**Using Extended Events to Track SQL Execution**

**Extended Events** (XEvents) is a lightweight, high-performance system for capturing events and data in SQL Server. You can use **Extended Events** to capture real-time query execution data and store it for later analysis. Unlike **Query Store**, which provides historical execution data over time, Extended Events captures real-time events as they happen.

**Key Features of Extended Events:**

1. **Real-time Data Capture**: You can capture SQL Server activity in real time as it happens.
2. **Flexible Event Filtering**: You can filter specific SQL Server events to capture.
3. **Low Overhead**: It is designed to capture events with minimal performance overhead, especially when compared to SQL Profiler.
4. **Storage Options**: You can configure it to store event data in memory or in files, making it easier to analyze after the event session completes.

**Where is Data Stored?**

Extended Events data can be stored in two main ways:

1. **Memory**: If you configure the event session to store data in memory, the event data will be available as long as the session is active. Once the session ends or the server is restarted, the data is lost.
2. **File-based Storage**: For persistent storage, you can configure Extended Events to store event data in a file, usually with a .xel extension. You can later read the .xel file for historical analysis using tools like SQL Server Management Studio (SSMS) or querying the file directly using T-SQL.

**Key Event to Track SQL Queries**

To track SQL queries, the most common event to use is sqlserver.sql\_batch\_completed (for SQL batch queries) or sqlserver.rpc\_completed (for stored procedure execution).

These events capture details like:

* SQL text
* Client information
* Execution times
* Query duration
* And more

**Steps to Set Up Extended Events for Tracking SQL Queries**

1. **Create an Extended Events Session**: You define an Extended Events session to capture specific events and store them in a file.
2. **Configure Event Filters**: You can add filters to capture only the events you're interested in, such as capturing only completed SQL batches or specific queries based on conditions (e.g., duration, database name).

**Example: Capture Completed SQL Batches with Extended Events**

Here is an example of how to create an Extended Event session to capture completed SQL batches (sql\_batch\_completed) and store them in a file.

-- Step 1: Create the Extended Event Session

CREATE EVENT SESSION [TrackSQLHistory] ON SERVER

ADD EVENT sqlserver.sql\_batch\_completed

(

ACTION(sqlserver.sql\_text, sqlserver.client\_hostname, sqlserver.username, sqlserver.database\_name, sqlserver.execution\_time)

WHERE (sqlserver.database\_name = 'YourDatabase') -- You can filter based on the database name

)

ADD TARGET package0.asynchronous\_file\_target

(

SET filename = 'C:\Temp\SQLHistory.xel', -- Specify the file location to store the events

metadatafile = 'C:\Temp\SQLHistory\_Metadata.xem' -- Optional metadata file

);

GO

-- Step 2: Start the Extended Event Session

ALTER EVENT SESSION [TrackSQLHistory] ON SERVER STATE = START;

GO

**Explanation:**

* **sqlserver.sql\_batch\_completed**: This event captures the completion of a SQL batch, including the SQL text and other relevant information.
* **ACTION**: This specifies the data you want to capture. In this example, we’re capturing the SQL text, client hostname, username, database name, and execution time.
* **WHERE**: Optional filter to capture only events related to a specific database (e.g., 'YourDatabase').
* **TARGET package0.asynchronous\_file\_target**: This specifies where to store the captured event data. In this case, we're storing it in a file (.xel), and also creating a metadata file (.xem).
* **ALTER EVENT SESSION**: This command starts the session to begin capturing events.

**Real-Time Data Capture**

Extended Events captures data in **real-time** as soon as the specified events occur. You can continue capturing data during the session and analyze it later. For example, in the case of the sql\_batch\_completed event, every time a batch finishes execution, the event is captured, and the data is written to the specified file or stored in memory if that option is selected.

**Step 3: Reading and Analyzing the Data**

After the Extended Event session has been running for a while, you can read the captured event data from the .xel file.

To query the .xel file and analyze the captured events, you can use the following T-SQL query:

-- Read events from the XEL file

SELECT

event\_data.event\_type AS EventName,

event\_data.sql\_text AS SQLText,

event\_data.client\_hostname AS ClientHostname,

event\_data.username AS UserName,

event\_data.database\_name AS DatabaseName,

event\_data.execution\_time AS ExecutionTime,

event\_data.cpu\_time AS CpuTime

FROM

sys.fn\_xe\_file\_target\_read\_file('C:\Temp\SQLHistory\*.xel', NULL, NULL, NULL) AS event\_data

ORDER BY

event\_data.execution\_time DESC;

**Explanation:**

* **sys.fn\_xe\_file\_target\_read\_file**: This function reads data from the .xel files that were generated by the Extended Event session. You can specify the path to the .xel file here.
* **Event Fields**: You can select specific columns of interest. In this example, I’m retrieving the SQL text, client hostname, username, database name, and execution time.

**Real-Time Analysis via sys.dm\_xe\_sessions**

If you prefer to analyze events in real-time without storing them in a file, you can query the active session data directly from the **sys.dm\_xe\_sessions** view. However, querying data in memory may only provide real-time access and might not persist after the session ends or the server restarts.

SELECT \*

FROM sys.dm\_xe\_sessions

WHERE name = 'TrackSQLHistory';

**Session Management: Start, Stop, and Drop**

* **Starting the Session**:  
  The session starts automatically when the session is created, or you can explicitly start it:

ALTER EVENT SESSION [TrackSQLHistory] ON SERVER STATE = START;

* **Stopping the Session**:  
  To stop the session (but keep the data in memory or on disk):

ALTER EVENT SESSION [TrackSQLHistory] ON SERVER STATE = STOP;

* **Dropping the Session**:  
  To permanently remove the session:

DROP EVENT SESSION [TrackSQLHistory] ON SERVER;

**Benefits of Extended Events for Real-Time Tracking:**

* **Low Overhead**: Unlike SQL Profiler, Extended Events is designed to have minimal performance impact.
* **Real-Time Tracking**: You can capture data as it happens in real-time, making it ideal for live monitoring.
* **Flexibility**: You can define exactly what events and data you want to capture, making it highly customizable.

**Conclusion:**

Extended Events in SQL Server is a powerful tool for capturing real-time event data. It's ideal for tracking SQL query execution, performance issues, and capturing detailed execution context information without the overhead of SQL Profiler. The data can be written to files or captured in memory, and you can analyze it in real-time or after the fact using T-SQL queries.

Let me know if you need more details on setting up, configuring, or analyzing Extended Events!