SQL Server Extended Events Basics

SQLMaestros Channels

Web	:	www.SQLMaestros.com
YouTube	:	https://www.youtube.com/user/SQLMaestros/videos
RSS	:	http://feeds.feedburner.com/SQLMaestros_AmitBansal
Twitter	:	www.twitter.com/SQLMaestros
LinkedIn	:	https://www.linkedin.com/showcase/14630987
Telegram	:	https://t.me/sqlmaestros
FB	:	www.facebook.com/SQLMaestros

HOLS Feedback	:	holfeedback@SQLMaestros.com
HOLS Support	:	holsupport@SQLMaestros.com

Terms of Use, Copyright & Intellectual Property

Table of Contents

Before You Begin

Estimated time to complete this lab
60 minutes

Objectives:
After completing this lab, we will learn:

- The basic overview of extended events
- Different components of extended events
- How to setup extended event
- Different targets available in extended events
- Some advanced features available on extended event GUI in SQL Server 2012

Lab Setup Requirements
Before executing this lab:

- You must have SQL Server 2012 Standard/Developer/Enterprise/Evaluation edition or higher. Click here to download SQL Server evaluation edition
- You must have AdventureWorks sample databases. It is recommended that you have AdventureWorks2012 or higher. Click here to download AdventureWorks sample databases

Prerequisites
Before executing this lab:

- It is recommended that you have basic experience with SQL Server
- You have met the Lab Setup Requirements mentioned above

Lab Scenario
SQL Server Extended Events (Extended Events) is a general event-handling system for server systems. The Extended Events infrastructure supports the correlation of data from SQL Server, and under certain conditions, the correlation of data from the operating system and database applications. In the first exercise, we will explore different components of extended events. In the second exercise we will create an event session and will also look at system catalog

views and DMV to view configuration of an existing event session. In the third exercise, we will look at different targets available in the extended event and in the fourth and final exercise of this we will look at some exciting features available in the SQL Server Management Studio extended event GUI.

Tips to complete this lab successfully

Following these tips will be helpful in completing the lab successfully in time

- All lab files are located in **SQL Server Extended Events Basics** folder
- The script(s) are divided into various sections marked with ‘Begin’, ‘End’ and ‘Steps’. As per the instructions, execute the statements between particular sections only or for a particular step
- Read the instructions carefully and do not deviate from the flow of the lab
- Practice this lab only in your test machine/environment. Do not run this lab in your production environment

Exercise 1: Extended Event Objects

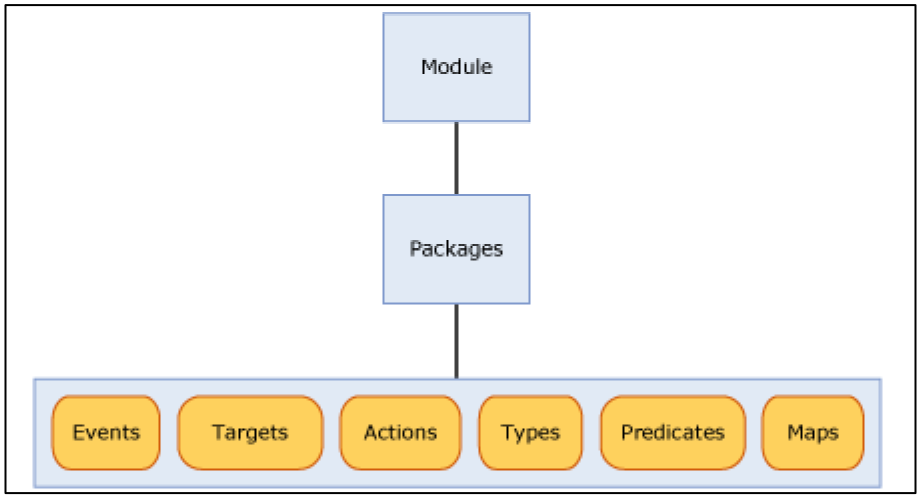
Overview:

All the components of extended event objects are contained in the package, which is a container for SQL Server Extended Events objects. There are three kinds of Extended Events packages, which include the following:

- package0 - Extended Events system objects. This is the default package
- Sqlserver - SQL Server related objects
- Sqlos - SQL Server Operating System (SQLOS) related objects

A package can contain any or all of the following objects, which are discussed in greater detail later in this exercise:

- Events
- Targets
- Actions
- Types
- Predicates
- Maps

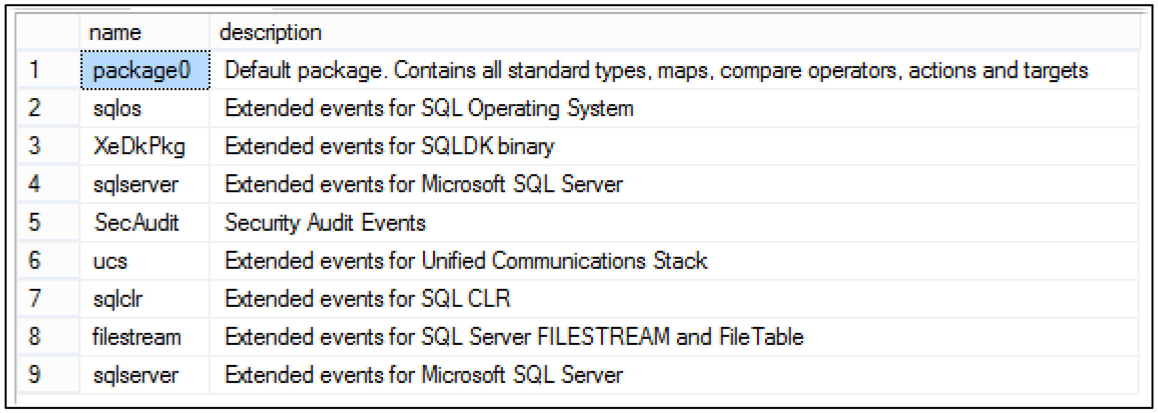


Scenario

In this exercise, we will look at different components of extended events.

Tasks	Detailed Steps
Launch **SQL Server Management Studio**	1. Click **Start** | **All Programs** | **SQL Server 2012** | **SQL Server Management Studio** 2. In the **Connect to Server** dialog box, click **Connect**
Open **1_EventObjects.sql**	1. Click **File** | **Open** | **File** or press (**Ctrl + O**) 2. In **Open File** dialogue box, navigate to **SQL Server Extended Events Basics\Scripts** folder 3. Select **1_EventObjects.sql** and click **Open**

View packages	Execute the following statement(s) to view available packages in extended event
	
	`SET NOCOUNT ON;`
	`-------------`
	`-- Packages`
	`-------------`
	`-- Step 1: View packages of extended events`
	`SELECT name, description FROM sys.dm_xe_packages;`
	`GO`
	
	Explanation: There are two packages named **sqlserver** in the below screenshot. These two packages are not same and they are loaded from two different module.



	name	description
1	package0	Default package. Contains all standard types, maps, compare operators, actions and targets
2	sqlos	Extended events for SQL Operating System
3	XeDkPkg	Extended events for SQLDK binary
4	sqlserver	Extended events for Microsoft SQL Server
5	SecAudit	Security Audit Events
6	ucs	Extended events for Unified Communications Stack
7	sqlclr	Extended events for SQL CLR
8	filestream	Extended events for SQL Server FILESTREAM and FileTable
9	sqlserver	Extended events for Microsoft SQL Server

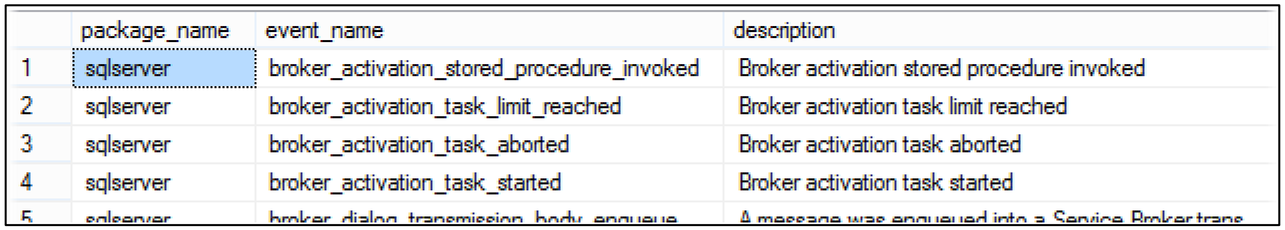
View events

Execute the following statement(s) to view events available in SQL Server 2012

Explanation: Events are monitoring points of interest in the execution path of a program, such as SQL Server. An event firing carries with it the fact that the point of interest was reached, and state information from the time the event was fired.

```
-------------
-- Events
-------------
-- Step 2: View all the extended events available
SELECT XP.name AS package_name,
              XO.name AS event_name,
              XO.description
      FROM sys.dm_xe_packages AS XP
      JOIN sys.dm_xe_objects AS XO
           ON XP.guid = XO.package_guid
      WHERE  XO.object_type = 'event';
GO
```

Note: There are total 627 events available in SQL Server 2012 but due to shortage of space the below screenshot contains only the first four events.

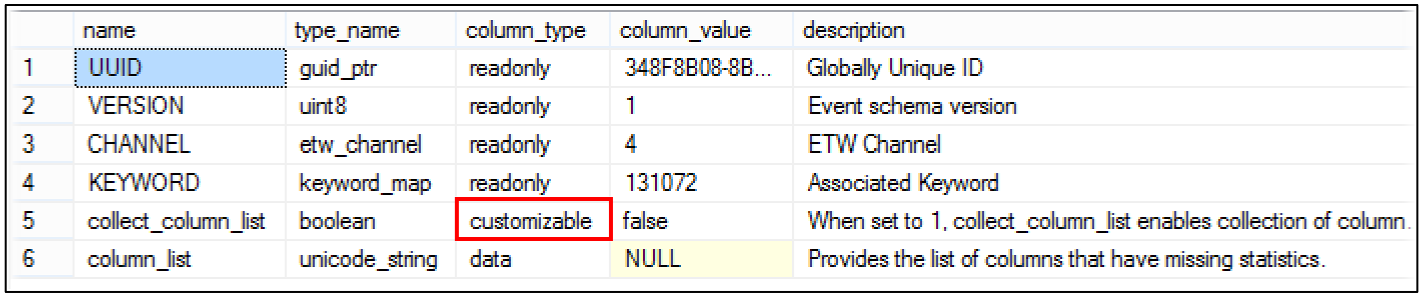


	package_name	event_name	description
1	sqlserver	broker_activation_stored_procedure_invoked	Broker activation stored procedure invoked
2	sqlserver	broker_activation_task_limit_reached	Broker activation task limit reached
3	sqlserver	broker_activation_task_aborted	Broker activation task aborted
4	sqlserver	broker_activation_task_started	Broker activation task started
5	sqlserver	broker_dialog_transmission_body_enqueue	A message was enqueued into a Service Broker trans

View data types returned by an event

Execute the following statement(s) to view data types returned by a particular event.

```sql
-----------------------
-- Event Data Elements
-----------------------
-- Step 3: View all the data type returned by a particular extended event
SELECT XOC.name, XOC.type_name, XOC.column_type, XOC.column_value, XOC.description FROM sys.dm_xe_objects
AS XO
INNER JOIN sys.dm_xe_object_columns AS XOC
ON XO.name = XOC.object_name WHERE XO.name = 'missing_column_statistics'
AND XO.object_type = 'event';
GO
```



	name	type_name	column_type	column_value	description
1	UUID	guid_ptr	readonly	348F8B08-8B...	Globally Unique ID
2	VERSION	uint8	readonly	1	Event schema version
3	CHANNEL	etw_channel	readonly	4	ETW Channel
4	KEYWORD	keyword_map	readonly	131072	Associated Keyword
5	collect_column_list	boolean	customizable	false	When set to 1, collect_column_list enables collection of column.
6	column_list	unicode_string	data	NULL	Provides the list of columns that have missing statistics.

Observation: In the above statement we are viewing data types returned by event **missing_column_statistics.**

Note: in the **column_type** column there are three distinct types. **Customizable** column data can be set by the user.

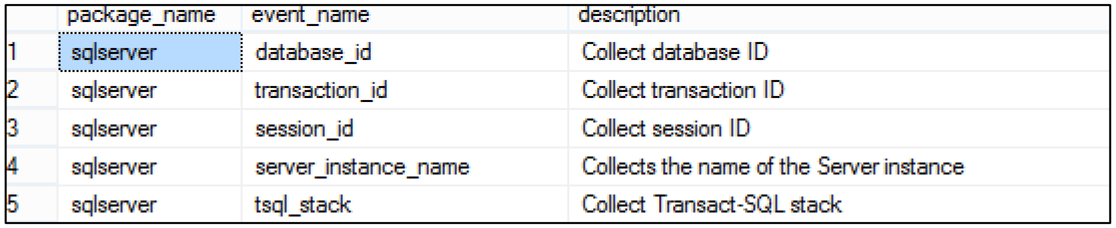
View available actions	Execute the following statement(s) to view available actions

Execute the following statement(s) to view available actions

Explanation: An action is a programmatic response or series of responses to an event. Actions are bound to an event, and each event may have a unique set of actions. An action bound to an event is invoked synchronously on the thread that fired the event. There are many types of actions and they have a wide range of capabilities.

```
-------------
-- Actions
-------------
-- Step 4: View all the actions available
SELECT XP.name AS package_name,
              XO.name AS event_name,
              XO.description
       FROM sys.dm_xe_packages AS XP
       JOIN sys.dm_xe_objects AS XO
            ON XP.guid = XO.package_guid
       WHERE  XO.object_type = 'action';
GO
```

Note: There are around 50 actions available in SQL Server 2012 but due to a shortage of space the below screenshot contains only the first five actions.



	package_name	event_name	description
1	sqlserver	database_id	Collect database ID
2	sqlserver	transaction_id	Collect transaction ID
3	sqlserver	session_id	Collect session ID
4	sqlserver	server_instance_name	Collects the name of the Server instance
5	sqlserver	tsql_stack	Collect Transact-SQL stack

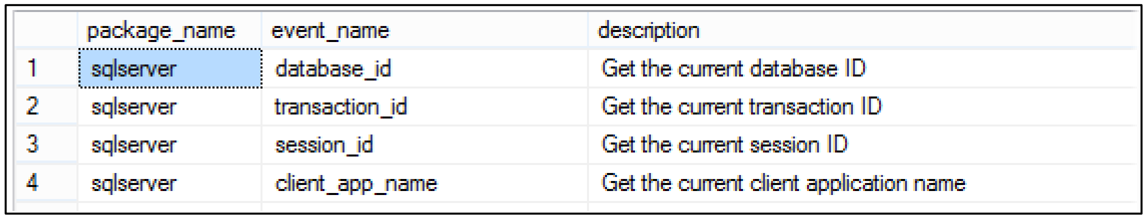
View predicates	Execute the following statement(s) to view predicates

Explanation: Predicates are a set of logical rules that are used to evaluate events when they are processed. This enables the Extended Events user to selectively capture event data based on specific criteria. There are two different types of predicate objects; source objects which provide the global state data elements for filtering on and comparators which provide the textual comparisons that can be performed between a data element and the specified value.

```
-------------
-- Predicates
-------------

-- Predicate Sources
-- Step 5: View all the predicate source available
SELECT XP.name AS package_name,
              XO.name AS event_name,
              XO.description
       FROM sys.dm_xe_packages AS XP
       JOIN sys.dm_xe_objects AS XO
            ON XP.guid = XO.package_guid
       WHERE  XO.object_type = 'pred_source';
GO
```

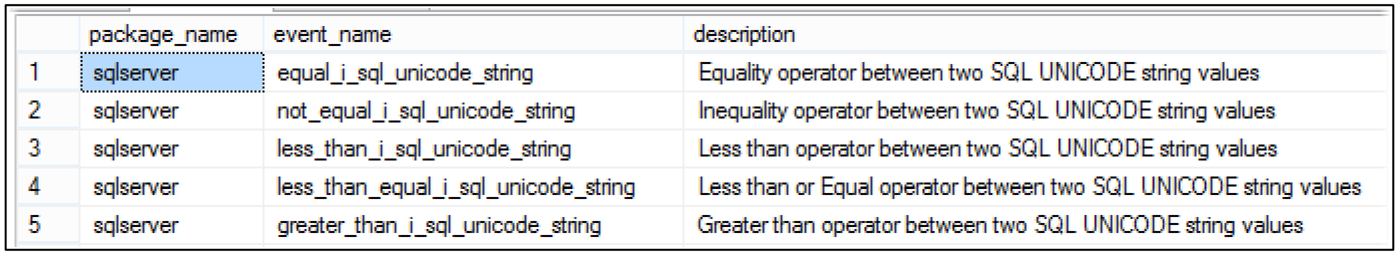
Note: There are around 44 predicate sources available in SQL Server 2012 but due to a shortage of space the below screenshot contains only the first four predicate sources.



	package_name	event_name	description
1	sqlserver	database_id	Get the current database ID
2	sqlserver	transaction_id	Get the current transaction ID
3	sqlserver	session_id	Get the current session ID
4	sqlserver	client_app_name	Get the current client application name

	(see below)

```
-- Predicate Comparators
-- Step 6: View all the predicate comparators available
SELECT XP.name AS package_name,
              XO.name AS event_name,
              XO.description
      FROM sys.dm_xe_packages AS XP
      JOIN sys.dm_xe_objects AS XO
           ON XP.guid = XO.package_guid
      WHERE  XO.object_type = 'pred_compare';
GO
```



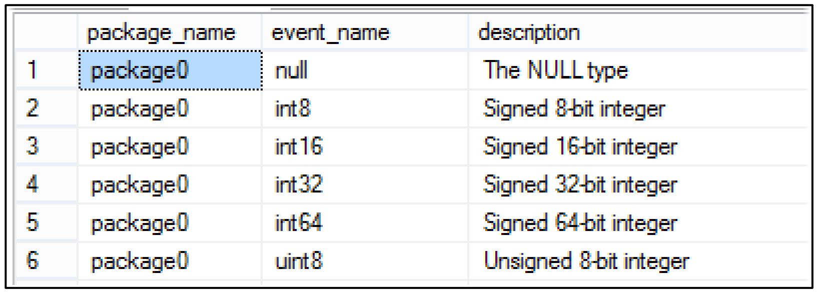
	package_name	event_name	description
1	sqlserver	equal_i_sql_unicode_string	Equality operator between two SQL UNICODE string values
2	sqlserver	not_equal_i_sql_unicode_string	Inequality operator between two SQL UNICODE string values
3	sqlserver	less_than_i_sql_unicode_string	Less than operator between two SQL UNICODE string values
4	sqlserver	less_than_equal_i_sql_unicode_string	Less than or Equal operator between two SQL UNICODE string values
5	sqlserver	greater_than_i_sql_unicode_string	Greater than operator between two SQL UNICODE string values

Note: There are around 77 predicate comparators available in SQL Server 2012 but due to a shortage of space the above screenshot contains only the first five predicate comparators.

View different data types returned by extended event

Execute the following statement(s) to view all the data types returned by extended event

```
-------------
-- Types
-------------
-- Step 7: View all the data types of the data returned by extended event
SELECT XP.name AS package_name,
              XO.name AS event_name,
              XO.description
      FROM sys.dm_xe_packages AS XP
      JOIN sys.dm_xe_objects AS XO
           ON XP.guid = XO.package_guid
     WHERE  XO.object_type = 'Type';
```



	package_name	event_name	description
1	package0	null	The NULL type
2	package0	int8	Signed 8-bit integer
3	package0	int16	Signed 16-bit integer
4	package0	int32	Signed 32-bit integer
5	package0	int64	Signed 64-bit integer
6	package0	uint8	Unsigned 8-bit integer

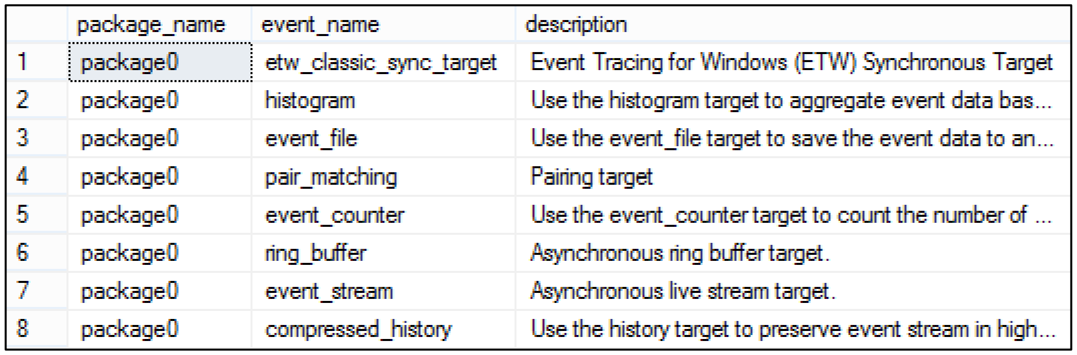
Note: There are around 28 data types available in SQL Server 2012 but due to a shortage of space the above screenshot contains only the first six types.

View all the available targets in extended event

Execute the following statement(s) to targets available in extended event

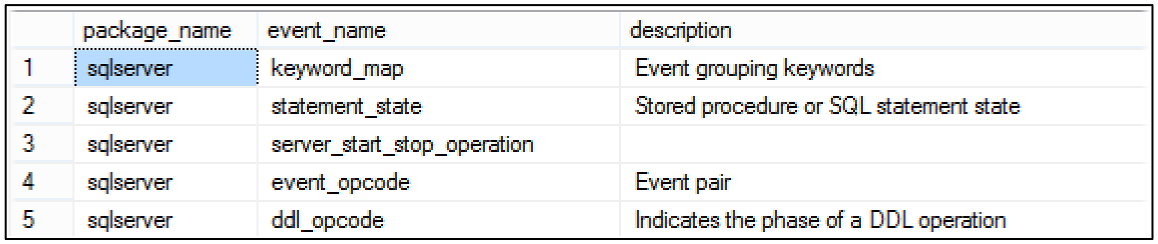
Explanation: SQL Server Extended Events targets are event consumers. Targets can write to a file, store event data in a memory buffer, or aggregate event data. Targets can process data synchronously or asynchronously.

```
-------------
-- Targets
-------------
-- Step 8: View all the available targets
SELECT XP.name AS package_name,
              XO.name AS event_name,
              XO.description
       FROM sys.dm_xe_packages AS XP
       JOIN sys.dm_xe_objects AS XO
            ON XP.guid = XO.package_guid
       WHERE  XO.object_type = 'target' AND XP.name = 'package0';
GO
```

	
View maps	Execute the following statement(s) to view maps available in extended events **Explanation**: A map table maps an internal value to a string, which enables a user to know what the value represents. Instead of only being able to obtain a numeric value, a user can get a meaningful description of the internal value. The following query shows how to obtain map values.

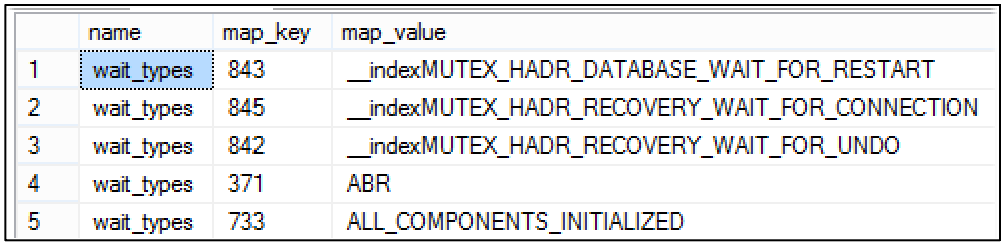
	package_name	event_name	description
1	package0	etw_classic_sync_target	Event Tracing for Windows (ETW) Synchronous Target
2	package0	histogram	Use the histogram target to aggregate event data bas...
3	package0	event_file	Use the event_file target to save the event data to an...
4	package0	pair_matching	Pairing target
5	package0	event_counter	Use the event_counter target to count the number of ...
6	package0	ring_buffer	Asynchronous ring buffer target.
7	package0	event_stream	Asynchronous live stream target.
8	package0	compressed_history	Use the history target to preserve event stream in high...

```
-------------
-- Maps
-------------
-- Step 9: View all the mapping objects available
SELECT XP.name AS package_name,
               XO.name AS event_name,
               XO.description
       FROM sys.dm_xe_packages AS XP
       JOIN sys.dm_xe_objects AS XO
            ON XP.guid = XO.package_guid
       WHERE  XO.object_type = 'map';
GO
```

	
Close all the query windows	Close all the query windows (✕) and if **SSMS** asks to save changes, click **NO**

	package_name	event_name	description
1	sqlserver	keyword_map	Event grouping keywords
2	sqlserver	statement_state	Stored procedure or SQL statement state
3	sqlserver	server_start_stop_operation	
4	sqlserver	event_opcode	Event pair
5	sqlserver	ddl_opcode	Indicates the phase of a DDL operation

```
-- Step 10: View mappings objects from map table
SELECT name, map_key, map_value
FROM sys.dm_xe_map_values
WHERE name = 'wait_types';
GO
```



	name	map_key	map_value
1	wait_types	843	__indexMUTEX_HADR_DATABASE_WAIT_FOR_RESTART
2	wait_types	845	__indexMUTEX_HADR_RECOVERY_WAIT_FOR_CONNECTION
3	wait_types	842	__indexMUTEX_HADR_RECOVERY_WAIT_FOR_UNDO
4	wait_types	371	ABR
5	wait_types	733	ALL_COMPONENTS_INITIALIZED

Summary

In this exercise, we have learned:

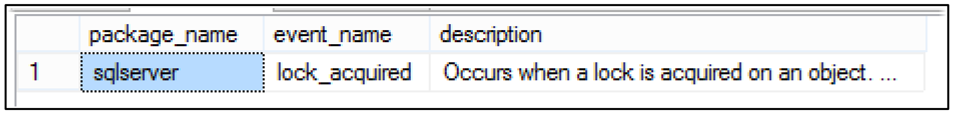
- Different components of extended events
- How to view details of each components using DMV

Exercise 2: Extended Event Sessions

Scenario

In this exercise, we will create an event session and will also monitor event session using system catalog views and DMV's.

Tasks	Detailed Steps
Open **2_EventSession.sql**	1. Click **File** | **Open** | **File** or press (**Ctrl + O**) 2. In **Open File** dialogue box, navigate to **SQL Server Extended Events Basics\Scripts** folder 3. Select **2_EventSession.sql** and click **Open**

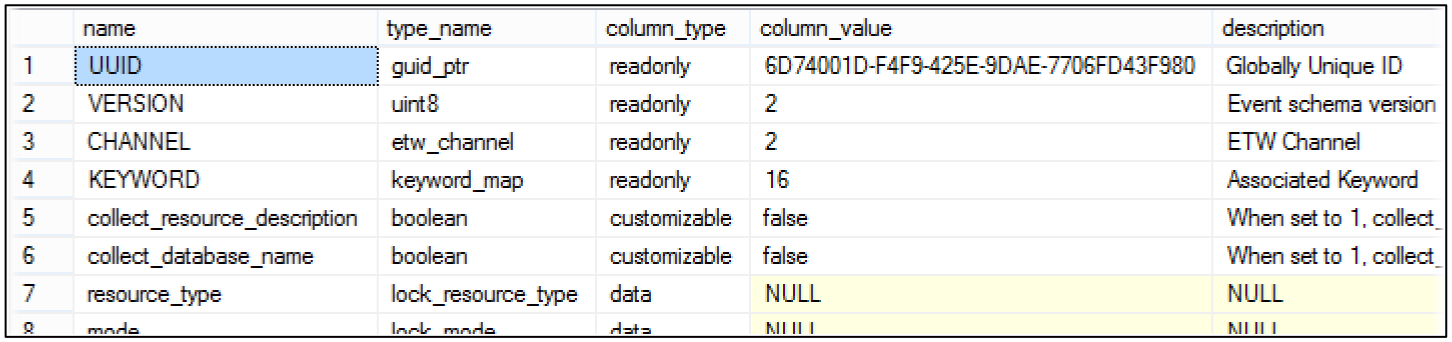
View event and package details	Execute the following statement(s) to view package and event details for **lock_acquired** event
	
	`-- Step 1: Find event information and package name`
	`SELECT XP.name AS package_name,`
	`              XO.name AS event_name,`
	`              XO.description`
	`      FROM sys.dm_xe_packages AS XP`
	`      JOIN sys.dm_xe_objects AS XO`
	`           ON XP.guid = XO.package_guid`
	`      WHERE  XO.name = 'lock_acquired';`
	`GO`
	
	

	package_name	event_name	description
1	sqlserver	lock_acquired	Occurs when a lock is acquired on an object. ...

View data elements for event **lock_acquired**

Execute the following statement(s) to view data types returned by **lock_acquired** event

```
-- Step 2: View data elements of lock_acquired extended event
SELECT XOC.name, XOC.type_name, XOC.column_type, XOC.column_value, XOC.description FROM
sys.dm_xe_objects AS XO
INNER JOIN sys.dm_xe_object_columns AS XOC
ON XO.name = XOC.object_name WHERE XO.name = 'lock_acquired'
AND XO.object_type = 'event';
GO
```



	name	type_name	column_type	column_value	description
1	UUID	guid_ptr	readonly	6D74001D-F4F9-425E-9DAE-7706FD43F980	Globally Unique ID
2	VERSION	uint8	readonly	2	Event schema version
3	CHANNEL	etw_channel	readonly	2	ETW Channel
4	KEYWORD	keyword_map	readonly	16	Associated Keyword
5	collect_resource_description	boolean	customizable	false	When set to 1, collect_
6	collect_database_name	boolean	customizable	false	When set to 1, collect_
7	resource_type	lock_resource_type	data	NULL	NULL
8	mode	lock_mode	data	NULL	NULL

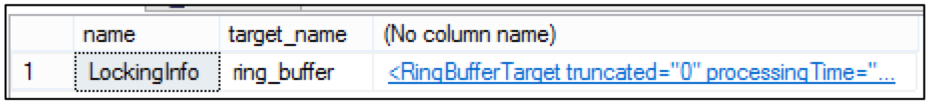
CREATE event session

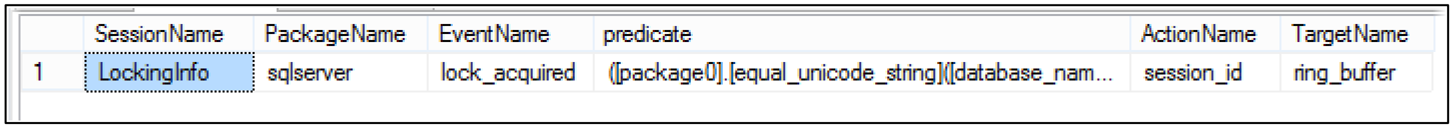
Execute the following statement(s) to **CREATE** event session **LockingInfo** for **lock_acquired** event

```
-- Step 3: Create event session
IF EXISTS(SELECT * FROM sys.server_event_sessions WHERE name='LockingInfo')
DROP EVENT SESSION [LockingInfo] ON SERVER;
CREATE EVENT SESSION [LockingInfo] ON SERVER
ADD EVENT sqlserver.lock_acquired(SET collect_database_name=(1),collect_resource_description=(1)
    ACTION(sqlserver.session_id)
    WHERE ([package0].[equal_unicode_string]([database_name],N'AdventureWorks2012')))
ADD TARGET package0.ring_buffer
WITH (MAX_DISPATCH_LATENCY=5 SECONDS)
GO
```

START event session	Execute the following statement(s) to **START** event session **LockingInfo**
	
	`-- Step 4: Start event session`
	`ALTER EVENT SESSION LockingInfo`
	`ON SERVER`
	`STATE=START;`
	`GO`

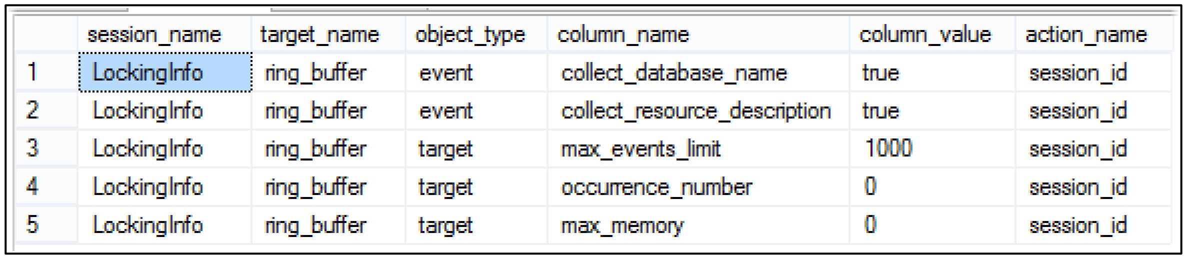
Execute an explicit transaction against **AdventureWorks2012** database	Execute the following statement(s) to run an explicit transaction against **AdventureWorks2012** database
	
	`-- Step 5: Execute an explicit transaction`
	`USE AdventureWorks2012`
	`BEGIN TRAN T1;`
	`UPDATE Person.Person SET FirstName = 'test'`
	`WHERE BusinessEntityID = 100;`
	`GO`

View event data	Execute the following statement(s) to view event data of **LockingInfo** event session
	
	`-- Step 6: View event data in XML format`
	`SELECT name, target_name, CAST(xet.target_data AS xml)`
	`FROM sys.dm_xe_session_targets AS xet`
	`JOIN sys.dm_xe_sessions AS xe`
	`   ON (xe.address = xet.event_session_address)`
	`WHERE xe.name = 'LockingInfo'`
	
	
	name / target_name / (No column name)
	1 / LockingInfo / ring_buffer / <RingBufferTarget truncated="0" processingTime="...

View event session details using system catalog view	Execute the following statement(s) to view event session details for **LockingInfo** event session using system catalog views
	
	`-- Step 7: View event session details from system catalogs`
	`SELECT sessions.name AS SessionName, sevents.package as PackageName,`
	`sevents.name AS EventName,`
	`sevents.predicate, sactions.name AS ActionName, stargets.name AS TargetName`
	`FROM sys.server_event_sessions sessions`
	`INNER JOIN sys.server_event_session_events sevents`
	`ON sessions.event_session_id = sevents.event_session_id`
	`INNER JOIN sys.server_event_session_actions sactions`
	`ON sessions.event_session_id = sactions.event_session_id`
	`INNER JOIN sys.server_event_session_targets stargets`
	`ON sessions.event_session_id = stargets.event_session_id`
	`WHERE sessions.name = 'LockingInfo'`
	`GO`
	
	
	
	| | SessionName | PackageName | EventName | predicate | ActionName | TargetName |
	| 1 | LockingInfo | sqlserver | lock_acquired | ([package0].[equal_unicode_string]([database_nam... | session_id | ring_buffer |

View configurable column and target details using **DMV**	Execute the following statement(s) to view configurable column and target details for LockingInfo event session
	
	`-- Step 8: View configurable column and target details from DMV`
	`SELECT DISTINCT s.name AS session_name,`
	`          t.target_name,`
	`       oc.object_type,`
	`       oc.column_name,`
	`       oc.column_value,`
	`          ea.action_name`
	`FROM sys.dm_xe_sessions AS s`
	`INNER JOIN sys.dm_xe_session_targets AS t`
	`     ON s.address = t.event_session_address`
	`INNER JOIN sys.dm_xe_session_events AS e`
	`     ON s.address = e.event_session_address`
	`INNER JOIN sys.dm_xe_session_object_columns AS oc`

```
       ON s.address = oc.event_session_address
INNER JOIN sys.dm_xe_session_event_actions as ea
       ON s.address = ea.event_session_address
       AND ((oc.object_type = 'target' AND t.target_name = oc.object_name)
       OR (oc.object_type = 'event' AND e.event_name = oc.object_name)) WHERE S.name = 'LockingInfo';
```



	session_name	target_name	object_type	column_name	column_value	action_name
1	LockingInfo	ring_buffer	event	collect_database_name	true	session_id
2	LockingInfo	ring_buffer	event	collect_resource_description	true	session_id
3	LockingInfo	ring_buffer	target	max_events_limit	1000	session_id
4	LockingInfo	ring_buffer	target	occurrence_number	0	session_id
5	LockingInfo	ring_buffer	target	max_memory	0	session_id

Cleanup

Execute the following script in Cleanup section

```
---------------------
-- Begin: Cleanup
---------------------
-- Rollback transaction T1
ROLLBACK TRAN T1

-- Stop the event session
ALTER EVENT SESSION LockingInfo
ON SERVER
STATE=STOP

-- Drop the event session
DROP EVENT SESSION LockingInfo
ON SERVER

---------------------
-- End: Cleanup
---------------------
```

Close all the query windows	Close all the query windows (☒) and if **SSMS** asks to save changes, click **NO**

Summary

In this exercise, we have learned:

- How to **CREATE** and **START** an event session
- How to view event data of an event session
- How to view event session details using system catalog view
- How to view configurable column and target details using DMV

Exercise 3: Extended Events Targets

Overview:

SQL Server Extended Events targets are event consumers. Targets can write to a file, store event data in a memory buffer, or aggregate event data. Targets can process data synchronously or asynchronously.
The Extended Events design ensures that targets are guaranteed to receive events once and only once per session.
Extended Events provide the following targets that we can use for an Extended Events session:

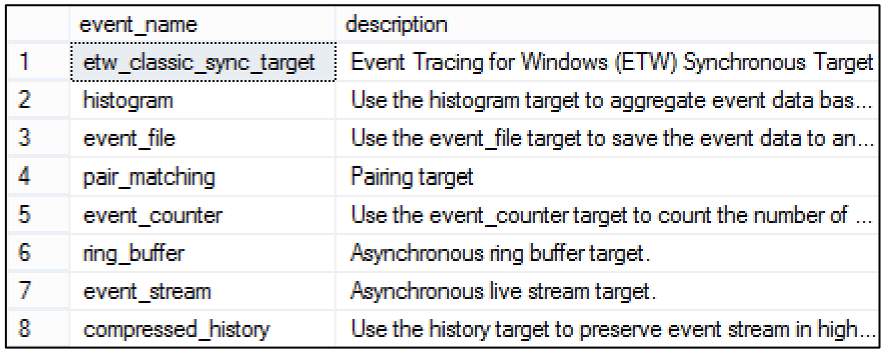
- **Event counter**
  Counts all specified events that occur during an Extended Events session. Use to obtain information about workload characteristics without adding the overhead of full event collection. This is a synchronous target.
- **Event file**
  Use to write event session output from complete memory buffers to disk. This is an asynchronous target.
- **Event pairing**
  Many kinds of events occur in pairs, such as lock acquires and lock releases. Use to determine when a specified paired event does not occur in a matched set. This is an asynchronous target.
- **Event Tracing for Windows (ETW)**
  Use to correlate SQL Server events with Windows operating system or application event data. This is a synchronous target.
- **Histogram**
  Use to count the number of times that a specified event occurs, based on a specified event column or action. This is an asynchronous target.
- **Ring buffer**
  Use to hold the event data in memory on a first-in-first-out (FIFO) basis or on a per-event FIFO basis. This is an asynchronous target.

Scenario

In this exercise, we will look at different targets available in SQL Server extended event.

Tasks	Detailed Steps
Open **3_EventTargets.sql**	1. Click **File** | **Open** | **File** or press (**Ctrl + O**) 2. In **Open File** dialogue box, navigate to **SQL Server Extended Events Basics\Scripts** folder 3. Select **3_EventTargets.sql** and click **Open**
View available targets	Execute the following statement(s) to view available targets (code and results below)

```
SET NOCOUNT ON;
-- Step 1: View all the available targets
SELECT
              XO.name AS event_name,
              XO.description
      FROM sys.dm_xe_packages AS XP
      JOIN sys.dm_xe_objects AS XO
           ON XP.guid = XO.package_guid
      WHERE  XO.object_type = 'target' AND XP.name = 'package0';
GO
```



	event_name	description
1	etw_classic_sync_target	Event Tracing for Windows (ETW) Synchronous Target
2	histogram	Use the histogram target to aggregate event data bas...
3	event_file	Use the event_file target to save the event data to an...
4	pair_matching	Pairing target
5	event_counter	Use the event_counter target to count the number of ...
6	ring_buffer	Asynchronous ring buffer target.
7	event_stream	Asynchronous live stream target.
8	compressed_history	Use the history target to preserve event stream in high...

CREATE and view event session with **event counter** target

Execute the following statement(s) one by one

In this step, we will do the following

1. **CREATE** an event session **LockingInfo1** with **event counter** as target
2. **START LockingInfo1** event session
3. Execute an explicit transaction against **AdventureWorks2012** database to generate event data
4. View **LockingInfo1** event data
5. **ROLLBACK** explicit transaction executed against **AdventureWorks2012** database

```
--------------------------------
-- Begin: Step 2 (Event Counter)
--------------------------------
-- Create event session
IF EXISTS(SELECT * FROM sys.server_event_sessions WHERE name='LockingInfo1')
DROP EVENT SESSION [LockingInfo1] ON SERVER;
CREATE EVENT SESSION [LockingInfo1] ON SERVER
ADD EVENT sqlserver.lock_acquired
ADD TARGET package0.event_counter
WITH (EVENT_RETENTION_MODE=ALLOW_SINGLE_EVENT_LOSS,MAX_DISPATCH_LATENCY=5 SECONDS)
GO

-- Start event session
ALTER EVENT SESSION LockingInfo1
ON SERVER
STATE=START;
GO

-- Execute an explicit transaction
USE AdventureWorks2012
BEGIN TRAN T1;
UPDATE Person.Person SET FirstName = 'test'
WHERE BusinessEntityID = 100;
GO
```

```
-- View event data
SELECT name, target_name, CAST(xet.target_data AS xml)
FROM sys.dm_xe_session_targets AS xet
JOIN sys.dm_xe_sessions AS xe
   ON (xe.address = xet.event_session_address)
WHERE xe.name = 'LockingInfo1'

-- Rollback transaction T1
ROLLBACK TRAN T1;

---------------------------
-- End: Step 2
---------------------------
```

Explanation: The event counter target counts all events that occur during an Extended Events session. By using the event counter target, you can obtain information about workload characteristics without adding the overhead of full event collection. This target has no customizable parameters.

CREATE and view event session with **event file** target

Execute the following statement(s) one by one

In this step, we will do the following

1. **CREATE** an event session **LockingInfo2** with **event file** as target
2. **START LockingInfo2** event session
3. Execute an explicit transaction against **AdventureWorks2012** database to generate event data
4. View **LockingInfo2** event data
5. **ROLLBACK** explicit transaction executed against **AdventureWorks2012** database

```
---------------------------
-- Begin: Step3 (Event File)
---------------------------
-- Create event session
IF EXISTS(SELECT * FROM sys.server_event_sessions WHERE name='LockingInfo2')
DROP EVENT SESSION [LockingInfo2] ON SERVER;
CREATE EVENT SESSION [LockingInfo2] ON SERVER
ADD EVENT sqlserver.lock_acquired
```

```sql
ADD TARGET package0.event_file(SET filename=N'C:\temp\LockingInfo2.xel')
WITH (MAX_DISPATCH_LATENCY=5 SECONDS)
GO
-- Start event session
ALTER EVENT SESSION LockingInfo2
ON SERVER
STATE=START;
GO

-- Execute an explicit transaction
USE AdventureWorks2012
BEGIN TRAN T1;
UPDATE Person.Person SET FirstName = 'test'
WHERE BusinessEntityID = 100;
GO

-- View event data
SELECT *, CAST(event_data AS XML) AS 'event_data_XML'
FROM sys.fn_xe_file_target_read_file('C:\temp\LockingInfo2*.xel', NULL, NULL, NULL)

-- Rollback transaction T1
ROLLBACK TRAN T1;

---------------------------
-- End: Step3
---------------------------
```

Explanation: The event file target is a target that writes complete buffers to disk. The first time that an event file target is created, the filename we specify is appended with _0_ and a long integer value. The integer value is calculated as the number of milliseconds between January 1, 1600, and the date and time the file is created. Subsequent rollover files also use this format. From examining the value of the long integer, we can determine the most current file.

CREATE and view event session with **histogram**

target

Execute the following statement(s) one by one

In this step, we will do the following

1. **CREATE** an event session **LockingInfo3** with **histogram** as target
2. **START LockingInfo3** event session
3. Execute an explicit transaction against **AdventureWorks2012** database to generate event data
4. View **LockingInfo3** event data
5. **ROLLBACK** explicit transaction executed against **AdventureWorks2012** database

```
---------------------------
-- Begin: Step 4 (Histograms)
---------------------------
-- Create event session
IF EXISTS(SELECT * FROM sys.server_event_sessions WHERE name='LockingInfo3')
DROP EVENT SESSION [LockingInfo3] ON SERVER;
CREATE EVENT SESSION [LockingInfo3] ON SERVER
ADD EVENT sqlserver.lock_acquired
ADD TARGET package0.histogram(SET
filtering_event_name=N'sqlserver.lock_acquired',source=N'mode',source_type=(0))
WITH (MAX_DISPATCH_LATENCY=3 SECONDS)
GO

-- Start event session
ALTER EVENT SESSION LockingInfo3
ON SERVER
STATE=START;
GO

-- Execute an explicit transaction
USE AdventureWorks2012
BEGIN TRAN T1;
UPDATE Person.Person SET FirstName = 'test'
WHERE BusinessEntityID = 100;
GO
```

	`-- View event data`
	`SELECT name, target_name, CAST(xet.target_data AS xml)`
	`FROM sys.dm_xe_session_targets AS xet`
	`   JOIN sys.dm_xe_sessions AS xe`
	`   ON (xe.address = xet.event_session_address)`
	`WHERE xe.name = 'LockingInfo3'`
	
	`-- Rollback transaction T1`
	`ROLLBACK TRAN T1;`
	
	`---------------------------`
	`-- End: Step 4`
	`---------------------------`
	
	Explanation: The histogram target groups occurrences of a specific event type based on event data. The groupings of events are counted based on a specified event column or action. We can use the histogram target to troubleshoot performance issues. By identifying which events are occurring most frequently, we can find "hotspots" that indicate a potential cause of a performance problem.

CREATE and view event session with **ring buffer** target	Execute the following statement(s) one by one
	
	In this step, we will do the following
	1. **CREATE** an event session **LockingInfo4** with **ring buffer** as target
	2. **START LockingInfo4** event session
	3. Execute an explicit transaction against **AdventureWorks2012** database to generate event data
	4. View **LockingInfo4** event data
	5. **ROLLBACK** explicit transaction executed against **AdventureWorks2012** database
	`---------------------------`
	`-- Begin: Step 5 (Ring Buffer)`
	`---------------------------`
	`-- Create event session`
	`IF EXISTS(SELECT * FROM sys.server_event_sessions WHERE name='LockingInfo5')`
	`DROP EVENT SESSION [LockingInfo4] ON SERVER;`

```
CREATE EVENT SESSION [LockingInfo4] ON SERVER
ADD EVENT sqlserver.lock_acquired
ADD TARGET package0.ring_buffer
WITH (MAX_DISPATCH_LATENCY=5 SECONDS)
GO

-- Start event session
ALTER EVENT SESSION LockingInfo4
ON SERVER
STATE=START;
GO

-- Execute an explicit transaction
USE AdventureWorks2012
BEGIN TRAN T1;
UPDATE Person.Person SET FirstName = 'test'
WHERE BusinessEntityID = 100;
GO

-- View event data
SELECT name, target_name, CAST(xet.target_data AS xml)
FROM sys.dm_xe_session_targets AS xet
JOIN sys.dm_xe_sessions AS xe
   ON (xe.address = xet.event_session_address)
WHERE xe.name = 'LockingInfo4'

-- Rollback transaction T1
ROLLBACK TRAN T1;

---------------------------
-- End: Step 5
---------------------------
```

Explanation: The ring buffer target briefly holds event data in memory. This target can manage events in one of two modes.

- The first mode is strict first-in-first-out (FIFO), where the oldest event is discarded when all the memory allocated to the target is used. In this mode (the default), the **occurrence_number** option is set to 0
- The second mode is per-event FIFO, where a specified number of events of each type is kept. In this mode, the oldest events of each type are discarded when all the memory allocated to the target is used. You can configure the **occurrence_number** option to specify the number of events of each type to keep.

CREATE and view event session with **event pairing** target

Execute the following statement(s) one by one

In this step, we will do the following

1. **CREATE** an event session **LockingInfo5** with **event pairing** as target
2. **START LockingInfo5** event session
3. Execute an explicit transaction against **AdventureWorks2012** database to generate event data
4. View **LockingInfo5** event data
5. **ROLLBACK** explicit transaction executed against **AdventureWorks2012** database

```
---------------------------
-- Begin: Step 6(Event Pairing)
---------------------------
-- Create event session
IF EXISTS(SELECT * FROM sys.server_event_sessions WHERE name='LockingInfo5')
DROP EVENT SESSION [LockingInfo5] ON SERVER;
CREATE EVENT SESSION [LockingInfo5] ON SERVER
ADD EVENT sqlserver.lock_acquired,
ADD EVENT sqlserver.lock_released
ADD TARGET package0.pair_matching(SET
begin_event=N'sqlserver.lock_acquired',end_event=N'sqlserver.lock_released')
WITH (MAX_DISPATCH_LATENCY=3 SECONDS)
GO

-- Start event session
ALTER EVENT SESSION LockingInfo5
ON SERVER
STATE=START;
GO
```

	`-- Execute an explicit transaction`
	`USE AdventureWorks2012`
	`BEGIN TRAN T1;`
	`UPDATE Person.Person SET FirstName = 'test'`
	`WHERE BusinessEntityID = 100;`
	`GO`
	
	`-- View event data`
	`SELECT name, target_name, CAST(xet.target_data AS xml)`
	`FROM sys.dm_xe_session_targets AS xet`
	`JOIN sys.dm_xe_sessions AS xe`
	`   ON (xe.address = xet.event_session_address)`
	`WHERE xe.name = 'LockingInfo5'`
	
	`-- Rollback transaction T1`
	`ROLLBACK TRAN T1;`
	
	`---------------------------`
	`-- End: Step 6`
	`---------------------------`
	
	Explanation: The event pairing target matches two events using one or more columns of data that are present in each event. Many events come in pairs, for example, lock acquires and lock releases. After an event sequence is paired, both events are discarded. Discarding matched sets allows for easy detection of lock acquisitions that have not been released.

Cleanup	Execute the following script in Cleanup section
	
	`-------------------`
	`-- Begin: Cleanup`
	`-------------------`
	`-- Stop the event sessions`
	`ALTER EVENT SESSION LockingInfo1`
	`ON SERVER`
	`STATE=STOP`

	```
	ALTER EVENT SESSION LockingInfo2
	ON SERVER
	STATE=STOP
	ALTER EVENT SESSION LockingInfo3
	ON SERVER
	STATE=STOP
	ALTER EVENT SESSION LockingInfo4
	ON SERVER
	STATE=STOP
	ALTER EVENT SESSION LockingInfo5
	ON SERVER
	STATE=STOP

	-- Drop the event sessions
	DROP EVENT SESSION LockingInfo1
	ON SERVER
	DROP EVENT SESSION LockingInfo2
	ON SERVER
	DROP EVENT SESSION LockingInfo3
	ON SERVER
	DROP EVENT SESSION LockingInfo4
	ON SERVER
	DROP EVENT SESSION LockingInfo5
	ON SERVER
	```
Close all the query windows	Close all the query windows (☒) and if **SSMS** asks to save changes, click **NO**

Summary

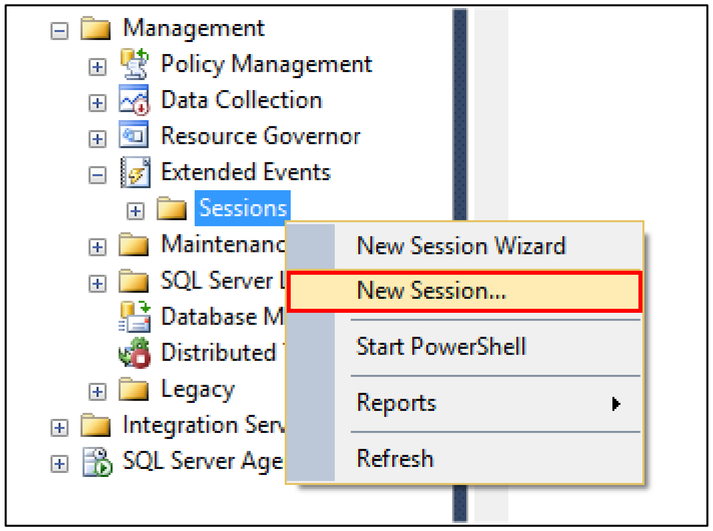
In this exercise, we have learned:

- Different types of targets available in SQL Server extended events
- How to setup event session with different targets
- How to view event session data for each target

Exercise 4: Extended Events Graphical User Interface

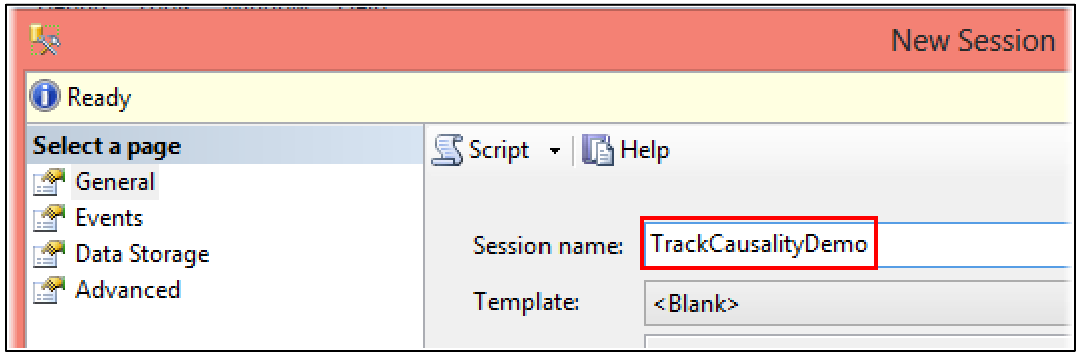
Scenario

In this exercise, we will look at different components and some advanced features of extended events graphical user interface.

Tasks	Detailed Steps
Launch **SQL Server Management Studio**	1. Click **Start** | **All Programs** | **SQL Server 2012** | **SQL Server Management Studio** 2. In the **Connect to Server** dialog box, click **Connect**
Open extended event graphical user interface	1. Click **View** | **Object Explorer** or press (**F8**) 2. Expand Server | Expand **Management** | Expand **Extended Events** 3. Right click on **New Session**  Management Policy Management Data Collection Resource Governor Extended Events Sessions Maintenanc SQL Server L Database M Distributed Legacy Integration Serv SQL Server Age New Session Wizard New Session... Start PowerShell Reports Refresh

CREATE an event session

1. Give **session name** as (Or any name)

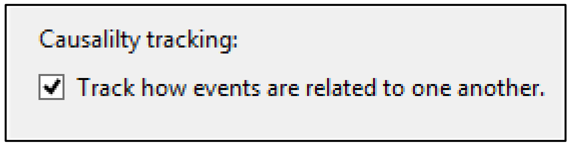


Session name: TrackCausalityDemo

Template: <Blank>

2. Enable **causality tracking** by selecting the checkbox

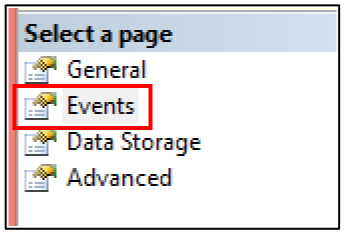
Note: It's located at the bottom of **General page**. Expand the **New Session** window, if unable to view.



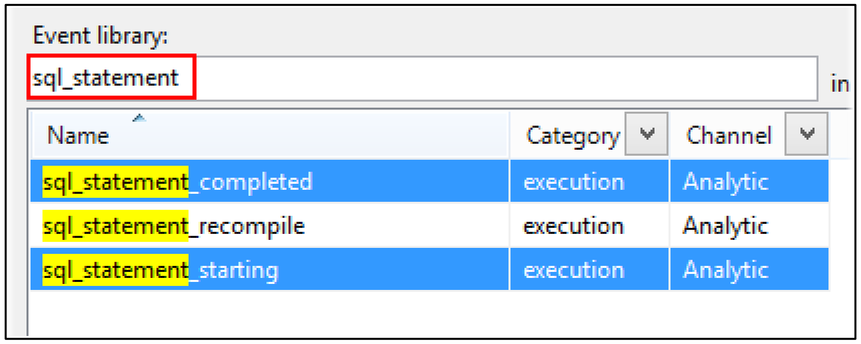
Causalilty tracking:

☑ Track how events are related to one another.

3. Click on **Events page**



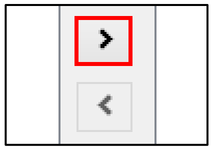
4. On the **Event library** type **sql_statement**



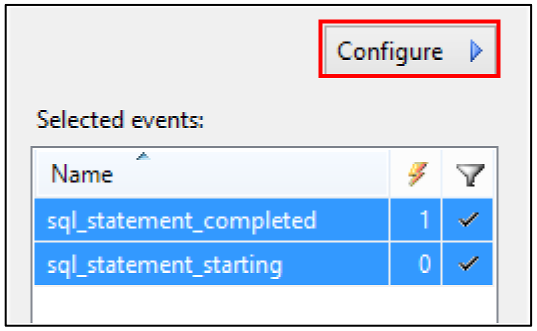
Event library:
sql_statement

Name	Category	Channel
sql_statement_completed	execution	Analytic
sql_statement_recompile	execution	Analytic
sql_statement_starting	execution	Analytic

5. Click on arrow facing right side to include the events for configuration



6. Click on **Configure** to configure event session

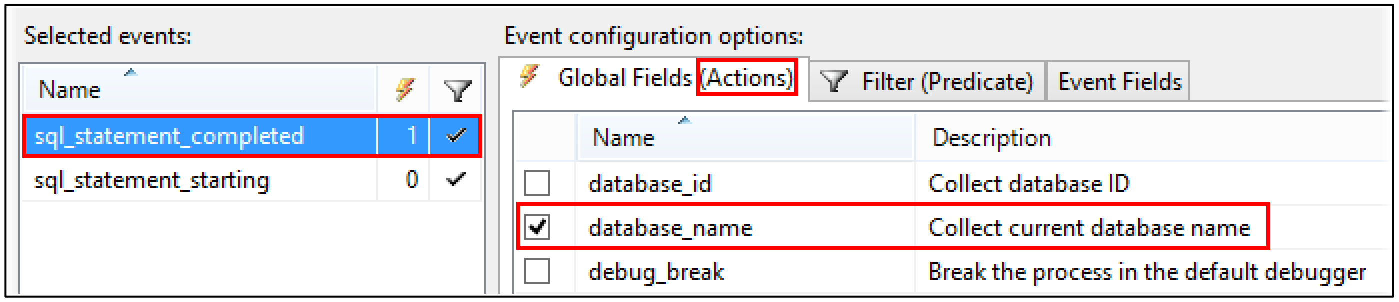


Selected events:

Name	⚡	▽
sql_statement_completed	1	✓
sql_statement_starting	0	✓

7. Select **sql_statement_completed** from the **Selected events** list and the mark **database_name** checkbox

   Note: We can configure **Actions** for an event session from this window



Selected events:

Name	⚡	▽
sql_statement_completed	1	✓
sql_statement_starting	0	✓

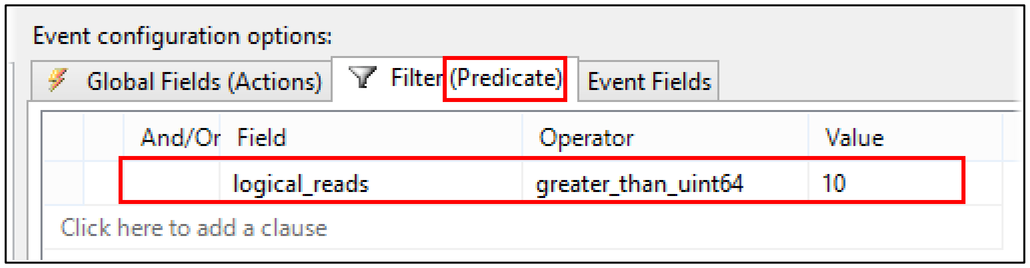
Event configuration options:

Global Fields (Actions) | Filter (Predicate) | Event Fields

	Name	Description
☐	database_id	Collect database ID
☑	database_name	Collect current database name
☐	debug_break	Break the process in the default debugger

8. Click on **Filter (Predicate)** tab and click just below field (**Click here to add clause**) and select **logical_reads** is equal to 10 as shown in the below diagram

   Note: We can configure **Predicates** for an event session from this window



Event configuration options:

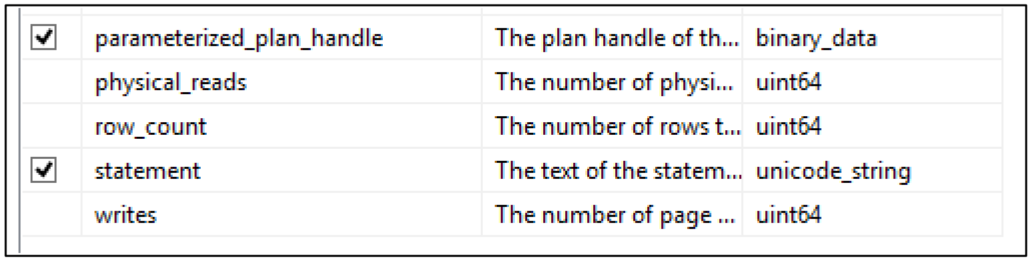
Global Fields (Actions) | Filter (Predicate) | Event Fields

And/Or	Field	Operator	Value
	logical_reads	greater_than_uint64	10

Click here to add a clause

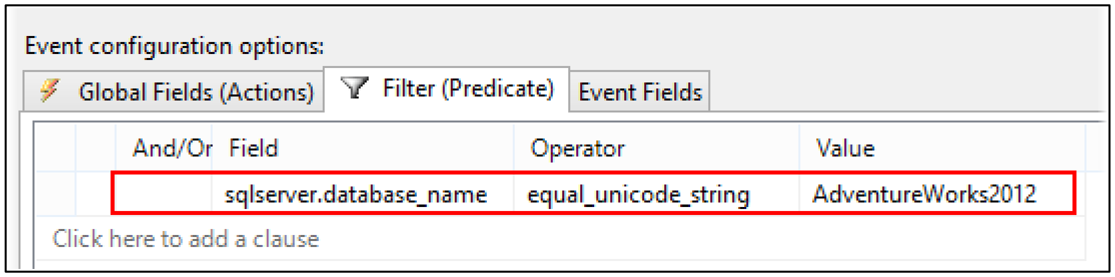
9. Click on **Event Fields** tab and mark **parameterized_plan_handle** and **statement** check box

   Note: We can configure a **configurable column** from this window if the event contains any **configurable column** in **Event Fields.**



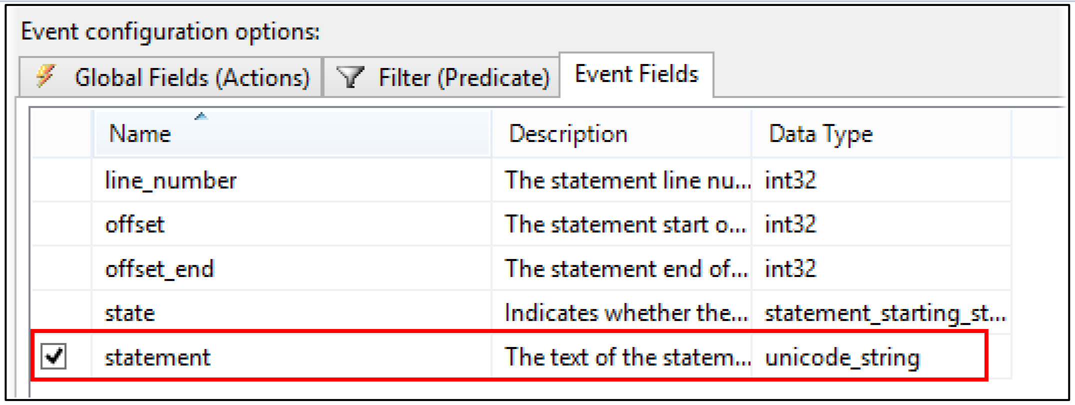
✓	parameterized_plan_handle	The plan handle of th...	binary_data
	physical_reads	The number of physi...	uint64
	row_count	The number of rows t...	uint64
✓	statement	The text of the statem...	unicode_string
	writes	The number of page ...	uint64

10. Select **sql_statement_starting** from **Selected events** and click on **Filter(Predicate)** tab. From the **Field** drop down list choose **sqlserver.database_name**, from the **Operator** drop-down list, choose **equal_unicode_string** and in the **Value** text box write **AdventureWorks2012**



And/Or	Field	Operator	Value
	sqlserver.database_name	equal_unicode_string	AdventureWorks2012

11. In the Event Fields for **sql_statement_starting** mark statement check box



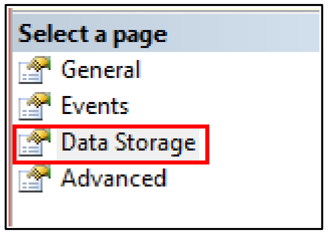
Event configuration options:

Global Fields (Actions) | Filter (Predicate) | Event Fields

	Name	Description	Data Type
	line_number	The statement line nu...	int32
	offset	The statement start o...	int32
	offset_end	The statement end of...	int32
	state	Indicates whether the...	statement_starting_st...
✓	statement	The text of the statem...	unicode_string

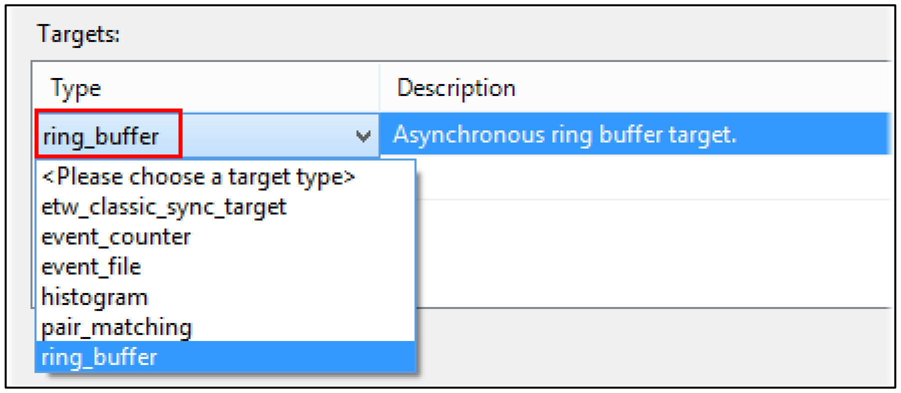
12. Click on **Data Storage** page

    Note: We can configure **Targets** for an event session from this window.



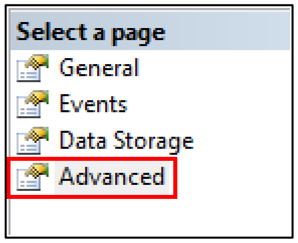
13. Select **ring_buffer** from the drop down list.

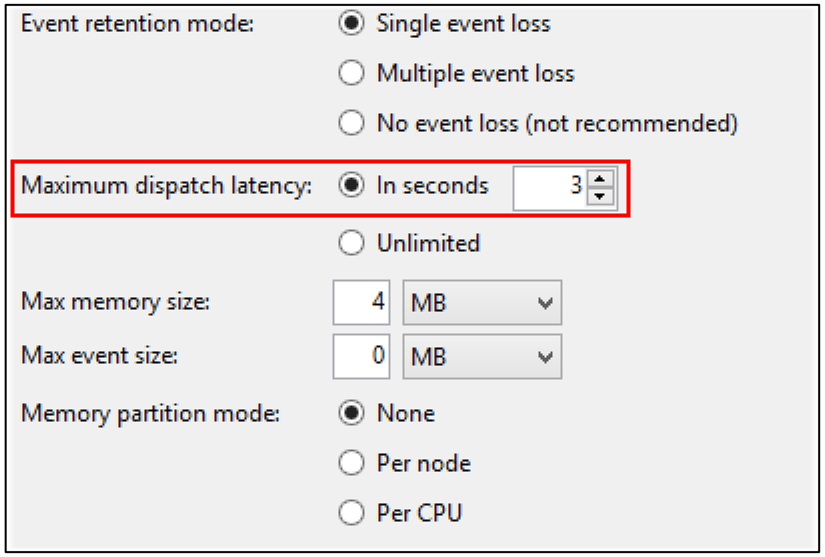
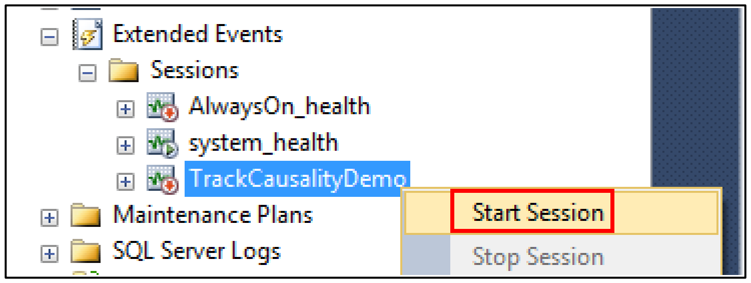
    Note: Each target has some configurable options, observe that in the bottom of this window.

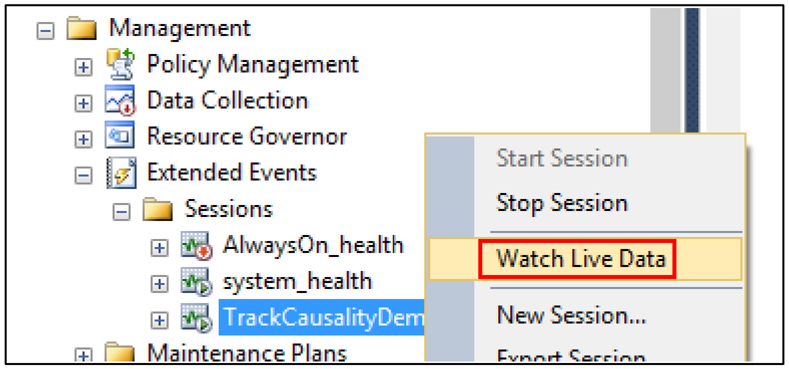

Targets:

Type	Description
ring_buffer	Asynchronous ring buffer target.

14. Click on **Advanced page**

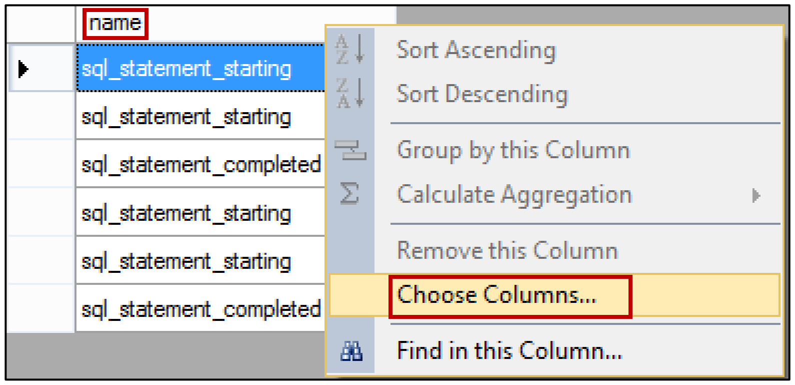


	15. Change the value of Maximum dispatch latency from default 30 seconds to 3 seconds  Event retention mode: [✓ Single event loss] Multiple event loss No event loss (not recommended) Maximum dispatch latency: [✓ In seconds] 3 Unlimited Max memory size: 4 MB Max event size: 0 MB Memory partition mode: [✓ None] Per node Per CPU 16. Click on **OK**
START the event session	1. Expand **Extended Events** | Expand **Sessions** 2. Right click on **TrackCausalityDemo** | Click on **Start Session**  Extended Events Sessions AlwaysOn_health system_health TrackCausalityDemo Maintenance Plans SQL Server Logs Start Session Stop Session

View event data	1. Right click on **TrackCausalityDemo** | Click on **Watch Live Data**  Management Policy Management Data Collection Resource Governor Extended Events Sessions AlwaysOn_health system_health TrackCausalityDem Maintenance Plans Start Session Stop Session Watch Live Data New Session... Export Session

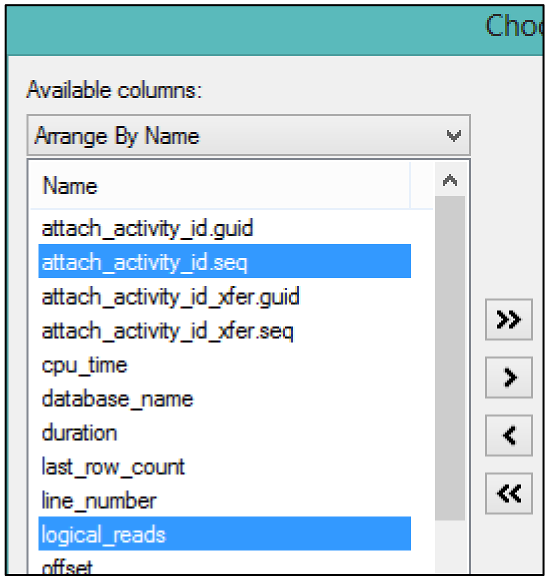
Execute a **SELECT** statement	Execute the following statement multiple times to **SELECT** data from **AdventureWorks2012** database
	
	`USE AdventureWorks2012;`
	`GO`
	`SELECT * FROM Person.Person`
	`GO`
	
	Note: Execute the above **SELECT** statement multiple times to get the **sql_statement_complete** event data.

View event data	Switch to **Live Data** query window 1. Right click on **name** field **|** click on **Choose columns**

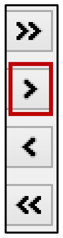
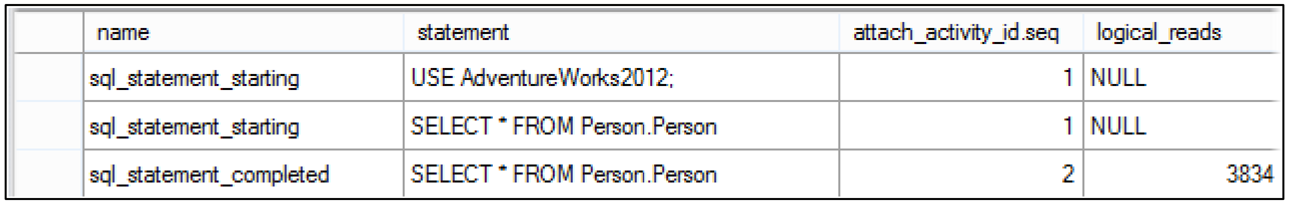
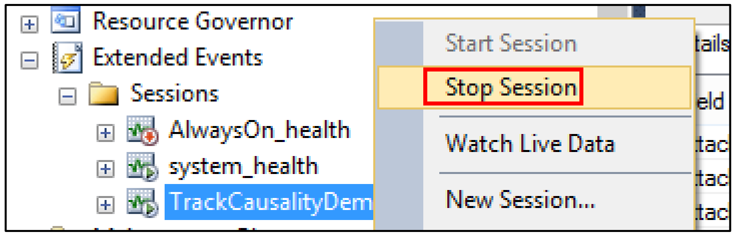


name
sql_statement_starting
sql_statement_starting
sql_statement_completed
sql_statement_starting
sql_statement_starting
sql_statement_completed

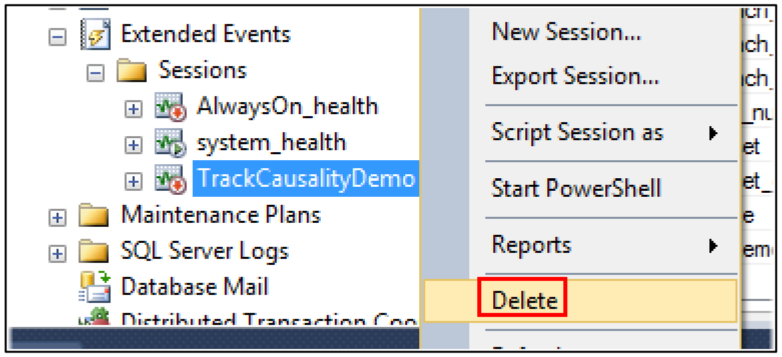
2. Select **attach_activity_id.seq** , **logical_reads** , **statement** from **Available Columns** list



attach_activity_id.guid
attach_activity_id.seq
attach_activity_id_xfer.guid
attach_activity_id_xfer.seq
cpu_time
database_name
duration
last_row_count
line_number
logical_reads
offset

	3. Click on arrow facing right side to include the columns and click on **OK**  4. observe the event data for **TrackCausality** event session 
Close all the query windows	Close all the query windows (✕) and if **SSMS** asks to save changes, click **NO**
STOP the event session	1. Expand **Extended Events** | Expand **Sessions** 2. Right click on **TrackCausalityDemo** | Click on **Stop Session** 

	name	statement	attach_activity_id.seq	logical_reads
	sql_statement_starting	USE AdventureWorks2012;	1	NULL
	sql_statement_starting	SELECT * FROM Person.Person	1	NULL
	sql_statement_completed	SELECT * FROM Person.Person	2	3834

DROP the event session	1. Expand **Extended Events** | Expand **Sessions** 2. Right click on **TrackCausalityDemo** | Click on **Delete** **|** Click on **OK**  Extended Events Sessions AlwaysOn_health system_health TrackCausalityDemo Maintenance Plans SQL Server Logs Database Mail New Session... Export Session... Script Session as Start PowerShell Reports Delete

Summary

In this exercise, we have learned:

- How to configure extended event session from GUI
- How to enable Track Causality
- How to start an event session from GUI
- How to view Live Data for an event session
- How to **STOP** and **DROP** an event session

Other learning opportunities from us that might interest you.

Brand	Description	Important Links
DataPlatformGeeks	DataPlatformGeeks is a community of Data, Analytics & AI professionals. It is a community initiative and all activities including webinars, events, etc., are free and community driven. You can join for free and access all learning resources for free. Join here: https://www.dataplatformgeeks.com/registration/ Follow on Twitter. Follow on LinkedIn.	www.DataPlatformGeeks.com YouTube: www.YouTube.com/SQLServerGeeks Telegram: https://t.me/dataplatformgeeks More Social Channels: https://www.dataplatformgeeks.com/our-social-channels/
Data Platform Summit	Data Platform Summit is Asia's largest Data, Analytics & AI learning event. Whole week of deep-dive training takes place in Bangalore each year in the month of August/September. World's best speakers and delegates join from more than 20 countries.	www.DPS10.com www.DataPlatformSummit.in
PEOPLEWARE INDIA TECHNOLOGY TRAINING SPECIALISTS	Peopleware India offers full length video courses on latest technologies.	www.PeoplewareIndia.com
SQLServerGeeks	Another community initiative, SQLServerGeeks is one of the most popular websites on SQL Server with thousands of blogs, articles, videos and learning resources on Microsoft SQL Server. SQLServerGeeks also organizes frequent webinars and events to impart free education on SQL Server.	www.SQLServerGeeks.com www.facebook.com/SQLServerGeeks YouTube: www.YouTube.com/SQLServerGeeks www.twitter.com/SQLServerGeeks Join SSG on LinkedIn
SQLशीघ्र SQLSHIGHRA	At SQLShighra.com you get quick SQL Server Tips and know-how related to SQL Server Internals, Troubleshooting and Performance Tuning. These are short & casual videos, mostly Amit Bansal's SQL brain dump and he records them anywhere, anytime (not literally).	www.SQLShighra.com
SQLMaestros	A team of specialists on SQL Server & Microsoft Data Platform, SQLMaestros offers affordable learning solutions and also offers SQL Server health check & baselining service. Video Courses. Hands-On-Labs. Learning Kits.	Follow/Join SQL Server Discussions: RSS Subscription, Telegram, YouTube, Twitter, LinkedIn, Facebook

*The above information is constantly updated here: www.eDominer.com/learning