УДК 519.682; 681.324.06

В. С. Рублёв, А. Н. Петров

Язык PL/ODQL и множества с индексами

Рассматривается расширение объектного языка запросов ODQL [2−4] для объектной СУБД Динамическая информационная модель DIM [5]. Для удобства работы с объектами в качестве одной из основных конструкций языка используется новая конструкция множества с индексами, позволяющая применять ее не только как обычное множество, но и как мультимножество, и списки объектов множества, упорядоченные по индексам.

Ключевые слова: объектная СУБД, объектный язык запросов, расширение языка запросов, множества с индексами.

V. S. Rublev, A. N. Petrov

The PL/ODQL Language and Sets with Indexes

Expansion of the object query language of ODQL [2−4] for object DBMS the Dynamic information DIM [5] model is considered. For convenience of work with objects as one of the main constructions of the language is used a new construction of a set with the indexes, allowing to use it not only as a usual set, but also as a multiset and lists of objects the sets ordered on indexes.

Keywords: object DBMS, an object query language, expansion of a query language, sets with indexes.

Введенное в [3] объектное расширение PL/ODQL языка запросов ODQL [6−8] СУБД DIM [4] предназначено для создания аппарата взаимодействий [4, 5] этой системы. В целях эффективной организации процедур для выполнения взаимодействий в этой работе введены «множества» объектов, классов, свойств и их значений. Но они являются не вполне множествами, так как, с одной стороны, элементы этих «множеств» могут повторяться, как в мультимножествах, а с другой стороны, в некоторых случаях они обрабатываются, как списки. Эти особенности не были учтены в [3], а потому не были организованы удобные операторы языка для работы с ними. Данная статья ликвидирует указанные пробелы и упорядочивает описание языка PL/ODQL в целях дальнейшей эффективной реализации взаимодействий.

В работе [4] классы рассматриваются как множество объектов и таковым являются, поскольку все объекты имеют различные объектные идентификаторы. Поэтому операции объединения, пересечения и дополнения для подмножеств объектов класса вполне допустимы, удобны и в реализации эффективны. Но если рассматривать совокупность значений некоторого свойства или группы свойств таких объектов, то она уже может не являться множеством, если не включать в каждый элемент совокупности объектный идентификатор, что не предусматривается.

Введенные индексы для элементов множества также представляют собой странную конструкцию: фактически они служат лишь как указатели на тот или иной объект подмножества, выделенный запросом, а в операциях над множествами их значения теряют смысл.

Вместе с тем, на множество объектов можно смотреть как на индексированный список. Анализ объектных языков запросов Tutorial D [2] и OQL [1] показывает, как отмечено в [3], что они используют и списки, и мультимножества. Введенная в [7] для эффективной организации выполнения запроса конструкция индекс-выборки позволяет с каждым подмножеством объектов класса организовать различное его упорядочивание, а следовательно, работать с ним как со списком.

1. Модули PL/ODQL

Основными конструкциями модулей PL/ODQL являются описания (переменных, процедур и функций), операторы, обработчики исключительных ситуаций и блоки, содержащие эти конструкции. Приведем синтаксис блоков и описаний процедур и функций.

<блок> ::= **BEGIN** {<ODQL-запрос> | <оператор PL/ODQL> | <блок >}

[{<ODQL-запрос> | <оператор PL/ODQL> | <блок >}]…}
[<обработчики исключений>] **END**

<описание процедуры> ::= <заголовок процедуры>
[<локальные описания: типов, констант, переменных, процедур, функций >]
<блок> **END** [<имя процедуры>]

<описание функции> ::= <заголовок функции>
[<локальные описания: типов, констант, переменных, процедур, функций >]
<блок> **END** [<имя функции>]

<заголовок процедуры> ::= **PROCEDURE** <имя процедуры>
([параметры процедуры]) **IS**

<заголовок функции> ::= **FUNCTION** <имя функции>
([параметры функции]) <тип функции> **IS**

Вызов функции является одним из выражений PL/ODQL и имеет вид:

<имя функции> ([<выражения аргументов>])

При вызове функции происходит передача параметров от аргументов: по значению аргументов простых типов (см. далее) или по ссылке для типов сущностных, массивов и множеств (см. далее).

В блоке описания функции обязателен оператор возвращения значения функции, который имеет вид:

RETURN <выражение возвращаемого значения>

Выражение возвращаемого значения может содержать операции, вызовы других функций и при возвращении приводится к типу функции.

Выполнение оператора **RETURN** завершает выполнение вызова функции и возвращает значение в точку вызова функции.

Вызов процедуры осуществляется оператором PL/ODQL вызова процедуры, который имеет вид:

<имя процедуры> ([<выражения аргументов>]);

Передача параметров происходит подобно вызову функции.

В блоке описания процедуры может быть оператор завершения процедуры, который имеет вид:

RETURN;

Он прекращает выполнение процедуры и передает управление следующему оператору за вызовом процедуры. Прекращение выполнения процедуры наступает и в том случае, когда выполнен последний оператор блока процедуры.

Модуль PL/ODQL имеет следующее синтаксическое описание:

<описание модуля> ::= <имя модуля> **BEGIN**
[<описание глобальных констант, переменных и типов модуля>]
[<описания процедур и функций>] [<обработчики исключений>]
END [<имя модуля>]

Комментарии в программе могут находиться в любой ее строке после символов `//'.

2. Простые типы и типы массивов

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

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

Определим простой тип как скалярный или структурный тип:

<простой тип> ::= <скалярный тип> | <структурный тип>

Константы и переменные простого типа определим следующим образом:

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

Переменная типа **integer** может принимать только целочисленные значения в диапазоне значений предопределенных скрытых системных 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. Примерами констант типа **float** могут быть следующие значения:

2012., 2012.0002, 2.0120002e3, -20120002e-4 .

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 **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 двойные кавычки (``''). Набор символов строковых значений определяется реализацией, но в него обязательно входят все строчные и прописные буквы кириллицы и латиницы, а также арабские цифры, скобки (круглые, квадратные, фигурные), кавычки (одинарные и двойные), знаки препинания, пробел и знаки операций (арифметических и логических).

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

Для констант типа **datetime** определены 3 формата:

- полный: YYYY.MM.DD|hh:mm:ss:lll Например, 2012.08.28|17:01:46:256
- даты: YYYY.MM.DD (время берется по умолчанию нулевым). Например, 2012.08.28 определяется значением 2012.08.28|00:00:00:000
- времени: hh:mm:ss:lll (дата по умолчанию берется текущей). Например, 17:01:46:256 определяется значением 2012.08.28|17:01:46:256

Описание массива задается следующей конструкцией:

ARRAY <простой тип> <имя массива> (n_1[, n_2] …)

[:= <выражение значения типа массива>];

где n_1, n_2,... – константы типа **integer**, определяющие количество элементов по каждой размерности массива, а <простой тип > есть тип элементов массива. Обращение к элементу массива имеет следующий синтаксис:

<имя массива> `['<список индексов элемента>`]'

<список индексов элемента> ::= <индекс> [, <индекс>] …

<индекс> ::= <выражение типа **integer**>

Значение выражения индекса по каждой соответствующей размерности массива должно быть неотрицательным и меньшим количества элементов по этой размерности.

3. Операции над величинами простых типов и массивами

Под величиной некоторого простого типа будем понимать переменную, константу, элемент массива такого типа, функцию, возвращающую значение этого типа, а также любое выражение, имеющее такой тип. Порядок выполнения операций определяется их приоритетом и скобками.

Введем традиционные арифметические операции сложения, вычитания, умножения и деления для величин типов **integer** и **float**, определив их обычным образом с обычным приоритетом. Для величин типа **boolean** введем логические операции *не, и, или* и *исключающее или*, которые будут обозначаться **not, and, or** и **xor** соответственно (приоритет операций в этом порядке).

Для величин типа **string** (<строка>) введем операцию конкатенации, которая будет обозначаться знаком ``+'', функцию

Length (<строка>),

возвращающую длину строки, а также функцию выделения подстроки

Substring (<строка>, m[, l]),

где m – позиция в строке начала выделения подстроки, а l − длина подстроки (если $m+l$-1 превосходит длину исходной строки, то возвращается часть строки, начиная с позиции m, так же, как и при отсутствии аргумента l).

Функция

Pozition (<строка 1>, <строка 2>)

возвращает номер позиции, с которой строка 1 полностью содержится в строке 2 (если это не так, то возвращает 0).

Для преобразования величины типа **string** к другому простому типу используются как неявные преобразования (если это требуется, например, оператором присваивания), так и явные преобразования функциями **StringToInteger**, **StringToFloat**, **StringToBoolean**, **StringToDate**. Обратные преобразования также могут выполняться неявно (например, оператором присваивания) или явно функциями **IntegerToString**, **FloatToString**, **BooleanToString** и **DateToString**.

Введем операции сложения и вычитания для переменных и констант типа **datetime**, которые будут обозначаться знаками ``+" и ``−" соответственно. Результатом этих операций будут данные того же типа. Второй операнд используется для увеличения или уменьшения мс, секунд, минут, часов, дня месяца, месяца и года с переносом. Например,

2012.10.20|23:50:00:000 + 0000.03.12|01:20:00:000 = 2013.02.02|01:10:00:000

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 составными частями значений типа **datetime** используют функции **DatePart**, **DateAdd**.

Функция **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.

Функция **DateAdd** возвращает значение типа **datetime** с указанной составляющей, большей на определенную величину (возможно, со знаком) типа **integer**:

DateAdd (<значение типа **datetime**>, <тип составляющей>, <величина>)

При этом может происходить коррекция соседних составляющих такого значения. Например,

DateAdd (2011.12.31|23:59:55:000, **s**, 10) = 2012.01.01|00:00:05:000.

Функции **Date** и **Time** возвращают, соответственно, значения даты и времени выражения аргумента типа **datetime**.

Выражение величин типа массив предполагает операции, соответствующие типу элементов массива. При этом если оба операнда являются выражениями типа массив, то оба массива должны иметь одинаковые размерности и количества элементов по каждой размерности. Если это не так, то выражения дополняются элементами, имеющими величину по умолчанию (0 для типов **integer**, **float** и **false** для **boolean,** а также пустую строку ``'' для **string**), и размерностями с элементами, также имеющими величину по умолчанию. Операция в этом случае выполняется над каждой парой элементов обоих массивов, обладающих одинаковыми индексами. Если же один из операндов является скалярной величиной, а другой – массивом, то операция выполняется для каждого элемента массива и этой скалярной величины.

4. Сущностные типы и типы множеств объектов

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

VAR <сущностный тип> <имя переменной>[:= <значение>];

<сущностный тип> ::= **property** | **class** | **object**

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

Для описания типов различных множеств объектов введен тип множеств следующей конструкцией:

TYPESET `<' <тип множества> `>' := <ODQL-запрос>;

Каждый элемент множества такого типа содержит свойства, описываемые фразой **select** запроса (для одного или нескольких связанных классов). Если класс один, то, как правило, рекомендуется в качестве имени типа множества выбирать имя класса.

5. Множества и операции над ними

Предоставим возможность работы не только с простыми и сущностными данными, но и с их множествами, а также с индексированными списками элементов каждого множества. Для этого введем в язык PL/ODQL тип множества (с индексами):

[**CONST**] **SET** {<тип множества> | <сущностный тип> | <простой тип>}
<имя множества>[:={<ODQL-запрос> | ``{"<список элементов>``}"}]
[**INDEX** <список индексов>];

<список элементов> ::= <элемент>[, <элемент>]…

<элемент> ::= <значение элемента типа множества>[1]

<список индексов> ::= <индекс>[,<индекс>]…

< индекс> ::= <имя индекса> : <выражение от свойств типа множества>

Данная конструкция позволяет присваивать переменной, имеющей тип **SET**, множество значений простого типа, или уже описанного типа множества, или множество значений сущностей системы DIM, которое будет получено в результате выполнения ODQL-запроса. В результирующем множестве могут присутствовать только значения указанного простого типа, или типа множества, или сущностного типа. При включении (добавлении) элемента в множество он получает по умолчанию индекс (главный), равный объектному идентификатору класса, объекта или свойства объекта в соответствии с сущностным типом множества, если источником элемента был запрос, или равный порядковому номеру включения элемента в множество, если источником элементов был список элементов. Непосредственно значение индекса элемента нельзя получить, так как он является указателем на элемент, и значение такого указателя зависит от реализации. Но в цикле **FOREACH** (см. далее), где перебираются все элементы множества, можно запомнить это значение в индексной переменной, которая объявляется следующим образом:

VAR index <имя индексной переменной>
[:= {<параметр цикла **FOREACH**> | <имя индекса>}];

Следовательно, каждый элемент множества при его создании получает индекс, с помощью которого к нему можно получить доступ следующим образом[2]:

<имя множества> `['<имя индексной переменной>`]'

Получив таким образом доступ к элементу множества, мы не можем изменять значение элемента и его индекс в множестве. При удалении элемента значению его главного индекса присваивается неопределенное значение **NULL**.

Помимо главного индекса, множество, тип которого не является сущностным типом **class**, **object** или **property**, может иметь дополнительные индексы, каждый из которых определяется именем и выражением значений свойств, связанных с типом множества. Если в операторе цикла **FOREACH** в качестве параметра используется имя индекса, то элементы множества перебираются в порядке увеличения значения индекса, и, кроме того, к элементам можно обратиться с помощью индексной переменной, которой можно присвоить значение индекса.

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

ADD ({<величина> [, <величина>]… | <запрос-ODQL>}) **into** <множество>;
<величина> ::= <переменная> | <константа> | <выражение>

Этот оператор добавляет в *множество* значения всех перечисленных величин, которые не присутствуют в нем.

Для удаления элементов из множества введем в язык PL/ODQL оператор **EXCL**, который имеет такой синтаксис:

EXCL ({<величина> [, <величина>]… |<запрос-ODQL>}) **from** <множество>;

Этот оператор удаляет из *множества* перечисленные значения, если они есть в нем.

---

[1] Значением элемента типа множества не может быть объектный идентификатор класса, объекта, свойства.

[2] В этом смысле множество подобно массиву, а обращение к элементу множества подобно обращению к переменной массива.

Введем 3 бинарных операции +, *, −, которые возвращают, соответственно, значение множества объединения, пересечения и разности множеств операндов. Это позволяет удобно строить формулы для множеств.

Дополним перечень операций с множествами бинарными операциями ``=’’ (равенства множеств), ``<=" (включения множества левого операнда в множество правого операнда) и **in** принадлежности элемента левого операнда, заданного *величиной*, множеству правого операнда. Эти операции возвращают значения типа **boolean**.

6. Свойства сущностей и их связей

В этом разделе понятие сущности ограничим классами и объектами. Последние могут иметь между собой различные типы связей, например, наследование классов или объектов, их включение. Для получения информации о связях введем следующие функции, которые в качестве аргумента содержат класс или объект:

Parent(сущность) − возвращает множество родительских сущностей,

Child (сущность) − возвращает множество дочерних сущностей,

Included(сущность) − возвращает множество включенных сущностей,

Includes (сущность) − возвращает множество включающих сущностей,

Inclusion(сущность 1, сущность 2) − возвращает множество связей включения двух аргументов, находящихся в отношении включения.

Для выделения свойств сущности DIM введем функцию

Properties (сущность),

которая возвращает множество свойств сущности DIM.

Для получения доступа к конкретному свойству сущности введем оператор [], который дает возможность получить значение свойства сущности по имени свойства. Синтаксис оператора следующий:

<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 четыре основные роли сущностей взаимодействия: *What, From, To* и *How*. Введем в язык PL/ODQL функции с переменным числом аргументов

(сущн.1, роль сущн.1[, сущн.2, роль сущн.2[, сущн.3, роль сущн.3]]),

которые для взаимодействий, где каждая сущность, указанная аргументом, находится в роли, указанной следующим аргументом (роли всех аргументов различны), возвращают множество сущностей взаимодействий, находящихся в другой роли, указанной названием функции:

WhatInteraction (аргументы),

FromInteraction (аргументы),

ToInteraction (аргументы),

HowInteraction (аргументы).

Например, функция

WhatInteraction (объект 1, *From*, объект 2, *To*)

возвращает множество объектов, для каждого из которых найдется взаимодействие, где они в роли *What*, а объекты, указанные аргументами, в ролях *From* и *To*.

Для связей отношения **HISTORY** введены 2 функции **Pred** и **Succ** с одним аргументом, возвращающие для него соответственно множество предшественников или множество последователей. Для этих функций аргументом может быть не только класс или объект, но и свойство.

7. Операторы PL/ODQL

Введем теперь полный перечень операторов PL/ODQL с описанием еще не рассмотренных.

<оператор PL/ODQL> ::= <оператор присваивания> | <оператор множества> |
<оператор ветвления> | <оператор цикла> | <оператор **BREAK**> |
<оператор **CONTINUE**> | <запрос-ODQL > | <оператор исключения>

<оператор присваивания> ::=
<имя переменной> := <выражение значения типа переменной>;

В *выражение значения типа переменной* могут входить определенные для этого типа бинарные операции, функции и скобки, обусловливающие порядок операций. Выражение, не имеющее типа переменной, должно быть преобразовано к нему.

<оператор множества> ::= <оператор **ADD**> | <оператор **EXCL**>

Семантика этих операторов описана в предыдущем разделе.

Оператор ветвления имеет следующий синтаксис:

<оператор ветвления> ::=
IF (<логическое выражение>)
{ <оператор PL/ODQL> | <блок >}
[**ELSIF** (<логическое выражение>)
{ <оператор PL/ODQL> | <блок >}] …
[**ELSE**
{ <оператор PL/ODQL> | <блок >}]

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 *оператор* или *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 *операторов* или *блоков* в пределах оператора. Синтаксис логического выражения определяется следующим образом:

<логическое выражение> ::=
<конъюнкция> | <логическое выражение> **or** <конъюнкция> |
<логическое выражение> **xor** <конъюнкция>

<конъюнкция> ::= <логич. элемент> | <конъюнкция> **and** <логич. элемент> <логический элемент> :=
<первичное логическое выражение> | **not** <логический элемент>

<первичное логическое выражение> ::=
<отношение выражений> | **true** | **false** | (<логическое выражение>)

<отношение выражений> ::=
<выраж. простого типа> <знак операции отношения> <выраж. того же типа> | <выраж. значения элемента типа множества> **in** <множество того же типа> | <выраж. множества> <знак операции отнош. множеств> <выраж. множества>

<знак операции отношения> ::= < | > | <= | >= | = | <>

<знак операции отнош. множеств> ::= <= | =

Перейдем к описанию синтаксиса и семантики операторов цикла.

<оператор цикла> ::= <цикл **WHILE**> | <цикл **FOR**> | <цикл **FOREACH**>

<цикл **WHILE**> ::= **WHILE** (<логическое выражение>) <тело цикла>

<тело цикла> ::= <оператор PL/ODQL> | <блок>

Цикл **WHILE** выполняется, пока истинно *логическое выражение*.

<цикл **FOR**> ::= **FOR** <параметр цикла>
from <начальное значение> **to** <конечное значение> <тело цикла>

Цикл **FOR** выполняется заданное число раз, пока значение параметра цикла принадлежит диапазону: от *начального значения* до *конечного значения* включительно. После каждой итерации цикла значение параметра цикла изменяется на единицу. Границы диапазона значений не могут изменяться внутри цикла. Переменную типа **integer**, используемую как параметр цикла, объявлять не обязательно, и в теле цикла ей можно задавать новое значение.

<цикл **FOREACH**> ::= **FOREACH** <параметр>
in <множество типа **SET**> <тело цикла>

<параметр> ::= <имя> | <имя индекса>

Если в качестве параметра используется необъявленное имя, то это представляет собой неявное объявление имени главного индекса множества, которое действует только в этом цикле. Элементы множества перебираются в порядке главного индекса множества. Если же параметр объявлен как *имя*

индекса, описанного ранее, то элементы множества перебираются в порядке этого индекса. Доступ к элементам в теле цикла осуществляется с помощью индекса (см. раздел 5).

Блок тела любого цикла может содержать операторы **CONTINUE** и **BREAK**. Первый из них ведет к выполнению следующей итерации тела цикла, а второй − к завершению выполнения цикла наибольшего уровня вложенности, в теле которого он выполняется.

8. Исключительные ситуации

Для описания исключительных ситуаций, которые могут возникнуть при выполнении кода, служит специальный тип переменных **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>]);

<код ситуации> ::= <константа типа **integer**>

<описание ситуации> ::= <константа типа **string**>

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

RAISE [**to call**] <переменная типа **EXCEPTION**>;

или для удобства в упрощенной форме:

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

Для обработки возникших уведомлений об исключительных ситуациях служит оператор **CATCH**, который должен находиться в конце ближайшего охватывающего блока PL/ODQL, где либо возникает исключительная ситуация (отсутствует опция **to call**), либо происходит вызов блока (при наличии опции **to call**), ведущего к исключительной ситуации. Его синтаксис таков:

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

{<оператор PL/ODQL> | <блок>}

Если указана *переменная описания ситуации*, то ее значением станет описание исключительной ситуации, указанное при создании переменной типа **EXCEPTION**, если оно было указано, или пустая строка (``"), если описание указано не было. Заранее определять *переменную описания ситуации* не требуется. Операторы *блока* используют *код ситуации* и *переменную описания ситуации* для обработки исключения: вывод сообщения из переменной и действия по продолжению (пустой блок), прекращение выполнения (оператор **BREAK**) или исправление ситуации ODQL-запросом.

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 должна быть продолжена работа над языком PL/ODQL, и для большего удобства в использовании он будет расширен другими конструкциями.

9. Пример

Проанализируем использование конструкций языка PL/ODQL на следующем примере из [7]. Рассматривается база данных кораблей (класс *Ships* с параметрами *IdShip, ShipName*), перевозящих грузы (класс *Cargoes* с параметрами *IdCargo, CargoName*) в порты (класс *Ports* с параметрами *IdPort, PortName*). Связь кораблей и грузов определяется включением с классом связи *Ships_Cargoes* (параметры *IdShip_Cargo, IdPort, Quantity*), который имеет родителем класс *Ports*. Требуется написать на PL/ODQL функцию *CargoesList*, возвращающую *список грузов, которые везут наиболее нагруженные корабли, идущие при этом в наименьшее число портов, за исключением наименее нагруженных кораблей, идущих при этом в наибольшее число портов*. Возвращение пустого списка является исключительной ситуацией, которую следует обработать в блоке вызова функции. Ниже приводится код части модуля *ShipsCargoesPorts*, в котором находится описание функции и обработчики возможного исключения.

ShipsCargoesPorts **BEGIN**

// *глобальные типы, константы и переменные модуля*

TYPESET <Cargoes> := **SELECT** *CargoName*;

TYPESET <Ships> := **SELECT OBJECT FROM** *Ships*;

```
EXCEPTION EmptyCargoesList 1, ``пустой список грузов'';
CONST SET <Cargoes> AllCargoes := SELECT CargoName;
CONST SET <Cargoes> EmptyCargoes := AllCargoes – AllCargoes;
……
FUNCTION CargoesList () Cargoes IS
  // локальные описания переменных
 SET <Ships>  Ships;
 SET <Ships> MaxSumCargoes :=
     ( SELECT objmaxsum (sc.Quantity) on s
       FROM Ships s, Ships_Cargoes sc, Cargoes c LINKS s contains (sc) c ) s1
  INTERSECTION
     ( SELECT objmincount (p.obj) on s1 FROM s1, Ports p
                                LINKS s1 contains (sc) c, p parent sc );
SET <Ships> MinSumCargoes :=
    ( SELECT objminsum (sc.Quantity) on s
       FROM Ships s, Ships_Cargoes sc, Cargoes c LINKS s contains (sc) c ) s1
  INTERSECTION
    ( SELECT objmaxcount (p.obj) on s1 FROM s1, Ports p
                                LINKS s1 contains (sc) c, p parent sc );
      // переменная списка грузов, упорядоченного по именам
  SET <Cargoes> List INDEX Name : CargoName;
     // блок функции
BEGIN
   IF ( MaxSumCargoes = MinSumCargoes ) RAISE to call EmptyCargoesList;
   Ships := MaxSumCargoes – MinSumCargoes;
   FOREACH Ship in Ships
     ADD  ( FOR obj.s=Ship SELECT CargoName.c
              FROM Cargoes c, Ships s, Ships_Cargoes sc
              LINKS  s contains (sc) c ) into List;
   RETURN  List;
END
END CargoesList
……..
SET <Cargoes> A := SELECT …
A := A + CargoesList ();
……..
CATCH 1, EmptyCargoesList RETURN EmptyCargoes;
……..
SET <Cargoes> B := SELECT …
B := B * CargoesList ();
……..
CATCH 1, EmptyCargoesList RETURN AllCargoes;
……..
END ShipsCargoesPorts
```

В этом примере обработка исключения производится различным образом в зависимости от места вызова функции.

Библиографический список

1. Гарсиа-Молина, Г. Системы баз данных. Полный курс [Текст] / Г. Гарсиа-Молина, Дж. Д. Ульман, Дж. Уидом. – Изд-во «Вильямс», 2003.

2. Дейт, К. Дж., Дарвен, Х. Основы будущих систем баз данных : третий манифест [Текст] / К. Дж Дейт, Х. Дарвен. – М. : Изд-во Янус-К, 2004.

3. Писаренко, Д. С. Объектное расширение PL/ODQL языка запросов ODQL для Динамической информационной модели DIM [Текст] / Д. С. Писаренко // Моделирование и анализ информационных систем. – 2008. – Т.15, №1, – С. 23–27.

4. Писаренко, Д. С., Рублев, В. С. Объектная СУБД Динамическая информационная модель DIM и ее основные концепции [Текст] / Д. С. Писаренко, В. С. Рублев // Моделирование и анализ информационных систем − 2009. – Т. 16, № 1. − С. 62−91.

5. Писаренко, Д. С., Рублев, В. С. Синтез аппарата взаимодействий системы управления базами данных DIM [Текст] / Д. С. Писаренко, В. С. Рублев // Материалы XVII международной школы-семинара «Синтез и сложность управляющих систем» имени академика О.Б. Лупанова −2008. − Новосибирск : Изд-во ИМ СО РАН, 2008. − С. 130–135.

6. Рублев, В. С. Запросная полнота языка ODQL динамической информационной модели DIM [Текст] / В. С. Рублев // Ярославский педагогический вестник. Естественные науки. – 2011. – Т. 3, № 1. − С. 69−75.

7. Рублев, В. С. Организация выполнения объектных запросов в динамической информационной модели DIM [Текст] / В. С. Рублев // Моделирование и анализ информационных систем. – 2011. – Т. 18, № 2. – С. 39–51.

8. Рублев, В. С. Язык объектных запросов динамической информационной модели DIM [Текст] / В. С. Рублев // Моделирование и анализ информационных систем. – 2010. – Т. 17, № 3. − С. 144−161.