Модел. и анализ информ. систем. Т.**15**, №**1** (2008) 27–33

Объектное расширение PL/ODQL языка запросов ODQL для Динамической информационной модели DIM

Писаренко Д.С.
Ярославский государственный университет
150 000, Ярославль, Советская, 14
E-mail : dimapi_ yaroslavl@mail.ru

получена 20 февраля 2008

Аннотация

Рассматривается задача создания вычислительно полного расширения объектно-динамического языка запросов ODQL [1], которое было бы удобно для работы с объектами этой системы. Приводится описание синтаксиса этого расширения.

Объектно-динамический язык запросов ODQL [1] модели данных DIM [2] не является вычислительно полным по той же причине, что и язык SQL (средствами SQL92 невозможно реализовать повторение шагов МТ до перехода МТ в состояние останова), хотя и обладает запросной полнотой [3].

Нашей целью является создание такого расширения языка ODQL (назовем его PL/ODQL), которое было бы вычислительно полным и позволяло удобно описывать различные алгоритмы взаимодействия сущностей системы DIM. Под *сущностью системы DIM* мы будем понимать классы, их свойства, отношения, объекты, значения свойств объектов.

Проведенный анализ имеющихся объектных языков запросов, таких как языки Tutorial D [4] и OQL [5], основанные на идеях третьего Манифеста Object Data Management Group [4], показывает, что в рассмотренных языках присутствуют структурные типы данных, позволяющие манипулировать некоторым количеством (множеством) однородных объектов: например, массив *кортежей* в Tutorial D или *множества*, *мультимножества*, *списки* в OQL. Но в обоих этих языках конструирование и использование этих множеств сложно, также отсутствуют конструкции, позволяющие добавлять в полученные множества и удалять из них единичные объекты. Еще одним недостатком этих языков является то, что в них нет конструкций, позволяющих описывать исключительные ситуации, которые могут возникать при выполнении различных алгоритмов, написанных с их (языков) помощью. Ну и как следствие, все эти языки не имеют конструкций для обработки таких исключительных ситуаций. При создании языка PL/ODQL некоторые вводимые конструкции подобны конструкциям рассмотренных языков, и, в свете рассмотренных недостатков изученных языков, вводятся новые конструкции. Мы потребуем от языка PL/ODQL вычислительной полноты еще и для того, чтобы избежать проблем, связанных с потребностью описывать алгоритмы взаимодействий на других языках (как было замечено для языка D в [6]). Немаловажно заметить, что разрабатываемый язык PL/ODQL будет полностью учитывать специфику и особенности (отношения включения, истории) системы DIM, что сделает его наиболее удобным для использования в системе.

Результатом выполнения одного ODQL SELECT запроса является множество однотипных сущностей системы DIM, например: множество классов, множество значений свойств объекта в разные промежутки времени. *Величинами* будем называть значения свойств объектов, скаляры и константы. Для начала определим скалярные типы:

<скалярный тип> ::= string | integer | float | boolean | datetime

Зададим синтаксис определения константы и скаляра следующим видом:

CONST <скалярный тип> <имя константы> := <значение>;
VAR <скалярный тип> <имя переменной> [:= <значение>];

Переменная типа string может принимать только строковые значения длиной не более предопределенной скрытой константы StringMaxLength, задаваемой реализацией языка PL/ODQL. Строковые значения должны всегда быть заключены в кавычки (“”).

Переменная типа integer может принимать только целочисленные значения в диапазоне значений предопределенных скрытых констант MinInteger и MaxInteger, задаваемых реализацией языка PL/ODQL.

Переменная типа float может принимать любое числовое значение с плавающей точкой в диапазоне значений констант MinFloat и MaxFloat с точностью до FloatPrecision десятичных знаков. Константы MinFloat, MaxFloat и FloatPrecision являются скрытыми и предопределенными реализацией языка PL/ODQL.

Переменная типа boolean может принимать только одно из двух значений: либо истина, либо ложь. Истина обозначается ключевым словом *true*, а ложь - ключевым словом *false*.

Переменная типа datetime может принимать любое значение даты и времени в диапазоне значений предопределенных скрытых констант MinDate и MaxDate, задаваемых реализацией языка PL/ODQL.

Определим синтаксис присвоения скаляру нового значения следующим образом:

<имя переменной> := <значение>;

Введем традиционные арифметические операции *сложения*, *вычитания*, *умножения* и *деления* для переменных и констант типов integer и float, определив их обычным образом.

Для переменных и констант типа string введем операцию *конкатенации*, которая будет обозначаться знаком "+".

Для переменных и констант типа boolean введем логические операции *и*, *или*, *не* и *исключающее или*, которые будут обозначаться AND, OR, NOT и XOR соответственно.

Введем операции сложения и вычитания для переменных и констант типа datetime, которые будут обозначаться знаками "+" и "-" соответственно. Результатом этих операций будут даты, но если результаты вычислений будут выходить за границы диапазона значений констант MinDate и MaxDate, то произойдет ошибка выполнения алгоритма, описанного на языке PL/ODQL. Для работы с составляющими частями значений типа datetime используются операторы DATEPART и DATEADD. Оператор DATEPART служит для выделения указанной составляющей из значения типа datetime и возвращает значение типа integer:

DATEPART(<значение типа datetime>, <тип составляющей>);
<тип составляющей> ::= Y | M | D | h | m | s | ms

<тип составляющей> Y означает год, M - порядковый номер месяца в году, D - день месяца, h - час, m - минуты, s - секунды и ms - миллисекунды. При выделении указанной составляющей остальные составляющие даты и времени игнорируются.

Оператор DATEADD позволяет увеличить указанную составляющую datetime на определенную величину:

DATEADD(<значение типа datetime>, <тип составляющей>, <величина>);
<тип составляющей> ::= Y | M | D | h | m | s | ms

Значение <типа составляющей> аналогично оператору DATEPART; <величина> может быть только типа integer.

Помимо скалярных типов переменных введем в язык PL/ODQL типы структурных переменных, служащих для описания свойств, классов и объектов. Синтаксис определения переменных структурных типов не отличается от синтаксиса определения переменных скалярных типов:

VAR <структурный тип> <имя переменной> [:= <значение>];
<структурный тип> ::= property | class | object

Переменные типа property могут принимать в качестве своего значения только свойства классов и объектов. Аналогично переменные типа class и object могут принимать в качестве своего значения только классы и объекты соответственно, и позволяют выполнять с ними различные операции, которые будут описаны далее.

Предоставим возможность работы не только с единичными сущностями системы DIM и единичными скалярами, но и с их множествами. Для этого введем в язык PL/ODQL тип *упорядоченное множество* (далее для простоты будем называть его *множество*).

SET "<"<структурный тип>">" <переменная> : =
<SELECT запрос > | "{" [<сущность> [, <сущность>]...] "}";

SET “<” <скалярный тип> “>” <переменная> : = “{” [<скаляр>[,< скаляр >]. . .] “}”;

Данная конструкция позволяет присваивать переменной, имеющей тип SET, множество скаляров или множество сущностей системы DIM, которое будет получено в результате выполнения запроса <SELECT запрос>. В результирующем множестве могут присутствовать только значения указанного структурного или скалярного типа. Каждый элемент полученного множества имеет порядковый номер, с помощью которого к нему можно получить доступ внутри множества следующим образом:

<имя переменной типа SET> “[”<порядковый номер>“]”

Получив, таким образом, доступ к элементу множества, мы не можем изменять значение элемента и не можем изменять порядковый номер элемента во множестве.

Для получения порядкового номера элемента множества введем в язык PL/ODQL оператор INDEXOF, возвращающий целочисленное значение. Первым параметром этого оператора служит множество, а вторым параметром - элемент этого множества. Синтаксис оператора таков:

INDEXOF(<множество типа SET>, <элемент множества>)

Введем в язык PL/ODQL операцию IS_IN, которая будет позволять определить, образует ли множество, переданное вторым параметром, подмножество множества, переданного первым параметром. И введем в язык PL/ODQL операцию IN, которая будет позволять определить, принадлежит ли величина, переданная вторым параметром, множеству, переданному первым параметром. Операторы IS_IN и IN будут возвращать значение типа boolean и могут использоваться в следующих видах:

IS_IN(<множество типа SET 1>, <множество типа SET 2>)
IN(<множество типа SET 1>, <переменная>)

Для добавления элементов в уже имеющиеся множества введем в язык PL/ODQL оператор INCLUDE со следующим синтаксисом:

INCLUDE (<величина> [, <величина>] . . .)
INTO <множество>;
<величина> ::= <переменная> | <константа> | <значение>

Этот оператор добавляет значения всех перечисленных переменных в <множество>, даже если они уже присутствуют в нем. Для удаления элементов из множества введем в язык PL/ODQL оператор EXCLUDE, который имеет такой синтаксис:

EXCLUDE (<величина> [, <величина>] . . .) FROM <множество>;

Этот оператор удаляет из <множества> перечисленные переменные. Если какой-либо переменной не было в <множестве>, то она не учитывается. Для каждой переменной из множества удаляется только 1 элемент, соответствующий этой переменной.

Дополним язык PL/ODQL операциями объединения, пересечения и разности множеств типа SET. Для объединения множеств введем оператор UNITE, который принимает в качестве параметров два множества типа SET и в результате выполнения возвращает новое множество, являющееся объединением множеств, указанных как параметры оператора.

Для нахождения пересечения двух множеств типа SET введем оператор INTERSECT. Этот оператор принимает в качестве параметров два множества типа SET и в результате выполнения возвращает новое множество, являющееся пересечением множеств, указанных как параметры оператора.

При работе с множествами часто требуется операция нахождения разности двух множеств. Для нахождения разности двух множеств типа SET в язык PL/ODQL введем оператор SUBSTRACT. Этот оператор принимает в качестве параметров два множества типа SET, первое из которых является уменьшаемым множеством, а второе - вычитаемым. В результате выполнения оператор возвращает новое множество, являющееся разностью множеств, указанных как параметры оператора.

Для упрощения описания операций объединения, пересечения и разности множеств можно использовать арифметические знаки “+”, “*” и “-”, которые эквивалентны операторам UNITE, INTERSECT и SUBSTRACT соответственно.

Классы и объекты могут иметь между собой различные типы связей, например, наследование классов и объектов, их включение. Для получения информации о связях введем операторы PARENTOF, CHILDRENOF, INCLUDES, INCLUDEDIN и INCLUSION. Все эти операторы принимают только один параметр, которым может быть или класс, или объект, и возвращают множество классов, если параметром был передан класс, или множество объектов, если параметром был передан объект. Результат выполнения первых четырех операторов следует из их названия, в комментариях нуждается только оператор INCLUSION, который возвращает объект связи при включении одной сущности в другую сущность, передаваемую первым параметром.

Для выделения свойств сущности DIM введем оператор PROPERTIES, который возвращает множество параметров сущности DIM, переданной оператору как параметр.

Для получения доступа к конкретному свойству сущности введем оператор [], который будет возвращать свойство сущности по имени свойства. Имя свойства должно быть значением типа string. Синтаксис оператора таков:

<сущность>“[”<имя свойства>“]”
<сущность> ::= <класс> | <объект>

Взаимодействия в DIM описываются с помощью специальных классов и объектов системы DIM. Они включают в себя классы и объекты, описывающие 4 основных составляющих взаимодействия: **What**, **From**, **To** и **How**. Введем в язык PL/ODQL операторы, которые позволят извлекать классы и объекты составляющих взаимодействия из классов и объектов, описывающих взаимодействия системы DIM. Этими операторами будут операторы WHATINTERACTION, FROMINTERACTION, TOINTERACTION и HOWINTERACTION соответственно. Все они принимают только один параметр, которым может быть или класс, или объект взаимодействия, и возвращают множество классов, если параметром был передан класс взаимодействия, или множество объектов, если параметром был передан объект взаимодействия.

Введем в язык PL/ODQL операции сравнения “=”, “<”, “>”, “<>”, “<=”, “>=” (*равно*, *меньше*, *больше*, *не равно*, *меньше или равно*, *больше или равно* соответственно).

<операция сравнения> ::= <первая величина> <знак сравнения> <вторая величина>
<знак сравнения> ::= { = | < | > | <> | <= | >= }

Сравниваемые величины должны быть одного типа либо быть скалярных приводимых типов. Например, величину типа integer можно привести к типу float и сравнить с другой величиной типа float. Любая скалярная величина приводима к типу string. Для структурных типов величин применимы только знаки сравнения “=” и “<>”.

Результат операции сравнения называется *истиной*, если знак сравнения в нем соответствует соотношению величин, участвующих в этом выражении сравнения. В противном случае результат операции сравнения называется *ложью*.

Операция сравнения является частным случаем логического выражения. В общем случае *логическое выражение* - это выражение, результатом вычисления которого является либо *истина*, либо *ложь*.

<логическое выражение> ::= <выражение сравнения> | <boolean значение > | true | false

Дополним язык PL/ODQL оператором ветвления IF, который имеет следующий синтаксис:

IF (<логическое выражение>)
  <блок >
[
ELSEIF (<логическое выражение>)
  <блок>
] . . .
[
ELSE
  <блок>
]

<блок> ::= оператор PL/ODQL>;
  | BEGIN <оператор PL/ODQL>; [<оператор PL/ODQL >;] . . . END

Оператор ветвления IF исполняет не более чем один <блок>, содержащийся в нем. Каждый блок может содержать один или более вложенных в него блоков. Если в операторе содержится ключевое слово ELSE, то выполняется точно один из <блоков>, содержащихся в операторе. Выражения <логическое выражение> проверяются на истинность в порядке их расположения в операторе, пока не будет встречено истинное <логическое выражение> или достигнуто ключевое слово ELSE. Если обнаружено истинное <логическое выражение> или встретилось ключевое слово ELSE, то выполняется непосредственно следующий <блок> и на этом исполнение оператора завершается. Оставшиеся <логические выражения> не вычисляются. Если не встретилось ни одного истинного <логического выражения> и не было найдено ключевое слово ELSE, выполнение оператора завершается без выполнения каких-либо <блоков> в пределах оператора.

Для удобства дальнейшего описания вводимых в язык PL/ODQL операторов циклов определим тело цикла следующим образом:

<тело цикла> ::= <оператор>; | BEGIN <оператор>; [<оператор>;] ... END
<оператор> ::= <оператор PL/ODQL> | BREAK; | CONTINUE;

BREAK и CONTINUE являются специальными операторами языка PL/ODQL, которые могут встречаться только в теле цикла. Оператор BREAK служит для прекращения выполнения цикла наибольшего уровня вложенности, в теле которого он встречается. Оператор CONTINUE служит для немедленного перехода к следующей итерации цикла наибольшего уровня вложенности, в теле которого он встречается.

Дополним язык PL/ODQL оператором цикла WHILE, который имеет следующий синтаксис:

WHILE (<логическое выражение>)
  <тело цикла>

Цикл WHILE выполняется пока истинно <логическое выражение>.

Дополним язык ODQL оператором цикла FOR, который имеет следующий синтаксис:

FOR <параметр цикла> FROM <начальное значение> TO <конечное значение>
  <тело цикла>

Цикл FOR выполняется заданное число раз, пока значение параметра цикла принадлежит диапазону от <начальное значение> до <конечное значение> включительно. После каждой итерации цикла значение параметра цикла изменяется на единицу. Границы диапазона значений не могут изменяться внутри цикла. Переменную, используемую как параметр цикла, объявлять не обязательно, и в теле цикла ей можно задавать новое значение.

Для последовательного перебора всех элементов множества типа SET и выполнения над ними однотипных операций дополним язык PL/ODQL оператором цикла FOREACH, который имеет следующий синтаксис:

FOREACH <переменная> IN <множество типа SET>
  <тело цикла>

Цикл FOREACH выполняется для каждого элемента <множества>. <множество> представляется переменной типа SET. Доступ ко всем элементам множества осуществляется с помощью <переменной>, которой на каждой итерации цикла присваивается новый элемент <множества>.

Для описания исключительных ситуаций, которые могут возникнуть при выполнении операций, описанных языком PL/ODQL, служит специальный тип переменных, тип EXCEPTION. Переменная этого типа содержит специальный код возникшей исключительной ситуации, и может содержать ее словесное описание. Синтаксис операции создания переменной типа EXCEPTION таков:

EXCEPTION <имя переменной> := (<код ситуации> [, <описание ситуации>]);

Создание переменной типа EXCEPTION не является уведомлением внешнего окружения программы о возникновении исключительной ситуации. Для этого служит оператор RAISE. Синтаксис использования оператора RAISE таков:

RAISE <переменная типа EXCEPTION>;

или для удобства в упрощенной форме:

RAISE EXCEPTION (<код ситуации> [, <описание ситуации>]);

Для обработки возникших уведомлений об исключительных ситуациях введем оператор CATCH, который должен находиться после всех операторов PL/ODQL. Его синтаксис таков:

CATCH <код ситуации> [, <переменная описания ситуации>]

Если указана <переменная описания ситуации>, то ее значением станет описание исключительной ситуации, указанное при создании переменной типа EXCEPTION, если оно было указано, и пустая строка (“”), если описание указано не было. Заранее определять переменную <переменная описания ситуации> не требуется.

Введенные конструкции являются широко распространенными во всех современных языках программирования и на данный момент обеспечивают необходимый минимум для удобного описания алгоритмов на языке PL/ODQL. В дальнейшем будет продолжена работа над языком PL/ODQL, и для большего удобства в использовании он будет расширен другими конструкциями.

Рассмотрим использование конструкций языка PL/ODQL на примере расчета общей суммы подоходного налога, уплачиваемой предприятием за своих работников из отдела IT в месяц:

```
CONST float percents = 0.13;
CONST float privilegesLimit = 20000;
VAR float taxesAmount = 0;

SET<Worker> workers = “for отдел = ‘IT’ select workers from workers”;
SET<Worker> workersWithoutPrivileges = ;

FOREACH worker IN workers
BEGIN
  taxesAmount = taxesAmount + worker[“ТекущаяЗарплата”] * percents;
  worker[“ТекущаяПолучка”] = worker[“ТекущаяЗарплата”] * percents;
  worker[“ЗаработалЗаГод”] = worker[“ЗаработалЗаГод”] + worker[“ТекущаяПолучка”];
  IF (worker[“ЗаработалЗаГод”] > privilegesLimit)
  BEGIN
    worker[“ПотерялПривилегию”] = true;
        INCLUDE (workersWithoutPrivileges,worker);
  END
END

SET <Worker> workersWithPriveleges = workers - workersWithoutPrivileges;

FOREACH worker IN workersWithPriveleges
BEGIN
  IF (worker[“ИмеетДетей”] = false)
    RAISE EXCEPTION (255, “Работник, не имеющий детей, получал налоговую скидку”);
END
```

Список литературы

1. Чехранов, Д.В. Основные концепции объектно-динамического языка запросов ODQL динамической информационной модели DIM / Д.В. Чехранов // Современные проблемы математики и информатики: Сборник научных трудов молодых ученых, аспирантов и студентов. Ярославль: ЯрГУ, 2006. - Вып. 8. С.143.

2. Рублев, В.С. Концепции объектной динамической информационной модели DIM / В.С. Рублев, А.Р. Юсупов. // Математика в Ярославском университете: Сборник обзорных статей к 30-летию математического факультета. - Ярославль: ЯрГУ, 2006. - С.335-354.

3. Рублев, В.С. Полнота объектно-динамического языка запросов динамической информационной модели DIM / В.С. Рублев, Д.В. Чехранов, А.Р. Юсупов. // Проблемы теоретической кибернетики: Тезисы докладов XIV Международной конференции (Пенза). - М., 2005. - С. 130.

4. Дейт, К. Дж. Основы будущих систем баз данных: третий манифест / К. Дж. Дейт, Хью Дарвен. - Янус-К, 2004.

5. Гарсиа-Молина, Г. Системы баз данных. Полный курс / Г. Гарсиа-Молина, Дж.Д. Ульман, Дж. Уидом - Вильямс, 2003.

6. Дейт, К. Дж. А теперь про нечто полностью вычислительное / К. Дж. Дейт; Перевод - Сергей Кузнецов // сайт www.citforum.ru (точный адрес http://www.citforum.ru/database/articles/comput-complete/)

PL/ODQL object enhancement of ODQL query language for Dynamic information model DIM

Pisarenko D. S.

Considered a task of creating a computing complete enhancement for ODQL [1] object-dynamic query language. The enhancement should be convenient in DIM system objects manipulating. The enhancement syntax description is stated.