Microsoft

SQL Server Query Tuning

Module 9

Learning Units covered in this Module

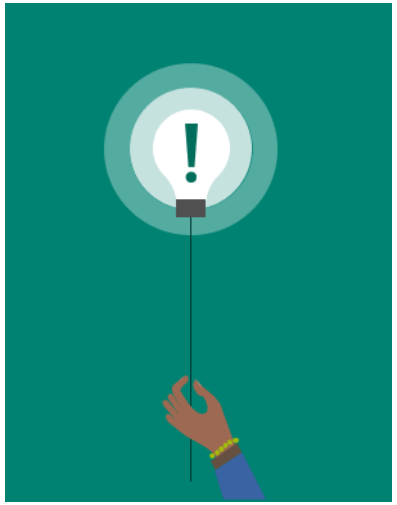
- Lesson 1: SARGable Expressions
- Lesson 2: Extended Events

Lesson 1: Sargable Expressions

Objectives

After completing this learning, you will be able to:

- Address SARGability Issues.
- Use computed columns for performance.
- Use constraints for performance.
- Understand parameter sniffing.

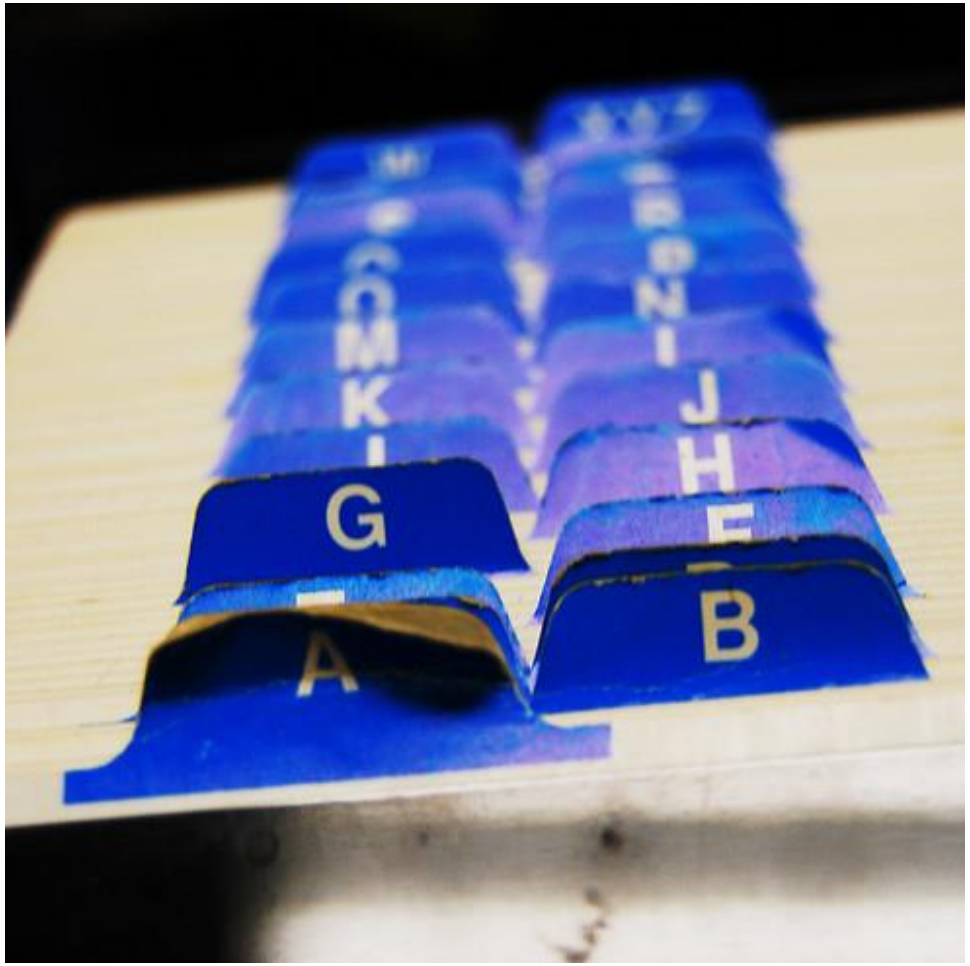


SARGability

What is this?

A SARGable item in a search predicate is able to use an index.

Non-SARGable expressions can significantly slow down queries.



Non-SARGable Expressions

Functions

WHERE ABS(ProductID) = 771

WHERE UPPER(City) = “London”

WHERE UPPER(surname) = “SMITHS”

Calculations / Expressions

WHERE Sales.Price + Sales.Tax > 100

WHERE Sales.Price * (1 + Sales.TaxRate) > 100

Using leading wildcard with LIKE operator

WHERE Employee.FirstName LIKE '%L%' is non-SARGable

WHERE Employee.FirstName LIKE ‘L%' is SARGable

Non-SARGable Expressions

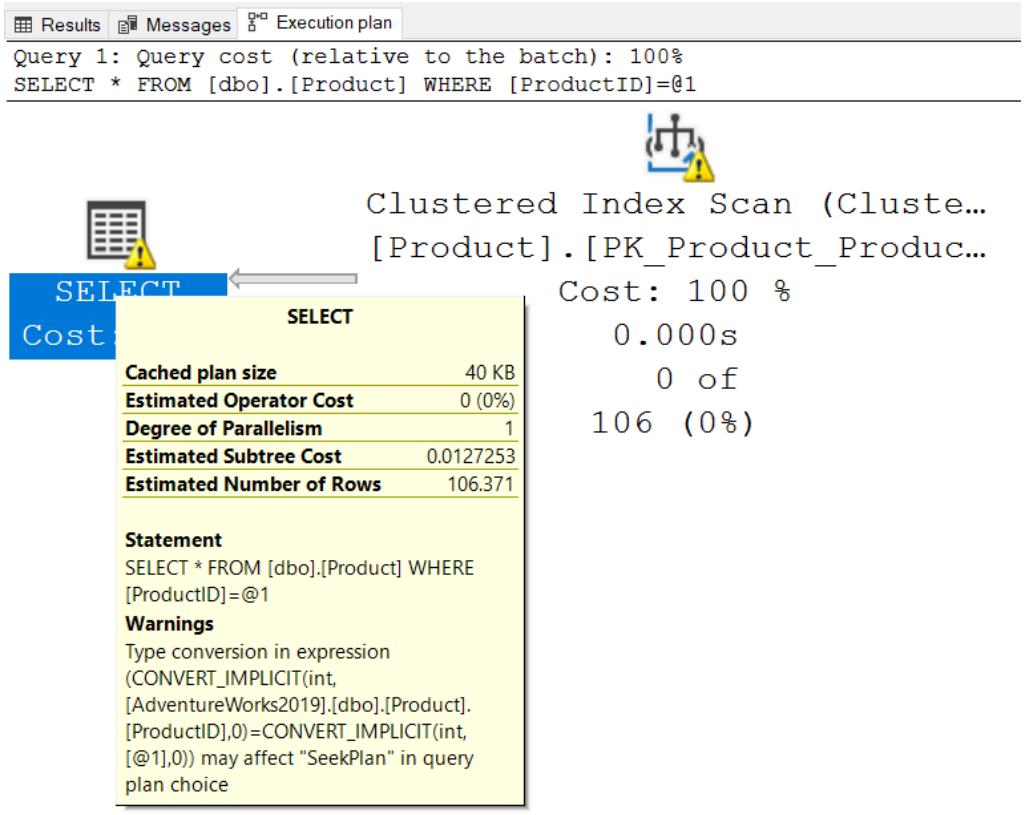
Continued

Implicit Conversions

ProductID is defined as **nvarchar(8)**

```
SELECT *
FROM [dbo].[Product]
WHERE [ProductID] = 7
```

```
SELECT *
FROM [dbo].[Product]
WHERE [ProductID] = N'7'
```



Using Computed Columns to Improve Performance

Resolving non-SARGable expressions

- Create computed column to Replace calculations / expressions.
- Create an index on computed column.

Selectivity issues for inequalities

- SQL Server assumes 30% selectivity on inequality comparisons.
- Computed column will have more accurate statistics.
- No need to specify computed column.

Using Computed Columns to Improve Performance

Persisted Computed Columns

Specifies that the Database Engine will physically store the computed values in the table.

Marking a computed column as PERSISTED allows an index to be created on a computed column that is deterministic.

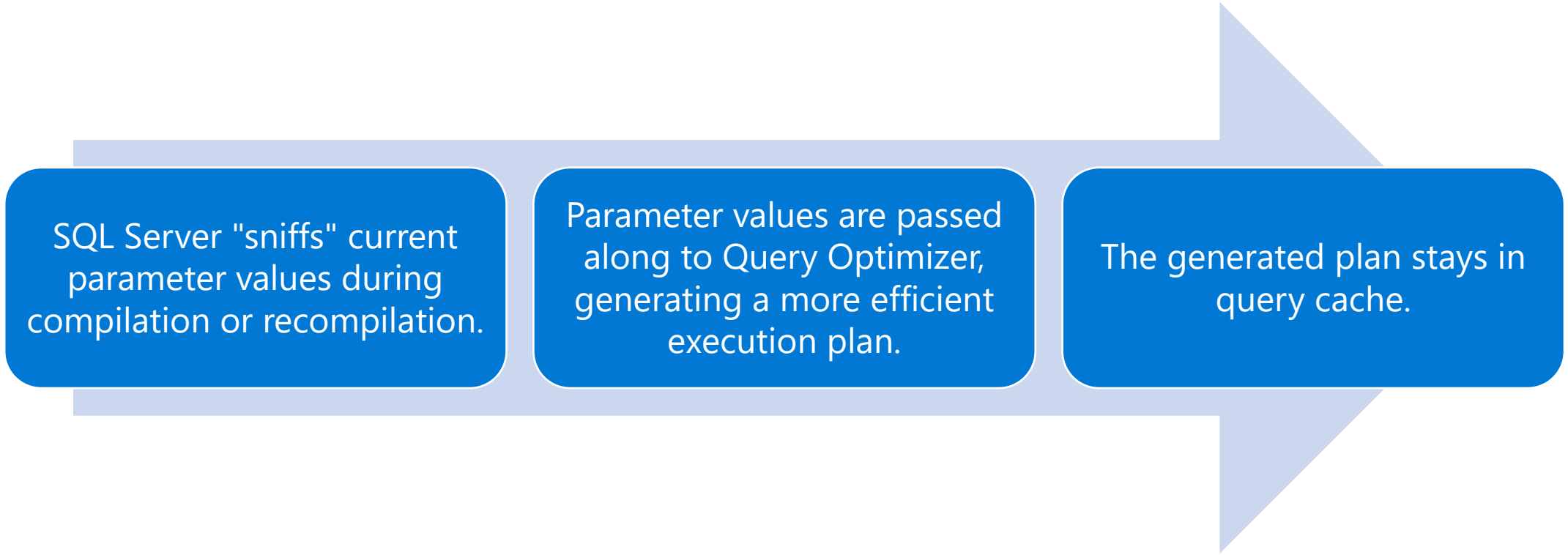
```
--Add a new computed column as persisted
ALTER TABLE [dbo].[ProductTest]
ADD stockValue AS (isnull(([UnitPrice] * ((1.0) - [UnitPriceDiscount])) *
[StockQty], (0.0))) PERSISTED;


--Create an index on new computed column to improve query performance
CREATE INDEX IX_StockValue ON [dbo].[ProductTest](stockValue);
```

Using Constraints to Improve Performance

Helping Query Optimizer to choose better plans

UNIQUE constraint	• DISTINCT property can be ignored • Extra columns in ORDER BY may be ignored
CHECK constraint	• CHECK constraints enforce domain integrity by limiting the values that are accepted by one or more columns.
Primary Key	• Special case of UNIQUE constraint
Foreign Key	• Declarative Referential Integrity (DRI) • Usually faster than triggers • Allows the query optimizer to identify unnecessary joins

Parameter Sniffing



Parameter sniffing can be seen on the following types of batches:

- Stored procedures
- Queries submitted via sp_executesql
- Prepared queries

Parameter Sniffing

Mitigation

RECOMPILE

- This workaround trades compilation time and increased CPU for better plan quality.

OPTION (OPTIMIZE FOR...)

- This option requires a good understanding of optimal parameter values and associated plan characteristics.

OPTION (OPTIMIZE FOR UNKNOWN)

- Overrides the actual parameter value and instead use the density vector.

DISABLE_PARAMETER_SNIFFING

- Disables parameter sniffing entirely. This hint name is equivalent to:
  - TRACE FLAG 4136
  - Database Scoped Configuration setting PARAMETER_SNIFFING = OFF

KEEPFIXEDPLAN

- This workaround assumes that the good-enough common plan is the one in cache already.

USE PLAN

- Force the plan by explicitly using this query hint by rewriting the query and adding the hint in the query text. Or set a specific plan by using Query Store.

Demonstration

Query Tuning Sargability

Query Tuning using

- Sargability
- Constraints
- Computed Columns

Query Tuning SARGability

- Applying SARGable expression to create better execution plans

Questions?

Knowledge Check

Give two examples of non-SARGable expressions.

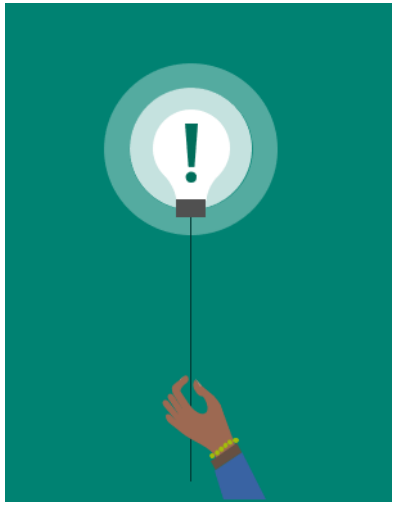
Explain three hints we can use to mitigate Parameter Sniffing.

Lesson 2 : Extended Events

Objectives

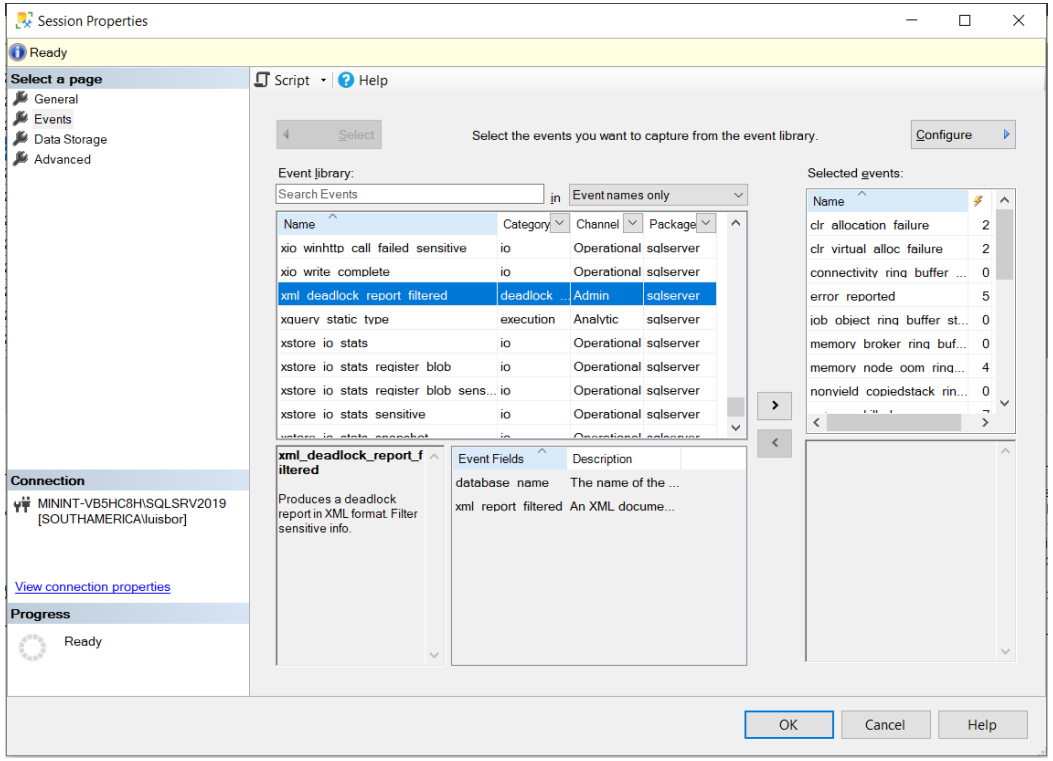
After completing this learning, you will be able to:

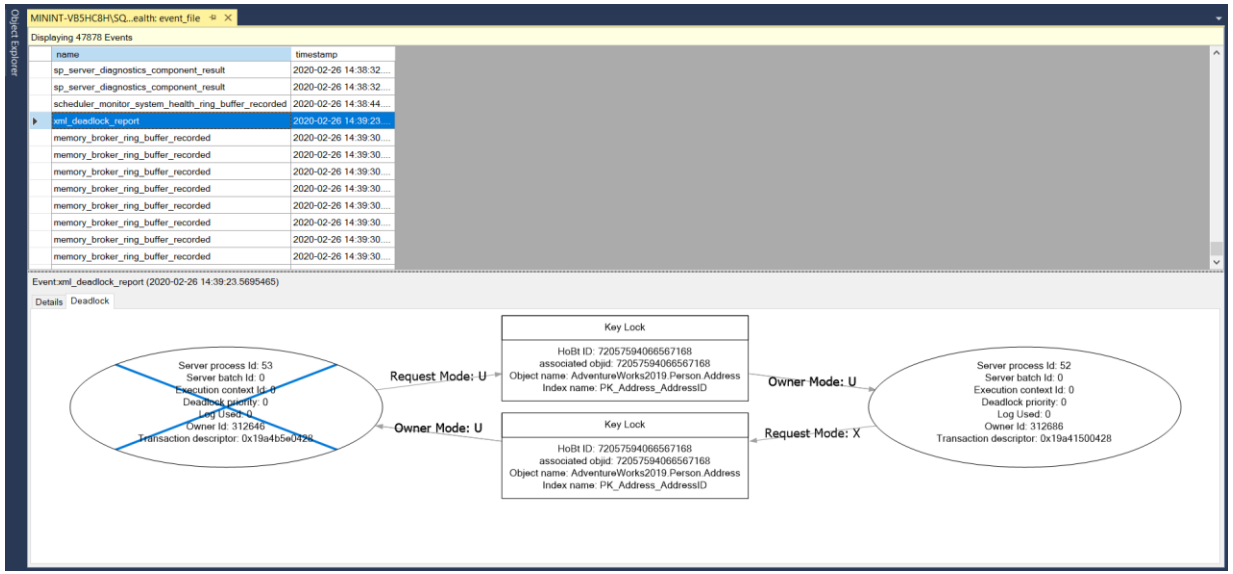
- Understand Extended Events objects.
- Create and configure Extended Event Sessions.
- Implement filtering, aggregation, and perform analysis via the UI in SSMS.
- Use Extended Events Profiler (XEProfiler).



Troubleshooting Tools

Extended Events – System Health Session





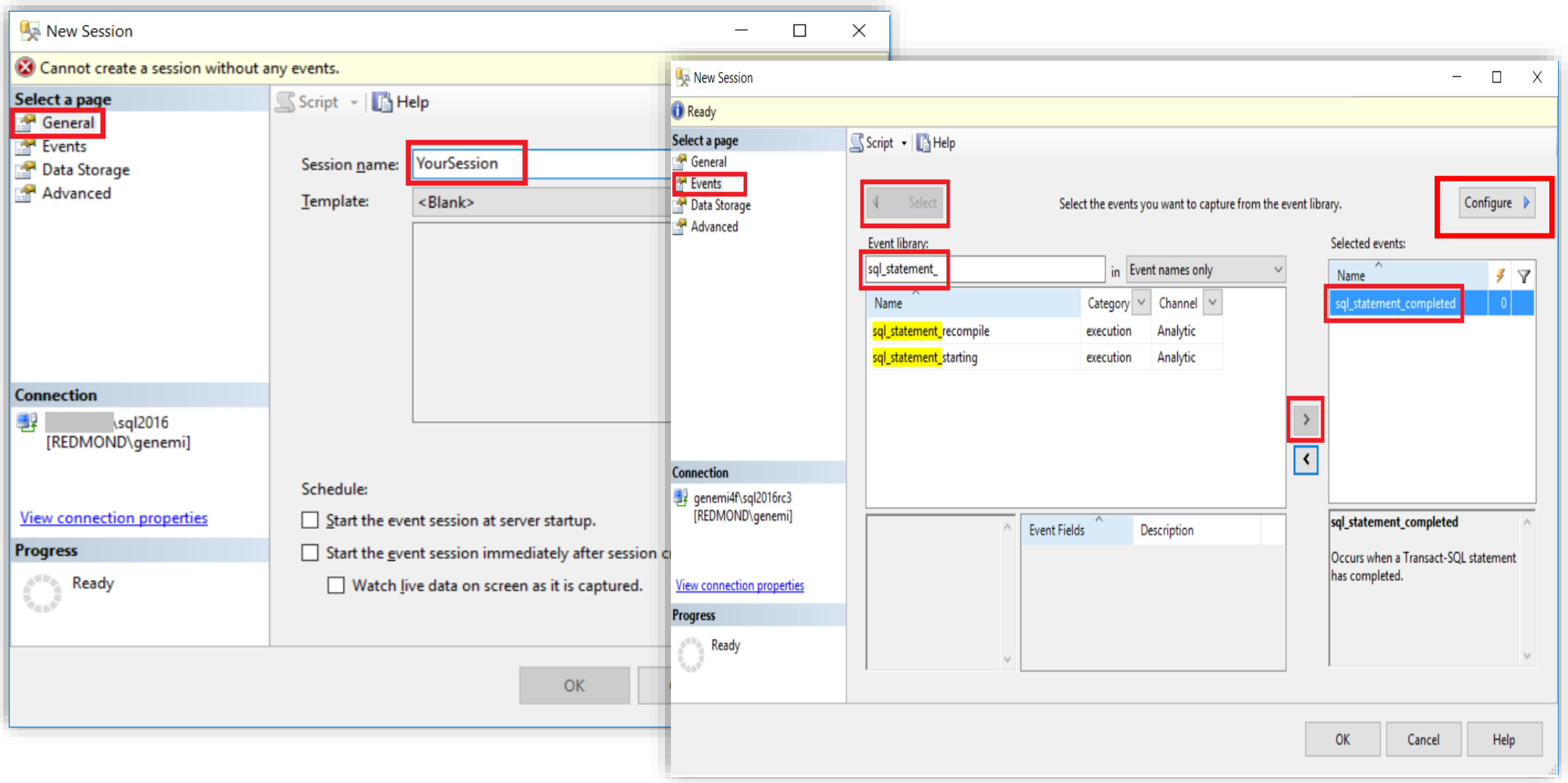
The system_health session is an Extended Events session that is included by default with SQL Server.

This session starts automatically when the SQL Server Database Engine starts and runs without any noticeable performance effects.

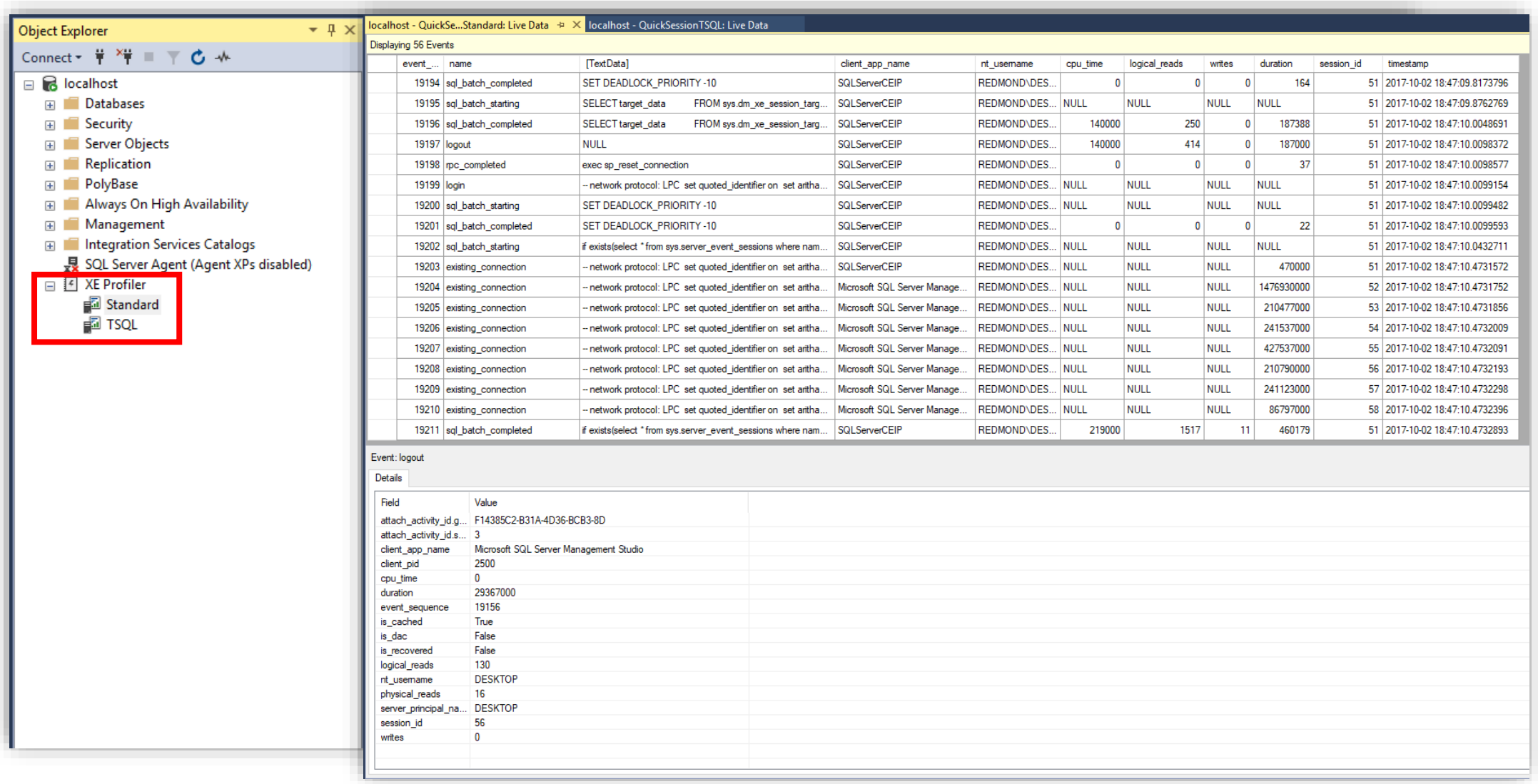
The session collects system data that you can use to help troubleshoot performance issues in the Database Engine.

Never change this session definition – always create your own according to your requirements.

Extended Events in SSMS



Extended Events Profiler



Demonstration

SQL Server Management Studio
Extended Events UI

XE Profiler

Extended Events

- Examining Extended Event Functionality

Questions?

Knowledge Check

What are the Extended Event objects?

What is XEProfiler?

When additional data elements capture is need, how can it be configured?

Which channel is disabled by default when creating a session?