Yet another Query Graph Model (YQGM)

Chavdar Botev    Lin Guo

12th April 2005

Computer Science Department
Cornell University

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

Contents

1 Introduction 2

2 The YQGM Model 2
   2.1 Elements and Relationships ........................................ 3
   2.2 Implementation Notes ............................................. 3

3 The YQGM Query Graph 3
   3.1 Query (Qry) .......................................................... 3
       3.1.1 Short Description ........................................... 3
       3.1.2 Relationships ................................................. 3
       3.1.3 Detailed Description ........................................ 3
   3.2 QueryWithFunction (Qry) ........................................... 4
       3.2.1 Short Description ........................................... 4
       3.2.2 Relationships ................................................. 4
       3.2.3 Detailed description ....................................... 4
   3.3 FunctionQuery (UsrFun) ........................................... 5
       3.3.1 Short Description ........................................... 5
       3.3.2 Attributes .................................................... 5
       3.3.3 Relationships ............................................... 5
       3.3.4 Detailed description ....................................... 5

4 Operators 5
   4.1 Operator (Opr) ..................................................... 5
       4.1.1 Short Description ........................................... 5
       4.1.2 Relationships ................................................. 5
       4.1.3 Detailed Description ....................................... 5
   4.2 ParamOperator ................................................... 7
       4.2.1 Short Description ........................................... 7
       4.2.2 Relationships ............................................... 7
       4.2.3 Detailed Description ....................................... 7

5 Quantifiers 7
   5.1 Quantifier (Qun) .................................................. 7
       5.1.1 Short Description ........................................... 7
       5.1.2 Relationships ............................................... 7
       5.1.3 Detailed Description ....................................... 9
   5.2 InputColumn (Icl) ................................................. 9
1 Introduction

This document describes Yet another Query Graph Model (YQGM) - a general model for representing queries. YQGM is inspired by the Query Graph Model (QGM) in the Starburst project [1], the XML Query Graph Model (XQGM) in XERANTO [3], and the RelOperator graph model in PREDATOR [6]. YQGM is more general because it is independent of the underlying data model and can support XML, relational and other data models.

2 The YQGM Model

In short, the model represents the query as a graph. The nodes in the graph are called operators and they are used to perform a specific kind of a computation. The edges are called quantifiers and represent the data flow in the graph. Each quantifier takes the output produced by an operator (called the output operator) and directs it to another operator (called the input operator).

The data flow in the model is encoded as a sequence of tuples, which can be viewed as a relation. Each tuple for a given operator output has a fixed number of attributes. The value of each attribute is a sequence as defined in the underlying data model [4].

As it can be seen, the data model is quite similar to the relational model. The major difference is that the value of an attribute in our model can be quite complex - it can be even a sequence of documents represented as document nodes. Nevertheless, through the rest of the exposition we will try to draw some analogies to SQL, which we hope will make the model clearer.

Let us first brush up the general structure of a SQL statement:
SELECT Expr1, Expr2, ...
FROM R1, R2, ...
WHERE Pred1 AND Pred2 ...
ORDER BY Key1, Key2, ...
GROUP BY GroupKey1, GroupKey2, ...

We now move to a more detailed description of the elements in the YQGM model.

2.1 Elements and Relationships

An YQGM Static Class Diagram (e.g. Figure 1) consists of three main types of graphical objects: classes (represented with boxes), generalizations (represented with arrows with hollow ends), and associations between classes (represented with lines with possible diamond heads or "stick" arrows). For more information about the UML language, refer to [2].

The classes describe the YQGM elements; these are the building blocks of the graph model. The associations describe how the YQGM elements relate to each other. In our model, we call them YQGM relationships. Each relationship has a name, which is specified in both of the corresponding association’s ends.

The following sections discuss the general YQGM elements (in library YQGM) and relationships grouped by type. The full YQGM class diagram is shown in section 8 on Figure 6. The standard YQGM elements are implemented in library YQGM_STD (Documents for the standard YQGM elements are stored in quark2/yqgm_std/doc ).

2.2 Implementation Notes

The YQGM implementation is in C++. All the classes related to it are in the "quark:yqgm:structures:" namespace.

In our implementation, each YQGM element is represented through a C++ class. The name of the class is the same as the name in the diagram.

We chose to implement relationships using template classes. There are two template classes with parameters the class of the parent element and class the children elements: RelN for 1-to-N relationships and Rel11 for 1-to-1 relationships. They define two subclasses, ParentAccess and ChildrenAccess or ChildAccess, which represent the view on the relationship from the children’s/child’s perspective and the parent’s perspective. The former allows access to the parent of the relationship and the latter allows access to the children/child in the relationship.

These relationships govern the way YQGM elements can be created. An element is constructed by taking as parameters all its parents in relationships where the element is a child. Therefore, the graph is always constructed top-down: first the query, then the operators, then the quantifiers, etc.

The names of the relationships are formed by the short name of the parent class (given in parenthesis in the YQGM element descriptions below), the short name of the children class, and possible additional tag identifying the relationship if there are more than one relationship between the parent and the child class.

3 The YQGM Query Graph

3.1 Query (Qry)

3.1.1 Short Description

The Query class is used to represent a query.

3.1.2 Relationships

<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>Parent</td>
<td>The operators in the query graph</td>
</tr>
<tr>
<td>qryOprTop</td>
<td>1</td>
<td>Parent</td>
<td>The output operator for the query</td>
</tr>
</tbody>
</table>

3.1.3 Detailed Description

The UML static class diagram for the Query class is shown on Figure 1. On this diagram (and the subsequent class diagrams) the classes in focus are shaded. All the other related classes are given in white boxes. Every query (without user defined functions) in YQGM is represented as an instance of the Query class. It consists of one or more operators (instances of the class Operator - see 4.1), connected by means of quantifiers (the class Quantifier - see 5.1), thus forming the query graph.
3.2 QueryWithFunction (Qry)

3.2.1 Short Description

The \textit{QueryWithFunction} class is used to represent a query with user defined functions.

3.2.2 Relationships

<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>Parent</td>
<td>The operators in the query graph</td>
</tr>
<tr>
<td>qryOprTop</td>
<td>1</td>
<td>Parent</td>
<td>The output operator for the query</td>
</tr>
<tr>
<td>qryUsrFun</td>
<td>1 – N</td>
<td>Child</td>
<td>User defined functions defined in this query</td>
</tr>
</tbody>
</table>

3.2.3 Detailed description

\textit{QueryWithFunction} is a query with user defined functions — \textit{FunctionQuery} (see 3.3). The \textit{qryUsrFun} relationship provides access to all the user defined functions in this query.

The class \textit{QueryWithFunction} inherits from \textit{Query} the \textit{qryOpr} relationship to all its operators and the \textit{qryOprTop} relationship to the top operator.
3.3 FunctionQuery(UsrFun)

3.3.1 Short Description

The FunctionQuery class is used to represent a user defined function.

3.3.2 Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>functionName</td>
<td>const char *</td>
<td>Name of the user defined function</td>
</tr>
<tr>
<td>paramOpr</td>
<td>ParamOperator</td>
<td>The operator which holds all input parameters of the function</td>
</tr>
</tbody>
</table>

3.3.3 Relationships

<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>Parent</td>
<td>The operators in the query graph</td>
</tr>
<tr>
<td>qryOprTop</td>
<td>1</td>
<td>Parent</td>
<td>The output operator for the query</td>
</tr>
<tr>
<td>qryUsrFun</td>
<td>1 – N</td>
<td>Parent</td>
<td>The containing query of this function</td>
</tr>
<tr>
<td>usrFunPuf</td>
<td>1 – N</td>
<td>Child</td>
<td>The PUserFunction objects which apply this function to associated parameters</td>
</tr>
</tbody>
</table>

3.3.4 Detailed description

FunctionQuery is a kind of query used to represent user defined functions (UDF). It stores the function name and a parameter operator ParamOperator (see 4.2) which holds all parameters of the UDF. The qryUsrFun relationship provides access to the containing query; and the PUserFunction (see 7.4) which uses this UDF is accessible through the usrFunPuf relationship.

The FunctionQuery inherits from Query the qryOpr relationship to all its operators and the qryOprTop relationship to the top operator.

4 Operators

4.1 Operator (Opr)

4.1.1 Short Description

The Operator class is used to represent an operator in the query graph.

<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>oprQm1pt</td>
<td>1 – N</td>
<td>Parent</td>
<td>The input quantifiers for the operator</td>
</tr>
<tr>
<td>oprQmOpt</td>
<td>1 – N</td>
<td>Parent</td>
<td>The output quantifiers for the operator</td>
</tr>
</tbody>
</table>

4.1.3 Detailed Description

The operator is the primary computational element in the YQGM graph. It may or may not take input data from its input quantifiers but it always produces some output, which is accessed by the other operators through its output quantifiers.

The Operator class is an abstract class that defines the common features of all operators (see Figure 2). There are eight types of operators that differ on the type of computation (functionality) they perform:

- SPJ (select-project-join): Generates the Cartesian product of the tuples from its input quantifiers (the Join phase), checks whether certain conditions are satisfied (the Select phase), and uses the product for some computational task involving scalar functions (the Project phase).
- Union: Forms the union of the input sequences of tuples.
- OrderBy: Orders its input tuples by the value of the specified key expression(s). 


Figure 2: UML Class Diagram for Operators
• GroupBy: Groups the values of specified input attributes into sequences based on the specified grouping keys expression(s).

• Unnest: Breaks an input tuple into a sequence of tuples by generating the Cartesian product of the values of the specified attributes.

• Param: Holds all the parameters of a user defined function (FunctionQuery).

• IdGen: Tags each tuple with an ID that uniquely identifies the tuple.

• LOJ: Implements the left outer join (LOJ) operation.

The common features among all these types of operators are the relationships listed in 4.2.2. The *qryOpr* relationship allows access to the containing query. The *oprOcl* relationship encompasses the output columns (see 6.2) of the operator that are used to compute the values for each attribute of the output tuple. The *oprQunLpt* relationship provides access to the input quantifiers for the operator, i.e. the quantifiers that contain the input sequences of tuples. The *oprQunOpt* relationship contains the output quantifiers of the operator; these are used by other operators to access the data produced by the operator.

Except for *ParamOperator*, all other specific operators are implemented in library YQGM_STD. The specifics of these operators are explained in quark2/yqgm_std/doc.

### 4.2 ParamOperator

#### 4.2.1 Short Description

The *ParamOperator* class is used to hold all parameters of a *FunctionQuery* object (see 3.3).

#### 4.2.2 Relationships

<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>oprQunLpt</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>

#### 4.2.3 Detailed Description

*ParamOperator* is a specific operator. We use it to hold all the parameter parse trees (see 7.1) of a *FunctionQuery* object. However, it doesn’t need to define its own relationships, since the parameter parse trees are wrapped in the output columns (see 6.2) of the parameter operator, which can be accessed by its relationship *oprOcl* (inherited from *Operator*). Please refer to section 6.2 for the relationships between output columns and parse trees.

### 5 Quantifiers

#### 5.1 Quantifier (Qun)

##### 5.1.1 Short Description

The *Quantifier* class represents the flow of data from one operator to another.

##### 5.1.2 Relationships

<table>
<thead>
<tr>
<th>Relationship</th>
<th>Type</th>
<th>Role</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>oprQunLpt</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>qunOcl</td>
<td>1 - N</td>
<td>Parent</td>
<td>The input columns of the quantifier</td>
</tr>
</tbody>
</table>
Figure 3: UML Class Diagram for Quantifiers
5.1.3 Detailed Description

Instances of the Quantifier class are used to connect one operator’s output as another operator’s input. The former is accessible through the oprQtntOpt relationship, and the latter is accessible through the oprQtntOptp relationship (see Figure 3). The tuples are accessible by the input operator through input columns (see 5.2). Each input column is used to access the value of an output column in the output operator. The input columns can be referred to using the qantcl relationship.

There are three types of quantifiers: EachQuantifier, AllQuantifier and SomeQuantifier. All of the three subclasses are implemented in library YQGM_STD.

5.2 InputColumn (Icl)

5.2.1 Short description

The InputColumn class allows the access to the value of an operator’s output column through a quantifier.

5.2.2 Relationships

<table>
<thead>
<tr>
<th>Relationship</th>
<th>Type</th>
<th>Role</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>qantcl</td>
<td>1 – N</td>
<td>Child</td>
<td>The quantifier containing the input column</td>
</tr>
<tr>
<td>oclcl</td>
<td>1 – N</td>
<td>Child</td>
<td>The output column producing the value</td>
</tr>
<tr>
<td>iclPcl</td>
<td>1 – N</td>
<td>Parent</td>
<td>The PColumns using the value supplied by the input columns</td>
</tr>
</tbody>
</table>

5.2.3 Detailed Description

The InputColumn class doesn’t inherit Quantifier, but is an accessor to the value of an attribute in the tuples produced by the output operator for a quantifier. The value is accessed through the output column used for its calculation. Then the value can be used in the input operator using a PColumn (see 7.2).

The containing quantifier is accessible through the qantcl relationship. The output column producing the value of the attribute is accessible through the oclcl relationship. The PColumns accessing the value of the input column can be found using the iclPcl relationship.

6 Expressions

6.1 Expression (Exp)

6.1.1 Short Description

The Expression class represents the concept of expression.

6.1.2 Relationships

<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>

6.1.3 Detailed Description

The Expression class is the abstract ancestor of all types of expressions used in YQGM. Through the expPrt relationship, it provides a parse tree (see 7.1) that can be used to compute the value of the expression.

There are five types of expressions. Except for OutputColumn, all the specific expressions are implemented in library YQGM_STD.

- OutputColumn - provides the value for an attribute in the output tuple of an operator
- Predicate - a predicate expression to filter out input tuples in an SPJ operator
- OrderByExpr - an expression used to compute the value of a sorting key for each tuple of the sequence to be sorted
- GroupByExpr - an expression used to compute the value of a grouping key
- UnnestExpr - An expression specifying the attributes of the input tuple to be unnested.
Figure 4: UML Class Diagram for Expressions
The Expression class corresponds to the expression that can be found in the different clauses in a SELECT statement in SQL.

6.2 OutputColumn (Ocl)

6.2.1 Short Description
The OutputColumn is used to compute the value of an attribute in the output tuple sequence of an operator.

6.2.2 Relationships

<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 output column; inherited from Expression</td>
</tr>
<tr>
<td>oprOcl</td>
<td>1−N</td>
<td>Child</td>
<td>The containing operator</td>
</tr>
<tr>
<td>oelOcl</td>
<td>1−N</td>
<td>Parent</td>
<td>The Quantifiers' input columns accessing the value of the output column</td>
</tr>
</tbody>
</table>

6.2.3 Detailed Description
The output column is a kind of expression (6.1) used to compute the value of a single attribute in the output tuple sequence of an operator. This value is then accessed by some of the input columns of the output quantifiers for the parent operator. The oprOcl relationship provides access to the containing operator. The expPrt relationship contains the parse tree used to compute the output column's value. The oelOcl relationship gives access to the input columns referring to the values produces by this output column.

The output column is analogous to the expressions in the SELECT clause of a SELECT query.

7 Parse Trees

7.1 ParseTree (Prt)

7.1.1 Short Description
The ParseTree class represents the concept of a tree generated by a parsed expression.

7.1.2 Relationships

<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 (if any)</td>
</tr>
<tr>
<td>pfnPrtParent</td>
<td>1−N</td>
<td>Child</td>
<td>The function taking the parse tree as a parameter (if any)</td>
</tr>
</tbody>
</table>

7.1.3 Detailed Description
The ParseTree is the base class to represent a (sub-)tree used to compute the value of an expression. The name comes from the fact that this tree is constructed by parsing the string representation of the expression.

Parse trees can be composed by supplying a parse tree as a parameter to a parse tree node representing a function (see below). There are three types of parse trees (see also Figure 5).

- **PColumn** - extracts the value of an input column (an attribute from an input tuple for an operator);
- **PFunction** - is an abstract class representing the application of a function to a set of parameter parse trees;
- **PConstant** - allows the use of constants in a parse tree, implemented in YQGM_STD.

The relationship expPrt points to the containing expression object and is set only for the root node of the parse tree. The relationship pfnPrtParent can be used to access the parent PFunction (see 7.3) which takes this parse tree as an input parameter.
Figure 5: UML Class Diagram for Parse Trees
7.2 PColumn (Pcl)

7.2.1 Short Description

The PColumn is a type of ParseTree used to access the value of an input column.

7.2.2 Relationships

<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>fnPrtParent</td>
<td>1−N</td>
<td>Child</td>
<td>The function taking the parse tree as a parameter</td>
</tr>
<tr>
<td>iclPcl</td>
<td>1−N</td>
<td>Child</td>
<td>The input column whose value the PColumn accesses</td>
</tr>
</tbody>
</table>

7.2.3 Detailed Description

The PColumn class can be used to access the value of an input column (see 5.2) in an expression. This class inherits from ParseTree the relationship to the containing expression (expPrt) and the relationship to the parent PFunction object (if any) - fnPrtParent. The PColumn class adds the iclPcl relationship to the input column whose value is accessed.

The PColumn corresponds to the reference to a column of a relation in the FROM clause in a SQL statement.

7.3 PFunction (Pfn)

7.3.1 Short description

The PFunction class represents the application of a function to a set of parameters in the form of ParseTrees (7.1).

7.3.2 Relationships

<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>fnPrtParent</td>
<td>1−N</td>
<td>Child</td>
<td>The function taking the parse tree as a parameter</td>
</tr>
<tr>
<td>fnPrtChildren</td>
<td>1−N</td>
<td>Parent</td>
<td>The parameters of the function</td>
</tr>
</tbody>
</table>

7.3.3 Detailed Description

PFunction is an abstract class used to relate a set of parameters (accessible by the fnPrtChildren relationship) to some functions (implemented by its subclasses). The parameters are other ParseTrees. This class inherits from ParseTree the relationships to the containing expression (expPrt) and the relationship to the parent PFunction object (if any) - fnPrtParent.

This class corresponds to the function application in SQL expressions. We implement two types of PFunctions (see also Figure 5).

- PUserFunction - Applies a user defined function (a FunctionQuery object) to the parameters.
- SetUnionPFunction - Applies a SetUnion function to the parameters. It is implemented in library YQGM_STD.

7.4 PUserFunction (Puf)

7.4.1 Short Description

PUserFunction represents an application of a user defined function to the parameters.

7.4.2 Relationships

<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>fnPrtParent</td>
<td>1−N</td>
<td>Child</td>
<td>The function taking the parse tree as a parameter</td>
</tr>
<tr>
<td>fnPrtChildren</td>
<td>1−N</td>
<td>Parent</td>
<td>The parameters of the function</td>
</tr>
<tr>
<td>usrfunPuf</td>
<td>1−N</td>
<td>Parent</td>
<td>The user defined function referred to</td>
</tr>
</tbody>
</table>

13
7.4.3 Detailed Description

`PUserFunction` is an implementation of class `PFunction` which applies a user defined function (accessed by relationship `usrFunPuf`) to the parameters. All parameters are accessible by the relationship `pfnPrtChildren` (inherited from `PFunction`). This class indirectly inherits from `ParseTree` the relationships to the containing expression (`expPrt`) and the relationship to the parent `PFunction` object (if any) - `pfnPrtParent`.

8 Full Diagram

See Figure 6.

References


[6] Praveen Seshadri, "PREDATOR Design and Implementation".
    http://www.distlab.dk/predator/papers/designdochtml/designdoc.html
Figure 6: YQGM Full Class Diagram