Filter Language – HBASE-4176

Anirudh Todi (anirudhtodi@berkeley.edu)

August 8, 2011

Use Case:

Currently, to use any of the filters, one has to explicitly add a scanner for the filter in the Thrift API making it messy and long. Thus, I am trying to add support for all the filters in a clean way.

Solution

The user specifies a filter via a string. The string is parsed on the server to construct the filter

General Filter String Syntax:

A simple filter expression is expressed as: *"FilterName (argument, argument, ... , argument)"*

You must specify the name of the filter followed by the argument list in parenthesis. Commas separate the individual arguments

If the argument represents a string, it should be enclosed in single quotes.

If it represents a boolean, an integer or a comparison operator like <, >, != etc. it should not be enclosed in quotes

The filter name must be one word. All ASCII characters are allowed except for whitespace, single quotes and parenthesis.

The filter's arguments can contain any ASCII character. **If single quotes are present in the argument, they must be escaped by a preceding single quote**

Compound Filters and Operators:

Currently, two binary operators – AND/OR and two unary operators – WHILE/SKIP are supported.

Note: the operators are all in uppercase

AND – as the name suggests, if this operator is used, the key-value must pass both the filters

OR – as the name suggests, if this operator is used, the key-value must pass at least one of the filters

SKIP – For a particular row, if any of the key-values don't pass the filter condition, the entire row is skipped

WHILE - For a particular row, it continues to emit key-values until a key-value is reached that fails the filter condition

Compound Filters: Using these operators, a hierarchy of filters can be created. For example:

"(Filter1 AND Filter2) OR (Filter3 AND Filter4)"

Order of Evaluation

Parenthesis have the highest precedence
The SKIP and WHILE operators are next and have the same precedence.
The AND operator has the next highest precedence followed by the OR operator

For example:
A filter string of the form: *“Filter1 AND Filter2 OR Filter3”* will be evaluated as: *“(Filter1 AND Filter2) OR Filter3”*
A filter string of the form: *“Filter1 AND SKIP Filter2 OR Filter3”* will be evaluated as: *“(Filter1 AND (SKIP Filter2)) OR Filter3”*

Compare Operator and Comparators

Compare Operator

A compare operator can be any of the following:

1. LESS (<)
2. LESS_OR_EQUAL (<=)
3. EQUAL (=)
4. NOT_EQUAL (!=)
5. GREATER_OR_EQUAL (>=)
6. GREATER (>)
7. NO_OP (no operation)

The client should use the symbols (<, <=, =, !=, >, >=) to express compare operators.

Comparators

A comparator can be any of the following:

1. **BinaryComparator** - This lexicographically compares against the specified byte array using Bytes.compareTo(byte[], byte[])
2. **BinaryPrefixComparator** - This lexicographically compares against a specified byte array. It only compares up to the length of this byte array.
3. **RegexStringComparator** - This compares against the specified byte array using the given regular expression. *Only EQUAL and NOT_EQUAL comparisons are valid with this comparator*
4. **SubStringComparator** - This tests if the given substring appears in a specified byte array. The comparison is case insensitive. *Only EQUAL and NOT_EQUAL comparisons are valid with this comparator*

The general syntax of a comparator is: ComparatorType:ComparatorValue

The ComparatorType for the various comparators is as follows:

1. **BinaryComparator** - binary
2. **BinaryPrefixComparator** - binaryprefix
3. **RegexStringComparator** - regexstring
4. **SubStringComparator** - substring

The ComparatorValue can be any value.

Example1: >, 'binary:abc' will match everything that is lexicographically greater than "abc"

Example2: =, 'binaryprefix:abc' will match everything whose first 3 characters are lexicographically equal to "abc"
Example3: !=, 'regexstring:ab*yz' will match everything that doesn't begin with "ab" and ends with "yz"
Example4: =, 'substring:abc123' will match everything that begins with the substring "abc123"

Example PHP Client Program that uses Filter Strings

```
<? $_SERVER['PHP_ROOT'] = realpath(dirname(__FILE__).'/..');
require_once $_SERVER['PHP_ROOT'].'/flib/__flib.php';
flib_init(FLIB_CONTEXT_SCRIPT);
require_module('storage/hbase');
$hbase = new HBase('<server_name_running_thrift_server>', <port on which thrift server is running>); $hbase->open();
$client = $hbase->getClient();

echo "Sample Program which uses the Filter Language";
$result = $client->scannerOpenWithFilterString('table_name', "(PrefixFilter ('row2') AND (QualifierFilter (>=,
'binary:xyz'))) AND (TimestampsFilter ( 123, 456))");
$to_print = $client->scannerGetList($result,1);

while ($to_print) {
    print_r($to_print);
    $to_print = $client->scannerGetList($result,1);
 }

$client->scannerClose($result); ?>
```

Example Filter Strings

- "PrefixFilter ('Row') AND PageFilter (1) AND FirstKeyOnlyFilter ()"

  will return all key-value pairs that match the following conditions:
  1) The row containing the key-value should have prefix "Row"
  2) The key-value must be located in the first row of the table
  3) The key-value pair must be the first key-value in the row

- "(RowFilter (=, 'binary:Row 1') AND TimeStampsFilter (74689, 89734)) OR ColumnRangeFilter ('abc', true, 'xyz', false))"

  will return all key-value pairs that match both the following conditions:
  1) The key-value is in a row having row key "Row 1"
  2) The key-value must have a timestamp of either 74689 or 89734
  Or it must match the following condition:
  1) The key-value pair must be in a column that is lexicographically >= abc and < xyz

- "SKIP ValueFilter (0)"

  will skip the entire row if any of the values in the row is not 0

Individual Filter Syntax

- **KeyOnlyFilter**

  Description: This filter doesn’t take any arguments. It returns only the key component of each key-value.

  Syntax: KeyOnlyFilter ()

  Example: "KeyOnlyFilter ()"

- **FirstKeyOnlyFilter**

  Description: This filter doesn’t take any arguments. It returns only the first key-value from each row.

  Syntax: FirstKeyOnlyFilter ()

  Example: "FirstKeyOnlyFilter ()"

- **PrefixFilter**

  Description: This filter takes one argument – a prefix of a row key. It returns only those key-values present in a row that starts with the specified row prefix

  Syntax: PrefixFilter (‘<row_prefix>’)

  Example: "PrefixFilter (‘Row’)"

- **ColumnPrefixFilter**

  Description: This filter takes one argument – a column prefix. It returns only those key-values present in a column that starts with the specified column prefix. The column prefix must be of the form:
  - “qualifier”

  Syntax: ColumnPrefixFilter (‘<column_prefix>’)

  Example: "ColumnPrefixFilter (‘Col’)"

- **MultipleColumnPrefixFilter**

  Description: This filter takes a list of column prefixes. It returns key-values that are present in a column that starts with any of the specified column prefixes. The column prefix must be of the form:
  - “qualifier” in which case it looks for key-values in columns whose qualifiers have the specified prefix

  Syntax: MultipleColumnPrefixFilter (‘<column_prefix>’, ‘<column_prefix>’, …, ‘<column_prefix>’)

  Example: "MultipleColumnPrefixFilter (‘Col1’, ‘Col2’)"

- **<u>ColumnCountGetFilter</u>**

  Description: This filter takes one argument – a limit. It returns the first limit number of columns in the table

  Syntax: ColumnCountGetFilter (‘<limit>’)

  Example: "ColumnCountGetFilter (4)"

- **<u>PageFilter</u>**

  Description: This filter takes one argument – a page size. It returns page size number of rows from the table.

  Syntax: PageFilter (‘<page_size>’)

  Example: "PageFilter (2)"

- **<u>ColumnPaginationFilter</u>**

  Description: This filter takes two arguments – a limit and offset. It returns limit number of columns after offset number of columns. It does this for all the rows

  Syntax: ColumnPaginationFilter (‘<limit>’, ‘<offest>’)

  Example: "ColumnPaginationFilter (3, 5)"

- **<u>InclusiveStopFilter</u>**

  Description: This filter takes one argument – a row key on which to stop scanning. It returns all key-values present in rows up to and including the specified row

  Syntax: InclusiveStopFilter (‘<stop_row_key>’)

  Example: "InclusiveStopFilter (‘Row2’)”

- **<u>TimeStampsFilter</u>**

  Description: This filter takes a list of timestamps. It returns those key-values whose timestamps matches any of the specified timestamps

  Syntax: TimeStampsFilter (<timestamp>, <timestamp>, ... ,<timestamp>)

  Example: "TimeStampsFilter (5985489, 48895495, 58489845945)"

- **<u>RowFilter</u>**

  Description: This filter takes a compare operator and a comparator. It compares each row key with the comparator using the compare operator and if the comparison returns true, it returns all the key-values in that row

Syntax: RowFilter (<compareOp>, ‘<row_comparator>’)

Example: "RowFilter (<=, ‘xyz)"

- **<u>Family Filter</u>**

  Description: This filter takes a compare operator and a comparator. It compares each family name with the comparator using the compare operator and if the comparison returns true, it returns all the key-values in that family

  Syntax: FamilyFilter (<compareOp>, ‘<family_comparator>’)

  Example: "FamilyFilter (>=, ‘FamilyB’)"

- **<u>QualifierFilter</u>**

  Description: This filter takes a compare operator and a comparator. It compares each qualifier name with the comparator using the compare operator and if the comparison returns true, it returns all the key-values in that column

  Syntax: QualifierFilter (<compareOp>, ‘<qualifier_comparator>’)

  Example: "QualifierFilter (=, ‘Column1’)"

- **<u>ValueFilter</u>**

  Description: This filter takes a compare operator and a comparator. It compares each value with the comparator using the compare operator and if the comparison returns true, it returns that key-value

  Syntax: ValueFilter (<compareOp>, ‘<value_comparator>’)

  Example: "ValueFilter (!=, ‘Value’)"

- **<u>DependentColumnFilter</u>**

  Description: This filter takes two arguments – a family and a qualifier. It tries to locate this column in each row and returns all key-values in that row that have the same timestamp. If the row doesn’t contain the specified column – none of the key-values in that row will be returned.

  The filter can also take an optional boolean argument – dropDependentColumn. If set to true, the column we were depending on doesn’t get returned.

  The filter can also take two more additional optional arguments – a compare operator and a value comparator, which are further checks in addition to the family and qualifier. If the dependent column is found, its value should also pass the value check and then only is its timestamp taken into consideration

  Syntax: DependentColumnFilter (‘<family>’, ‘<qualifier>’, <boolean>, <compare operator>, ‘<value comparator’)

  DependentColumnFilter (‘<family>’, ‘<qualifier>’, <boolean>)

  DependentColumnFilter (‘<family>’, ‘<qualifier>’)

  Example: "DependentColumnFilter (‘conf’, ‘blacklist’, false, >=, ‘zebra’)"

"DependentColumnFilter (‘conf’, ‘blacklist’, true)"

"DependentColumnFilter (‘conf’, ‘blacklist’)"

- **SingleColumnValueFilter**

  Description: This filter takes a column family, a qualifier, a compare operator and a comparator. If the specified column is not found – all the columns of that row will be emitted. If the column is found and the comparison with the comparator returns true, all the columns of the row will be emitted. If the condition fails, the row will not be emitted.

  This filter also takes two additional optional boolean arguments – filterIfColumnMissing and setLatestVersionOnly

  If the filterIfColumnMissing flag is set to true the columns of the row will not be emitted if the specified column to check is not found in the row. The default value is false.

  If the setLatestVersionOnly flag is set to false, it will test previous versions (timestamps) too. The default value is true.

  These flags are optional and if you must set neither or both

  Syntax: SingleColumnValueFilter (<compare operator>, ‘<comparator>’, ‘<family>’, ‘<qualifier>’, <filterIfColumnMissing_boolean>, <latest_version_boolean>)

  Syntax: SingleColumnValueFilter (<compare operator>, ‘<comparator>’, ‘<family>’, ‘<qualifier>’)

  Example: "SingleColumnValueFilter (<=, ‘abc’, ‘FamilyA’, ‘Column1’, true, false)"

  Example: "SingleColumnValueFilter (<=, ‘abc’, ‘FamilyA’, ‘Column1’)"

- **SingleColumnValueExcludeFilter**

  Description: This filter takes the same arguments and behaves same as SingleColumnValueFilter – however, if the column is found and the condition passes, all the columns of the row will be emitted except for the tested column value.

  Syntax: SingleColumnValueExcludeFilter (<compare operators> <comparator> <family> <qualifier> <latest_version_boolean> <filterIfColumnMissing_boolean>)

  Syntax: SingleColumnValueExcludeFilter (<compare operator> <comparator> <family> <qualifier>)

  Example: "SingleColumnValueExcludeFilter (‘<=’, ‘abc’, ‘FamilyA’, ‘Column1’, ‘false’, ‘true’)"

  Example: "SingleColumnValueExcludeFilter (‘<=’, ‘abc’, ‘FamilyA’, ‘Column1’)"

- **ColumnRangeFilter**

  Description: This filter is used for selecting only those keys with columns that are between minColumn and maxColumn. It also takes two boolean variables to indicate whether to include the minColumn and maxColumn or not. If you don’t want to set the minColumn or the maxColumn – you can pass in an empty argument.

  Syntax: ColumnRangeFilter (‘<minColumn >’, <minColumnInclusive_bool>, ‘<maxColumn>’, <maxColumnInclusive_bool>)

  Example: "ColumnRangeFilter (‘abc’, true, ‘xyz’, false)"