PLT Fall 2021

MQL: Minimalistic Query Language
Language Reference Manual

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

Contents

1 Overview

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, MQL is 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.

1.1 Types of programs that can be written in MQL

MQL is an imperative programming language. There are seven kinds of tokens: variables, reserved keywords, strings, numeric types, table operators, expression operators, and other separators.

Blanks, tabs, newlines, and comments will be ignored. However, users can include it for code readability. The input stream will be parsed by order of token precedence where expression operators will have the same precedence as mathematics operation (parentheses, multiplication, division, addition, subtraction) followed by table operators (`IMPORT, CREATE, INSERT, SELECT, WHERE, JOIN, EXTEND, DISTINCT, DELETE`) which will have the same precedence.

- **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.
- **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.

2 Comments

2.1 Single and Multi-line Comments

Single and Multi-line comments can be denoted as:

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

/* one line comment */
```

3 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.

MQL will only accept .csv files for table import.

```
IMPORT table.csv as Table1
IMPORT table2.csv as Table2
```

In order to use imported tables, users must rename it in the import statement (where a variable such as `t` will be the identifier of the `.cvs`) like the following:

```
IMPORT table.csv as t

t
.WHERE(column1 == “something”)

t
.WHERE(column1 == “item1” && column2 == “item2”)
```

4 Table and Other Data Types

4.1 Tables

A table is a collection of data, organized in terms of rows and columns. In MQL tables are used with table operations including `IMPORT, CREATE, INSERT, SELECT, WHERE, JOIN, EXTEND, DISTINCT, DELETE`. A table can hold multiple columns with any combination of types. Tables can be created or imported into MQL.

In order to access rows, users must do so by providing conditions using table operations.

In order to access columns, users can do so with `table.ColumnName`

4.2 Strings

A string is a sequence of characters surrounded by double quotes “ “. Strings can be assigned to a variable in order to be referenced throughout the program. Strings can be any length more specifically, to denote a single character, users should also denote using “ ”.

```
string mystr = “I walked my dog today.”;

string mychar = “a”;
```

4.3 Constants

MQL supports all constants including integer, double, float, character, and boolean constants, and other literals inside expressions. A character is a string of length 1 and string literals can be of 1 or more characters.

4.4 Overview of all Data Types

Data type	**Operator**	**Example**

`TABLE`	`All table operators`
	`- SELECT`
	`- INSERT`
	`- WHERE`
	`- JOIN`
	`- EXTEND`
	`- DISTINCT`
	`- DELETE`
		`students.INSERT([“Lily”, 14])`

`COLUMN`	No direct operator. Can only be accessed in reference to `TABLE`.	
`int`	All	`int x = 3;`
`float`	All	`float x = 2.7;`
`boolean`	Logical, Comparison	`bool x = true;`

`string`	Concat	`string x = “Hello, MQL”;`

5 Table Manipulation Operations

MQL supports basic database operations including `SELECT`, `WHERE`, `JOIN`, `DISTINCT`, `INSERT`, `DELETE` and `EXTEND` new columns. Users also have the ability to create temporary tables using the `TABLE` keyword or create new variables of a certain type to then store in the memory and use later.

```
IMPORT Buildings.csv as Buildings;
IMPORT Courses.csv as Courses;

TABLE b = Buildings.WHERE(Campus == “MorningSide”);

Courses
.WHERE(Name == “PLT” && Professor == “Stephen”)
.JOIN(INNER, b, Id, courseId)
.EXTEND(“BuildingName”, b.Name ++ “ at ” ++ b.Campus)
.DISTINCT(Id, BuildingName);
```

Table manipulation operations	**Operator**	**Example**
`TABLE`	`TABLE`	`TABLE variable = TABLE { int col1, string col2, string col3 };`
`INSERT`	`table.`**`INSERT`**`(column_names_tupe)`	`students.INSERT(“Lily”, 14);`

`DELTE`	`table.`**`DELETE`**`(`*`condition`*`)`	`IMPORT table.csv as T`
		
		`T.DELETE(column1 == “name”);`

`WHERE`	`table.`**`WHERE`**`(`*`condition`*`)`	`IMPORT table.csv as T`
		
		`T.WHERE(column1 ==`

		"name");
JOIN	*table*.**JOIN**(join_type, table, [l_table_col1, … ,l_column_n], [r_table_col1, … ,r_column_n]) 3 types of joins: **.JOIN**(INNER, table_name, [l_table_col1, … ,l_column_n], [r_table_col1, … ,r_column_n]) **.JOIN**(LEFT, table_name, [l_table_col1, … ,l_column_n], [r_table_col1, … ,r_column_n]) **.JOIN**(RIGHT, table_name, [l_table_col1, … ,l_column_n], [r_table_col1, … ,r_column_n])	IMPORT table.csv as T IMPORT table2.csv as TT T.JOIN(left, TT, [tName] , [ttName]);
EXTEND	table.**EXTEND**(*new_column_name*, *value*)	IMPORT table.csv as T T.EXTEND("IsAdult", BirthYear > 2003)
DISTINCT	table.**DISTINCT**(column_1,…, column_n)	IMPORT table.csv as T T.DISTINCT(name, DOB)

6 Reserved Words

The following identifiers are reserved for use as keywords, and may not be used otherwise:

```
if else while return let
int float string bool void
true false
TABLE COLUMN INNER LEFT RIGHT DELETE INSERT JOIN SELECT EXTEND
DISTINCT WHERE
print
```

7 Statements

7.1 Variable declarations

7.1.1 Non-Table Type Variable Declarations

For all the primitive types except TABLE, the form of a variable declaration is:

```
Type variable = value;
```

The operator *'='* assigns a value from the right to the left. *Type* here is one of the following: *int, boolean, float, string.*

For example:

```
int a = 5+3;
```

assigns 8 to an integer variable named *a.*

7.1.2 Table Type Variable Declarations

For the TABLE type exclusively, the form of a variable declaration is:

```
TABLE variable = TABLE { type col1, type col2, type col3 };
```

The statement above declares a new table with three columns. Any number of columns is accepted. *Type* here is one of the following: int, boolean, float, string, it defines what value the column can hold. For example,

```
TABLE t = TABLE { int id, string name };
```

declares a table *t* with two columns, where the first one is called *id,* holding integer values, and the second one is called *name*, holding string values.

7.2 Control Flows

MQL processes operations 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.

There are several control flows included in our MQL, this part will show the `if, else, while, return` in detail and other control flows in a general view.

Control Flow	**Description**	**Example**
`if/else`	Conditional statement	*`if(day == "Sun"){`* *`  return true;`* *`}else{`*

		`  return false;` `}`

`while`	Iterative loop	`int i = 0;`
		`int count = 0;`
		`while(i < 3){`
		`  count = count * i;`
		`  i = i+1;`
		`}`

7.2.1 Conditional Statements (If / Else)

If statements consist of a condition (an expression) and a series of statements. The first statement is executed if the condition evaluates to True. Otherwise, we will come to the second statement if there's an 'else' statement.

```
if (condition) {statement1} else {statement2}
if (condition) {statement2}
```

7.2.2 While Loop

The while loop statement consists of a condition and a series of statements. The statements are repeatedly executed when the condition remains true before every iteration. Every statement should end with a semicolon.

MQL does not support ++ and --. In order to increment or decrement the iterator for `while` loops users need to do so by adding or subtracting to the iterator variable.

```
int i = 0
while(condition){
  /*
     series of statements;
  */
  i =  i + 1;
}
```

7.2.3 Return

The return statement has the form:

```
return expression;
```

It returns the value of the expression from a user-defined function.

7.3 Print

Users can print variables, function returns, or anything else to the output.

```
string x = "Hello World";
print(x);
```

8 Variable

A variable can be a sequence of letters and digits. The first character must be alphabetic. Underscores "_" will be counted as alphabetic. Upper and lower case letters are considered different. While there is no maximum variable name length, it should be no more than 10 characters and should not overlap with reserved words.

9 Operators and Arithmetic

9.1 Mathematical Computation Operators

The main mathematical computation operators we will use include `+`, `-`, `*` and `/`. Users can make use of them to fulfill their mathematical requirements. The table below and examples show how to use these operators in MQL.

```
int a = 3;
float b = 1.5;
```

Mathematical Computation Operators	**Description**	**MQL**

+	let the first element plus the second	`float c = a + b;`
		`print (result);`
		/*the result is 4.5*/

-	let the first element minus the second	`float c = a - b;`
		`print (result);`
		/*the result is 1.5*/

*	let the first element times the second	`float c = a * b;`
		`print (result);`
		/*the result is 4.5*/

/	let the first element divide the second	`float c = a / b;`
		`print (result);`
		/*the result is 2*/

9.2 String Operators

MQL supports string concatenation using operator '++'.

```
string c = "Hello";
string d = "MQL";
```

Operation	Description	MQL
++	Concatenate strings	`string e = c ++ d;` /* string e is "Hello MQ" */

9.3 Boolean Operators

Except for the mathematical operators, we also offer users with boolean operators like `++`, `==`, `!=`, `>`, `<`, `>=` and `<=`. The table below and examples show how to use these operators in MQL.

```
int a = 3;
float b =  1.5;
```

Operation	Description	MQL
==	Boolean Equal	`a == b` –> False
!=	Boolean Not equal	`a != b` –> True

>	Greater than	`a > b` –> True
<	Less than	`a < b` –> False
>=	Greater than or equal to	`a >= b` –> True
<=	Less than or equal to	`a <= b` –> False

9.4 Logical Operators

Besides, we also provide `&&, ||, NOT` as logical operators. The cases of these operators are shown below.

Logical Operators	**Description**	**MQL**
`&&` (and)	True only if both two items are true	(True, True) –> True (True, False) –> False
`||` (or)	True as long as one of these items is True	(True, True) –> True (True, False) –> False (False, False) –> False
`NOT` (not)	Change a value from False to True, or from True to False	True –> False False –> True

10 Functions

MQL allows users to build user-defined functions. There are only two things that can be done with a function: calling a function, and defining a function. If a function is referenced beyond the scope of where it is defined that function is said to be “called,” along with any passed in variables and finally the function returns a value.

The syntax of defining a function using MQL is to start with the keyword “`let`” followed by a space, a return type, and a user-defined function name, arguments of the function put in parenthesis right next to the function name. MQL allows functions to have zero or more arguments, each with a type and a name.

An equal sign and curly braces should come after arguments, statements of the function body should all be inside the curly braces.

A return statement is required to end the function.

```
let return_type functionName(argType arg)= {
```

```
       functionBody;
       return;
};
```

A sample of an MQL function is provided below.

```
let boolean checkWeekend(string day)= {
     if(day == "Sun" || day == "Sat"){
          return True;
     }else{
          return False;
     }
};
```

A sample of an MQL function that does not have a return value (void) is provided below.

```
let void addColumn(TABLE t)= {
       T.EXTEND("NewCol", col1 + col1 )
       return;
};
```

---

11 Sample Code

```
/* This program finds names of all female students from the computer
science department */

/* advisor tables has attributes: "advisorID", "studentID",
"firstName", "lastName", "departmentID" */

IMPORT advisor.csv as advisor

TABLE student = TABLE{int studentID,
                      string gender,
                      string firstName,
                      string lastName,
                      int birthYear,
                      string major,
                      int advisorID};
```

```
student.INSERT(111, "M", "John", "Smith", 1989, "Math", 1)
.INSERT(112, "F",  "Jade", "Johnson", 1990, "Math", 1)
.INSERT(113, "F", "Nancy", "Tan", 1995, "CS", 2)
.INSERT(113, "F", "Michelle", "Watt", 2004, "CS", 2);

TABLE department = student.JOIN(INNER, advisor, [advisorID],
[advisorID]);

bool femaleOnly = true;
if(femaleOnly){
     Table result = department
                .EXTEND("IsAdult", birthYear > 2003)
                .DISTINCT(firstName, lastName, departmentID)
                .WHERE(departmentID == "computer_science")
                .WHERE(gender == "F");
     print(result);
}else{
     Table result = department
                .EXTEND("IsAdult", birthYear > 2003)
                .DISTINCT(firstName, lastName, departmentID)
                .WHERE(departmentID = "computer_science")
                .WHERE(gender == "F");
     print(result);
}
```

Below is what the imported advisor table looks like:

advisor.csv as advisor

advisorId	firstName	lastName	departmentId
1	Max	Green	math
2	Tony	Li	computer_science

The resulting table after running the code above will look like the following:

result

firstName	lastName	departmentId	IsAdult
Jade	Johnson	math	true
Nancy	Tan	computer_science	true

Michelle	Watt	computer_science	false