**Splunk SPL for SQL users**

<http://docs.splunk.com/Documentation/Splunk/latest/SearchReference/SQLtoSplunk>

This is not a perfect mapping between SQL and Splunk Search Processing Language (SPL), but if you are familiar with SQL, this quick comparison might be helpful as a jump-start into using the search commands.

**Concepts**

The Splunk platform does not store data in a conventional database. Rather, it stores data in a distributed, non-relational, semi-structured database with an implicit time dimension. Relational databases require that all table columns be defined up-front and they do not automatically scale by just plugging in new hardware. However, there are analogues to many of the concepts in the database world.

|  |  |  |
| --- | --- | --- |
| **Database 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 by Splunk software, 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 a Splunk search is a list of fields (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 are returned dynamically from a search, meaning that one search might return a set of fields, while another search might return another set. After teaching Splunk software how to extract more fields from the raw underlying data, the same search will return more fields that it previously did. Fields are not tied to a datatype. |
| database/schema | index/app | A Splunk 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 application. |

**From SQL to Splunk SPL**

The examples below use the value of the Splunk field "source" as a proxy for "table". In Splunk software, "source" is the name of the file, stream, or other input from which a particular piece of data originates, for example /var/log/messagesor UDP:514.

When translating from any language to another, often the translation is longer because of idioms in the original language. Some of the Splunk search examples shown below could be more concise, but for parallelism and clarity, the table and field names are kept the same from the sql. Also, searches rarely need the FIELDS command to filter out columns as the user interface provides a more convenient method; and you never have to use "AND" in boolean searches, as they are implied between terms.

|  |  |  |
| --- | --- | --- |
| **SQL command** | **SQL example** | **Splunk Enterprise 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)  Several commands use a by-clause to group information, including [chart](http://docs.splunk.com/Documentation/Splunk/6.4.1/SearchReference/Chart), [rare](http://docs.splunk.com/Documentation/Splunk/6.4.1/SearchReference/Rare), [sort](http://docs.splunk.com/Documentation/Splunk/6.4.1/SearchReference/Sort),[stats](http://docs.splunk.com/Documentation/Splunk/6.4.1/SearchReference/Stats), and [timechart](http://docs.splunk.com/Documentation/Splunk/6.4.1/SearchReference/Timechart). |
| 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 in Splunk SPL is nearly impossible in SQL: to search all fields for a substring. The following search returns 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 Enterprise so that future access is much faster. This current example is atypical but shown for comparison to the SQL command. The 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 Enterprise. First, you can just add the new values into Splunk Enterprise (see INSERT INTO) and not worry about deleting the old values, because Splunk Enterprise 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). |