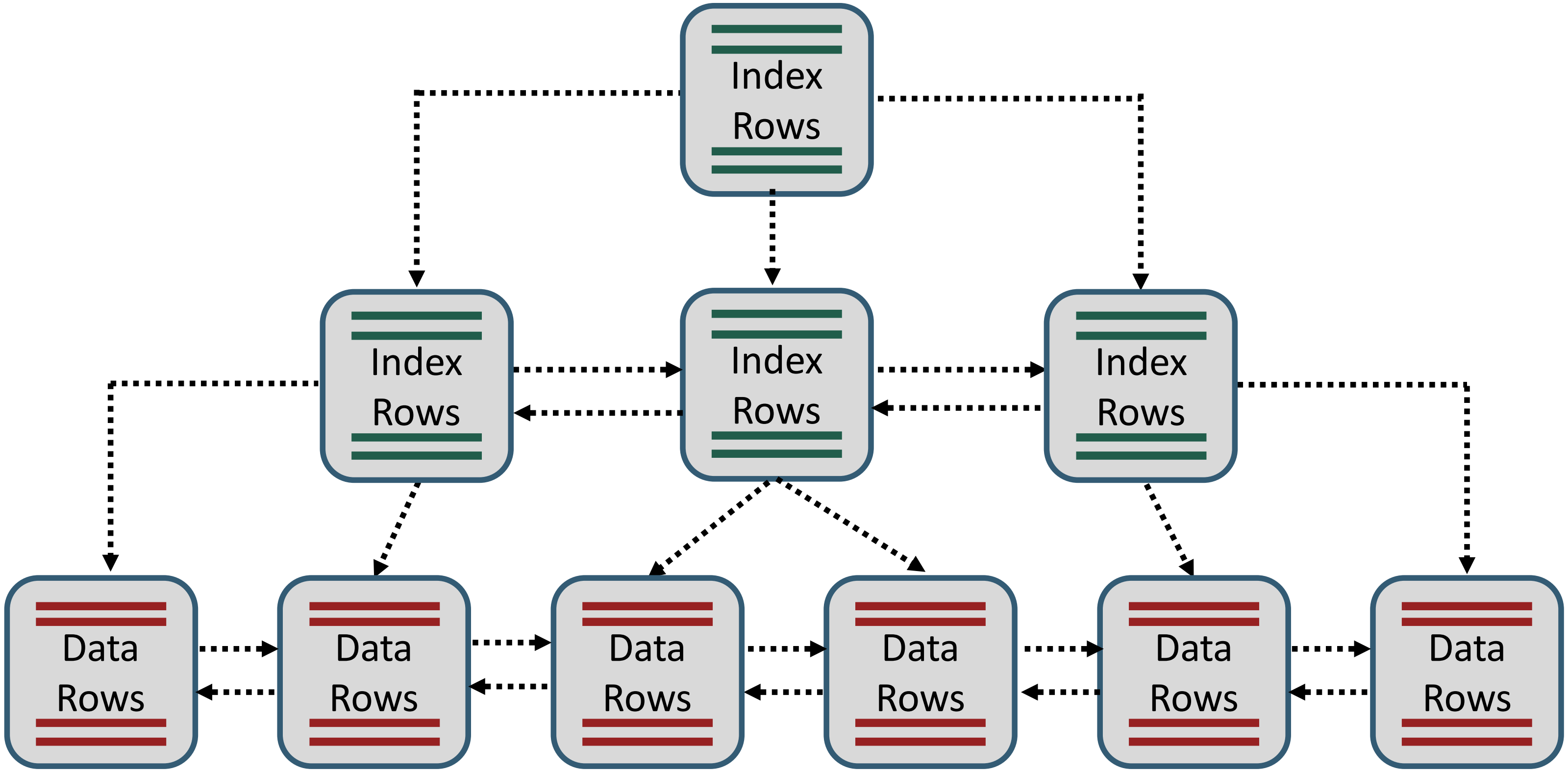
A **normalized, dimensionless value** that combines together both the **CPU** and **IO** cost of executing a statement into a **single value**

Lower cost statements will **run faster** and use **fewer resources** on the database server

Scan Operations

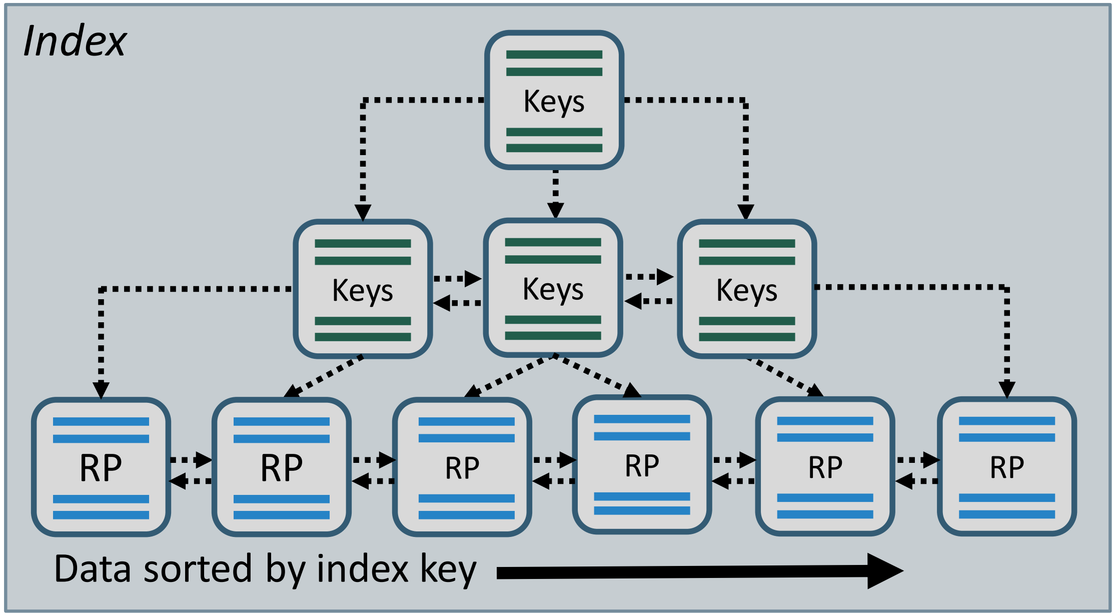
Think of these like a **linear search** through an array. Fast if the table (array) is small. **Very expensive** if the table (array) is large

How Data is Stored in SQL Server



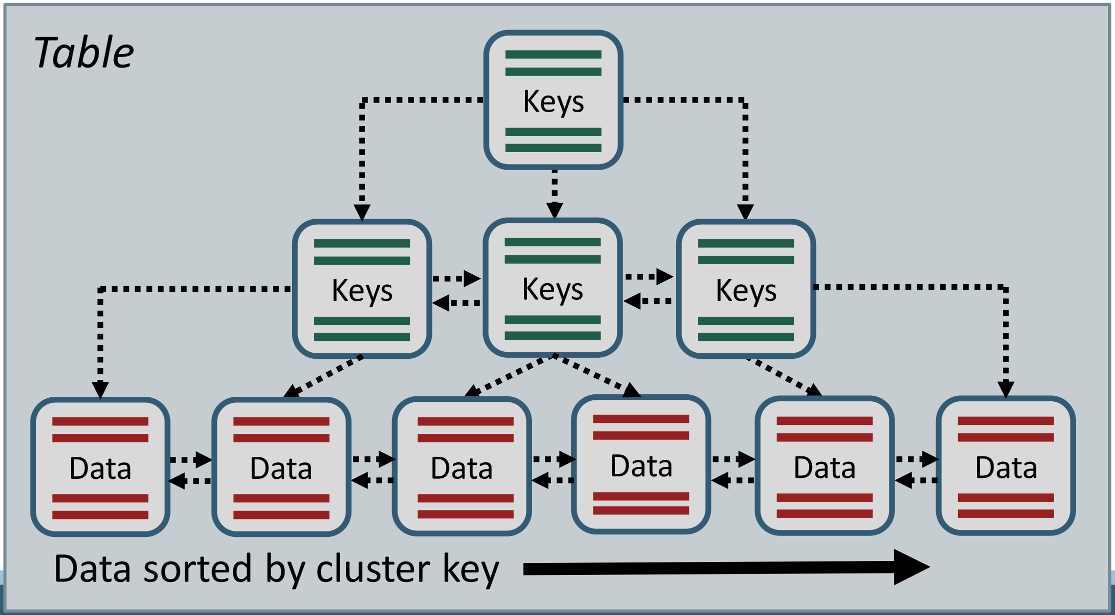
Data is stored in sorted order by the cluster key ⇢

How SQL Server Uses an Index

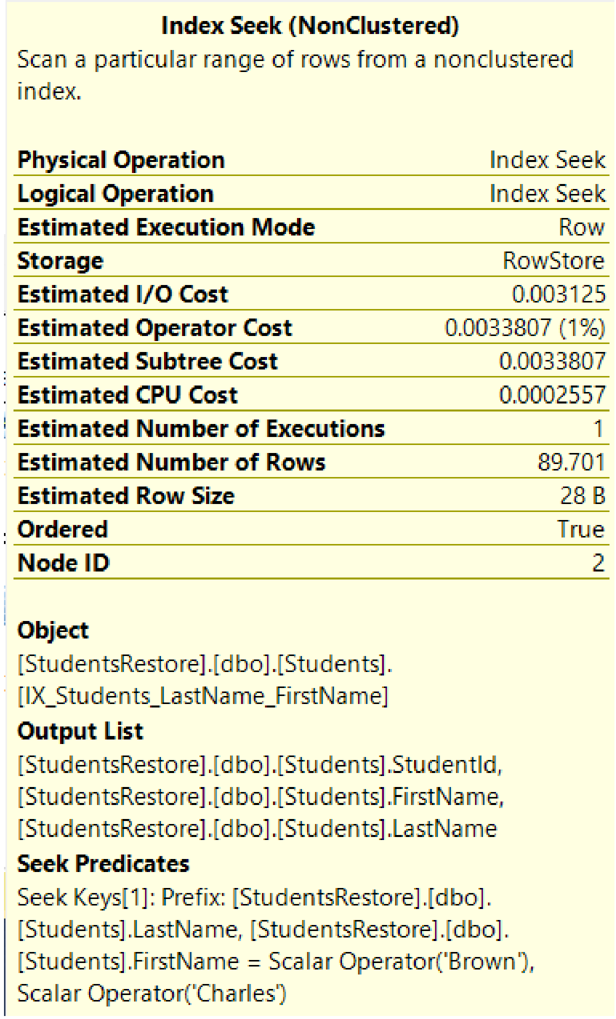


1. Matching keys are found in the index

2. Data is looked up in the table using the row pointers



Plan Operation Information



Index Seek (NonClustered)

Scan a particular range of rows from a nonclustered index.

Physical Operation	Index Seek
Logical Operation	Index Seek
Estimated Execution Mode	Row
Storage	RowStore
Estimated I/O Cost	0.003125
Estimated Operator Cost	0.0033807 (1%)
Estimated Subtree Cost	0.0033807
Estimated CPU Cost	0.0002557
Estimated Number of Executions	1
Estimated Number of Rows	89.701
Estimated Row Size	28 B
Ordered	True
Node ID	2

Object

[StudentsRestore].[dbo].[Students].[IX_Students_LastName_FirstName]

Output List

[StudentsRestore].[dbo].[Students].StudentId,
[StudentsRestore].[dbo].[Students].FirstName,
[StudentsRestore].[dbo].[Students].LastName

Seek Predicates

Seek Keys[1]: Prefix: [StudentsRestore].[dbo].[Students].LastName, [StudentsRestore].[dbo].[Students].FirstName = Scalar Operator('Brown'), Scalar Operator('Charles')

Description

Operation Cost

How many times this operation is executed

Estimated rows returned by this operation

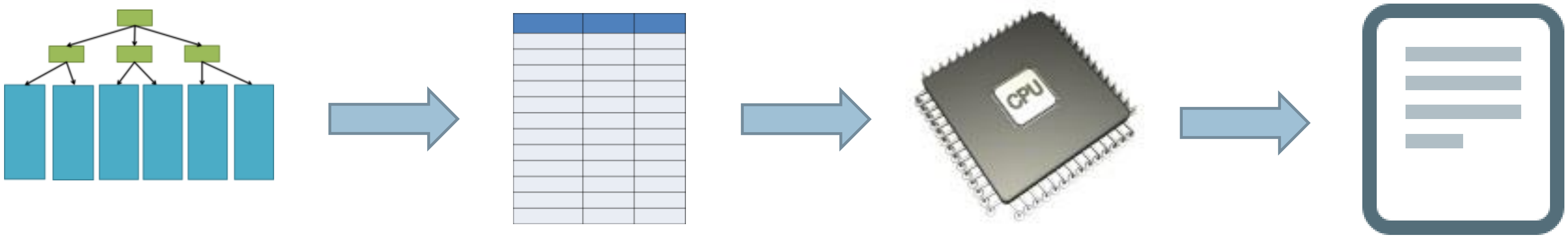
Target object

Predicate (will match WHERE clause or join operation)

Number of Rows Matters

```sql
SELECT *
    FROM Students
WHERE LastName = 'Jones'
AND FirstName = 'Renee';
```

Index on LastName only



Index Seek
Estimate 914 rows returned

Row Lookup
Read 914 rows from table

Filter
Filter through rows to match FirstName

Results
1 row returned

Number of Rows Matters

More rows mean **more data** SQL Server is processing. This means more IO, more CPU and more time to execute

We want to help SQL Server trim down the data it is looking through as early as possible by using **specific WHERE clauses** and **selective indexes**

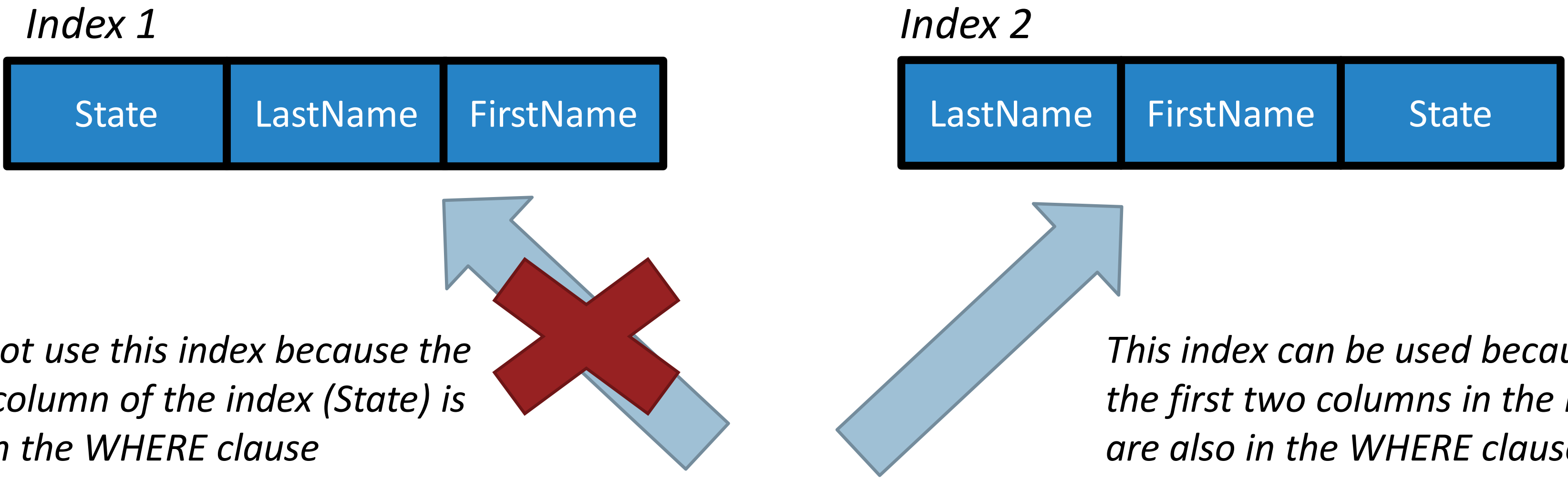
Statement Statistics

```
SET STATISTICS TIME ON;
SET STATISTICS IO ON;

// Run SQL Statement

// In the Messages Window
SQL Server parse and compile time:
   CPU time = 0 ms, elapsed time = 0 ms.
SQL Server parse and compile time:
   CPU time = 0 ms, elapsed time = 0 ms.

(8 row(s) affected)
Table 'Students'. Scan count 1, logical reads 3747, physical reads 0, read-ahead reads 0, lob logical reads 0, lob physical reads 0, lob read-ahead reads 0.

 SQL Server Execution Times:
   CPU time = 16 ms,  elapsed time = 12 ms.
```

Common Execution Plan Operations

Operation	Description
Clustered Index Scan	SQL Server reads all of the rows of a table looking for the rows that match the criteria. Very slow for large tables.
Clustered Index Seek	Traverses the tree structure of the table stored as a clustered index to find the needed row(s)
Index Scan	Reads all of the key values of the index to find the matching data
Index Seek	Traverses the tree structure of the index to find the matching key values
Key Lookup	Finds the data for a row in a table by looking the row up by its key (usually after reading an index)

Index Column Order Matters



Index 1

State | LastName | FirstName

Index 2

LastName | FirstName | State

Cannot use this index because the first column of the index (State) is not in the WHERE clause

This index can be used because the first two columns in the index are also in the WHERE clause

```
SELECT *
    FROM Students
    WHERE LastName = ‘Anderson’
        AND FirstName = ‘Stacy’
```

Index Selectivity Matters

Selectivity: A measure of how unique the values in an index are

$$\textit{Selectivity} = \frac{\textit{Number of Rows in Table}}{\textit{Number of Unique Keys in Index}}$$

Low Selectivity Index – Large number of rows per index key

High Selectivity Index – Few of rows per index key. Unique indexes have the highest possible selectivity

Index Selectivity Example

Index

LastName	FirstName	State

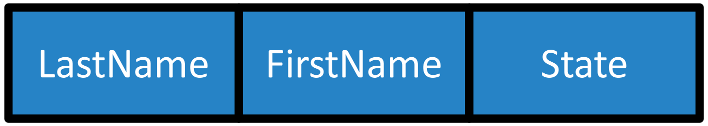
Statement

```
SELECT *
    FROM Students
    WHERE LastName = ‘Anderson’
        AND FirstName = ‘Stacy’
        AND State = ‘WI’
```

Selectivity is determined by the **unique combinations of all three columns** in the index since all three columns are used in the WHERE clause

Another Index Selectivity Example

Index



Statement

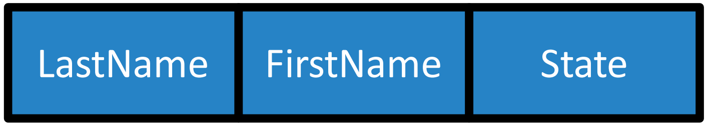
```
SELECT *
    FROM Students
    WHERE LastName = ‘Anderson’
        AND FirstName = ‘Stacy’
```

Selectivity is determined by the **unique combinations of the first two columns** since only these columns are in the WHERE clause.

For columns to count for selectivity, the must be consecutive from the front of the index

Like Clauses and Selectivity

Index



Statement One

```
SELECT *
    FROM Students
    WHERE LastName = ‘Ander%’
        AND FirstName = ‘Sta%’
```

Selectivity is determined by the number of characters before the wildcard character (%) in the WHERE clause. Most likely, the index can be used

Statement Two

```
SELECT *
    FROM Students
    WHERE LastName = ‘%Ander%’
        AND FirstName = ‘%Sta%’
```

Index cannot be used because we have a leading wildcard character in our WHERE clause values

Creating Effective Indexes

Arrange columns in an index so the columns most frequently used in the WHERE clause are at the front of the index

Make sure columns in an index have enough selectivity so only a small number of rows exist for each index key

What Should I Index?

Primary Keys

Index will be created by default

Foreign Keys

Helps JOIN performance. Queries are often traverse foreign keys

Search Columns

Match use cases of how your application searches for data

Dynamic Management Views

Real time diagnostic information about what is happening inside SQL Server

- Nothing extra to install – views exist automatically and collect data by default
- Views cover all aspects of what is happening inside of SQL Server, not just performance

Requires VIEW SERVER STATE permission to access these views

- Some shops comfortable giving this to developers, some are not

Useful queries for performance tuning

- Queries we will discuss are at http://bit.ly/PluralsightCourseDmvQueries

Current Sessions Connected to SQL Server

```
SELECT
    database_id,    -- SQL Server 2012 and after only
    session_id,
    status,
    login_time,
    cpu_time,
    memory_usage,
    reads,
    writes,
    logical_reads,
    host_name,
    program_name,
    host_process_id,
    client_interface_name,
    login_name as database_login_name,
    last_request_start_time
FROM sys.dm_exec_sessions
WHERE is_user_process = 1
ORDER BY cpu_time DESC;
```

Current Sessions Connected to SQL Server

```
SELECT
        [DatabaseName] = db_name(rq.database_id), s.session_id, rq.status,
        [SqlStatement] = SUBSTRING (qt.text,rq.statement_start_offset/2,
            (CASE WHEN rq.statement_end_offset = -1 THEN LEN(CONVERT(NVARCHAR(MAX),
            qt.text)) * 2 ELSE rq.statement_end_offset END - rq.statement_start_offset)/2),
        [ClientHost] = s.host_name, [ClientProgram] = s.program_name,
        [ClientProcessId] = s.host_process_id, [SqlLoginUser] = s.login_name,
        [DurationInSeconds] = datediff(s,rq.start_time,getdate()),
        rq.start_time, rq.cpu_time, rq.logical_reads, rq.writes,
        [ParentStatement] = qt.text,
        p.query_plan, rq.wait_type,
        [BlockingSessionId] = bs.session_id,
        [BlockingHostname] = bs.host_name,
        [BlockingProgram] = bs.program_name,
        [BlockingClientProcessId] = bs.host_process_id,
        [BlockingSql] = SUBSTRING (bt.text, brq.statement_start_offset/2,
            (CASE WHEN brq.statement_end_offset = -1 THEN LEN(CONVERT(NVARCHAR(MAX),
            bt.text)) * 2 ELSE brq.statement_end_offset END - brq.statement_start_offset)/2)
    FROM sys.dm_exec_sessions s
    INNER JOIN sys.dm_exec_requests rq
        ON s.session_id = rq.session_id
    CROSS APPLY sys.dm_exec_sql_text(rq.sql_handle) as qt
    OUTER APPLY sys.dm_exec_query_plan(rq.plan_handle) p
    LEFT OUTER JOIN sys.dm_exec_sessions bs
        ON rq.blocking_session_id = bs.session_id
    LEFT OUTER JOIN sys.dm_exec_requests brq
        ON rq.blocking_session_id = brq.session_id
    OUTER APPLY sys.dm_exec_sql_text(brq.sql_handle) as bt
    WHERE s.is_user_process =1
        AND s.session_id <> @@spid
        AND rq.database_id = DB_ID()  -- Comment out to look at all databases
    ORDER BY rq.start_time ASC;
```

Most Expensive SQL Statements

```
SELECT TOP 20
        DatabaseName = DB_NAME(CONVERT(int, epa.value)),
        [Execution count] = qs.execution_count,
        [CpuPerExecution] = total_worker_time / qs.execution_count ,
        [TotalCPU] = total_worker_time,
        [IOPerExecution] = (total_logical_reads + total_logical_writes) / qs.execution_count ,
        [TotalIO] = (total_logical_reads + total_logical_writes) ,
        [AverageElapsedTime] = total_elapsed_time / qs.execution_count,
        [AverageTimeBlocked] = (total_elapsed_time - total_worker_time) / qs.execution_count,
        [AverageRowsReturned] = total_rows / qs.execution_count,
        [Query Text] = SUBSTRING(qt.text,qs.statement_start_offset/2 +1,
            (CASE WHEN qs.statement_end_offset = -1
                THEN LEN(CONVERT(nvarchar(max), qt.text)) * 2
                ELSE qs.statement_end_offset end - qs.statement_start_offset)
            /2),
        [Parent Query] = qt.text,
        [Execution Plan] = p.query_plan,
        [Creation Time] = qs.creation_time,
        [Last Execution Time] = qs.last_execution_time
    FROM sys.dm_exec_query_stats qs
    CROSS APPLY sys.dm_exec_sql_text(qs.sql_handle) as qt
    OUTER APPLY sys.dm_exec_query_plan(qs.plan_handle) p
    OUTER APPLY sys.dm_exec_plan_attributes(plan_handle) AS epa
    WHERE epa.attribute = 'dbid'
        AND epa.value = db_id()
    ORDER BY [AverageElapsedTime] DESC; --Other column aliases can be used
```

Missing Indexes

```
SELECT
    TableName = d.statement,
    d.equality_columns,
    d.inequality_columns,
    d.included_columns,
    s.user_scans,
    s.user_seeks,
    s.avg_total_user_cost,
    s.avg_user_impact,
    AverageCostSavings = ROUND(s.avg_total_user_cost *
        (s.avg_user_impact/100.0), 3),
    TotalCostSavings = ROUND(s.avg_total_user_cost *
        (s.avg_user_impact/100.0) * (s.user_seeks + s.user_scans),3)
FROM sys.dm_db_missing_index_groups g
INNER JOIN sys.dm_db_missing_index_group_stats s
    ON s.group_handle = g.index_group_handle
INNER JOIN sys.dm_db_missing_index_details d
    ON d.index_handle = g.index_handle
WHERE d.database_id = db_id()
ORDER BY TableName, TotalCostSavings DESC;
```

Index Usage Statistics

```
SELECT
    [DatabaseName] = DB_Name(db_id()),
    [TableName] = OBJECT_NAME(i.object_id),
    [IndexName] = i.name,
    [IndexType] = i.type_desc,
    [TotalUsage] = IsNull(user_seeks, 0) + IsNull(user_scans, 0) +
         IsNull(user_lookups, 0),
    [UserSeeks] = IsNull(user_seeks, 0),
    [UserScans] = IsNull(user_scans, 0),
    [UserLookups] = IsNull(user_lookups, 0),
    [UserUpdates] = IsNull(user_updates, 0)
FROM sys.indexes i
INNER JOIN sys.objects o
    ON i.object_id = o.object_id
LEFT OUTER JOIN sys.dm_db_index_usage_stats s
    ON s.object_id = i.object_id
    AND s.index_id = i.index_id
WHERE
    (OBJECTPROPERTY(i.object_id, 'IsMsShipped') = 0)
ORDER BY [TableName], [IndexName];
```

SQL Tracing

Allows us to record activity in SQL Server

- What statements are run
- What order they are run in
- Performance statistics about the statement

SQL Tracing Tools

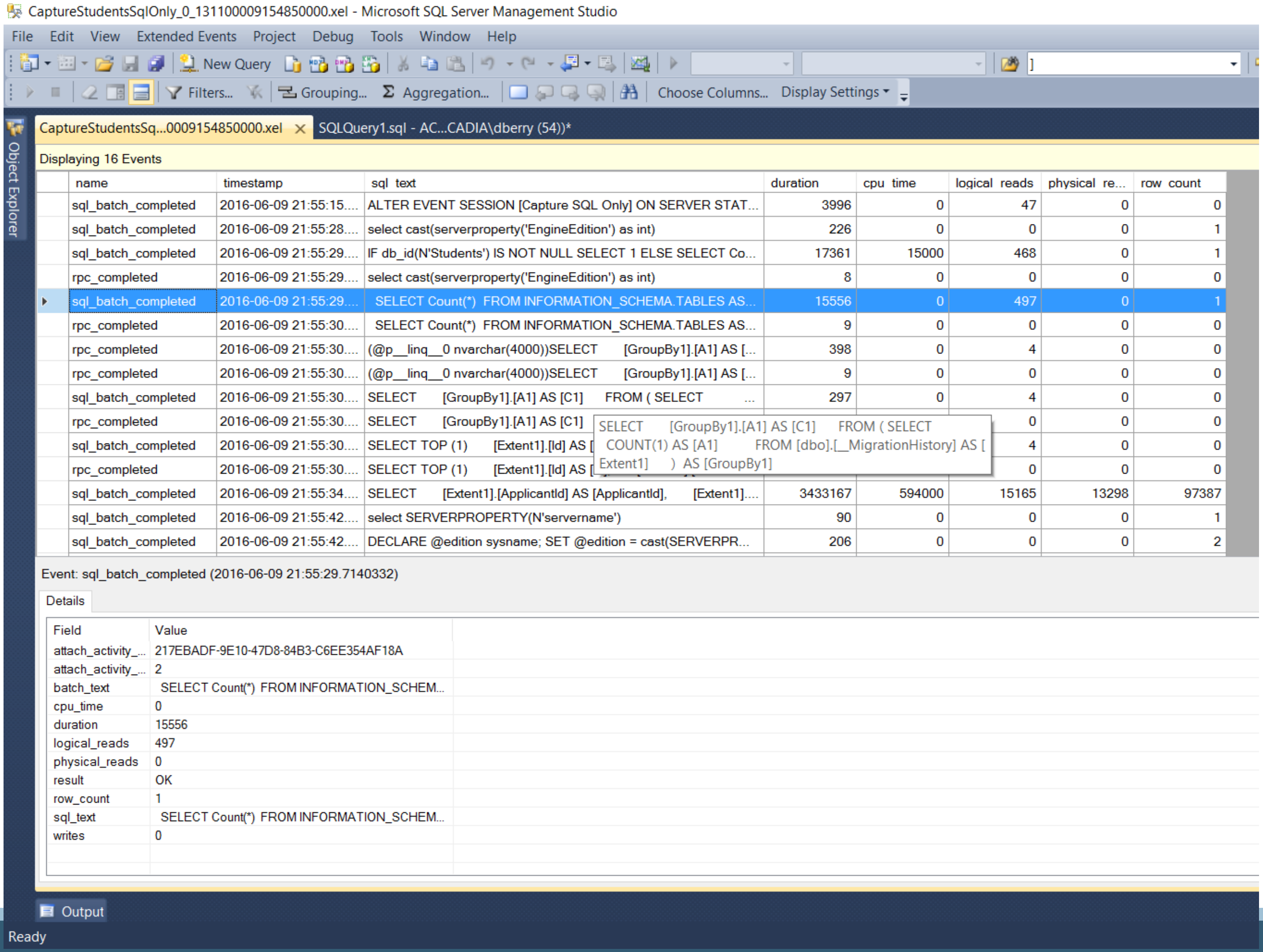
SQL Profiler

Supported in SQL 2005 – SQL 2014
Run as separate executable
Profile trace can also be run server side via stored procs

Extended Events

Supported in SQL 2008R2 and beyond (including SQL Azure)
Run from Management Studio
Better tools for filtering and sorting captured data

Tracing Output



CaptureStudentsSqlOnly_0_131100009154850000.xel - Microsoft SQL Server Management Studio
File Edit View Extended Events Project Debug Tools Window Help
New Query
Filters... Grouping... Aggregation... Choose Columns... Display Settings
CaptureStudentsSq...0009154850000.xel
SQLQuery1.sql - AC...CADIA\dberry (54))*
Object Explorer
Output
Ready

Displaying 16 Events

name	timestamp	sql text	duration	cpu time	logical reads	physical re...	row count
sql_batch_completed	2016-06-09 21:55:15....	ALTER EVENT SESSION [Capture SQL Only] ON SERVER STAT...	3996	0	47	0	0
sql_batch_completed	2016-06-09 21:55:28....	select cast(serverproperty('EngineEdition') as int)	226	0	0	0	1
sql_batch_completed	2016-06-09 21:55:29....	IF db_id(N'Students') IS NOT NULL SELECT 1 ELSE SELECT Co...	17361	15000	468	0	1
rpc_completed	2016-06-09 21:55:29....	select cast(serverproperty('EngineEdition') as int)	8	0	0	0	0
sql_batch_completed	2016-06-09 21:55:29....	SELECT Count(*) FROM INFORMATION_SCHEMA.TABLES AS...	15556	0	497	0	1
rpc_completed	2016-06-09 21:55:30....	SELECT Count(*) FROM INFORMATION_SCHEMA.TABLES AS...	9	0	0	0	0
rpc_completed	2016-06-09 21:55:30....	(@p__linq__0 nvarchar(4000))SELECT [GroupBy1].[A1] AS [...	398	0	4	0	0
rpc_completed	2016-06-09 21:55:30....	(@p__linq__0 nvarchar(4000))SELECT [GroupBy1].[A1] AS [...	9	0	0	0	0
sql_batch_completed	2016-06-09 21:55:30....	SELECT [GroupBy1].[A1] AS [C1] FROM (SELECT ...	297	0	4	0	0
rpc_completed	2016-06-09 21:55:30....	SELECT [GroupBy1].[A1] AS [C1]			0	0	0
sql_batch_completed	2016-06-09 21:55:30....	SELECT TOP (1) [Extent1].[Id] AS [			4	0	0
rpc_completed	2016-06-09 21:55:30....	SELECT TOP (1) [Extent1].[Id] AS [			0	0	0
sql_batch_completed	2016-06-09 21:55:34....	SELECT [Extent1].[ApplicantId] AS [ApplicantId], [Extent1]....	3433167	594000	15165	13298	97387
sql_batch_completed	2016-06-09 21:55:42....	select SERVERPROPERTY(N'servername')	90	0	0	0	1
sql_batch_completed	2016-06-09 21:55:42....	DECLARE @edition sysname; SET @edition = cast(SERVERPR...	206	0	0	0	2

```
SELECT      [GroupBy1].[A1] AS [C1]      FROM ( SELECT
 COUNT(1) AS [A1]        FROM [dbo].[__MigrationHistory] AS [
Extent1]     ) AS [GroupBy1]
```

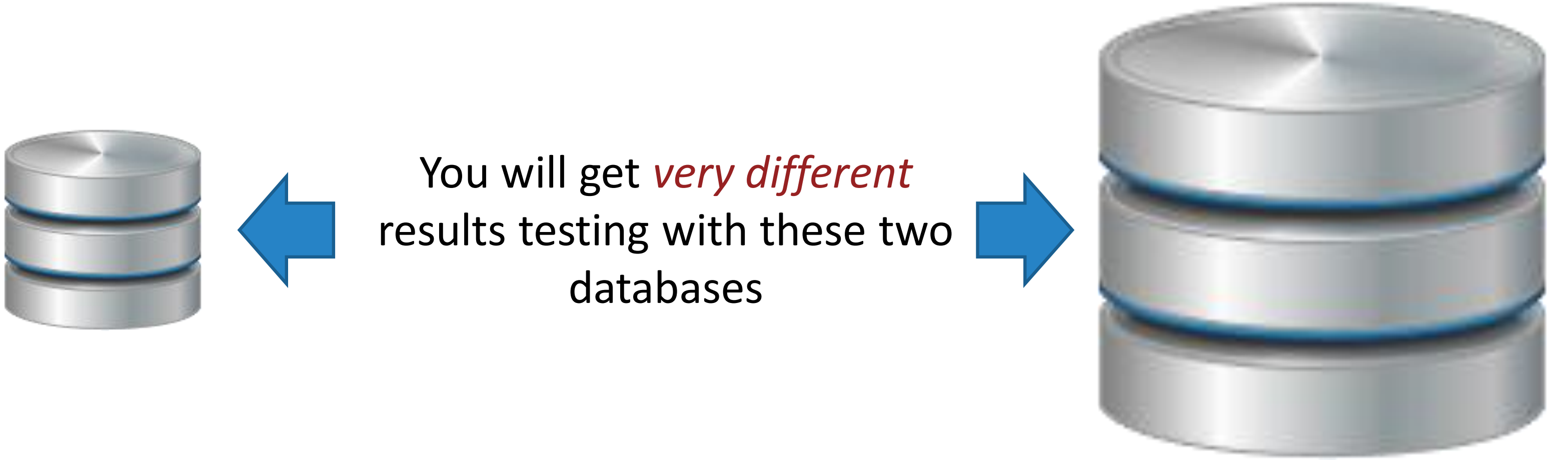
Event: sql_batch_completed (2016-06-09 21:55:29.7140332)

Details

Field	Value
attach_activity_...	217EBADF-9E10-47D8-84B3-C6EE354AF18A
attach_activity_...	2
batch_text	SELECT Count(*) FROM INFORMATION_SCHEM...
cpu_time	0
duration	15556
logical_reads	497
physical_reads	0
result	OK
row_count	1
sql_text	SELECT Count(*) FROM INFORMATION_SCHEM...
writes	0

Performance Practices

Test with a Production Size Database



You will get *very different* results testing with these two databases

Make sure you have relatively the same ***amount*** *and* ***distribution*** *of data in a test system as you do in production*

Keep SQL Statements Focused

Long, complex SQL statements are **hard to get right** and often have **poor performance**

Each statement should have a **clear purpose** and return **only the data needed** by the application process

Analyze Key Statements for Performance

Key Statements

- Statements against large tables
- Statements with multiple joins
- Frequently called statements
- Statements in critical processes

What to Analyze

- Execution plan
- Statement CPU and IO
- Trace all statements run for an application process

Monitor DMV Queries

Frequency

- After a major release
- Every 1-2 weeks for a stable application

What to look for?

- Expensive SQL statements
- How often statements are run
- Missing indexes
- Unused indexes

You have taken your first steps into a larger world

Photo Credits

Construction Worker Minifig - https://flic.kr/p/b33qCr

Minifig in Green Sweater - https://flic.kr/p/8ZVNgZ

Programmer Minifig - https://flic.kr/p/c7dagS

Sailor Minifig - https://flic.kr/p/c7di2U

Minifig with Magnifying Glass - https://flic.kr/p/dMHWQR

Luke Skywalker Minifig - https://flic.kr/p/8xfaLG

Lifecycle of a SQL Statement

Parse Phase

Check SQL Syntax
Check Object Permissions

Query Optimization

Evaluate Statistics
Create Execution Plan

Execution Phase

Read Blocks
Filter Rows
Sort Data