Splunk and Log parser

Splunk users know how to translate SQL queries to Splunk search queries.  We use search operators for this. For non-Splunk users we could provide Splunk as a data store with log parser as a SQL interface.  Therefore, we will look into providing Splunk searchable data as a COM input to log parser. A COM input simply implements a few methods for the log parser and abstracts the data store. These methods are :

OpenInput: Opens your data source and sets up any initial environment settings

GetFieldCount: returns the number of fields that the plugin provides

GetFieldName: returns the name of a specified field

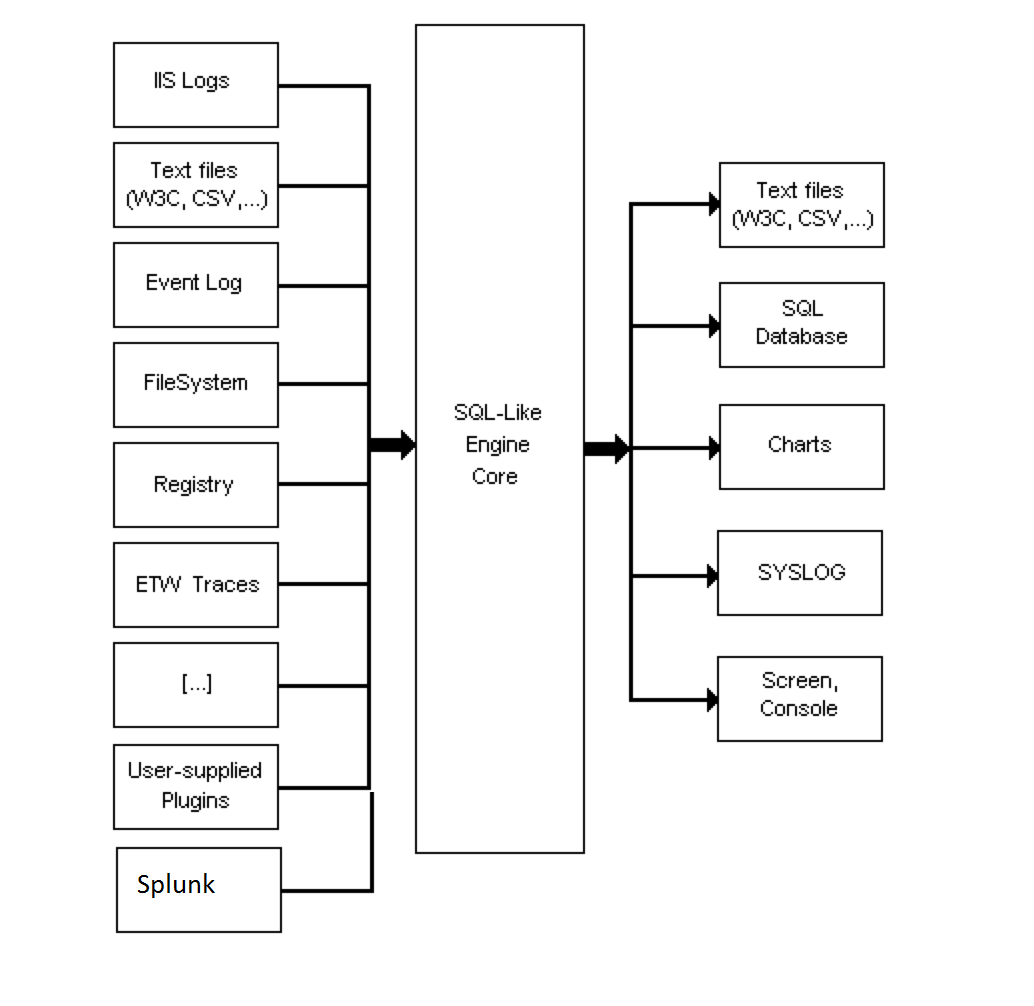
GetFieldType : returns the datatype of a specified field

GetValue : returns the value of a specified field

ReadRecord : reads the next record from your data source

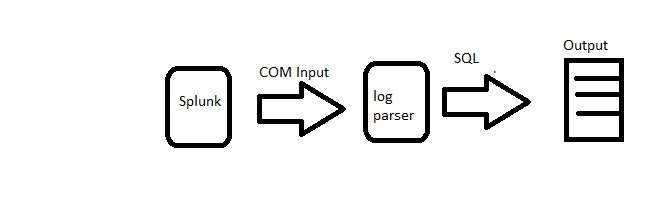
CloseInput: closes the data source and cleans up any environment settings

Architecture:



Detail:

Splunk acts as data store for COM input to log parser which can then be queried in SQL for the desired output.



Log Parser SQL queries example:

The queries are issued to logparser in the following syntax. The last term denotes the output to console. Other options are available.

logparser "(sql clause)" -i:COM -iProgID:BAR -rtp:-1

where (sql clause) is like select \* from FOO where FOO is a datastore that makes sense to the COM input plugin denoted by BAR.

Top 10 items retrieved:

SELECT TOP 10 cs-uri-stem as Url, COUNT(cs-uri-stem) AS Hits FROM FOO GROUP BY cs-uri-stem ORDER BY Hits DESC

Top 10 slowest items:

SELECT TOP 10 cs-uri-stem AS Url, MIN(time-taken) as [Min], AVG(time-taken) AS [Avg], max(time-taken) AS [Max], count(time-taken) AS Hits FROM FOO WHERE time-taken < 120000 GROUP BY Url ORDER BY [Avg] DESC

All Unique Urls retrieved:

SELECT DISTINCT TO\_LOWERCASE(cs-uri-stem) AS Url, Count(\*) AS Hits FROM FOO WHERE sc-status=200 GROUP BY Url ORDER BY Url

HTTP errors per hour:

SELECT date, QUANTIZE(time, 3600) AS Hour, sc-status AS Status, COUNT(\*) AS Errors FROM FOO WHERE (sc-status >= 400) GROUP BY date, hour, sc-status HAVING (Errors > 25) ORDER BY Errors DESC

HTTP errors ordered by Url and Status:

SELECT cs-uri-stem AS Url, sc-status AS Status, COUNT(\*) AS Errors FROM FOO WHERE (sc-status >= 400) GROUP BY Url, Status ORDER BY Errors DESC

Win32 error codes by total and page:

SELECT cs-uri-stem AS Url, WIN32\_ERROR\_DESCRIPTION(sc-win32-status) AS Error, Count(\*) AS Total FROM FOO WHERE (sc-win32-status > 0) GROUP BY Url, Error ORDER BY Total DESC

HTTP methods (GET, POST, etc) used per Url:

SELECT cs-uri-stem AS Url, cs-method AS Method, Count(\*) AS Total FROM FOO WHERE (sc-status < 400 or sc-status >= 500) GROUP BY Url, Method ORDER BY Url, Method

Bytes sent from the server:

SELECT cs-uri-stem AS Url, Count(\*) AS Hits, AVG(sc-bytes) AS Avg, Max(sc-bytes) AS Max, Min(sc-bytes) AS Min, Sum(sc-bytes) AS TotalBytes FROM FOO GROUP BY cs-uri-stem HAVING (Hits > 100) ORDER BY [Avg] DESC

Bytes sent from the client:

SELECT cs-uri-stem AS Url, Count(\*) AS Hits, AVG(cs-bytes) AS Avg, Max(cs-bytes) AS Max, Min(cs-bytes) AS Min, Sum(cs-bytes) AS TotalBytes FROM FOO GROUP BY Url HAVING (Hits > 100) ORDER BY [Avg] DESC

SQL to Splunk queries:

From Splunk documentation:

|  |  |  |
| --- | --- | --- |
| **DB Concept** | **Splunk Concept** | **Notes** |
| SQL query | Splunk search | A Splunk search retrieves indexed data and can perform transforming and reporting operations. Results from one search can be "piped", or transferred, from command to command, to filter, modify, reorder, and group your results. |
| table/view | search results | Search results can be thought of as a database view, a dynamically generated table of rows, with columns. |
| index | index | All values and fields are indexed in Splunk, so there is no need to manually add, update, drop, or even think about indexing columns. Everything can be quickly retrieved automatically. |
| row | result/event | A result in Splunk is a list of field (i.e., column) values, corresponding to a table row. An event is a result that has a timestamp and raw text. Typically an event is a record from a log file, such as:  173.26.34.223 - - [01/Jul/2009:12:05:27 -0700] "GET /trade/app?action=logout HTTP/1.1" 200 2953 |
| column | field | Fields in Splunk are dynamically returned from a search, meaning that one search might return a set of fields, while another search might return another set. After teaching Splunk how to extract out more fields from the raw underlying data, the same search will return more fields that it previously did. Fields in Splunk are not tied to a datatype. |
| database/schema | index/app | In Splunk, an index is a collection of data, somewhat like a database has a collection of tables. Domain knowledge of that data, how to extract it, what reports to run, etc, are stored in a Splunk app. |

Here’s a translation of common SQL commands to native Splunk search language queries.

|  |  |  |
| --- | --- | --- |
| **SQL command** | **SQL example** | **Splunk example** |
| SELECT \* | SELECT \*  FROM mytable | source=mytable |
| WHERE | SELECT \*  FROM mytable  WHERE mycolumn=5 | source=mytable mycolumn=5 |
| SELECT | SELECT mycolumn1, mycolumn2  FROM mytable | source=mytable  | FIELDS mycolumn1, mycolumn2 |
| AND/OR | SELECT \*  FROM mytable  WHERE (mycolumn1="true" OR mycolumn2="red") AND mycolumn3="blue" | source=mytable  AND (mycolumn1="true" OR mycolumn2="red")  AND mycolumn3="blue" |
| AS (alias) | SELECT mycolumn AS column\_alias  FROM mytable | source=mytable  | RENAME mycolumn as column\_alias  | FIELDS column\_alias |
| BETWEEN | SELECT \*  FROM mytable  WHERE mycolumn  BETWEEN 1 AND 5 | source=mytable mycolumn>=1 mycolumn<=5 |
| GROUP BY | SELECT mycolumn, avg(mycolumn)  FROM mytable  WHERE mycolumn=value  GROUP BY mycolumn | source=mytable mycolumn=value  | STATS avg(mycolumn) BY mycolumn  | FIELDS mycolumn, avg(mycolumn) |
| HAVING | SELECT mycolumn, avg(mycolumn)  FROM mytable  WHERE mycolumn=value  GROUP BY mycolumn  HAVING avg(mycolumn)=value | source=mytable mycolumn=value  | STATS avg(mycolumn) BY mycolumn  | SEARCH avg(mycolumn)=value  | FIELDS mycolumn, avg(mycolumn) |
| LIKE | SELECT \*  FROM mytable  WHERE mycolumn LIKE "%some text%" | source=mytable mycolumn="\*some text\*"  **Note:** The most common search usage in Splunk is actually something that is nearly impossible in SQL -- to search all fields for a substring. The following search will return all rows that contain "some text" anywhere:  source=mytable "some text" |
| ORDER BY | SELECT \*  FROM mytable  ORDER BY mycolumn desc | source=mytable  | SORT -mycolumn |
| SELECT DISTINCT | SELECT DISTINCT mycolumn1, mycolumn2  FROM mytable | source=mytable  | DEDUP mycolumn1  | FIELDS mycolumn1, mycolumn2 |
| SELECT TOP | SELECT TOP 5 mycolumn1, mycolumn2  FROM mytable | source=mytable  | TOP mycolumn1, mycolumn2 |
| INNER JOIN | SELECT \*  FROM mytable1  INNER JOIN mytable2  ON mytable1.mycolumn=mytable2.mycolumn | source=mytable1  | JOIN type=inner mycolumn [ SEARCH source=mytable2 ]  **Note:** There are two other methods to do a join:   * Use the lookup command to add fields from an external table:   ... | LOOKUP myvaluelookup mycolumn OUTPUT myoutputcolumn   * Use a subsearch:   source=mytable1 [  SEARCH source=mytable2 mycolumn2=myvalue  | FIELDS mycolumn2  ] |
| LEFT (OUTER) JOIN | SELECT \*  FROM mytable1  LEFT JOIN mytable2  ON mytable1.mycolumn=mytable2.mycolumn | source=mytable1  | JOIN type=left mycolumn [ SEARCH source=mytable2 ] |
| SELECT INTO | SELECT \*  INTO new\_mytable IN mydb2  FROM old\_mytable | source=old\_mytable  | EVAL source=new\_mytable  | COLLECT index=mydb2  **Note:** COLLECT is typically used to store expensively calculated fields back into Splunk so that future access is much faster. This current example is atypical but shown for comparison with SQL's command. source will be renamed orig\_source |
| TRUNCATE TABLE | TRUNCATE TABLE mytable | source=mytable  | DELETE |
| INSERT INTO | INSERT INTO mytable  VALUES (value1, value2, value3,....) | **Note:** see SELECT INTO. Individual records are not added via the search language, but can be added via the API if need be. |
| UNION | SELECT mycolumn  FROM mytable1  UNION  SELECT mycolumn FROM mytable2 | source=mytable1  | APPEND [ SEARCH source=mytable2]  | DEDUP mycolumn |
| UNION ALL | SELECT \*  FROM mytable1  UNION ALL  SELECT \* FROM mytable2 | source=mytable1  | APPEND [ SEARCH source=mytable2] |
| DELETE | DELETE FROM mytable  WHERE mycolumn=5 | source=mytable1 mycolumn=5  | DELETE |
| UPDATE | UPDATE mytable  SET column1=value, column2=value,...  WHERE some\_column=some\_value | **Note:** There are a few things to think about when updating records in Splunk. First, you can just add the new values into Splunk (see INSERT INTO) and not worry about deleting the old values, because Splunk always returns the most recent results first. Second, on retrieval, you can always de-duplicate the results to ensure only the latest values are used (see SELECT DISTINCT). Finally, you can actually delete the old records (see DELETE). |

Together splunk and log parser brings the power of splunk to log parser users without requiring them to know about Splunk search commands. At the same time, they have the choice to search the Splunk indexes directly. The ability to use SQL makes Splunk more common and inviting to windows users.