Table of Contents

Introduction .......................................................... 3
AST for Query expressions ........................................ 4
ForBinds .................................................................... 4
Variable/ctx-param .................................................. 4
allow-duplicates .................................................................. 6
不允许使用…位的Saxon MP:qsxml namespace ............. 7
VariableExp ......................................................... 10
AST ........................................................................... 10
YQGM ........................................................................ 10
GetContextExp ...................................................... 12
AST ........................................................................... 12
YQGM ........................................................................ 12
OperatorExp .......................................................... 13
AST ........................................................................... 13
YQGM ........................................................................ 15
IfExp ........................................................................ 17
AST ........................................................................... 17
QuantifiedExp ........................................................ 19
AST ........................................................................... 19
YQGM ........................................................................ 20
FLWRExpr ........................................................... 21
AST ........................................................................... 21
YQGM ........................................................................ 22
CreateContextExp ............................................... 23
AST ........................................................................... 23
YQGM ........................................................................ 24
Introduction

This document describes the library `xquery_yqgm_conversion`, which is a visitor for the XQuery Abstract Syntax Tree (AST) model. It provides a visitor for the XQuery Abstract Syntax Tree model (in the `xquery_ast` library) that converts the syntax tree to a logical YQGM representation. Each of the AST nodes from `xquery_ast` has a mirror in `xquery_ast_yqgm_conversion`. Each of these mirror nodes has a `toYQGM` method which takes in the corresponding `xquery_ast` node and returns it as a YQGM graph. The following will describe the conversion process for XQuery expressions.

AST for XQuery expressions

The AST is the syntax tree that the XQuery gets converted to by the parser. It contains a construct for each type of expression in the grammar. Here's the general inheritance hierarchy:

```
Expr
  -type : LiteralType
  -value : const char *

LiteralExpr

FLWRExpr
...
```

These are the nodes which are gotten from the `xquery_ast` library. In the `xquery_ast_yqgm_conversion` library, we have a similar inheritance hierarchy:

```
+toYQGM (in astExpr, in ...)
+canPreserveCtx ()
+canHandleMultiTuples ()

ASTYQGMExpr

ASTYQGMFLWRExpr

ASTYQGMLiteralExpr
```

The `ASTYQGMExpr` contains a function `toYQGM`, which creates a correlation if necessary and calls `createYQGM` (a virtual function in `Expr`) to convert the actual expression to YQGM.
To determine whether a correlation needs to be created, the YQGM receives two flags: `preserveCtx` and `handleMultiTuples`. The `preserveCtx` flag indicates to the function whether the context of the original YQGM operator needs to be preserved. Here's what the YQGM graph looks like if `preserveCtx` is true:
preserveCtx

expr refers to the current expression that is being converted to YQGM. In the diagram above, expr->toYQGM represents the YQGM graph that is generated by the AST->YQGM conversion. The output columns of this box represent the output columns that exist after the evaluation of these operators (let's call this the context columns). This box can be viewed as the top operator of the YQGM graph created by expr->toYQGM. The arrow pointing to res represents the output column of this expression.

In this case, we can see that the context columns of the original ctxOpr has not been altered. Additionally, during the evaluation of the expr there may be some output columns added that are not necessarily required for the output columns of the graph (called exprOcls in the graph above). These do not necessarily have to be removed at this stage. Even if they're not being used by the later parts of the query, it can be removed by optimizations later on if need be for the graph.

If preserveCtx is false, that means this toYQGM function does not have to preserve the context of the original expr if it doesn't want to. Of course, it still has to preserve it if it is needed for the execution of the query. In this case, our parser chooses not to preserve the context.
In this case, the extraneous ctx columns have been removed since they are no longer needed.

**handleMultiTuples**

This variable indicates whether the YQGM graph created by the AST is capable of handling incoming tuples of cardinality other than 1. For example, suppose we have the query `SELECT * FROM foo a`. One possible mapping of this to YQGM would be something like:

```
PathExpr
  ctx
  Spj (apply predicate name=a to)
  ctx
  Unnest
  Unnest
  GroupBy (the foo)
  // Unnest
  // GroupBy (the foo)
```

```
Here we do the setting. Let’s apply a predicate name of element = a to each of the /*. Let us say we want to feed more than one tuple at a time to this graph, maybe get the /* of two different documents at once. If the first document has 2 a’s and the second document has 1 a, then the top GroupBy Operator would receive three tuples. However, the GroupBy Operator would not know which tuple belongs to which original input tuple. Since the semantics of the converted YQGM graph is that we get one output tuple for every input tuple, the conversion’s graph does not know which tuple to group with each other. This means that the GroupBy Operator would not know which tuple belongs to which original input tuple.

canPreserveCtx() & canHandleMultiTuples()

Each Expr node in the AST also has the virtual functions canPreserveCtx() and canHandleMultiTuples(). These invoke whether the YQGM graph produced by the YQGM Conversion is capable of handling multiple input tuples at a time. This is determined by checking the flags preserveCtx and handleMultiTuples, which are passed as arguments to the canPreserveCtx() and canHandleMultiTuples() virtual functions. The flags are set to indicate whether context and/or multiple tuples are preserved.

YQGM Conversion

The AST to YQGM conversion is done by the toYQGM() function of each expression node in the AST. This function sets up a correlation if necessary and calls createYQGM to actually generate the YQGM graph. Here is the interface:

toYQGM (visitor, ctxOpr, env, preserveCtx, handleMultiTuples)

- **visitor**: an object of type ExprVisitor which is used as an AST context to include memory management, etc., during the conversion.
- **ctxOpr**: the YQGM operator containing the context for the current expression to be evaluated. The operator will feed this expression tuples as they are evaluated, or can store the context to be used if the expression is delayed for some reason.
- **env**: the environment which contains variable bindings to ocls (i.e. for $x in 5 return 5, $x would be bound to the output column containing the result of the evaluation of 5 when we convert the return expression).
- **preserveCtx**: a flag to specify whether the context of ctxOpr should be preserved in the output operator (see description of last section).
- **handleMultiTuples**: a flag to specify whether the expression needs to handle 0-n tuples at a time (see description of last section).

Here’s a chart of when to create correlations and their corresponding figures:

<table>
<thead>
<tr>
<th>Condition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>preserveCtx = true, handleMultiTuples = true</td>
<td>Create a correlation</td>
</tr>
<tr>
<td>preserveCtx = false, handleMultiTuples = true</td>
<td>Do not create a correlation</td>
</tr>
<tr>
<td>preserveCtx = true, handleMultiTuples = false</td>
<td>Create a correlation</td>
</tr>
<tr>
<td>preserveCtx = false, handleMultiTuples = false</td>
<td>Do not create a correlation</td>
</tr>
<tr>
<td>Scenario</td>
<td>Query</td>
</tr>
<tr>
<td>------------</td>
<td>---------</td>
</tr>
<tr>
<td>Scenario 1</td>
<td>Query 1</td>
</tr>
<tr>
<td>Scenario 2</td>
<td>Query 2</td>
</tr>
<tr>
<td>Scenario 3</td>
<td>Query 3</td>
</tr>
<tr>
<td>Scenario 4</td>
<td>Query 4</td>
</tr>
<tr>
<td>Scenario 5</td>
<td>Query 5</td>
</tr>
<tr>
<td>Scenario 6</td>
<td>Query 6</td>
</tr>
<tr>
<td>Scenario 7</td>
<td>Query 7</td>
</tr>
<tr>
<td>Scenario 8</td>
<td>Query 8</td>
</tr>
<tr>
<td>Scenario 9</td>
<td>Query 9</td>
</tr>
<tr>
<td>Scenario 10</td>
<td>Query 10</td>
</tr>
</tbody>
</table>

Figures A to D illustrate different scenarios and their outcomes.

---

**Figures:**
- Fig A
- Fig B
- Fig C
- Fig D
Of course, the toYQGM function doesn't have to check all of these cases. Here's the pseudo code:

```cpp
if (handleMultiTuples && !canHandleMultiTuples) {
    Create a Correlation with the expr->createYQGM graph
} else {
    Call expr->createYQGM
}
```

createYQGM is an internal virtual function used by toYQGM. It has the exactly same interface as toYQGM and converts the expression toYQGM but doesn't have to worry about whether it needs to set up a correlation (correlations within the expression will be handled by other toYQGM calls in createYQGM).
VariableExpr

- varName : QString

YQGM

canPreserveCtx = true
canHandleMultiTuples = true

Add binding varName->ocl in the environment
GetContextExpr

AST

YQGM

canPreserveCtx = true
canHandleMultiTuples = true

Get the column binding for PT_ITEM, PT_POSITION, or PT_SIZE depending on
the type. Return a pointer to this column back to the main function.
OperatorExpr

AST

[Diagram of AST with nodes and edges]
myElements[0]->toYQGM(PRESERVE_CTX, handleMultiTuples)

myElements[1]->toYQGM(PRESERVE_CTX, handleMultiTuples)

myElements[2-n]->toYQGM(PRESERVE_CTX, handleMultiTuples)

spj

oprType(expr1…exprn)

QuantifiedExpr

AST

- type : QuantifiedExprType
  - varName : QString
  - someEveryClauses1..*
    - satisfiesExpr1
      - expr1
Unnest se0

someEveryClauses[2-n]->toYQGM(PRESERVE_CTX, HANDLE_MULTI)

Unnest sen

satisfiesExpr->toYQGM(NO_PRESERVE_CTX, HANDLE_MULTI)

existsQuantifier – true if one qunpred is true

forallQuantifier – true if all qunpred is true
producer->toYQGM(PRESERVE_CTX, NO_HANDLE_MULTI)

consumer->toYQGM(NO_PRESERVE_CTX, NO_HANDLE_MULTI)

groupby
createAggSequence(consumerRes)