XQ: An XML Query Language

Language Reference Manual

Kin Ng

kn2006@columbia.edu

1. Introduction

XQ is a query language for XML documents. This language enables programmers to express queries in a few simple statements. This manual defines the XQ language proposed by the white paper. For the most part, this document follows the board outline in the white paper.

2. Lexical Conventions

2.1 Comments

A comment starts with the characters /* and terminates with the characters */. Comments do not nest, and they do not occur within string or character literals.

2.2 Identifiers

An identifiers is a series of letters and digits with the first character being a letter or the underscore. Upper and lower case letters are distinct.

2.3 Keywords

The following identifiers are keywords:

ASC	*DISTINCT*	*integer*	*SQLCODE*
AVERAGE	*else*	*INTO*	*String*
break	*FETCH*	*MAX*	*SUM*
BY	*document*	*MIN*	*Tag*
CLOSE	*float*	*NULL*	*WHERE*
continue	*for*	*OPEN*	*While*
COUNT	*FROM*	*ORDER*	*WITH*
cursor	*function*	*return*	
DESC	*if*	*SELECT*	

2.4 Constants

2.4.1 Integer Constants

An integer is a series of digits is taken to be decimal.

2.4.2 Floating Constants

A floating constants consists of an integer part, a decimal point, a fraction part, and e or E and an optionally signed integer exponent. Either the integer part or the fraction part (not both) may be missing; either the decimal part of the e and the exponent (not both) may be missing.

2.4.3 String Literals

A string literals is a series of characters enclosed by double quotes, as in *". . ."*. A double quote inside a string literal is represented by \".

2.5 Other Tokens
The following are the remaining tokens:

{	}	(	)	[	]	,	;	+	-
*	/	%	=	+=	-=	*=	/=	>=	<=
>	<	==	!=	!	&&	||	:	&	

3. Meaning of Identifies
3.1 Types
In this language, all variables are statically typed and must be declared before they are used. Types that can be interpreted as numbers will be referred to as arithmetic type. There are several basic types including:

cursor:	pointer to navigate within the query result set
document:	pointer to the data structures to hold the XML document
float:	64-bit IEEE floating point numbers
integer:	32-bit integers
tag:	represent the name of the tags in the data file
string:	sequence of characters

3.2 Scope
There are two kinds of scope: block and file. The scope of an identifier begins when its declaration is seen. Although it is impossible to have two declarations of the same identifier active in the same scope, no conflict occurs if the instances are in different scopes.

4. Conversions
4.1 Integer and Floating
When a floating value is converted to an integral value, the rounded value is preserved as long as it does not overflow. When an integral value is converted to a floating value, the value is preserved.

5. Expressions and Operators
5.1 Primary Expressions
5.1.1 Identifiers
An identifier is an lvalue expression.

5.1.2 Constants
A constant's type is determined by its form and value.

5.1.3 String Literals
A string literal's type is *string*.

5.1.4 Parenthesized Expressions
A parenthesized expressions's type and value are identical to those of the un-parenthesized expressions.

5.1.5 Function Calls
A function call contains a function identifier and a (possibly empty) comma-separated list of expressions that are the arguments to the function.

5.2 Arithmetic Expressions
5.2.1 Unary Operators
Expressions with unary operators associate from right to left. The result of the unary – operator is the negative of its operand. The result of the unary operator + is the value of the operand. The operand must be arithmetic type.

5.2.2 Multiplicative Operators
The multiplicative operators *, /, and % group from left to right. Operands of * and / must have arithmetic type. Operands of % must have integral type. The binary operators *, / and % represents multiplication, division and modulo.

5.2.3 Additive Operators
The additive operators + and – associate from left to right. The result of the + operator is the sum of the operands. The result of the - operator is the difference of the operands. The operands must be both arithmetic type.

5.3 Relational Expressions
The relational operators associate from left to right. The operators <, >, <=, >=, == and != represents greater than, less than or equal to, greater than or equal to, equal to and not equal to respectively. They all yield a result of type *integer* with value 0 if the specific relation is false and 1 if it is true.

5.4 Logical Expressions
The unary operator ! is the logical negation. Its result is 1 if the value of its operand is 0 and vice versa. The && and || operators represent logical AND and OR. The result from both operators has type *integer*. For && operator, if neither of the operands evaluates to 0, the result has a value of 1. Otherwise it has a value of 0. For || operator, if neither of the operands evaluates to 0, the result has a value of 1. Otherwise it has a value of 0.

5.5 Assignment Expressions
Assignment operators consists =, *=, /= , %=, +=, -= . They are all of associate from right to left. Assignment operators require a modifiable lvalue as their left operand. The type of an assignment expression is that of its unqualified left operand. The result is not an lvalue. Its value is the value stored in the left operand after the assignment.

6. Statements

A statement is a complete instruction to the computer. Statements are executed in sequence.

6.1 Expression Statements

Most statements are expression statements. They are evaluated for their side effects, such as assignments or function calls.

6.2 Conditional Statements

The conditional statements have the following form:

if (expression)
  statement

or

if (expression)
  statement
else
  statement

Conditional statements choose one of a set of statements to execute, based on the evaluation of the expression. The expression is the controlling expression. For both forms of the *if* statement, the first statement is executed if the controlling expression evaluates to nonzero. For the second form, the second statement is executed if the controlling expression evaluates to zero. An *else* clause that follows multiple sequential *else-less if* statements is associated with the most recent *if* statement in the same block.

6.3 Iterative Statements

Iteration statements execute the attached statement repeatedly until the controlling expression evaluates to zero.

6.3.1 For Statement

The *for* statement has the following form:

for (expression; expression; expression)
  statement

The first expression specifies initialization for the loop. The second expression is the controlling expression which is evaluated before each iteration. The third expression often specifies incrementation which is evaluated after each iteration.

6.3.2 While Statement

The *while* statement has the following form:

while (expression)
  statement

The controlling expression of a *while* statement is evaluated before each execution of the body.

6.3.3 Break Statement

The *break* statement can appear only in the body of an iteration statement. It transfers control to the statement immediately following the smallest enclosing iteration.

6.3.4 Continue Statement

The *continue* statement can appear only in the body of an iteration statement. It causes the *while* or *for* statement to the end of the loop.

6.4.2 Function Return Statement

A function returns to its caller by the *return* statement. When *return* is followed by an expression, the value is returned to the caller fo the function.

6.4 Query Statements

6.4.2 SET and SELECT Statement

The SET, SELECT statement has the following form:

SET cursor-identifier WITH query-clause

The *SET* and *WITH* keyword first associates a cursor-identifier with the query clause. The query clause has the following form:

SELECT sel-clause
FROM doc-identifier
WHERE expression order-by-clause

The *sel-clause* consists of a *list of tag-identifiers* and performs a projection on the target document. Operational key words such as *AVERAGE, COUNT, SUM, MIN* and *MAX* can be applied to the *sel-clause* to perform aggregate operations. Another keyword *DISTINCT* can be used so there will not be duplicates in the result set.
The *where-clause* consists of expression and follows by an optional *order-by-clause*.
The *where-clause* expression is used to perform a selection on the target document. The optional *order-by-clause* has the following form:

ORDER BY list of tag-identifiers ASC/DESC

The *order-by-clause* is used to sort the result set in order to retrieve in a particular order. *ASC* and *DESC* are used to represent ascending and descending order respectively.

6.4.3 CLOSE Statement
The close statement has the following form:

CLOSE cursor-identifier

The CLOSE statement associates with an identifier of cursor type. Any resource that link to the cursor will be released after the CLOSE statement is executed.

6.4.4 FETCH Statement
The fetch statement has the following form:

FETCH cursor-identifier INTO identifier-list
The fetch statement retrieve information from result set through the a cursor type identifier. The result is loaded into a list of identifiers.

7. Declarations
A declaration specifies the interpretation given to a set of identifiers. Declarations have the following form:

declaration-specifiers init-declarator-list

The *init-declarator-list* is a comma-separated sequence of declarators, each of which can have an initializer. The declarators in the *init-declarator-list* consist of a sequence of type.

8. Function Definition
A function definition has the following form:

```
function func-identifier (identifier-list) {
  statement
}
```

func-identifier represents the name of the function. *Expression-list* is a list of identifier or arguments, separated by commas. The list of argument can be empty. The return statement should be used in function block in order to return a value.

9. Library Functions
9.1 Console Output Functions
9.1.1 Print Function
Print function takes an argument list and print them to the standard output. The return value is number of characters written or negative if an error occurred.

9.2 File I/O Functions

9.2.1 Load Function

The load function has the following form:

load(char-identifier, char- identifier)

The load function takes the name of the XML document and the DTD file as the first and second argument respectively, and returns a *document* type which is pointing to the data structures. If the function fails, the function returns NULL.

9.2.2 Release Function

The release function has the following form:

release(document-identifier)

The release function takes an document identifier as the argument, and free resources that are linked to the document identifier.

9.3 Mathematical Functions

The following mathematical functions are supported by this language:

ceil(x)	smallest integer not less than x, as a float
exp(x)	exponential function e^x
fabs(x)	absolute value $\|x\|$
floor(x)	largest integer not greater than x, as a float
log(x, y)	logarithm of x with base y $log_y x$, $x > 0$
pow(x, y)	x^y. A domain error occurs if $x=0$ and $y \leq 0$, or if $x<0$ and y is not an integer