Standard YQGM

Lin Guo Chavdar Botev

12th April 2005

Copyright (C) 2003-2005 Cornell University. All Rights Reserved.

Computer Science Department

Cornell University

Contents

1 Introduction 2

2 Operators 4

2.1 SPJOperator (Spj) ........................................... 4
  2.1.1 Short Description .................................... 4
  2.1.2 Relationships ......................................... 4
  2.1.3 Detailed Description ................................ 4

2.2 OrderByOperator (Obo) .................................. 4
  2.2.1 Short Description .................................... 4
  2.2.2 Relationships ......................................... 4
  2.2.3 Detailed Description ................................ 4
  2.2.4 Restrictions ......................................... 5

2.3 GroupByOperator (Gbo) .................................. 5
  2.3.1 Short Description .................................... 5
  2.3.2 Relationships ......................................... 5
  2.3.3 Detailed Description ................................ 5
  2.3.4 Restrictions ......................................... 5

2.4 UnnestOperator (Uno) .................................. 5
  2.4.1 Short Description .................................... 5
  2.4.2 Relationships ......................................... 5
  2.4.3 Detailed Description ................................ 5
  2.4.4 Restrictions ......................................... 5

2.5 Union Operator (Uio) .................................. 6
  2.5.1 Short description .................................... 6
  2.5.2 Detailed Description ................................ 6
  2.5.3 Restrictions ......................................... 6

2.6 IdGen Operator (Igo) .................................. 6
  2.6.1 Short Description .................................... 6
  2.6.2 Attributes ............................................. 6
  2.6.3 Detailed Description ................................ 6
  2.6.4 Restrictions ......................................... 6
3 Quantifiers

3.1 Each Quantifier (Eq) ................................................................. 6
  3.1.1 Short Description ............................................................. 6
  3.1.2 Detailed Description ......................................................... 8
3.2 Some Quantifier (Sq) ............................................................... 8
  3.2.1 Short Description ............................................................. 8
  3.2.2 Detailed Description ......................................................... 8
3.3 All Quantifier (Aq) ................................................................. 8
  3.3.1 Short Description ............................................................. 8
  3.3.2 Detailed Description ......................................................... 8

4 Expressions

4.1 Predicate (Prd) ................................................................. 8
  4.1.1 Short Description ............................................................. 8
  4.1.2 Relationships ................................................................. 8
  4.1.3 Detailed Description ......................................................... 10
4.2 OrderByExpr (Oxp) ............................................................... 10
  4.2.1 Short Description ............................................................. 10
  4.2.2 Attributes ................................................................. 10
  4.2.3 Relationships ................................................................. 10
  4.2.4 Detailed Description ......................................................... 10
4.3 GroupByExpr (Gxp) ............................................................... 10
  4.3.1 Short Description ............................................................. 10
  4.3.2 Relationships ................................................................. 10
  4.3.3 Detailed Description ......................................................... 10
4.4 UnnestExpr (Uxp) ................................................................. 10
  4.4.1 Short Description ............................................................. 10
  4.4.2 Relationships ................................................................. 11
  4.4.3 Detailed Description ......................................................... 11
  4.4.4 Restrictions ................................................................. 11

5 Parse Trees

5.1 P Constant (Pcn) ................................................................. 12
  5.1.1 Short Description ............................................................. 12
  5.1.2 Detailed Description ......................................................... 12
5.2 SetUnionFunction (Supf) ........................................................ 12
  5.2.1 Short description ............................................................ 12
  5.2.2 Relationships ................................................................. 12
  5.2.3 Detailed Description ......................................................... 12

6 Full Diagram ......................................................... 12

1 Introduction

This document describes the implementation of the standard YQGM elements. Please refer to quark2/yqgm/doc/yqgm.ps for descriptions of the YQGM model and the general YQGM elements.

Throughout this document, we use the same YQGM Static Class Diagram as in quark2/yqgm/doc/yqgm.ps. The following sections discuss the standard YQGM elements and relationships grouped by type. The full YQGM class diagram is shown in section 6 on Figure 5.

The standard YQGM implementation is in C++. All the classes related to it are in the "quark::yqgm_std" namespace. In our implementation, each standard YQGM element is represented through a C++ class. The name of the class is the same as the name in the diagram.
Figure 1: UML Class Diagram for Operators
2 Operators

In this section, we will describe seven standard YQGM operators: GroupByOperator, OrderByOperator, SPJOperator, UnionOperator, UnnestOperator, IdGenOperator and LOJOperator. They all inherit from class Operator the following relationships — the relationship to the containing query (qryOpr, qryOprTop), to its output columns (oprOcl), to its input quantifiers(oprQunInp), and to its output quantifiers (oprQunOpt). We won’t repeat them in the following description.

<table>
<thead>
<tr>
<th>Relationship</th>
<th>Type</th>
<th>Role</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>qryOpr</td>
<td>1 − N</td>
<td>Child</td>
<td>The containing query</td>
</tr>
<tr>
<td>qryOprTop</td>
<td>1</td>
<td>Child</td>
<td>Refers to the containing query if this is the top operator for the query</td>
</tr>
<tr>
<td>oprOcl</td>
<td>1 − N</td>
<td>Parent</td>
<td>The output columns of the operator</td>
</tr>
<tr>
<td>oprQunInp</td>
<td>1 − N</td>
<td>Parent</td>
<td>The input quantifiers for the operator</td>
</tr>
<tr>
<td>oprQunOpt</td>
<td>1 − N</td>
<td>Parent</td>
<td>The output quantifiers for the operator</td>
</tr>
</tbody>
</table>

2.1 SPJOperator (Spj)

2.1.1 Short Description

The SPJOperator class allows the application of scalar transformations on a set of input data streams.

2.1.2 Relationships

<table>
<thead>
<tr>
<th>Relationship</th>
<th>Type</th>
<th>Role</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>spjPrl</td>
<td>1 − N</td>
<td>Parent</td>
<td>The predicates that filter out the associated input tuples</td>
</tr>
</tbody>
</table>

2.1.3 Detailed Description

The SPJ operator is a kind of YQGM operator that can apply scalar transformation on a set of input tuple sequences. The name of the operator comes from the three phases of its work. First is the Join phase that takes the input streams of tuples and joins them by forming their Cartesian product. Then comes the Select phase, when a set of predicates (see 4.1) are checked against the formed input tuple. Those tuples that do not satisfy the predicates are filtered out. The last phase is the Project phase when the values of the output columns are computed for the tuples that satisfy the selection criteria.

It adds the spjPrl relationship to its filtering predicates. The operator may have 0 or more predicates. If the number of predicates is zero then all input tuples are processed. The SPJ operator is analogous to a SELECT statement in SQL with SELECT, FROM, and WHERE clauses.

2.2 OrderByOperator (Obo)

2.2.1 Short Description

The OrderByOperator class allows the ordering of an input data stream according to a specified set of ordering keys.

2.2.2 Relationships

<table>
<thead>
<tr>
<th>Relationship</th>
<th>Type</th>
<th>Role</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>oboOxp</td>
<td>1 − N</td>
<td>Parent</td>
<td>The sequence of ordering expressions that specify the way the input tuples should be ordered</td>
</tr>
</tbody>
</table>

2.2.3 Detailed Description

The OrderBy operator allows the ordering of the input sequence of tuples. The order is determined by a sequence of ordering keys represented as order-by expressions (see 4.2). The order-by expressions are evaluated in the order they are specified. The first order-by expression has the highest priority: it is used to order tuples that produce different values for it. Tuples resulting in the same values for the first order-by expressions are then tested with the second order-by expression, etc.

It adds the oboOxp relationship to its ordering expressions (4.2). The OrderBy operator is analogous to an ORDER BY clause in a SELECT statement in SQL.
2.2.4 Restrictions
An OrderBy operator can have only one input quantifier.

2.3 GroupByOperator (Gbo)

2.3.1 Short Description
The GroupByOperator class allows the grouping of an input data stream according to a specified set of grouping keys.

2.3.2 Relationships

<table>
<thead>
<tr>
<th>Relationship</th>
<th>Type</th>
<th>Role</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gboGxp</td>
<td>1 – N</td>
<td>Parent</td>
<td>The sequence of ordering expressions that specify the way the input tuples are grouped</td>
</tr>
</tbody>
</table>

2.3.3 Detailed Description
The GroupBy operator allows the grouping of the input sequence of tuples. Each set of input tuples that have the same values for the grouping expressions are transformed into one output tuple where the values of the attributes in the output tuples for each group are formed by concatenating the values of the corresponding output columns.

It adds the gboGxp relationship to its grouping expressions (4.3). The GroupBy operator is analogous to a GROUP BY clause in a SELECT statement in SQL.

2.3.4 Restrictions
The GroupBy operator can have only one input quantifier.

2.4 UnnestOperator (Uno)

2.4.1 Short Description
The UnnestOperator class allows the transformation of a tuple into multiple tuples by breaking down the value of an attribute from a sequence of items to multiple singleton sequences.

2.4.2 Relationships

<table>
<thead>
<tr>
<th>Relationship</th>
<th>Type</th>
<th>Role</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>unoUxp</td>
<td>1 – 1</td>
<td>Parent</td>
<td>The sequence of unnesting expressions that specify the attributes of the input tuples to be unnested</td>
</tr>
</tbody>
</table>

2.4.3 Detailed Description
The Unnest operator transforms each input tuple into a sequence of tuples. This is done by breaking down the values of a set of attributes from sequences to multiple singleton sequences (consisting only of one item). The resulting tuples essentially form the Cartesian product of the values of the unnested attributes with preserving the values of the rest of the attributes.

The attributes to be unnested are specified using the UnnestExpr (see 4.4) contained by the operator. Each UnnestExpr is a reference to an input column from the input quantifier that should be unnested. It adds the unoUxp relationship to its unnesting expressions. The unnest operator has no analogous clause or statement in SQL.

2.4.4 Restrictions
The Unnest operator can have only one input quantifier. The Unnest operator must have at least one unnest expression. The parse trees (Section 5) of output columns of an Unnest operator can contain only objects of PColumn and objects of implementations of PFunction.
2.5 Union Operator (Uio)

2.5.1 Short description

The UnionOperator allows the computation of the union of several input streams.

2.5.2 Detailed Description

The Union operator forms the union of the tuples from its input quantifiers. The tuples in each input quantifier must have the same number of attributes. The duplicates are eliminated using the sequence comparison as defined in the XQuery specification [5]. The Union operator is analogous to the UNION operator in SQL.

2.5.3 Restrictions

The PFUnct nodes in ParseTrees of a union operator can only be a SetUnionPFunction (see 5.2).

2.6 IdGen Operator (Igo)

2.6.1 Short Description

The IdGenOperator tags each tuple with an ID that uniquely identifies it.

2.6.2 Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>isOrdered</td>
<td>boolean</td>
<td>Indicate whether the IDs produced in order</td>
</tr>
</tbody>
</table>

2.6.3 Detailed Description

The IdGen operator allows generating of uniquely identified tuples. It is the consumer operator of those source operators whose output tuples need to be identified. It produces N+1 output columns where the first N propagate the output columns of the source operators and the last one produces IDs of the tuples. It adds the attribute isOrdered to indicate whether the IDs are produced in order.

2.6.4 Restrictions

The IdGen operator must have exactly one input quantifier. It must have exactly one output column whose parse tree is an IdGen PFunction (with zero parameters); the parse trees of the rest of the output columns must be PColumns.

3 Quantifiers

In this section, we will describe three standard YQGM quantifiers: AllQuantifier, SomeQuantifier and EachQuantifier. They all inherit from class Quantifier the following relationships — the relationship to the data-stream-consuming operator (oprQunIpt), the relationship to the data-stream-producing operator (oprQunOpt), and the relationship to its input columns (qunId). We won’t repeat them in the following description.

<table>
<thead>
<tr>
<th>Relationship</th>
<th>Type</th>
<th>Role</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>oprQunIpt</td>
<td>1 − N</td>
<td>Child</td>
<td>The operator consuming the data stream</td>
</tr>
<tr>
<td>oprQunOpt</td>
<td>1 − N</td>
<td>Child</td>
<td>The operator producing the data stream</td>
</tr>
<tr>
<td>qunId</td>
<td>1 − N</td>
<td>Parent</td>
<td>The input columns of the quantifier</td>
</tr>
</tbody>
</table>

3.1 EachQuantifier (Eq)

3.1.1 Short Description

The Each Quantifier class represents the flow of data from one operator to another where each produced tuple by the producer operator is supplied to the consumer operator.
Figure 2: UML Class Diagram for Quantifiers
3.1.2 Detailed Description
Instances of the EachQuantifier class are used to connect one operator’s output as another operator’s input. This is the most widely used type of quantifier. It specifies that every tuple produced by the output operator is supplied to the input operator.

3.2 SomeQuantifier (Sq)
3.2.1 Short Description
The SomeQuantifier class represents the flow of data from one operator to another allowing existential quantification over the tuples produced by the producer operator; the Boolean result is supplied to the consumer operator.

3.2.2 Detailed Description
Instances of the SomeQuantifier class are used to connect one operator’s output as another operator’s input. It implements the existential quantification in first-order logic. The value required by the predicate is computed for every tuple produced by the output operator. If some tuples satisfy the predicate, the quantifier produces one tuple with the value true; otherwise, it produces one tuple with the value false.

3.3 AllQuantifier (Aq)
3.3.1 Short Description
The AllQuantifier class represents the flow of data from one operator to another allowing universal quantification over the tuples produced by the producer operator; the Boolean result is supplied to the consumer operator.

3.3.2 Detailed Description
Instances of the AllQuantifier class are used to connect one operator’s output as another operator’s input. It implements the universal quantification in first-order logic. The value required by the predicate is computed for every tuple produced by the output operator. If all tuples satisfy the predicate, the quantifier produces one tuple with the value true; otherwise, it produces one tuple with the value false.

4 Expressions
In this section, we will describe four standard YQGM expressions: Predicate, OrderByExpr, GroupByExpr and UnnestExpr. They all inherit from class Expression the relationship to the parse trees which compute its value (expPrt). We won’t repeat it in the following description.

<table>
<thead>
<tr>
<th>Relationship</th>
<th>Type</th>
<th>Role</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ExpPrt</td>
<td>1-1</td>
<td>Parent</td>
<td>The parse tree used to compute the value of the expression</td>
</tr>
</tbody>
</table>

4.1 Predicate (Prd)
4.1.1 Short Description
The Predicate is an kind of Expression that is interpreted as returning Boolean values.

4.1.2 Relationships

<table>
<thead>
<tr>
<th>Relationship</th>
<th>Type</th>
<th>Role</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>spjPrd</td>
<td>1-N</td>
<td>Child</td>
<td>The containing SPJ operator</td>
</tr>
</tbody>
</table>
Figure 3: UML Class Diagram for Expressions
4.1.3 Detailed Description

The Predicate is an expression that is used in SPJ operators (2.1) to filter out input tuples. It is a regular expression but its values are interpreted as "true" or "false" as defined in the XQuery standard [5]. It adds the spjPnd relationship to the containing SPJ operator. The Predicate is analogous to a predicate expression in the WHERE clause of a SELECT statement in SQL.

4.2 OrderByExpr (Oxp)

4.2.1 Short Description

The OrderByExpr is a kind of expression that is interpreted as an ordering key in a OrderBy operator (see 2.2).

4.2.2 Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>orderDir</td>
<td>OrderDirection</td>
<td>Indicates the sorting order.</td>
</tr>
<tr>
<td>nullLoc</td>
<td>NullLocation</td>
<td>Indicates where the tuples with NULL value should be placed.</td>
</tr>
</tbody>
</table>

4.2.3 Relationships

<table>
<thead>
<tr>
<th>Relationship</th>
<th>Type</th>
<th>Role</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>oboOxp</td>
<td>1 − N</td>
<td>Child</td>
<td>The containing OrderBy operator</td>
</tr>
</tbody>
</table>

4.2.4 Detailed Description

The OrderByExpr class provides a key value to be used for sorting of input tuples in an OrderByOperator. The key value is computed based on the expression’s parse tree for every input tuple.

The OrderByExpr also contains additional modifiers for the ordering operation. These are the order direction (orderDir) and NULL location (nullLoc). The former defines the sorting order and can be the default (implementation-defined) order, ascending order, or descending order. The latter specifies where the tuples with NULL value should be placed for the OrderBy expression. The options are the default implementation-defined location, to be considered as having the greatest key value, or to be considered as having the smallest key value.

An OrderByExpr corresponds to a single ordering key expression in the ORDER BY clause of a SELECT statement.

4.3 GroupByExpr (Gxp)

4.3.1 Short Description

The GroupByExpr is a kind of Expression that is interpreted as a grouping key in a GroupBy operator (see 2.3).

4.3.2 Relationships

<table>
<thead>
<tr>
<th>Relationship</th>
<th>Type</th>
<th>Role</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gboGxp</td>
<td>1 − N</td>
<td>Child</td>
<td>The containing GroupBy operator</td>
</tr>
</tbody>
</table>

4.3.3 Detailed Description

The GroupByExpr provides a key value to be used for grouping tuples in a GroupByOperator. The input tuples with the same values for each grouping expressions are transformed into one output tuple as described in 2.3.

The GroupByExpr corresponds to the comma-delimited expressions in the GROUP BY clause of a SELECT statement in SQL.

4.4 UnnestExpr (Uxp)

4.4.1 Short Description

The UnnestExpr is a kind of expression that specifies the unnesting of the values of an attribute of the input in an unnest operator (see 2.4).
### 4.4.2 Relationships

<table>
<thead>
<tr>
<th>Relationship</th>
<th>Type</th>
<th>Role</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>unoUxp</td>
<td>1 − N</td>
<td>Child</td>
<td>The containing Unnest operator</td>
</tr>
</tbody>
</table>

### 4.4.3 Detailed Description

The `UnnestExpr` specifies how an attribute is to be unnested in a unnest operator. The parse tree of an `UnnestExpr` can contain only `PColumn` nodes or `PFunction` nodes of a superscalar function. The `UnnestExpr` has no analogous expression in SQL.

### 4.4.4 Restrictions

The parse tree of an `UnnestExpr` can contain only `PColumn` nodes or `PFunction` nodes of a superscalar function.
5 Parse Trees

In this section, we will describe two standard YQGM parse trees: PConstants and SetUnionPFunction. They both inherit from class ParseTree the following relationships — the relationship to the containing expression (expPrt) and the relationship to the function taking the parse tree as a parameter (pfnPrtParent). We won’t repeat it in the following description.

<table>
<thead>
<tr>
<th>Relationship</th>
<th>Type</th>
<th>Role</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>expPrt</td>
<td>1 − 1</td>
<td>Child</td>
<td>The expression whose value the parse tree computes</td>
</tr>
<tr>
<td>pfnPrtParent</td>
<td>1 − N</td>
<td>Child</td>
<td>The function taking the parse tree as a parameter</td>
</tr>
</tbody>
</table>

5.1 PConstant (Pcn)

5.1.1 Short Description

The PConstant class represents a constant in an parse tree.

5.1.2 Detailed Description

PConstant is a kind of ParseTree node that allows the use of constants in a parse tree. It is an abstract class that defines interfaces to access the constant (encoded as an atomic value in the XQuery data model). The PConstant corresponds to the constants used in SQL expressions.

5.2 SetUnionPFunction (Supf)

5.2.1 Short Description

The SetUnionPFunction represents the application of a SetUnion function to a set of parameter parse trees.

5.2.2 Relationships

<table>
<thead>
<tr>
<th>Relationship</th>
<th>Type</th>
<th>Role</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pfnPrtChildren</td>
<td>1 − N</td>
<td>Parent</td>
<td>The parameters of the function</td>
</tr>
</tbody>
</table>

5.2.3 Detailed Description

SetUnionPFunction is an implementation of class PFunction. It is used to apply a SetUnion function on a set of parameters accessible through the pfnPrtChildren relationship (inherited from PFunction). It is used by Union operator.

6 Full Diagram

See Figure 5.

References

Figure 5: YQGM Full Class Diagram