MQL: Minimalistic Query Language

Programming Language Proposal

Yiqu Liu | yl4617
Pitchapa Chantanapongvanij | pc2806
Peihan Liu | pl2804
Daisy Wang | yw3753

Overview of MQL:

Minimalistic Query Language (MQL) is a static and imperative programming language that is used to perform commands to process data extracted from a user-provided source table. In comparison with SQL, we envision MQL to be capable of performing complex processing in a simple way. MQL will use a CSV library (ocaml-csv) of OCaml to adequately fulfill query tasks. MQL will have simple and readable syntax, it is designed for everyone to understand easily regardless of programming background.
**This proposal outlines how we expect MQL to be, and may be subject to changes in the future.*

What sorts of programs are meant to be written in MQL:

We aim to make MQL an imperative programming language instead of a declarative query language. It will have the major characteristics of imperative languages, for example, MQL will parse code in a step-wise manner, each statement will be executed separately and sequentially. In this case, MQL makes database querying more flexible.

- **Queries Require string processing and comparisons**
  Users will be able to easily query the provided source table to look up the specific tale entries given type string conditions. More specifically, MQL will be able to do substring pattern matching as well as exact pattern matching.
- **Queries Require User-defined functions**
  User-defined functions in MQL have a very similar syntax to other imperative languages. This enables users to define functions that involve loops, if-else statements to communicate with a database. Therefore, making it more approachable to programmers.
- **Queries Require Complex Calculations**
  Users will be able to carry out involved nested queries or queries with multiple conditions in a straightforward manner with MQL's simplified syntax. For example, finding specific entries in a table with multiple conditions.

Parts of MQL

---

1. **Imports**

Most of the operations in MQL are provided in the standard library, import statements in MQL are used for the purpose of importing database tables from the local host that users could access using the current script.

```
#import table.csv
#import table2.sql
```

MQL also supports renaming in import statements

```
# import table.csv as t
```

2. Operations

Operation	Description
++	Increment
--	Decrement
==	Equal
!=	Not equal
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
&&	And
||	Or

3. Data Types

Data type	Operator	Example
Table	Select Where Summarize	Table a = row[]
int	All	int x = 3
float	All	float x = 2.7

boolean	Logical, Comparison	bool x = true
array	Comparison	Int [3] x = [1,2,3]
string	Concat	str x = ‘Hello, MQL’

4. **Control Flow**

Control Flow	**Description**	**Example**
/* */	Multi-line Comment	/*comment1 comment2*/
//	Single line comment	//comment
print	Print statement	`print(x);`

if/else if/else	Conditional statement	`if(day == “Sun”){`
		`  return True;`
		`}`

while loop	Iterative loop	`let calendars = [calendar1, calendar2, calendar3];`
		`let i = 0;`
		`while(i < 3){`
		
		`print(holiday_count(calendars[i]));`
		`i++;`
		`}`

`let myfunc = (column_name: x){`
`  return x;`
`};`
	Functions	`let holiday_count = (calendar: Table){`
		`  return calendar.where(holiday == True).count();`
		`};`
		
		`print(holiday_count(Calendar));`

5. **Comments**

   MQL will support comments for programmers to document the code.

   Example of one-line comment in MQL:

   `//single-line comment example`

   Example of multi-line comments in MQL:

```
/*multi-line comments example
      Multi-line comments example */
```

6. **Indentation**
   Indentation in MQL programs is for formatting purposes only. MQL compiler will NOT parse script based on indentation. However, users should indent their programs by differentiating 1 or 2 spaces between sequential lines and stick with this formatting throughout the entire program.

How to implement MQL:

In our language, the operations are processed in a pipeline. Every operation takes a table as an input and returns a table as a result. Generally, after every operation, an intermediate table is generated as a result of the previous step, which is then fed as the input to the next operation.

```
Table  <-source table

Table
.where(column1 == 'something')  <- filtered source table

Table
.where(column1 == 'something')
.top(10)  <- top 10 results of the filter source table
```

Features:

- Support basic database operation, including select, where, join, summarize and extend new columns.
- Allow users to create temporary variables to store a table in the memory and use it later.
  Example:

```
SELECT DISTINCT c.Id, BuildingName = CONCAT(b.Name, ' at ',
b.Campus)
FROM Courses as c
INNER JOIN (
      SELECT * FROM Building WHERE Campus = 'MorningSide'
) as b on c.Id = b.courseId
WHERE Name = 'PLT' AND Professor = 'Stephen'
```

  Could be expressed in MQL as:

```
let b = Buildings.where(Campus == 'MorningSide');
Courses
.where(Name == 'PLT' && Professor == 'Stephen')
.join("inner", availableBuildings, Id, courseId)
```

```
.extend("BuildingName", b.Name + " at " + Campus)
.distinct("Id", "BuildingName");
```

- MQL allows users to build user-defined functions.
  Example:
  ```
  let checkWeekend = (day: varchar) {
        if(day == "Sun" || day == "Sat") return True;
        return False;
  };
  Calendar
  .extend("IsWeekend", checkWeekend(Day)); //comments
  ```

- MQL will support advanced string processing methods, such as concat(), substring(), begin_with(), charAt() and regex(), all of which will be stored in mql-strings. Basically, users can use string manipulation methods just like what they can do with Python or Java.
- MQL has some syntactic sugar to make the language 'sweeter':
  - Use '=~' for Regex matching