Splunk SPL for SQL users

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.

From SQL to Splunk SPL

SQL is designed to search relational database tables which are comprised of columns. SPL is designed to search events, which are comprised of fields. In SQL, you often see examples that use "mytable" and "mycolumn". In SPL, you will see examples that refer to "fields". In these examples, the "source" field is used 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/messages` or `UDP:514`.

NOTE: 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 and more efficient, but for parallelism and clarity, the SPL table and field names are kept the same as the SQL example.

- SPL searches rarely need the FIELDS command to filter out columns because the user interface provides a more convenient method for filtering. The FIELDS command is used in the SPL examples for parallelism.
- With SPL, you never have to use the AND operator in Boolean searches, because AND is implied between terms. However when you use the AND or OR operators, they must be specified in uppercase.
- SPL commands do not need to be specified in uppercase. In the these SPL examples, the commands are specified in uppercase for easier identification and clarity.
- Although some SPL commands loosely correspond to specific SQL commands as shown in the following table, your SPL searches might not produce the desired results if you "think in SQL." For this reason, avoid directly translating from SQL to SPL when you design your searches. See About the search language in the *Search Manual* for an overview of SPL.

SQL command	SQL example	Splunk SPL 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"`
		
		Note: The AND operator is implied in SPL and does not need to be specified. For this example you could also use:
		
		`source=mytable`
		`(mycolumn1="true"`
		`  OR mycolumn2="red")`
		`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, rare, sort, stats, and 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 SPL 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`
		
		In SPL you use a negative sign (-) in front of a field name to sort in descending order.

SELECT DISTINCT	`SELECT DISTINCT` `  mycolumn1,` `mycolumn2` `FROM mytable`	`source=mytable` `| DEDUP mycolumn1, mycolumn2` `| FIELDS mycolumn1, mycolumn2`
SELECT TOP	`SELECT TOP(5)` `mycolum1,` `mycolum2` `FROM mytable1` `WHERE mycolum3 =` `"bar"` `ORDER BY mycolum1` `mycolum2`	`Source=mytable1 mycolum3="bar"` `| FIELDS mycolum1 mycolum2` `| SORT mycolum1 mycolum2` `| HEAD 5`
INNER JOIN	`SELECT *` `FROM mytable1` `INNER JOIN mytable2` `ON` `mytable1.mycolumn=` `  mytable2.mycolumn`	`index=myIndex1 OR index=myIndex2` `| stats values(*) AS * BY myField` Note: There are two other methods to join tables: • 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]` If the columns that you want to join on have different names, use the `rename` command to rename one of the columns. For example, to rename the column in mytable2:

		`source=mytable1`
		`| JOIN type=inner mycolumn`
		`  [ SEARCH source=mytable2`
		`    | RENAME mycolumn2`
		`    AS mycolumn]`
		
		To rename the column in myindex1:
		
		`index=myIndex1 OR index=myIndex2`
		`| rename myfield1 as myField`
		`| stats values(*) AS * BY myField`
		
		You can rename a column regardless of whether you use the search command, a lookup, or a subsearch.

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 your Splunk deployment 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 to your Splunk deployment (see INSERT INTO) and not worry about deleting the old values, because Splunk software 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).

See also

- Understanding SPL syntax