NwQL: A declarative relief from common IP network headaches

Language reference manual

Name: Jehanzeb khan
Uni:JK3305

Table of Contents

1 Introduction

NwQL is a declarative programming language based on relational algebra. NwQL is a complied language that translates statements into a program that sends sequence of NorthBound API messages to an SDN controller, processes the responses and prints the output. NwQL provides simple yet powerful constructs which network administrator may use to query Network Information Base without having to worry about imperative aspects of their queries such as How to process each of the responses? Where to store the responses? An expression written in NwQL takes at least one relation as its operand and supports different types of unary or binary operators.

2 Tokens

NwQL tokens are classified in to eight types: comments, identifiers, keywords, operators, data types, constants, separators and relational tokens.

2.1 Comments

In NwQL comments may span single or multiple lines. The `/*` characters mark beginning of comments while `*/` mark end of the comments.

2.2 Identifiers

An identifier must begin with an uppercase or lowercase letter and may be followed by a combination of letters [including upper and lowercase], digits and underscore(s) in any sequence. Identifiers in NwQL are case sensitive. Valid identifiers and keywords in NyQL are disjoint sets.

2.3 Keywords

The following keywords in NwQL are reserved and may not be used as identifiers. Please note some of these keywords are operators in NwQL and identified as such in the next section.

```
and           insert into      update

as            or               values

delete        select           where

from          set
```

2.4 Operators

NyQL specifies following unary and binary operators.

2.4.1 Unary operators

2.4.1.1 Selection operator

Selection operator in NwQL is a unary operator mathematically represented as $\sigma_{a\theta v}R$ where 'a' is the name of an attribute, 'v' is a constant value, θ is binary operator from a set of {=,! =,<, >, <=,>=} and R is the relation on which select operator is being applied. The keyword used for select operator is `'where'`. Multiple selections can be combined using `and` and `or` operators.

Select and project operators are combined to form a single NyQL query

2.4.1.2 Projection operator

Projection operator in NwQL is a unary operator mathematically represented as πa1.a2.a3…an R where 'a' is a list of attributes names to be projected and R is the relation on which project operator is being applied. The keyword used for project operator is `'select'`

2.4.1.3 Delete operator

The `delete` operator is used to remove rows of a given table in NyQL database. Algebraically deletion is equivalent to performing selection(s) on an existing relation and overriding the resulting relation to the existing relation variable.

2.4.1.4 Update operator

The `update` operator is used to update values of existing rows of a given table in NyQL database. Keyword `set` is used to specify new values and the keyword `where` is used to select the rows to be updated.

2.4.2 Binary operators

2.4.2.1 Insertion operator

The `insert into` operator is used to add new rows of data in NyQL database.

2.4.2.2 Equal operator

The equal operator is one of the operators that can be used with selection operation. The equal operator is represented in NyQL as =

2.4.2.3 Not Equal operator

The not equal operator is one of the operators that can be used with selection operation. The equal operator is represented in NyQL as <>

2.4.2.4 Less than

The less than operator is one of the operators that can be used with selection operation. The equal operator is represented in NyQL as <

2.4.2.5 Greater than

The greater than operator is one of the operators that can be used with selection operation. The equal operator is represented in NyQL as >

2.4.2.6 Less than equals

The less than equal operator is one of the operators that can be used with selection operation. The equal operator is represented in NyQL as <=

2.4.2.7 Greater than equals

The greater than equal operator is one of the operators that can be used with selection operation. The equal operator is represented in NyQL as >=

2.5 Data types

All of the operators in NyQL takes at least one relation as its operand and return another relation as the output. Hence NyQL specifies a single data type that is typically used to hold relations returned as a result of executing different operations on the relations, also known as aliases. Aliases are declared using `as` keyword.

2.6 Constants

In NyQL there are several types of constants where each of the constants belongs to either scalar or composite data types.

2.6.1 Scalar data types

2.6.1.1 Boolean constants

Boolean constants can have one of the two possible values: true or false.

2.6.1.2 String constants

Character strings of variable length.

2.6.1.3 Integer constants

A whole number in the range of (-2**31) to (2**31)-1.

2.6.2 Composite data types

2.6.2.1 Node

A Node describes generic network equipment and is identified using the following two attributes.

NodeIdentifier:String

NodeType:String

2.7 Separators

2.7.1 Comma

In NyQL queries commas are used to separate attributes and table names in the statements.

2.7.2 Parenthesis

In NyQL parenthesis are used to separate subqueries within selection operations

2.8 Relational tokens

NwQL allows abstraction and manipulation of the following OpenDayLight API as a relation.

2.8.1 FlowProgrammer

FlowProgrammer table is an abstraction of OpenFlow flow table and it contains the following attributes.

`Actions`	`IdleTimeout`	`Nwsrc`
`Cookie`	`ingressPort`	`Priority`
`Dldst`	`installInHw`	`Tosbits`
`Dlsrc`	`Name`	`VlanID`
`Ethertype`	`Node`	`VlanPriority`
`HardTimeout`	`Nwdst`	

3 Syntax

3.1 Statements

A NyQL program consists of a set of statements. A statement may be composed of select, select-project, insert, update and delete statements.

```
statement -> project | project_select | insert| update | delete
```

3.1.1 Project

3.1.1.1 A project statement may be composed of **`select`** `column_names` **`from`** `table_names`.

```
Example:Select * from FlowProgrammer
```

```
Example:Select Node from FlowProgrammer
```

```
Example:Select CollisionCount from PortStatistics
```

The return type of projection statements is a relation which can be stored as an alias.

```
Example:Select * from FlowProgrammer as myAlias
```

3.1.2 Project_select

A project_select is one of more selections followed by projection.

```
Example:Select * from FlowProgrammer where action=drop or IdleTimeout
> 0
```

The selection clause in the project_select type of statement may include a sub statement of a restricted type whose result is always a single value.

```
Example:Select * from FlowProgrammer where NodeId = (select NodeId
from PortStatistics where CollisionCount > 5000 )
```

3.2 Insert

3.2.1.1 Insert statements are composed of

INSERT INTO table_name (attribute1, attribute2, attribute3…)

VALUES (value1, value2, value3...)

```
Example:insert INTO

FlowProgrammer

(installInHw...)VALUES (false…)
```

3.3 Update

3.3.1.1 Update statements are composed of

update table_name

SET attribute1=value1, attribute2=value2.. attribute$_n$=value$_n$

```
where column_names =value;
```

3.4 Delete

3.4.1.1 Delete statements are composed of

```
DELETE FROM table_name
WHERE some_column=some_value;

Or

DELETE * FROM table_name
```