Module 4: Query Optimization

Student Lab Manual

SQL Server 2012: Performance Tuning – Design, Internals, and Architecture

Version 1.0

Conditions and Terms of Use

Microsoft Confidential - For Internal Use Only

This training package is proprietary and confidential, and is intended only for uses described in the training materials. Content and software is provided to you under a Non-Disclosure Agreement and cannot be distributed. Copying or disclosing all or any portion of the content and/or software included in such packages is strictly prohibited.

The contents of this package are for informational and training purposes only and are provided "as is" without warranty of any kind, whether express or implied, including but not limited to the implied warranties of merchantability, fitness for a particular purpose, and non-infringement.

Training package content, including URLs and other Internet Web site references, is subject to change without notice. Because Microsoft must respond to changing market conditions, the content should not be interpreted to be a commitment on the part of Microsoft, and Microsoft cannot guarantee the accuracy of any information presented after the date of publication. Unless otherwise noted, the companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted herein are fictitious, and no association with any real company, organization, product, domain name, e-mail address, logo, person, place, or event is intended or should be inferred.

Copyright and Trademarks

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

For more information, see Use of Microsoft Copyrighted Content at
http://www.microsoft.com/about/legal/permissions/

Microsoft®, Internet Explorer®, and Windows® are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. Other Microsoft products mentioned herein may be either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. All other trademarks are property of their respective owners.

Query Optimization

Introduction

In this lab you will set up an extended events session to monitor autoupdate statistics. You will also find expensive queries in cache.

Objectives

After completing this lab, you will be able to:

- Determine the difference between synchronous and asynchronous statistics updates
- Set up an extended events session
- Use XML plan to navigate query plan structures.

Prerequisites

Familiarity with SQL Server Management Studio.

Estimated time to complete this lab

120 minutes

Scenario

You have been asked to tune expensive queries on a SQL Server instance. There are also queries exhibiting varying completion times and you have been asked to determine the cause.

Exercise 1: Asynchronous Update Statistics

Objectives

In this exercise, you will:

- Identify the difference between synchronous or asynchronous updates
- Use DMVs to examine stats queues
- Use Extended Events to view update statistics events

Prerequisites

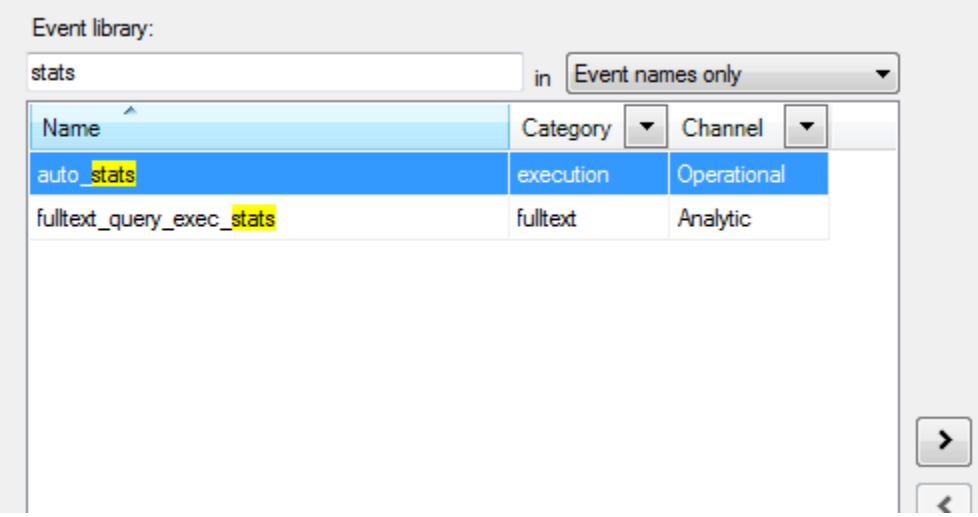
- Connect to the SQL2012PT Virtual Machine
- Log in using the following credentials
  User: Administrator
  Password: P@ssw0rd

> **Note:** The Virtual Machine for this workshop is time bombed for security purposes. You may need to rearm the virtual machine if the activation has expired. If the VM issues a message that it needs to be reactivated, you can use slmgr.vbs with the rearm option as follows:
>
> 1. Open an elevated command prompt (right click on "Command Prompt" in the Start menu and click "Run As Administrator)
> 2. Execute the following command
>   *slmgr.vbs –rearm*

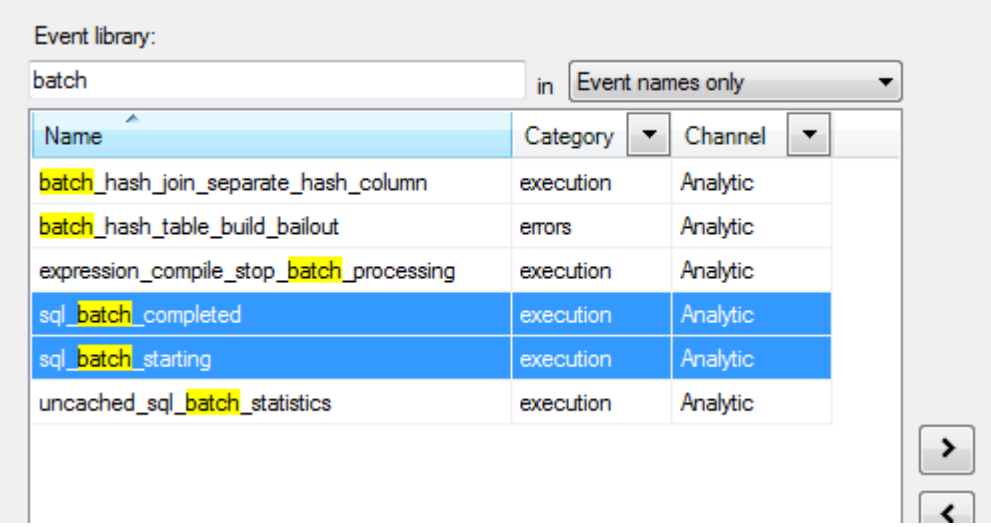
Task 1

1. Navigate to the folder C:\Labs\Module4\Exercise1 and double-click on **Restart SQL.cmd**. This will recycle SQL Server for this exercise.
2. Open the Extended Events session wizard.
   a. Expand *Management* by clicking on the Plus sign
   b. Expand *Extended Event* by clicking on the Plus sign
   c. Right Click on *Sessions* and choose *New Session Wizard*
3. Click 'Next'
4. Give the Session a name, such as AutoStats XEvents Session
5. Click 'Next'
6. Click 'Do Not Use A Template'
7. Click 'Next'

8. Type ‘stats’ in the Event Library box



9. Add auto_stats to the ‘Selected Events’ pane by clicking ‘>’
10. Type ‘batch’ into the Event Library box



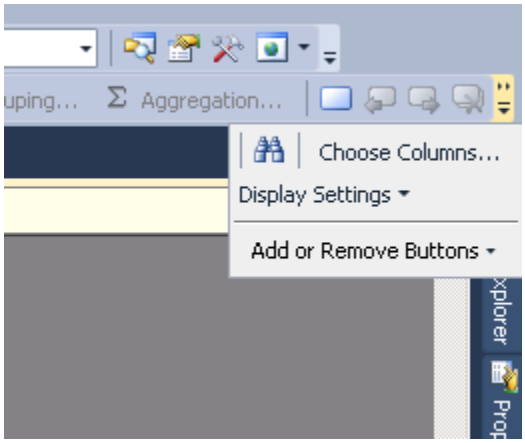
11. Add sql_batch_starting and sql_batch_completed to ‘Selected Events’ pane by clicking ‘>’’
12. Click ‘Next’
13. Click the checkbox next to ‘database_name’
14. Click ‘Next’
15. Leave the Event Session Filters page empty. Click ‘Next’
16. Click ‘Next’

17. Click ‘Finish’
18. Click the check boxes ‘Start the event session immediately after session creation’ and ‘Watch live data on screen as it is captured’
19. Click ‘Close’
20. Open the file **C:\Labs\Module4\Exercise1\scenario1.sql** in Management Studio and execute it.

```
USE master
GO

IF DB_ID('async_stats_test') IS NOT NULL
BEGIN
    DROP DATABASE async_stats_test
END
GO

CREATE DATABASE async_stats_test
GO
ALTER DATABASE async_stats_test SET AUTO_UPDATE_STATISTICS ON;
ALTER DATABASE async_stats_test SET AUTO_UPDATE_STATISTICS_ASYNC ON;
GO

USE async_stats_test
GO

IF OBJECT_ID('t1') IS NOT NULL
BEGIN
    DROP TABLE t1
END
GO

CREATE TABLE t1 (INSERT_date DATETIME)
GO

INSERT INTO t1 VALUES ('2005-06-09')
GO

CREATE INDEX stats_INSERT_date ON t1 (INSERT_date)
GO

INSERT INTO t1 VALUES ('2005-06-09')
INSERT INTO t1 VALUES ('2005-06-09')
INSERT INTO t1 VALUES ('2005-06-10')
INSERT INTO t1 VALUES ('2005-06-10')
INSERT INTO t1 VALUES ('2005-06-11')
INSERT INTO t1 VALUES ('2005-06-11')
INSERT INTO t1 VALUES ('2005-06-12')
INSERT INTO t1 VALUES ('2005-06-12')
INSERT INTO t1 VALUES ('2005-06-13')
INSERT INTO t1 VALUES ('2005-06-13')
INSERT INTO t1 VALUES ('2005-06-13')
INSERT INTO t1 VALUES ('2005-06-14')
INSERT INTO t1 VALUES ('2005-06-14')
GO
```

```
DECLARE @i INT
SET @i = 0

WHILE @i < 500
BEGIN
        INSERT INTO t1 VALUES ('2005-06-15')
        SELECT @i = @i + 1
END
GO

SET STATISTICS PROFILE ON
GO

SELECT * FROM t1 WHERE INSERT_date = '2005-06-13'
GO

WAITFOR DELAY '00:00:05'
GO

SELECT * FROM t1 WHERE INSERT_date = '2005-06-13'
GO

SET STATISTICS PROFILE OFF
GO
```

21. Go to the XEvents session you just configured and watch it.
22. Click the "Choose columns" button and add all the columns. If you don't see the button on the toolbar, you may need to click the dropdown at the end of the bar as below:



23. After the script finishes there should be three AutoStats events (you can add a filter using the "Filters…" button on the toolbar to make it easier to find the events).
    a. Why are there three trace events (hint: look at the status column)?
    b. The batch_text column is blank. Which column do you have to look at to determine what statistics/index are being maintained?

c. From the data in the trace events themselves can you confirm that the stats were updated asynchronously?

24. Close the XEvents Live Data window, then stop the XEvents session - do not delete it.

Task 2

1. Close the query window for scenario1.sql if you have not already done so.
2. Start the XEvents session from Task 1.
3. Open the second SQL file scenario2.sql and run it.

```
USE master
GO

IF DB_ID('async_stats_test') IS NOT NULL
BEGIN
    DROP DATABASE async_stats_test
END
GO

CREATE DATABASE async_stats_test
GO

ALTER DATABASE async_stats_test SET AUTO_UPDATE_STATISTICS ON;
ALTER DATABASE async_stats_test SET AUTO_UPDATE_STATISTICS_ASYNC ON;
GO

USE async_stats_test
GO

IF object_id('t1') IS NOT NULL
BEGIN
    DROP TABLE t1
END
GO

CREATE TABLE t1 (INSERT_date DATETIME)
GO

INSERT INTO t1 VALUES ('2005-06-09')
GO

CREATE STATISTICS stats_INSERT_date ON t1 (INSERT_date)
GO

INSERT INTO t1 VALUES ('2005-06-09')
INSERT INTO t1 VALUES ('2005-06-09')
INSERT INTO t1 VALUES ('2005-06-10')
INSERT INTO t1 VALUES ('2005-06-10')
INSERT INTO t1 VALUES ('2005-06-11')
INSERT INTO t1 VALUES ('2005-06-11')
INSERT INTO t1 VALUES ('2005-06-12')
INSERT INTO t1 VALUES ('2005-06-12')
INSERT INTO t1 VALUES ('2005-06-13')
```

```
INSERT INTO t1 VALUES ('2005-06-13')
INSERT INTO t1 VALUES ('2005-06-13')
INSERT INTO t1 VALUES ('2005-06-14')
INSERT INTO t1 VALUES ('2005-06-14')
GO

declare @i INT
SET @i = 0

while @i < 500
BEGIN
    INSERT INTO t1 VALUES ('2005-06-15')
    SELECT @i = @i + 1
END
GO

SET STATISTICS profile ON
GO

BEGIN TRAN
GO

ALTER TABLE t1 ADD c2 INT NULL
GO

-- ThIS time there was a DDL operation against the TABLE in the same
-- xact as WHERE we are trying to update stats
SELECT * FROM t1 WHERE INSERT_date = '2005-06-15'
GO

ROLLBACK
GO

SET STATISTICS profile OFF
GO
```

4. Stop the XEvents session when this completes (this may generate an error because you still have the Live Data window open).
5. Look at the XEvents session capture
   a. Were the statistics updated asynchronously (again, check the status column)?
   b. Why or why not (hint: examine the code you executed in step 3)?
6. Query the sys.dm_exec_background_job_queue_stats DMV
   a. How many stats jobs have run?
   b. What are their average and max response times?
7. Close the XEvents Live Data window and any other windows you have open in Management Studio. Delete the AutoStats XEvents Session.

Exercise 2: Identifying and tuning expensive queries in the cache

Objectives

In this exercise, you will be able:

- Understand how to use sys.dm_exec_query_stats to identify expensive queries
- Use the XML Showplan to identify query plan constructs

Scenario

You have been asked to find current expensive queries on a SQL Server instance and tune them.

Task 1

1. Open a query window in Management Studio and run the Exercise2_setup.sql.

```
USE AdventureWorksPTO
GO
if  EXISTS (SELECT * FROM sys.objects WHERE object_id =
OBJECT_ID(N'[dbo].[SalesOrderHeader_pto]') AND type IN (N'U'))
DROP TABLE [dbo].[SalesOrderHeader_pto]
GO
IF  EXISTS (SELECT * FROM sys.objects WHERE object_id =
OBJECT_ID(N'[dbo].[SalesOrderdetail_pto]') AND type IN (N'U'))
DROP TABLE [dbo].[SalesOrderdetail_pto]
GO

IF  EXISTS (SELECT * FROM sys.objects WHERE object_id =
OBJECT_ID(N'[dbo].[Customer_pto]') AND type IN (N'U'))
DROP TABLE [dbo].[customer_pto]
GO
SELECT * INTO SalesOrderHeader_pto
FROM Sales.SalesOrderHeader
GO
SELECT * INTO SalesOrderdetail_pto
FROM Sales.SalesOrderDetail
GO
SELECT * INTO Customer_pto
FROM Sales.Customer
GO

IF  EXISTS (SELECT * FROM sys.objects WHERE object_id =
OBJECT_ID(N'[dbo].[perf_proc1]') AND type IN (N'P', N'PC'))
DROP PROCEDURE [dbo].[perf_proc1]
GO

CREATE PROCEDURE perf_proc1
AS

SELECT * FROM Production.Product
  WHERE ProductID=1
```

```
SELECT a.CustomerID, --customertype,
   OrderDate, ShipDate, ProductID,
   OrderQty, UnitPrice, UnitPriceDiscount
FROM dbo.Customer_pto a
    JOIN dbo.SalesOrderHeader_pto b ON a.CustomerID=b.CustomerID
    JOIN dbo.SalesOrderdetail_pto c ON b.SalesOrderID= c.SalesOrderID
WHERE ShipDate between '07/01/2001' AND '07/31/2001'
```

2. Open a query window in Management Studio and run execute_perf_proc1.sql.

```
USE AdventureWorksPTO
GO

DBCC freeproccache
GO

EXEC perf_proc1
GO
```

3. Open another Query Window and execute Expensives_Queries_based_on_IO_1.sql to identify the expensive queries based on IO

```
SELECT TOP 20 last_execution_time,
    (total_physical_reads + total_logical_writes
    + total_logical_reads) AS [Total IO],
    text, qp.query_plan, statement_start_offset,
    statement_end_offset, sql_handle, plan_handle
FROM sys.dm_exec_query_stats AS qs
    CROSS APPLY sys.dm_exec_sql_text(qs.sql_handle) AS st
    CROSS APPLY sys.dm_exec_query_plan(qs.plan_handle) as qp
ORDER BY [Total IO] DESC;
```

   a. How many rows are returned that list the procedure perf_proc1 in the text column? Write down the value for the [total io] column for each row which has perf_proc1 in the text column

   b. For the rows which list create procedure perf_proc1, are the sql_handle columns different for each row? How about the plan_handle columns? Note the value for the sql_handle, and the plan_handle

4. Open another Query Window and execute Expensives_Queries_based_on_IO_2.sql.

```
SELECT TOP 20 last_execution_time,
    (total_physical_reads + total_logical_writes
           + total_logical_reads) AS [Total IO],
    qp.query_plan, sql_handle, plan_handle,
    (SELECT SUBSTRING(text, statement_start_offset/2,
           (CASE WHEN statement_end_offset = -1
                  THEN LEN(CONVERT(nvarchar(max),text)) * 2
                  ELSE statement_end_offset
           END - statement_start_offset)/2)
    FROM sys.dm_exec_sql_text(sql_handle)) AS query_text
FROM sys.dm_exec_query_stats qs
```

```
    CROSS APPLY sys.dm_exec_query_plan(qs.plan_handle) as qp
ORDER BY [Total IO] DESC;
```

a. Find the rows with the same sql_handle that you noted in the previous step. Are you able to determine based on the query the select statements are part of a stored procedure?

5. Open another Query Window and execute Expensives_Queries_based_on_IO_3.sql.

```
SELECT TOP 20 last_execution_time,
    (total_physical_reads + total_logical_writes
           + total_logical_reads) AS [Total IO],
    sql_handle, plan_handle, qp.*,
    (SELECT SUBSTRING(text, statement_start_offset/2,
           (CASE WHEN statement_end_offset = -1
                  THEN LEN(CONVERT(nvarchar(max),text)) * 2
                  ELSE statement_end_offset
           END - statement_start_offset)/2)
    FROM sys.dm_exec_sql_text(sql_handle)) AS query_text
FROM sys.dm_exec_query_stats qs
    CROSS APPLY sys.dm_exec_query_plan(qs.plan_handle) as qp
ORDER BY [Total IO] DESC;
```

a. How many rows have the same values for the dbid column and objectid?

b. How can you get the name for the object?

Exercise 3: Using Extended Events to identify query performance issues

Objectives

In this exercise, you will be able:

- Configure Extended Events (XEvents) to find
  - expensive queries
  - inaccurate cardinalities

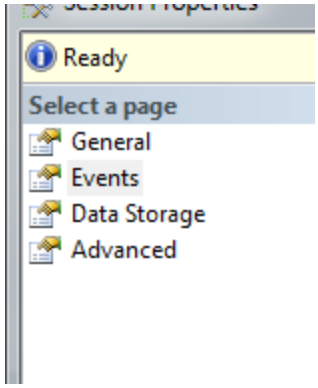
Scenario

Overall server performance troubleshooting: When you haven't identified any particular query, you can choose to capture all execution plans that consumed CPU_Time or duration exceeding certain threshold.

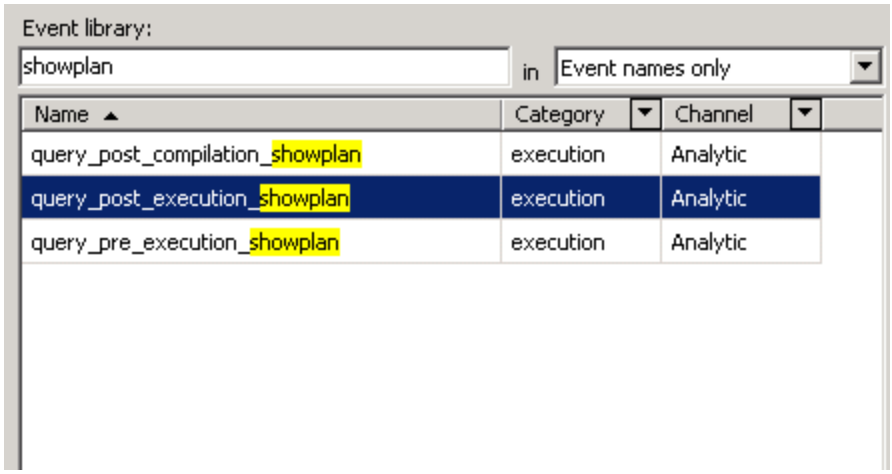
Prior to SQL Server 2012, tracing execution plan is all or nothing approach. In profiler, if you select execution plan, you will get everything every occurrence of every execution plan. This can bloat trace file very quickly. You need to find a way to monitor for expensive queries that uses fewer resources.

Task 1 - Expensive queries

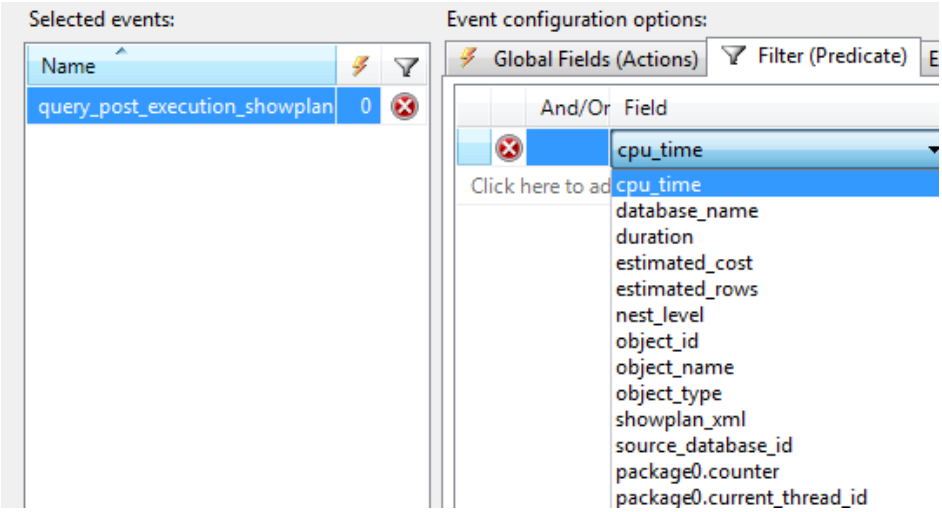
1. Open Extended Events
2. Right click Sessions. And choose 'New Session'
3. Enter a name for the session e.g. Expensive Queries
4. Check the boxes for 'Start the session immediately after session creation' and 'Watch live data on screen as it is captured'
5. Click on 'Events' in the 'Select a page' pane



6. In Event library, search using keyword "showplan".



7. Click on query_post_execution_showplan. Note that cpu_time and duration are among available event fields.
8. Move query_post_execution_showplan to the selected events pane.
9. Click on query_post_execution_showplan in the Selected events pane
10. Click on 'Configure'
11. Click on the "Filter (Predicate)" tab.
12. Choose cpu_time. Note: cpu_time is in *micro*seconds (1microsecond = milliseconds/1000)



13. Choose operator > and Value = 1000000

14. Under global fields (actions), choose sql_text
15. Click 'OK'
16. When you are finished, click OK
17. Open a query window in Management Studio and run scenario3.sql.

```
USE AdventureWorksPTO
GO

--create CPU stress.
SELECT top 200000 *
FROM Production.ProductListPriceHistory
    INNER JOIN Production.ProductCostHistory
            INNER JOIN Production.Product
                    ON Production.ProductCostHistory.ProductID
                            = Production.Product.ProductID
            INNER JOIN Production.ProductDocument AS ProductDocument_1
                    ON Production.Product.ProductID
                            = ProductDocument_1.ProductID
            INNER JOIN Production.ProductInventory
                    ON Production.Product.ProductID
                            = Production.ProductInventory.ProductID
            ON Production.ProductListPriceHistory.ProductID
                    = Production.Product.ProductID
    INNER JOIN Production.ProductModel
            ON Production.Product.ProductModelID
                    = Production.ProductModel.ProductModelID
    INNER JOIN Production.ProductSubcategory
            ON Production.Product.ProductSubcategoryID
                    = Production.ProductSubcategory.ProductSubcategoryID
    INNER JOIN Production.ProductCategory
            ON Production.ProductSubcategory.ProductCategoryID
                    = Production.ProductCategory.ProductCategoryID
                    AND Production.ProductSubcategory.ProductCategoryID
                            = Production.ProductCategory.ProductCategoryID
    CROSS JOIN Production.ProductDescription Test
    INNER JOIN Production.ProductDescription
            ON Test.ProductDescriptionID
                    = Production.ProductDescription.ProductDescriptionID
    CROSS JOIN Person.Address
```

18. Open a query window in Management Studio and run scenario4.sql.

```
USE AdventureWorksPTO
SELECT *
FROM Person.Address
    INNER JOIN Person.BusinessEntityAddress
            ON Person.Address.AddressID
                    = Person.BusinessEntityAddress.AddressID
    INNER JOIN Person.BusinessEntity
            ON Person.BusinessEntityAddress.BusinessEntityID
                    = Person.BusinessEntity.BusinessEntityID
    INNER JOIN Person.BusinessEntityContact
            ON Person.BusinessEntity.BusinessEntityID
                    = Person.BusinessEntityContact.BusinessEntityID
GO
```

19. What do you see in the XEvents session window?
    a. Are you able to identify which query used more than a second of CPU? Look in the details pane at the bottom of the screen.
    b. Click the Query Plan tab to view the graphical execution plan of the query.
    c. Do you think it would be possible to use XEvents sessions to find long running queries? How would you do it?
    d. If time permits, try changing the session configuration to find long running queries. Rerun scenario3 and scenario4 to see if you can capture just the query that runs longer.
20. Close any open windows in Management Studio, including the XEvents Live Data. Stop the Expensive Queries XEvent session.

Task 2 – Inaccurate Cardinalities

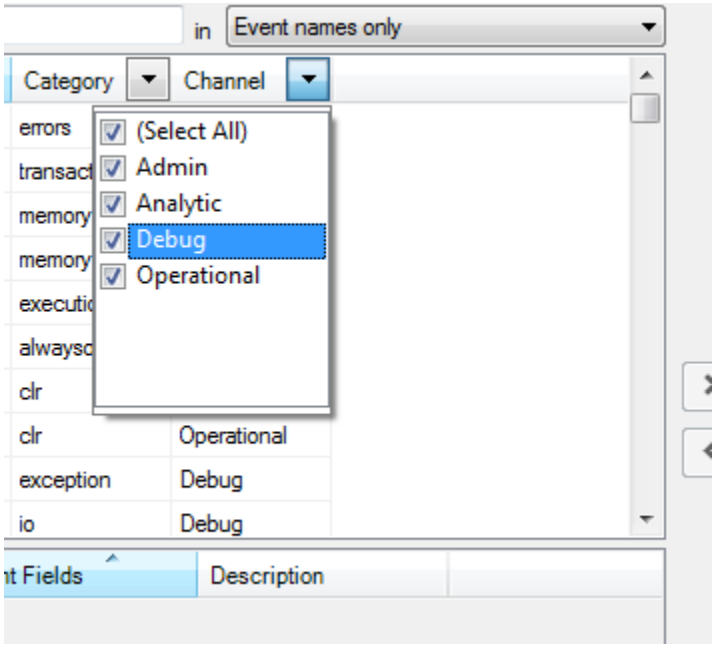
Up until SQL Server 2012, detecting cardinality estimate issues is a manual process. XEvents can raise an event when an operator outputs significantly more rows than estimated by the query optimizer.

1. Run the script prepare_scenario5.sql

```
USE AdventureworksPTO
GO

IF object_id ('dbo.cardtable') IS NOT NULL
DROP TABLE dbo.cardtable
GO

CREATE TABLE dbo.cardtable (c1 INT)
GO
SET NOCOUNT ON
BEGIN TRAN
DECLARE @i INT = 0
WHILE @i < 10000
BEGIN
    IF (@i < 10)
    BEGIN
            INSERT INTO cardtable VALUES (-1)
    END
    IF (@i < 8)
    BEGIN
            INSERT INTO cardtable VALUES (-2)
    END
    INSERT INTO cardtable VALUES (@i)
    INSERT INTO cardtable VALUES (@i)
    SET @i = @i + 1
END
COMMIT TRAN

CREATE INDEX ix_C1 ON dbo.cardtable (c1)
```

```
GO
IF object_id ('dbo.p_test') IS NOT NULL
DROP PROCEDURE dbo.p_test
GO
CREATE PROCEDURE dbo.p_test @i INT
AS
SELECT * FROM dbo.cardtable WHERE c1 = @i
GO
```

2. Create a new Xevents session
3. Name the session “Inaccurate Cardinality”
4. Check the boxes for ‘Start the session immediately after session creation’ and ‘Watch live data on screen as it is captured’
5. Click on ‘Events’ in the ‘Select a page’ pane
6. Enable the debug channel



7. Search for ‘cardinality’
8. Click ‘inaccurate_cardinality_estimate’ and move it into the Selected Events pane
9. Click on ‘Configure’
10. Under global fields (actions), choose sql_text
11. Click ‘OK’.
12. Run the script scenario5.sql

```
USE AdventureworksPTO
GO
EXEC dbo.p_test 1
GO
EXEC dbo.p_test -1
```

```
GO
EXEC dbo.p_test -2
```

13. Look at the captured events in the XEvents session.
    a. How many warning event did you get?
    b. Which one of the executions should get the warning event? Why?
14. Run the above three queries with execution plan. Compare estimated rows and actual rows.
15. Close all open windows in Management Studio, including the XEvents Live Data window. Stop the Inaccurate Cardinality XEvents session.