ИПМ им.М.В.Келдыша РАН • Электронная библиотека

Препринты ИПМ • Препринт № 9 за 2007 г.

ISSN 2071-2898 (Print)
ISSN 2071-2901 (Online)

Н.Л. Брошкова

Об одном языке манипулирования данными

Рекомендуемая форма библиографической ссылки: Брошкова Н.Л. Об одном языке манипулирования данными // Препринты ИПМ им. М.В.Келдыша. 2007. № 9. 33 с.
https://library.keldysh.ru/preprint.asp?id=2007-9

РОССИЙСКАЯ АКАДЕМИЯ НАУК
ОРДЕНА ЛЕНИНА ИНСТИТУТ ПРИКЛАДНОЙ МАТЕМАТИКИ
ИМЕНИ М.В. КЕЛДЫША

Н.Л. Брошкова

ОБ ОДНОМ ЯЗЫКЕ МАНИПУЛИРОВАНИЯ ДАННЫМИ

Москва, 2007 г.

УДК 510.6

Брошкова Н.Л. Об одном языке манипулирования данными. Препринт Института прикладной математики им. М.В. Келдыша РАН. Москва, 2007 г.

Описывается язык манипулирования данными, основанный на принципах логического моделирования. Он объединяет две парадигмы программирования: логическую и операторную, логическая - служит для декларативного описания предметной области, операторная – используется, главным образом, при создании запросов к базам данных. Предлагаемый язык манипулирования данными содержательно более наглядный, чем стандартный язык запросов *SQL* Опыт показывает, что он являться эффективным средством решения задач над базами данных со сложными структурами.

Broshkova N.L. About one data manipulation language. Preprint of the Keldysh Institute of Applied Mathematics of RAS. Moscow, 2007.

The data manipulation language is described, founded on princes-groin of logical modeling. He unites two paradigms of the programming: logical and operational, logical - serves for declarative descriptions of the application domain, operational - is used for making request to database. The proposed data manipulation language given profound more demonstrative, than standard language SQL requests. Experience shows that it is an efficient facility of the decision of the tasks on data bases with complex structure.

Введение. При создании сложных приложений немыслимо обойтись без подсистемы обработки данных, предполагающей работу с типами данных наподобие отношений реляционных БД. В этом случае стандартные средства СУБД позволяют получать изящные решения, экономя тем самым ресурсы, как разработчика, так и вычислителя. Обычным средством создания приложений в СУБД является язык *SQL*, семантика которого базируется на реляционной алгебре.

С другой стороны, основываясь на принципах логического моделирования, удается построить язык манипулирования данными, содержательно более наглядный, чем *SQL*. Он объединяет две парадигмы программирования: логическую и операторную, логическая – служит для декларативного описания предметной области (ПО), операторная – используется, главным образом, при создании запросов к БД. Опыт показывает, что язык являться эффективным средством решения задач над БД со сложными структурами.

Приведем основные характеристики языка.

1) Его декларативная часть основана на логическом синтаксисе и служит для описания отношений ПО. При этом используется синтаксис языка первого порядка. А механизм порождения новых кортежей отношений базируется на операционной семантике логических формул (см. [1]).

2) Язык не имеет ограничений на используемые типы данных, чем характеризуется стандартный *SQL*. Разрешено определять новые типы, которые образуют частичный порядок. Это качество удобно при работе с ПО, обладающими широкими наборами типов данных.

3) Введено средство для описания эквивалентностей. Последнее эффективно, если иметь в виду ориентацию языка на решение задач в ПО со сложной структурой данных.

4) Предлагаемый язык манипулирования данными позволяет по спецификациям отношений ПО создать логическое исчисление, для которого

отношения БД служат моделью. Тем самым автоматически решается задача поддержания целостности БД.

5) Средство создания приложений языка включает как стандартные средства, имеющиеся в *SQL*, так и дополнительные, позволяющие создавать достаточно сложные приложения. В этом случае приложения представляются, в определенном смысле, расширением логического исчисления, описывающего отношения БД.

6) Отличительной чертой языка является присутствие операторного компонента, который служит для описания стратегий поиска решений. Его присутствие позволяет эффективно управлять выполнением декларативной части.

7) Язык основывается на основном принципе логического моделирования, формулируемом следующим образом: *для решения задачи достаточно описать ее в стандартных логических терминах*, т.е. представить в виде логического исчисления. Затем решение ищется автоматически, основываясь на операционной семантике логических формул.

Основная задача настоящей работы состоит в демонстрации того, как стандартные конструкции *SQL* реализуются средствами предлагаемого языка. Поэтому он выступает не только средством манипулирования данными, но и для написания приложений на основе хранящейся в БД информации.

1. Основные понятия, определения. Опишем основные понятия и особенности предлагаемого языка манипулирования данными. Решение всякой задачи с его помощью представляет собой определенные действия над отношениями БД. Действия представляются логическими формулами первого порядка, которые представляет собой аксиомы прикладного исчисления. В результате решение задачи над БД сводится к построению модели этого исчисления.

Каждое действие над отношениями БД имеет вид *продукции*:

ЕСЛИ <$условие_1$> ТО <$действие_1$> И

ЕСЛИ <$условие_2$> ТО <$действие_2$> И

. . .

ЕСЛИ <$условие_n$ > ТО <$действие_n$ >

над определенной сигнатурой. Сигнатура определяется типами данных, отношениями и функциями ПО, а также набором встроенных функций и отношений языка. Применение продукции состоит в проверке условий из левых частей импликаций и, при их выполнении, реализации соответствующих действий, которые сводятся к добавлению или исключению кортежей отношений БД.

Поиск решения состоит в порождении кортежей из элементов ПО и построении интерпретации логического исчисления. Особенность состоит в том, что построение интерпретации осуществляется под управлением стратегии, описывающей порядок применения продукций. Это дает возможность манипулировать продукциями и элементами отношений БД. Стратегия описывается на языке управления решением, базисные конструкции которого позволяют манипулировать продукциями и кортежами отношений БД. Манипулирование продукциями сводится к их запуску в определенные моменты, а манипулирование отношениями БД – к удалению и добавлению кортежей в соответствии с определенными условиями.

Окончательное решение представляет собой отношения БД, на которых выполняются условия, описанные в спецификации задачи. Тем самым, решение задачи в виде построения модели прикладного исчисления определяется двумя компонентами: декларативной частью - описывающей понятия ПО и соотношения между ними, и процедурной – управляющей решением.

Предлагаемая методика решения как построение модели прикладного исчисления сводится, во-первых, к описанию аксиоматики ПО и, во-вторых, к определению схемы перебора, используемой при построении искомой модели. Модель построена, если для нее выполняются условия, которым должно удовлетворять заключительное состояние задачи.

2. Язык описания предметной области. Язык описания ПО представляет собой многосортный язык первого порядка. Опишем иерархию из двух уровней его конструкций, так, что конструкции более высокого уровня строятся на основе низших.

Конструкции нулевого уровня – *имена* суть обычные идентификаторы, т.е. это последовательности букв (латиница и кириллица) и цифр, а также специальных знаков, начинающиеся с буквы или специального знака (заглавные и строчные буквы различаются).

Конструкции первого уровня (так называемые индивидные конструкции) включают *константы*, *переменные*, *функции*, из них строятся индивидные термы и предикаты.

Описание ПО состоит из нескольких разделов, в каждом из которых описываются его составляющие, необходимые для последующих вычислений. Основные конструкции – *продукции* суть формулы многосортного исчисления первого порядка. Они определяют прикладное исчисление.

Первым разделом описания ПО является раздел *DECLARATIONS*, в котором перечисляются все элементы сигнатуры. Это суть сорта переменных, констант и функций; константы и функции, используемые для описания ПО,

а также предикаты, в числе которых особо выделяются эквивалентности. Раздел *DECLARATIONS* выглядит следующим образом:

DECLARATIONS

SORTS < описание сортов и порядка на них>

CONSTANTS <объявление констант>

PREDICATES <объявление предикатов>

FUNCTIONS <объявление функций>

EQUIVALENCES <объявление эквивалентностей>

В каждом из разделов определяются соответствующие конструкции. Рассмотрим их по порядку.

Сорт есть идентификатор из фиксированного для данной ПО множества. На сортах определен частичный порядок "<" таким образом, что они образуют решетку с одним наибольшим и одним наименьшим элементами (соответственно, *TERM* и *NUL*).

Сорта объявляются в разделе *SORTS* следующим образом: *сорт*1 < *сорт*2, что означает наличие частичного порядка *сорт*1 < *сорт*2.

***Пример* 1.** Определение типов треугольников может выглядеть следующим образом:

SORTS

TERM < треугольник < прямоугольный< NUL;

TERM < треугольник < равнобедренный < равносторонний < NUL;

Таким образом, сорт «*равнобедренный треугольник*» является частным случаем сорта «*треугольник*», а «*равносторонний треугольник*» - частным случаем «*равнобедренный треугольник*».

Используются сорта с фиксированным значением: *TERM, NUL, INT, RAT, STR. TERM* – самый общий сорт, любой другой сорт есть частный случай сорта *TERM*; *NUL* – пустой сорт, является частным случаем любого другого сорта; *INT* – целые числа, константа с этим сортом записывается как

INT.n, где *n* – целое число; *RAT* – рациональная дробь, константа с этим сортом записывается как *RAT.n/m,* где *n, m* – целые числа, *n* – числитель, *m* – знаменатель; *STR* - символьные переменные и константы, константа с этим сортом записывается как *STR.strins,* где *strins* – произвольная последовательность символов.

Константы служат для обозначения объектов ПО. Каждая константа характеризуется сортом и именем. Объявление константы производится либо ее записью в разделе *CONSTANTS,* либо появлением в продукциях. Записываются константы следующим образом:

имя_сорта **.** *имя_константы.*

Каждая константа однозначно определяется своим именем, у различных констант не может быть одного имени. Но разные вхождения одной константы могут иметь разные сорта. В этом случае сорта должны быть сравнимы относительно частичного порядка, заданного на сортах данной ПО.

***Пример* 2.** Если БД имеет отношение к геометрии и сорта задают типы треугольников, то допустимо следующее описание констант:

DECLARATIONS

SORTS

TERM < *треугольник* < *прямоугольный*< NUL;

TERM < *треугольник* < *равнобедренный* < *равносторонний* < *NUL*;

CONSTANTS

треугольник.ABC

равнобедренный.EFG

прямоугольный.ABC

В последующем константа *треугольник.ABC* может входить в таблицы или в логические формулы как *треугольник.ABC, равносторонний.ABC, прямоугольный.ABC* или *равнобедренный.ABC*. В процессе работы программы

могут возникнуть другие константы с этими сортами, например, *треугольник.DEF* или *прямоугольный.PQR*.

Переменные, как и обычные индивидные переменные в логических формулах, служат для обозначения элементов ПО. Как и константы, они определяются сортом и именем, но в отличие от констант переменные заранее не определяются. Они используются только в продукциях, где и определяются. Каждая переменная является локальной для отдельной продукции. Поэтому переменные с одинаковыми именами в разных продукциях считаются независимыми. Записываются переменные в виде ***&*** *сорт* **.** *имя.*

***Пример* 3.** В следующей продукции:

NAME: *pr1*

KEYS: *P*(*&INT.x, &INT.y, Flag.U*)

COND: (*&INT.x >= INT.*1) *AND* (*&INT.y >= INT.*1) *AND*

(*&INT.x = INT.*1) *OR* (*&INT.x = &INT.y*)

CONC: *P*(*&INT.x - INT.*1, *&INT.y - INT.*1, *Flag.R*)

COM: *Расширение отношения путем добавления кортежа*

END

Переменные *&INT.x* и *&INT.y* определяются в ключах продукции. Затем они используются в условии (раздел *COND*) и в заключении (раздел *CONC*) продукции. Здесь же используются две константы *Flag.U* и *Flag.R.*

Для описания ПО предусмотрены предикаты двух типов – *вычисляемые* и *символические*. Первые служат для установления традиционных отношений между величинами ПО, например равенства, порядка, включения и т.п. Вторые фиксируют новые отношения, которые характерны для ПО и используются в процессе поиска решения. В каждом состоянии эти предикаты интерпретируются одноименными таблицами: каждое отношение - единственной таблицей, имя таблицы совпадает с именем предиката. Поиск решения сводится к доопределению таблиц с учетом определенных условий. В каж-

дом состоянии таблицы содержат интерпретации одноименных предикатов, определяя, тем самым, интерпретацию всех логических выражений.

Предикаты определяются перечислением в разделе *PREDICATES* их имен и следующих за ними строк аргументов. Определения предикатов необходимо для задания начальных условий задачи, например, описание отношения БД при начальном ее вводе. Для их записи предусмотрены средства в специализированном редакторе.

***Примера* 4.** Раздел описаний может выглядеть следующим образом.

DECLARATIONS

SORTS

TERM < Flag < NUL;

CONSTANTS

Flag.U

Flag.R

PREDICATES

P(*INT*.3, *INT*.3, *Flag.U*)

FUNCTIONS

EQUIVALENCES

PRODUCTIONS

Вычисляемые предикаты используются в условиях продукции. Каждому из них соответствует процедура, вычисляющая значение *истина* или *ложь* в зависимости от значений аргументов. Так, равенство записывается стандартным образом: *терм*1 = *терм*2, ему соответствует процедура, результатом которой есть *истина*, когда *терм*1 совпадает с *терм*2 и *ложь* – в противном случае.

Основной набор таких предикатов включает – равенство (“=”), сравнения (“>”, “>=”, “<”, “<=”) и неравенство (“!=”).

Для образования условий используются обычные логические операторы

AND, OR, XOR, NOT.

***Пример* 5.** Термы *INT*.2 и *RAT*.4/2 – равны, то есть предикат *INT*.2 = *RAT*.4/2 после вычисления принимает значение *истина*. Термы *SORT*.2 и *SORT*.4/2 не равны, поэтому предикат *SORT*.2 = *SORT*.4/2 принимает значение *ложь*. Истинны предикаты *INT*.2 < *RAT*.5/1, *SORT*.2 >= *SORT*.5/1, *namec*(*треугольник.ABC*) != *namec*(*треугольник.DEF*).

***Пример* 6.** В условной части продукции из примера 3 используются вычислимые предикаты равенства и сравнения.

Функции так же, как и предикаты, бывают вычислимые и символические. Назначение символических функций заключается в единообразном представлении сложных объектов предметной области - стеков, деревьев, схем и т.п., поэтому они не обладают иным значением кроме их собственного вида. Символические функции определяются в разделе *FUNCTIONS* так: *сорт_функции*: *имя_функции***(***список_сортов_аргументов***).**

Вычисляемые функции реализуются встроенными процедурами. С их помощью реализованы операции сложения, вычитания, деления и умножения (“+”, “-”, “:”, “*”). Функции – аналоги математических операций записываются в принятой форме. Функции, реализующие не стандартные математические операции, записываются как имя функции, за которым в круглых скобках следует список аргументов. Некоторые функции обладают не фиксированным числом аргументов.

***Пример* 7**. Функция *set*() с не фиксированным количеством аргументов определяет упорядоченное множество из своих аргументов, а при помощи функции *firstfromset*(), единственным аргументом которой должна быть функция *set*() – получить первый элемент множества. Результат вычисления функции *firstfromset*(*set*(*SORT.A, SORT.B, SORT.C*)) равен константе *SORT.A.*

В последующем потребуется понятие *терма*, которое будем использовать в традиционном математическом смысле: терм есть суперпозиция

функций, констант и переменных. Чтобы терм был *правильным,* должно выполняться требование соответствия сортов подставляемых констант и переменных сортам аргументов функций.

***Пример* 8.** Термы “*INT*.1*”, ”INT*.1+*RAT*.12/5*”* – правильные. Терм “*INT*.1 + *треугольник.ABC”* – неправильный.

Каждое отношение *эквивалентности* задается идентификатором, оно определяет соответствующие классы эквивалентности. По свойству рефлексивности, любой терм эквивалентен сам себе и поэтому образует класс эквивалентности, состоящий из одного элемента. Эквивалентности являются полезным инструментом для представления множеств элементов, обладающих общим свойством.

Продукции – это, с одной стороны, логические формулы, но, с другой, выступают как операторы для порождения новых строк таблиц, интерпретирующих символические предикаты. Каждая продукция в традиционном логическом формализме выглядит так:

$$\Gamma \Rightarrow (F_1 \Rightarrow A_1) \,\&\, \ldots \,\&\, (F_n \Rightarrow A_n).$$

Здесь $\Gamma = (L_1 \,\&\, \ldots \,\&\, L_m)$, где L_i , $i = 1, 2, \ldots, m$ суть литеры, образованные из символических предикатов (они называются *ключами* продукции), аргументами которых служат переменные, константы или термы, образованные с помощью символических функций. Эта конъюнкция есть, в определенном смысле, глобальная посылка всей продукции. Переменные из положительных литер, входящих в Г, называются *определяемыми*. В каждом отрицательном ключе, по меньшей мере, одна индивидная переменная должна быть определена. Ключами продукции не могут служить эквивалентности.

F_i, $i = 1, 2, \ldots, m$ суть бесквантворные логические формулы, образованные из вычисляемых и символических предикатов и отношений эквивалентности. Их все индивидные переменные определены в Г. Каждая из формул F_i представляет собой *условие*, которое проверяется прежде, чем будет осу-

ществлено действие, описанное в A_i , i = 1, 2, ..., m. При этом вычисление условия происходит следующим образом.

Вычисляемые предикаты реализованы соответствующими процедурами, поэтому по каждому набору аргументов выдают значение *истина* или *ложь*. Для символических предикатов проверяется присутствие строки аргументов в соответствующей таблице. Если кортеж в таблице присутствует, то предикат также принимает значение *истина*, в противном случае – *ложь*. Если аргументом условия служит эквивалентность, то проверяется присутствие двух термов в одном классе эквивалентности. Присутствие их в одном классе влечет значение истина, отсутствие – *ложь*. В результате этих действий каждое условие принимает в точности одно значение – *истина* или *ложь*.

A_i, i = 1, 2, ..., m суть литеры, образованные из символических предикатов, чьи индивидные переменные так же, как и в F_i, определены в Г. Они представляют собой заключения продукции, каждое заключение выполняется , если соответствующее условие истинно.

Продукции задаются следующим образом:

NAME: *имя продукции*

KEYS: *список ключей продукции*

COND: *условие*$_1$

CONC: *заключение*$_1$

. . .

COND: *условие*$_m$

CONC: *заключение*$_m$

COM: *содержательный комментарий*

END

Имя продукции используется для различения продукций.

Ключи продукции есть список символических предикатов или их отрицаний с аргументами в виде термов. Вместо несущественных аргументов можно подставлять символ _ (подчеркивание). Так, содержимое ключа *P*(_,

_, &сорт$_1$.*имя*$_1$) может унифицироваться с любой строкой таблицы *P*, в которой на первом и втором месте может стоять любой терм, а на третьем - терм, сорт которого являющийся подсортом сорта сорт$_1$ или самим этим сортом.

Условия представляют собой бесквантворные логические формулы, состоящие из вычисляемых функций, предикатов, аргументами которых служат термы, состоящие лишь из вычисляемых функций и переменных ключей. Для записи логических условий используются стандартные логические операторы *AND, OR, NOT.*

Заключения представляет собой наборы литер, образованных из символических предикатов, на аргументных местах которых стоят произвольные термы, зависящие от переменных, определенных в ключах продукции. Вычисление продукции состоит в том, что в случае выполнения условия в одноименную таблицу добавляется эта строка термов с соответствующими подстановками на места переменных (случай положительного вхождения) или исключается строка (случай отрицательного вхождения). При этом предикат может быть не описан в разделе *PREDICATES*, если соответствующая таблица изначально пуста.

В заключительной части

CONC: *P*(*&INT.x - INT.*1, *&INT.y + INT.*1, *INT.*0)

продукции происходит изменение таблицы *P*. При этом добавляемая строка определяется строкой аргументов.

Комментарий используется в объясняющем компоненте системы, назначение которого состоит в объяснении найденного решения, определяя траектории изменения таблиц и комментируя каждое действие.

Порождение новой информации продолжается по правилам, задаваемым стратегиями, описанными на языке управления решением. В частности, для отдельно взятой продукции оно может осуществляться один раз до тех пор, пока не будет достигнута неподвижная точка (т. е. не порождается новая

информация, и поэтому каждое новое применение этой продукции не создает новых строк для соответствующих таблиц) или не будет окончено выполнение стратегии, например, по достижении предиката *END*() и т.п.

3. Команды и операторы языка управления решением. Язык управления решением (ЯУР) служит для описания стратегий реализации запросов и является *метаязыком* по отношению к ЯОП, так как данными для ЯУР, в том числе, являются и конструкции ЯОП. В итоге каждая программа состоит из двух фрагментов, описанных на языках описания ПО и управления решением. Если ЯОП задает аксиоматику прикладного исчисления первого порядка, то ЯУР построен по типу обычного операторного языка. Для управления решением в нем имеются стандартные конструкции алгоритмического языка высокого уровня, например условные операторы, операторы цикла и перехода по метке, и набор оригинальных операторов, предназначенных для манипулирования содержимым таблицами и выполнения продукций. Ниже перечислены все наиболее значимые конструкции, содержащиеся в ЯУР. В следующем разделе приводятся конкретные примеры формирования запросов средствами разработанного языка.

FIXEDPOINT(<*имя продукции*>)	Истина, если после очередного вычисления указанной продукции не было зафиксировано изменений в таблицах, соответствующих предикатам заключения этой продукции
EMPTY(<*имя предиката*>)	Истина, если таблица пуста
HAVE(<*имя предиката*>, <*строка аргументов*>)	Истина, если соответствующая таблица содержит указанную строку аргументов

Для непосредственной работы с таблицами используются так называемые *директивные* операторы, предназначенные для варьирования содержимого таблиц символических предикатов. Они позволяют изменять, очищать, копировать и перемещать строки из соответствующих таблиц.

В зависимости от выполняемого действия директивный оператор записывается в одном из следующих двух шаблонов:

(1) <*действие*$_1$> < *место_действия*$_1$> *IN* <*предикат*$_1$> *TO*

<место_действия$_2$> IN <предикат$_2$>

(2) *<действие$_2$> <место_действия$_1$> IN <предикат$_1$>*

При этом,

действие$_1$ ::= *MOVE | COPY,*

действие$_2$ ::= *ERASE|CUT|PASTE|ADD* –

определяют выполняемое действие (перемещение, копирование, очищение, вырезка в буферную таблицу, вставка из буферной таблицы и добавление строк соответственно).

место_действия ::= *FIRSTLINE|LASTLINE|ALL|*

FROM <разделитель> UPTO <разделитель> |

FROM <разделитель> DOWNTO <разделитель> |

MATCH <строка аргументов>|

NOT MATCH<строка аргументов> | BEFORE <разделитель> |

AFTER <разделитель>| BEFORE <строка аргументов> -

определяет строки, над которыми выполняется *действие*. Для разных действий множество допустимых мест действия различно. Семантика определения места действия и допустимые конфигурации директивного оператора описаны дальше в таблицах.

Описание места действия директивного оператора

Синтаксис	*Сокращенное обозначение*	*Семантика, в зависимости от типа таблицы*	
		Таблица-источник	*Таблица-приемник*
FIRSTLINE	*FL*	Первая строка	Вместо первой строки
LASTLINE	*LL*	Последняя строка	Вместо последней строки
ALL	*AL*	Все строки	Вместо всех строк
FROM DOWNTO	*DN*	От первого разделителя до второго сверху вниз	Вместо строк от первого разделителя до второго сверху вниз
FROM UPTO	*UP*	От первого разделителя до второго снизу вверх	Вместо строк от первого разделителя до второго снизу вверх
MATCH (*строка аргументов*)	*MA*	Строки, совпадающие с маской	Вместо совпадающих с маской строк
NOT MATCH	*NM*	Не совпадающие с мас-	Вместо несовпадающих с мас-

(*строка аргументов*)		кой строки	кой строк

Осуществляемые действия и способы указания места в таблицах позволяют вырезать, копировать и перемещать строки из таблицы в таблицу и из таблицы в буфер и обратно. Выполнение этого оператора возможно также для части таблицы, удовлетворяющей определенному условию. Если действие директивного оператора затрагивает сразу несколько таблиц, то без особых оговорок его можно выполнять только над таблицами с одинаковым числом столбцов (т.е. одинаковой ширины).

Допустимые конфигурации операторов вида (1).

Действие	*Допустимое место_действия*$_1$	*Допустимое место_действия*$_2$
COPY, MOVE	*FL, LL AL, UP, DN, MA, NM, BE, AF*	*FL, LL, AL, UP, DN, MA, NM, BE, AF AL, UP, DN, MA, NM, BE, AF*

Допустимые конфигурации операторов вида (2).

Действие	*Допустимое место_действия*1
ERASE, CUT	*FL, LL, AL, UP, DN, MA, NM, BE, AF*
PASTE, ADD	*AL, UP, DN, MA, NM, BE, AF*

Строка аргументов в директивном операторе состоит из констант и символа “_” (подчеркивание). Символ “_” определяет замещаемые (немаскируемые) места в маске.

Действия директивных операторов следующие:

ADD <строка>	Добавляет новую строку <*строка*>, описанную на языке описания предикатов, в таблицу, над которой осуществляется действие. Добавляемая строка не может содержать переменных и иметь не соответствующее ширине таблицы количество элементов. Например, оператор *ADD* (s_1.*NAME*$_1$, s_2.*NAME*$_2$) *BEFORE*_START *IN pred*$_1$ добавляет строку s_1.*NAME*$_1$, s_2.*NAME*$_2$ в начало таблицы *pred*$_1$.
ERASE	Удаляет указываемый фрагмент таблицы. Например, оператор *ERASE ALL IN bit* удалит все строки из таблицы *bit*.

COPY	Копирует указываемый фрагмент одной таблицы в другую таблицу. Например, оператор *COPY* ON <*условие*$_1$> *MATCH IN* *пред*$_1$ TO *ALL IN пред*$_2$ копирует строки, удовлетворяющих условию *условие*$_1$ из таблицы *пред*$_1$ в конец таблицы *пред*$_2$. Зарезервированное слово BUFFER в качестве имени указывает, что копировать фрагмент требуется в буфер. Например, оператор *COPY LASTLINE IN pred*$_1$ *TO* BUFFER скопирует в буфер последнюю строку предиката *pred*$_1$. Если таблица *pred*$_1$ пуста, содержимым буфера будет пустая таблица такой же ширины, что и таблица *pred*$_1$. Для копирования содержимого одной таблицы в другую используется оператор *COPY ALL IN pred*$_1$ *TO pred*$_2$ Он скопирует все строки таблицы *pred*$_1$ в таблицу *pred*$_2$
MOVE	Перемещает указываемый фрагмент таблицы и добавляет в таблицу с именем <*имя*$_1$>.

Выделение *места действия* в директивных операторах осуществляется одним из следующих способов:

ALL	Указывает всю таблицу, с первой по последнюю строку. В рассматриваемом примере директива *ERASE IN Tabl ALL* очищает таблицу *Tabl*.
FROM a_1 *TO* a_2	Указывает фрагмент таблицы от разделителя a_1 до разделителя a_2. Способ установки и работа с разделителями описаны ниже. Вместо a_1 можно поставить *TOP*, т.е. 1, вместо a_2 - BOT*TOM*, т.е. номер ее последней записи
FIRSTLINE и *LASTLINE*	Указывают в качестве места действия оператора первую и последнюю строки таблицы соответственно
LINE_BEFORE a_1 и *LINE_AFTER* a_1	Указывают в качестве места действия оператора единственную строку предиката соответственно перед разделителем и после разделителя a_1

***Пример* 9.** Пусть исходная таблица *pred* имеет следующий вид:

(*S.A, S.B, S.C*)

(*S.A, S.C, S.C*)

(*S.A, S.D, S.C*)

(*S.A, S.E, S.A*)

(*S.A, S.F, S.A*)

Применение директивного оператора

ERASE IN pred ON pred(_,*S.D*,_) *MATCH*

изменит эту таблицу на

(*S.A, S.B, S.C*)
(*S.A, S.C, S.C*)
(*S.A, S.E, S.A*)
(*S.A, S.F, S.A*)

т.е. будет удалена третья строка.

Оператор

ERASE IN pred ON pred (*S.A*, _, _) *MATCH*

очистит таблицу целиком, так как в каждой строке на первом месте расположена константа *S.A*.

Продуктивные операторы осуществляют выполнение продукций. Все они построены по следующей схеме:

RUN <имя_продукции> <тип выполнения> <место выполнения>.

Тип выполнения задает, как выполняется продукция - один раз или несколько. Типы выполнения описаны в следующей таблице.

UNTIL_FIX <*предикат*>	Выполнять до достижения таблицей неподвижной точки, соответствующей указанному предикату.
ONCE	Выполнить один раз. По умолчанию, если после имени продукции не стоит ONCE, она также выполняется один раз
FOR a_1	Выполнить один раз для строки, на которую указывает разделитель a_1
DEEPTO *имя_предиката* (*строка*)	Выполнить в глубину, до достижения указанной строки или построения неподвижных точек всех предикатов, перечисленных в заключениях продукции
DEEP *число*	Выполнить в глубину, *число* раз или до построения неподвижных точек всех таблиц, перечисленных в заключениях продукции

Место выполнения указывает для каждой таблицы тот фрагмент, который унифицируется с соответствующим предикатом из ключей продукции. Эта часть продуктивного оператора имеет такой же вид, как соответствующая часть для директивного оператора, т.е. используются конструкции вида

IN имя_таблицы место_в_таблице.

Из директив, указывающих место действия, используются директивы типа *FROM* a_1 *TO* a_2, где a_1 и a_2 суть разделители, *FIRSTLINE* и *LASTLINE*. Тем самым указывается фрагмент таблицы либо от разделителя a_1 до разделителя a_2, либо первая, или последняя записи таблицы. Вместо a_1 можно использовать *TOP*, т.е. указание на первую запись таблицы, вместо a_2 - *BOTTOM*, т.е. номер ее последней записи. По умолчанию предполагается, что место действия есть вся таблица.

4. Примеры запросов к БД. Приведем примеры моделирования средствами предлагаемого языка наиболее распространенных операторов языка *SQL*. Как показывает опыт, такая реализация запросов нагляднее, чем на стандартном *SQL*. При кажущейся громоздкости программ они прозрачны с содержательной точки зрения и поэтому легко воспринимаются. Помимо этого язык обладает рядом свойств, которые позволяют говорить о его большей выразительной возможности, чем стандартный *SQL*. Поэтому реализация СУБД на принципах логического моделирования является интересной альтернативой общепринятому подходу.

Чтобы лучше ориентироваться в запросах, приведем схему БД, к которой они относятся (рис. 1).

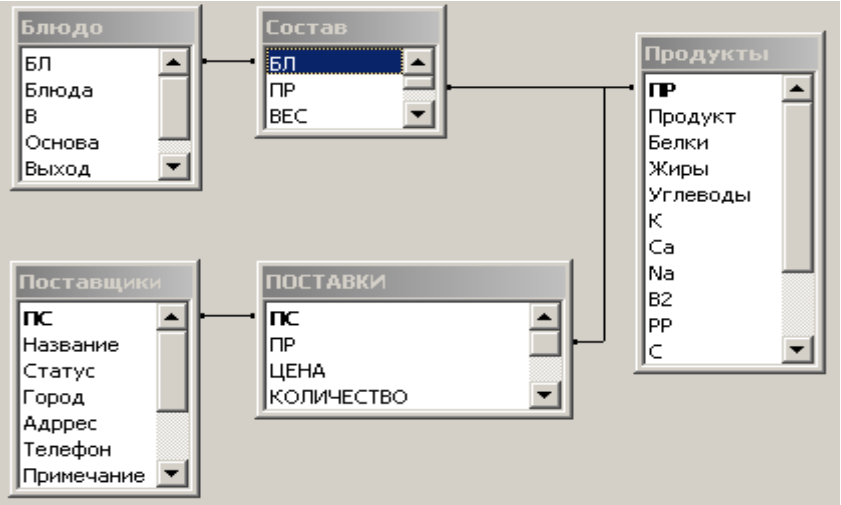

Рис. 1

Пример* 10**. Моделирование оператора ***Max

Запрос на *SQL*:

SELECT MAX(Выход)

FROM Блюда

Предлагаемая программа на языке:

NAME: prMax

KEYS: ВремБлюдо(*&INT*.A, &Блюда.B, &B.C, &Основа.D, *&INT*.E, *INT*.F)

Блюдо(*&INT*.A1, &Блюда.B1, &B.C1, &Основа.D1, *&INT*.E1, &*INT*.F1)

COND: (&*INT*.E > &*INT*.E1)

CONC: *NOT* Блюдо(&*INT*.A1,&Блюда.B1,&B.C1,&Основа.D1,&*INT*.E1, &*INT*.F1)

COM: Моделирование оператора MAX. ВремБлюдо – вспомогательная таблица. Ее аргументы упорядочены так же, как в таблице Блюдо

END

BEGIN

L1: *MOVE FIRSTLINE IN* Блюдо *TO ALL IN* ВремБлюдо

RUN prMax

IF EMPTY(Блюдо) *THEN GOTO* L2 *ELSE GOTO* L1

L2: *PRINT* ВремБлюдо

END.

Пример* 11**. Моделирование оператора ***Count

Запрос на *SQL*:

SELECT COUNT (БЛ)

FROM Блюдо

Предлагаемая программа:

NAME: prCount

KEYS: ВремБлюдо(&*INT*.A, &Блюда.B, &B.C, &Основа.D, &*INT*.E, &*INT*.F)

SummCount(&*INT*.U)

COND:

CONC: SummCount(&*INT*.U + *INT*.1) *NOT* SummCount(&*INT*.U)

COM: Моделирование оператора Count

END

BEGIN

L1: *MOVE FIRSTLINE IN* Блюдо *TO ALL IN* ВремБлюдо

RUN prCount

IF EMPTY(Блюдо) *THEN GOTO* L2 *ELSE GOTO* L1

L2: *PRINT* SummCount

END.

Пример* 12**. Моделирование оператора ***GROUP BY. Требуется вычислить общую массу каждого из продуктов, поставляемых в настоящее время поставщиками.

SELECT Основа, ***SUM***(Выход) ***AS*** *SUM*

FROM Блюдо

GROUP BY Основа

Предлагаемая программа:

NAME: prОснованСумм1

KEYS:

ВремБлюдо(&*INT*.A, &Блюда.B, &B.C, &Основа.D, &*INT*.E, &*INT*.F)

COND:

CONC: ОснованСумм(&Основа.D, *INT*.0)

COM: Сумма инициализируется значением 0.

END

NAME: prОснованСумм2

KEYS: SummWieght(&*INT*.A) ОснованСумм(&Основа.X, *INT*.0)

COND:

CONC: ОснованСумм(&Основа.X, &*INT*.A)

NOT ОснованСумм(&Основа.X, *INT*.0)

COM:

END

NAME: prGroup_by

KEYS: Блюдо(&*INT*.A, &Блюда.B, &B.C, &Основа.D, &*INT*.E, &*INT*.F)
ВремБлюдо(&*INT*.A1, &Блюда.B1, &B.C1, &Основа.D, &*INT*.E1, &*INT*.F1)

COND:

CONC: ВремБлюдоAVR(&*INT*.A, &Блюда.B, &B.C, &Основа.D, &*INT*.E, &*INT*.F) *NOT* Блюдо(&*INT*.A, &Блюда.B, &B.C, &Основа.D, &*INT*.E, &*INT*.F)

COM: Моделирование оператора ***GROUP BY***. Таблица ВремБлюдо содержит одну запись, поэтому в таблице ВремБлюдоAVR включены все записи с одним признаком Основа.

END

NAME: prSummaWieght

KEYS: ВремБлюдо(&*INT*.A1, &Блюда.B1, &B.C1, &Основа.D, &*INT*.E1, &*INT*.F1) ОснованСумм(&Основа.D, &*INT*.S0)

COND:

CONC: ОснованСумм(&Основа.D, &*INT*.E + &*INT*.S0)

NOT ОснованСумм(&Основа.D, &*INT*.S0)

COM: Суммирует выход продукта в одной группе.

END

BEGIN

ERASE ALL IN ВремБлюдоAVR

ERASE ALL IN ОснованаСумм
L1: *COPY FIRSTLINE IN* Блюдо *TO ALL IN* ВремБлюдо
RUN prGroup_by
COPY FIRSTLINE IN ВремБлюдоAVR *TO ALL IN* ВремБлюдо
RUN prОснованаСумм1
L2: *MOVE FIRSTLINE IN* ВремБлюдоAVR *TO ALL IN* ВремБлюдо
RUN prSummaWeight
ERASE ALL IN ВремБлюдо
/* суммирование в пределах одной группы Основа */
IF EMPTY(Блюдо) *THEN GOTO* LE *ELSE GOTO* L1
LE: *PRINT* ОснованаСумм
END.

***Пример* 13**. Найти поставщиков помидоров, которые в БД являются продуктом с номером 11.

```
SELECT   Название, Статус
FROM     Поставщики
WHERE    ПС IN
      (SELECT ПС
      FROM    Поставки
      WHERE   ПР IN
      (SELECT ПР
      FROM    Продукты
      WHERE   Продукт = 'Помидоры'));
```

Предлагаемая программа:
NAME: pr_Temp_PR_0
KEYS: Продукты(&ПР.A, Продукт.Помидоры, &Бел._, &Жир._, &Угл._, &K._, &Ca._, &Na._, &B2._, &PP._, &C._) Поставки(&ПС.A2, &ПР.A, &*INT*.C4, &*INT*._) Поставщики(&ПС.A2, &Название.B3, &Статус.D3, &Город.E3, &Адрес.F3, &Тел.G3)
COND:

CONC: *FINTabl*(&Название.B3, &Статус.D3)

COM:

END

RUN pr_Temp_PR_0

PRINT FINTabl

END.

***Пример* 14**. Выдать номера поставщиков, находящихся в том же городе, что и поставщик с номером 6.

SELECT ПС

FROM Поставщики

WHERE Город =

(***SELECT*** Город

FROM Поставщики

WHERE ПС = 6);

Предлагаемая программа:

NAME: pr_Town

KEYS: Поставщики(ПС.6, &Название.B, &Статус.C, &Город.D, &Адрес.E, &Тел.F)

COND:

CONC: *TempTown*(&Город.D)

COM:

END

NAME: prFIN

KEYS: Поставщики(&ПС.A,&Название.B,&Статус.C,&Город.D,&Адрес.E, &Тел.F) *TempTown*(&Город.D)

COND:

CONC: *FINPS*(&ПС.A)

COM:

END

***Пример* 15**. Выдать название и статус поставщиков продукта с номером 11.

SELECT Название, Статус
FROM Поставщики
WHERE 11 ***IN***
(***SELECT*** ПР
FROM Поставки
WHERE ПС = Поставщики.ПС);

Предлагаемая программа:

1-й вариант.

NAME: pr_Temp_PR
KEYS: TempPostavchik(&ПС.A, &Название.B, &Статус.C, &Город.D, &Адрес.E, &Тел.F) Поставки(&ПС.A, &ПР.B1, &*INT*.C1, &*INT*.D1)
COND:
CONC: TempPR(&ПР.B1)
COM:
END
NAME: pr_Temp_PR1
KEYS: TempPostavchik(&ПС.A, &Название.B, &Статус.C, &Город.D, &Адрес.E, &Тел.F) TempPR(ПР.11)
COND:
CONC: *FINTabl*(&Название.B, &Статус.C)
COM:
END
BEGIN
ERASE ALL IN FINTabl
L2: *ERASE ALL IN* TempPR
MOVE FIRSTLINE IN Поставщики *TO ALL IN* TempPostavchik
RUN pr_Temp_PR
PRINT TempPR

RUN pr_Temp_PR1

IF EMPTY(Поставщики) *THEN GOTO* L3 *ELSE GOTO* L2

L3: *PRINT FINTabl*

END.

2-й вариант.

NAME: pr_Temp_PR

KEYS: TempPostavchik(&ПС.A, &Название.B, &Статус.C, &Город.D, &Адрес.E, &Тел.F) Поставки(&ПС.A,ПР.11,&*INT*.C1,&*INT*.D1)

COND:

CONC: *FINTabl*(&Название.B, &Статус.C)

COM: Из поставщиков название и статус с признаком ПР = 11

BEGIN

ERASE ALL IN FINTabl

L2: *ERASE ALL IN* TempPR

MOVE FIRSTLINE IN Поставщики *TO ALL IN* TempPostavchik

RUN pr_Temp_PR

IF EMPTY(Поставщики) *THEN GOTO* L3 *ELSE GOTO* L2

L3: *PRINT FINTabl*

END.

***Пример* 16**. Выдать номера всех продуктов, поставляемых только одним поставщиком.

SELECT DISTINCT *X*.ПР
FROM Поставки *X*
WHERE *X*.ПР ***NOT IN***
(***SELECT*** *Y*.ПР
FROM Поставки *Y*
WHERE *Y*.ПС <> *X*.ПС);

Предлагаемая программа:

NAME: pr_Temp_PR_0

KEYS: TempPostavki(&ПС.A, &ПР.B, &*INT*.C, &*INT*.D)

MidPostavki(&ПС.A1, &ПР.B1, &*INT*.C1, &*INT*.D1)

COND: (&ПС.A != &ПС.A1)

CONC: TempPR(&ПР.B1)

COM: выделяем все продукты, которые не поставляет поставщик &ПС.A. Таблица TempPostavki сдержит одну запись, а MidPostavki – все записи из таблицы Поставки.

END

NAME: pr_Temp_PR_1

KEYS: TempPostavki(&ПС.A, &ПР.B, &*INT*.C, &*INT*.D) TempPR(&ПР.B)

COND:

CONC: Flag(*INT*.1) *NOT* Flag(*INT*.0)

COM: если среди продуктов, поставляемых другими поставщиками (не поставщиком &ПС.A) есть продукт, поставляемый поставщиком &ПС.A, то это фиксируется признаком Flag(*INT*.1).

NAME: pr_FIN

KEYS: TempPostavki(&ПС.A, &ПР.B, &*INT*.C, &*INT*.D) Flag(&*INT*.X)

COND: (&*INT*.X = *INT*.0)

CONC: *FINTabl*(&ПР.B)

COND: (&*INT*.X = *INT*.1)

CONC: *NOT* Flag(&*INT*.X) Flag(*INT*.0)

COM: Если продукт &ПР.B поставляется только поставщиком &ПС.A (чему соответствует Flag(*INT*.0)), то заносим его в окончательную таблицу *FINTabl*. Если он поставляется и другими поставщиками (чему соответствует Flag(*INT*.1)), то не заносим.

END

BEGIN

ERASE ALL IN FINTabl

COPY ALL IN Поставки *TO ALL IN* MidPostavki

L1: *ERASE ALL IN* TempPR

MOVE FIRSTLINE IN Поставки *TO ALL IN* TempPostavki

RUN pr_Temp_PR_0

RUN pr_Temp_PR_1
RUN pr_FIN
IF EMPTY(Поставки) *THEN PRINT FINTabl ELSE GOTO* L1
END.

***Пример* 17**. Выдать названия поставщиков, поставляющих продукт с номером 11.

SELECT Название
FROM Поставщики
WHERE EXISTS
(***SELECT ****
FROM Поставки
WHERE ПС = Поставщики.ПС ***AND*** ПР = 11);

Предлагаемая программа:
NAME: pr_Temp_PR_0
KEYS: TempPostavchik(&ПС.A, &Название.B, &Статус.C, &Город.D, &Адрес.E, &Тел.F) Поставки(&ПС.A, ПР.11, &*INT*.C1, &*INT*.D1)
COND:
CONC: *FINTabl*(&Название.B, &Статус.C)
COM:
END
BEGIN
ERASE ALL IN FINTabl
L1: *MOVE FIRSTLINE IN* Поставщики *TO ALL IN* TempPostavchik
RUN pr_Temp_PR_0
IF EMPTY(Поставщики) *THEN GOTO* L2 *ELSE GOTO* L1
PRINT FINTabl
END.

***Пример* 18**. Выдать название и статус поставщиков, не поставляющих продукт с номером 11.

SELECT Название, Статус

FROM Поставщики

WHERE NOT EXISTS

(***SELECT ****

FROM Поставки

WHERE ПС = Поставщики.ПС ***AND*** ПР = 11);

Предлагаемая программа:

NAME: pr_Temp_PR_0

KEYS: TempPostavchik(&ПС.A, &Название.B, &Статус.C, &Город.D, &Адрес.E, &Тел.F) Поставки(&ПС.A, ПР.11, &*INT*.C1, &*INT*.D1)

COND:

CONC: TempPS(&ПС.A)

COM: Выделяем всех поставщиков, поставляющих продукт ПР.11.

END

NAME: pr_Temp_PR_1

KEYS: TempPostavchik(&ПС.A, &Название.B, &Статус.C, &Город.D, &Адрес.E, &Тел.F)

COND:

CONC: *FINTabl*(&Название.B, &Статус.C)

COM: Выделяем название и статус поставщиков, не поставляющих помидоры (ПР.11)

END

BEGIN

ERASE ALL IN FINTabl

L1: *ERASE ALL IN* TempPS

MOVE FIRSTLINE IN Поставщики *TO ALL IN* TempPostavchik

RUN pr_Temp_PR_0

IF EMPTY(TempPS) THEN RUN pr_Temp_PR_1 *ELSE GOTO* L1

IF EMPTY(Поставщики) *THEN GOTO* L3 *ELSE GOTO* L1

L3: *PRINT FINTabl*

END.

***Пример* 19**. Выдать поставщиков продуктов для летнего салата, которые поставляют эти продукты по минимальной цене.

SELECT Продукт, Цена, Название, Статус
FROM Продукты, Состав, Блюда, Поставки, Поставщики
WHERE Продукты.ПР = Состав.ПР ***AND*** Состав.БЛ = Блюда.БЛ
AND Поставки.ПР = Состав.ПР ***AND*** Поставки.ПС = Поставщики.ПС
AND Блюда = 'Салат летний' ***AND*** Цена =
(***SELECT MIN*** (Цена)
FROM Поставки *X*
WHERE *X*.ПР = Поставки.ПР);

Предлагаемая программа:

NAME: pr_Sel
KEYS: Блюдо(&БЛ.A, Блюда.Салат_летний, &B.C, &Основа.D, &*INT*.E, &*INT*.F) Состав(&ПС.A2, &ПР.B1, &*INT*.C2, &*INT*.D2)
COND:
CONC: *Stoim*(&ПР.B1, &*INT*.C2, &ПС.A2)
COM: По БЛ выбираем из состава необходимые коды продуктов (ПР), добавляя их стоимость и код поставщика.
END
NAME: pr_Minstoim
KEYS: *Stoim*(&ПР.A, &*INT*.B, &ПС.C) *Stoim*(&ПР.A, &*INT*.B1, &ПС.C1)
COND: (&*INT*.B1 > &*INT*.B)
CONC: *NOT Stoim*(&ПР.A, &*INT*.B1, &ПС.C1)
COM: Выбираем продукты с минимальной стоимостью
END
NAME: pr_Temp_PR_0
KEYS: *Stoim*(&ПР.A, &*INT*.B, &ПС.C) Поставщики(&ПС.C1, &Название.B1, &Статус.C5, &Город.D1, &Адрес.E1, &Тел.F1)
Продукты(&ПР.A, &Продукт.B3, &Бел.C3, &Жир.D2, &Угл.E2, &K.F2, &Ca.D3, &Na.E3, &B2.F3, &PP.F4, &C.F5)

COND:
CONC: *NOT FINTabl*(&Продукт.B3, &*INT*.B, &Название.B1, &Статус.C5,)
COM: Выбираем по коду поставщика название и статус, а по коду продукта – его название (поле «Продукт»).
END
BEGIN
ERASE ALL IN Stoim
RUN pr_Sel
RUN pr_*Minstoim*
RUN pr_Temp_PR_0
PRINT Stoim
PRINT FINTable
END.

Литература

1. Брошкова Н.Л. Единая среда проектирования и эксплуатации информационной системы, основанная на формально-семантическом подходе. Магистерская диссертация, М.: МАДИ, 2005. 155 с.

2. Попов С.В. Логическое моделирование, М.: Тровант, 2006. 256 с.

3. Мартин Дж. Организация баз данных в вычислительных системах. М.: Мир, 1980. 662 с.

4. Дейт К. Введение в системы баз данных. М.: Наука, 1980.

5. Мейер Д. Теория реляционных баз данных. М.: Мир, 1987. 608 с.

6. Мартен Д. Базы данных: практические методы. М.: Радио и связь, 1983. 168 с.