Monitoring Databases

This topic explains how to configure monitoring for databases.

Sections include:

- Why Monitor Databases?
- Configuring Event Monitor to Monitor a Database
- Notes

Why Monitor Databases?

The Event Monitor allows you to execute a SQL query on a database of your choice after each test executes within the test suite.

Although the DB Tool can be used for similar purposes, the Event Monitor Database mode is better suited for retrieving database rows when events occur as a result of your application logging messages into a database.

The Event Monitor is different than the DB Tool in a number of ways:

- A single Event Monitor in your test suite can execute database queries automatically after each test execution. This relieves you from having to add a DB Tool directly after each test.
- It allows delayed query execution (see the "Event polling delay after each test finishes execution" option under the Options tab). This is important for many logging databases because the logged entries may not reach the database in real time.
- It can consolidate the database entries into a single flow of events within a test suite, while the DB Tool gives you the flexibility to execute isolated and different queries at the desired points of your use case scenario.
- It helps you isolate the entries that were added to the database during test execution.

Configuring Event Monitor to Monitor a Database

1. Double-click the Event Monitor tool to open up the tool configuration panel.
2. In the **Event Source** tab, select **Database** as the platform.
3. With **Local** selected, enter the **Driver, URL, Username**, and **Password** for the database you want to query. For details on completing these fields, see Database Configuration Parameters.
4. (Optional) In the **Constraint SQL Query** field, enter a value that identifies the last logged value from the database before a test executes. Event Monitor expects that query to return a single value. Typically this would be a table key, an entry number, or a Timestamp.
    - Using a constraint query is useful when your logging database is cumulative and it is not cleaned/restored after each scenario executes. By executing this query before the test suite tests execute, the Event Monitor can distinguish the pre-existing entries from the new entries that will be logged into the database during test execution.
    - Such queries would typically use the `SQL MAX` function. For example, the query `select max(MESSAGE_TIMESTAMP) from MESSAGE_LOG` assumes that you have a table named `MESSAGE_LOG` that contains a column named `MESSAGE_TIMESTAMP` of type `Timestamp`. It will return a single value representing the latest message entry currently present in that database. Event Monitor will execute that query first and keep that timestamp value.
5. In the **Event SQL Query** field, specify the SQL for retrieving the log or event entry from the database. For example, such a query might look like:
    ```
    select * from MESSAGE_LOG
    where MESSAGE_TIMESTAMP > $[CONSTRAINT]
    order by MESSAGE_TIMESTAMP DESC
    ```
    - Note that $[CONSTRAINT] is a special SOAtest variable. It tells the Event Monitor to use the value it received from the first constraint query (described in the previous step) and automatically provide it in the event query. The event query executes after test suite execution completes (and after the delay specified in the Event Monitor’s Options tab). It retrieves the rows that were added to the database after the test executed.
    - Use of $[CONSTRAINT] in event queries is not required.
6. In the **Options** tab, modify settings as needed.
    - **Clear the event viewer before each event monitor run** determines whether SOAtest automatically clears the Event Monitor event view (both text and graphical) whenever Event Monitor starts monitoring.
    - **Include test execution events in the XML event output** specifies whether the Event Viewer tab and XML output display show only the monitored messages and events, or if they also indicate when each test started and completed. Enabling this option is helpful if you have multiple tests in the test suite and you want to better identify the events and correlate them to your test executions.
    - **Wrap monitored messages with CDATA to ensure well-formedness of the XML event output** should be disabled if you expect the monitored events’ message content to be well-formed XML. This will make the messages inside the events accessible via XPaths, allowing the message contents to be extracted by XML Transformer or validated with XML Assertor tools.
        - If the message contents are not necessarily XML, you should enable this option to ensure that the XML output of the Event Monitor tool (i.e. the XML Event Output for chaining tools to the Event Monitor, not what is shown under the Event Viewer) is well-formed XML by escaping all the message contents. This will make the content of these messages inaccessible by XPath since the message technically becomes just string content for the parent element.
        - Note that the Diff tool’s XML mode supports string content that is XML. In other words, no matter which option you select here, the Diff tool will still be able to diff the messages as XML, including the ability to use XPaths for ignoring values.
    - **Maximum monitor execution duration** specifies the point at which the test should timeout—in case another test in the test suite hangs, or if no other tests are being run (e.g., if you execute the Event Monitor test apart from the test suite, then use a custom application to send messages to system).
    - **Event polling delay after each test finishes execution (milliseconds)** specifies how long Event Monitor waits between the time the test ends and the time it retrieves the events.

Notes

- The **Event Viewer** tab will display each row retrieved by the Event SQL query as a box in the event sequence flow. Double-clicking the box opens a dialog with data details.
- You can chain XML tools to a database Event Monitor. Since the database rows are output in XML, they can be diffed, validated with XML Assertor, etc.