**Навигация по глобалам. Поиск в глубину и ширину.**

**Деревья и разреженные массивы.**

Теперь, изучив ObjectScript, мы может заняться организацией иерархических баз данных в Cache. Поскольку запросы в них имеют навигационный характер, необходимо освоить команды, позволяющие определить свойства узлов, организовать поиск в ширину и глубину.

Дополнительно будет рассмотрена команда копирования индексированных переменных, которая упростит групповые вставки.

**Разреженные массивы и их представление деревьями.**

Представим набор записей в виде массива, в котором имя массива — это имя набора, а имена индексов это имена полей записи. Значениями индексов в такой модели хранения данных будут значения полей. Если хотя бы один домен имеет мощность континуума, например, домен с вещественными числами, то ограничиться целочисленными индексами, как в массивах, изучаемых в математике, не удастся.

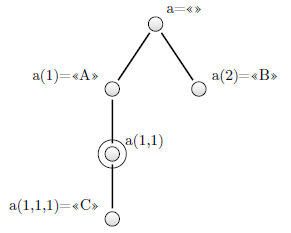
В языках общего назначения при определении массива с конечным набором значений индексов необходимо задать его размерность. Это необходимо, чтобы выделить область памяти, достаточную для хранения массива. Такие традиционные массивы будем называть плотными. В практически важных моделях данных такой массив сильно разрежен. Большую часть выделенного объёма занимают пустые места, соответствующие комбинациям значений индексов, которые не могут быть использованы или ещё не использованы.

Разреженные массивы, реализованные в Cache, отличаются следующими особенностями:

* Хранятся только имеющиеся данные, место под все возможные комбинации индексов не выделяется.
* Размерность массива теоретически не ограничена, а практически ограничивается только длиной строки, в которую помещается запись всех значений индексов.

Вторая ипостась разреженного многомерного массива — это дерево. Нетрудно догадаться, что если присутствуют не все узлы, необходимые для образования дерева, то, может быть, придётся вводить в рассмотрение так называемые виртуальные узлы.

Рассмотрим пример. Создадим дерево в виде локала из следующих узлов: a(1)="A", a(2)="B", a(1,1,1)="C". В узле a(1)="A" единица — это индекс, "А" — значение узла a(1). Строго говоря, дерево получается (рисунок 3.22), только если считать, что к заданным узлам добавлен корневой узел a="" (его значение пустое, но могло быть любое другое) и промежуточный узел a(1,1). Последний называется виртуальным узлом. Такие узлы имеют имя и индексы, но не имеют значения. Их предназначение — связать узлы, отличающиеся на два и более индексов, чтобы на каждом следующем уровне дерева добавлялся ровно один индекс. Виртуальные узлы будем изображать двумя концентрическими окружностями.



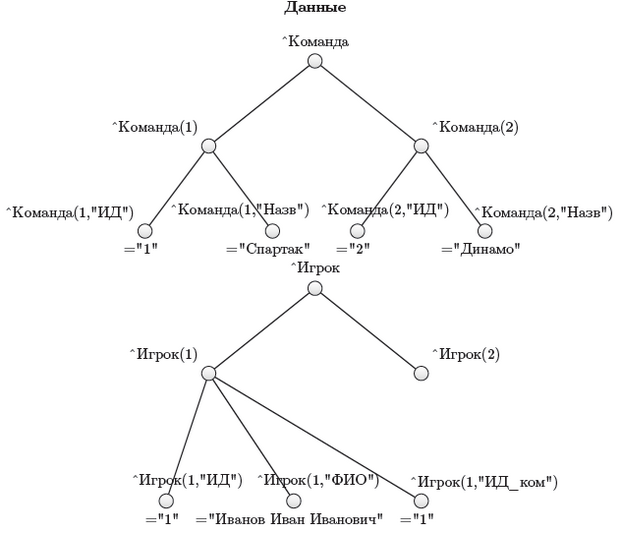
**Рис. 3.22.** Виртуальный узел а(1,1)

Не думайте, что виртуальные узлы вводятся в угоду теории. В следующем разделе станет понятно, что без них невозможна навигация по деревьям.

*Замечание*. Вы, конечно, заметили, что мы следуем противоестественной привычке математиков высаживать деревья вверх корнями.

Набор деревьев (лес) всегда можно свести в одно дерево или небольшое их число. На рисунке 3.23 представлены в экземплярах две связанные сущности "Команда" и "Игрок".

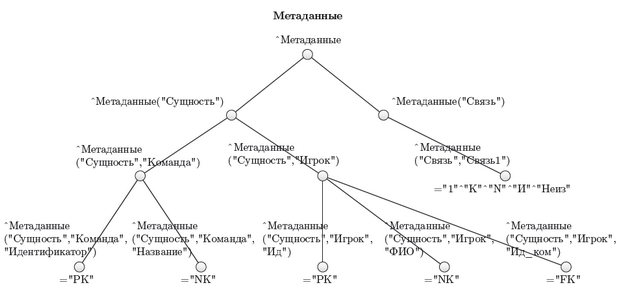
Сущность "Команда" с экземплярами представлена на рисунке 3.23 деревом глубины 2. Заметим, что можно было бы ограничиться деревом глубины 1. В этом случае в нем на уровне 1 существовали бы узлы вида ^Команда (1) =" 1 "Спартак" или ^Команда (1)=$lb(" 1","Спартак") или что-нибудь ещё.



**Рис. 3.23.** Представление двух связанных сущностей глобалами: Данные

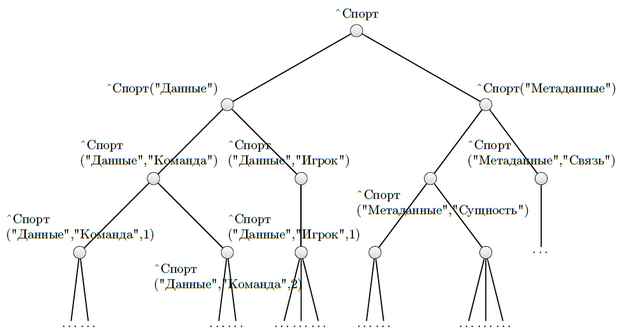
Позже вы увидите, что именно так хранятся строки таблиц в реализации реляционной модели.

В варианте, представленном на рисунке 3.23, данные каждой сущности хранятся в своём дереве, а для всех метаданных использовано дополнительное дерево (рисунке 3.24). Поясним значения листовых узлов. РК—первичный ключ, NK — не ключ, FK — внешний ключ. Значения узла ^Метаданные("Связь","Связь1") представлено строкой с разделителем "^"



**Рис. 3.24.** Представление двух связанных сущностей глобалами: Метаданные

Можно объединить все деревья в одно, выполнив необходимые переименования (рисунок 3.25)

[](https://www.intuit.ru/EDI/06_03_17_3/1488752433-23564/tutorial/959/objects/3/files/03_25.png)

**Рис. 3.25.** Представление базы данных сущностей одним глобалом

Кроме констант и простых переменных узел может принимать следующие значения:

* строка с разделителями, например,

K ^a S ^a(1)="a%b%c" W ^a(1)

* список, например,

K ^a S ^a(1,1,1)=$LB("a","b")

W ^a(1,1,1),"~",$P(^a(1,1,1),2)

* значение узла другого массива, например,

K ^b S ^b(1)="b1", ^b(2)="b2", ^b(1,1)="b11"

S ^a(1,1,1)=^b(1) W ^a(1,1,1)

В соответствии с известным выражением, вы вольны расширять глобал в любую сторону своей души. Это не означает, конечно, что все варианты будут работать одинаково эффективно.

В результате создана база, содержащая данные и метаданные, организованные в иерархии. Система управления базой отсутствует. Все манипуляции с данными и метаданными и запросы данных придётся выполнять врукопашную. Контроль ограничений целостности не организован. Связь обозначена, но необходимо дописать процедурную часть, обеспечивающую её работу.

**3.3.2 Навигация по данным**

В иерархической модели данных для работы с базой необходимо, прежде всего, обеспечить навигацию по данным. Это означает, что всегда можно определить, в какой структуре, в каком элементе данных находится система управления базой и как попасть в нужное место. Для программной навигации необходимо хранить метаданные в словаре. В примерах предыдущего раздела метаданные хранились в глобалах, а процедурная часть словаря проработана не была.

В деревьях Cache навигация осуществляется на физическом уровне. Необходимо обеспечить движение в ширину (функция $ORDER) и движение в глубину (функция $QUERY). При конструировании глобалов их узлы создаются и удаляются командами SET и KILL. Копирование поддеревьев обеспечивает команда MERGE.

Существуют функции, поставляющие дополнительную информацию, необходимую для обеспечения навигации:

* какой тип имеет узел (функция $DATA), сколько индексов у переменной (функция $QLENGTH);
* какое значение имеет указанный индекс (функция $QSUBSCRIPT);
* каково значение узла (функция $GET)

Начнём с получения вспомогательных данных.

###### Определение типа узла, уровня дерева, извлечение индекса и значения

Узлы можно характеризовать двумя признаками — наличием потомков и наличием значений. Если у узла нет потомков, но есть значение, то это листовой узел (на рисунке 3.22 — узлы a(2), a(1,1,1)). Если у узла есть потомки и значения, то это либо корень дерева, либо реальный промежуточный узел (узлы a, a(1)). Узел, имеющий потомков, но не имеющий значений — виртуальный (узел a(1,1)).

Тип узла определяет функция $DATA. Её значения сведены в таблицу 3.3

|  |  |  |  |
| --- | --- | --- | --- |
| Таблица 3.3. Значение функции $DATA | | | |
| Потомки | Значение узла | Код | Значение $DATA |
| 0 | 0 | 00 | 0 |
| 0 | 1 | 01 | 1 |
| 1 | 0 | 10 | 10 |
| 1 | 1 | 11 | 11 |

Cache не печатает ведущие (стоящие слева) нули. Поэтому возвращаются значения $DATA такие, как указано в последнем столбце таблицы.

В следующую таблицу 3.4 сведены варианты условий, проверяемых функцией $DATA.

|  |  |  |
| --- | --- | --- |
| Таблица 3.4. Условия, проверяемые с помощью  $DATA | | |
| Условие | Значение $DATA | Запись условий |
| Потомки имеются | 10,11 | $DATA(узел)\10 |
| Значения имеются | 1,11 | $DATA(узел)#10 |
| Узел не существует | 0 | $DATA(узел)=0 |
| Узел виртуальный | 10 | $DATA(узел)=10 |
| Узел существует | 1,10,11 | $DATA(узел) |

Создадим локал с древесной структурой и проверим типы узлов с помощью функции $DATA (пример 3.38).

USER>S a="", a(1)="A", a(2)="B", a(1,1,1)="C"

USER>W $D(a),?10,$D(a(1)),?20,$D(a(2)),?30,$D(a(1,1,1))

11 11 1 1

USER>W $D(a(1,1))

10

Пример 3.38. Проверка типа узла функцией $DATA

Уровень узла, то есть число индексов в записи узла определяется функцией $QL[ENGTH]. Корень дерева имеет уровень 0 (у него нет индексов), на уровне 1 находятся узлы с одним индексом и т. д.

Для только что введенного дерева, просмотрим уровень всех узлов (пример 3.39).

Функция $QL не просматривает реального узла, а только изучает текст своего аргумента. Поэтому можно определить глубину несуществующего узла несуществующего дерева (пример 3.39, последняя командная строка).

USER>S a="", a(1)="A", a(2)="B", a(1,1,1)="C"

USER>W $QL("a"),?10,$QL("a(1)"),?20,$QL("a(1,1)")

0 1 2

USER>W $QL("a(1,1,1)")

3

USER>K W $QL("a(2,5)")

2

Пример 3.39. Функция $QLENGTH

Функция $QS[UBSCRIPT] работает с полным именем переменной, в том числе извлекает индексы.

Полное имя включает имя области имён, которое окружается символами "|" и "|". Записывают его между знаком "^" и именем переменной с индексами (пример 3.40).

Частные случаи команды:

* $QS("имя",-1) возвращает окружение, если оно есть в полном имени;
* $QS("имя",0) возвращает имя;
* $QS("имя",n), где n > 1 возвращает индекс, если он есть, и пустую строку, если его нет.

USER>S ^a(1)=77

USER>ZN "%SYS"

%SYS>S a=^|"USER"|a(1) w a

77

%SYS>W $QS("^|""USER""|a(1)", -1)

USER

%SYS>W $QS("^|""USER""|a(1)", 0)

^a

%SYS>W $QS("^|""USER""|a(1)", 1)

1

%SYS>W $QS("^|""USER""|a(1)", 2)

%SYS>

Пример 3.40. Полное имя

Функция $G[ET] (аргумент) возвращает значение узла (пример 3.41). Если аргумент отсутствует, вернётся пустая строка.

USER>K y S a="0", a(1)="B"

USER>W "a=",$G(a),?10,"a(1)=",$G(a(1)),?20,"a(y)=",$G(y)

a=0 a(1)=B a(y)=

Пример 3.41. Функция $GET

###### Неполная глобальная ссылка

Почти все команды или функции, обращающиеся к глобалам, изменяют указатель, называемый индикатором неполной глобальной ссылки. Он запоминает предка последнего узла, к которому производилось обращение.

В программе неполная глобальная ссылка записывается как знак "^", за которым в круглых скобках следуют индексы, которые должны быть добавлены к индексам, хранящимся в индикаторе.

Примеры использования неполной глобальной ссылки приведены в пример 3.42.

USER>S ^a="",^a(1)="A",^a(2)="B",^a(1,1)="C"

USER>S ^a(1,1,1)="D",^a(1,1,2)="E"

USER>S x=^a(1) w ^(2)

B

USER>S x=^a(1,1,1) w ^(2)

E

Пример 3.42. Неполная глобальная ссылка

Неполная глобальная ссылка становится не определённой после ссылки на неиндексированную переменную и после вызова функции $QUERY, которая будет описана в следующем разделе. Там же станет понятно, что неполная глобальная ссылка хорошо работает вместе с функцией $ORDER.

Последняя функция $NA[ME] возвращает имя с вычисленными индексами. Более точно,

\$NA(имя(i_1,...i_n), m),

где m — число, вернёт:

при m=0 — только имя без индекса;

при m>0, m \leq n — имя с n индексами;

при m<0 — ошибка;

при m>n — то же, что m=n.

Следует помнить, что обращения к узлу не происходит. Он может вообще не существовать.

Обратите внимание на то, что глобал при записи сокращается за счёт удаления повторяющихся значений индексов. Рассмотрим пример из документации Cache. Глобал:

^Data(1999) = 100

^Data(1999,1) = "January"

^Data(1999,2) = "February"

^Data(2000) = 300

^Data(2000,1) = "January"

^Data(2000,2) = "February"

будет храниться примерно в таком виде:

Data(1999):100|1:January|2:February|2000:300|1:January| 2:February|...

Такая запись может уменьшить количество читаемых блоков памяти.

**Поиск в ширину и в глубину**

Оба вида поиска в этом разделе будут демонстрироваться на глобалах. Однако всё сказанное применимо и к локальным многомерным массивам.

###### *Поиск в ширину (функция $ORDER)*

В разреженных массивах практически никогда нельзя предугадать следующий индекс на выбранном уровне индексации. Точно также не известны ни предыдущий индекс, ни первый, ни последний индексы на каждом уровне индексации.

Функция $O[RDER] (сокращённо $O) использует в качестве аргумента узел дерева и возвращает значение индекса следующего по порядку элемента. Первый аргумент даёт имя локала или глобала. Второй аргумент указывает направление обхода. Значение +1, оно же значение по умолчанию, определяет движение по возрастанию. Значение -1 задаёт движение по убыванию.

![\$O[RDER](имя\_переменной, направление\_обхода)](data:image/png;base64,)

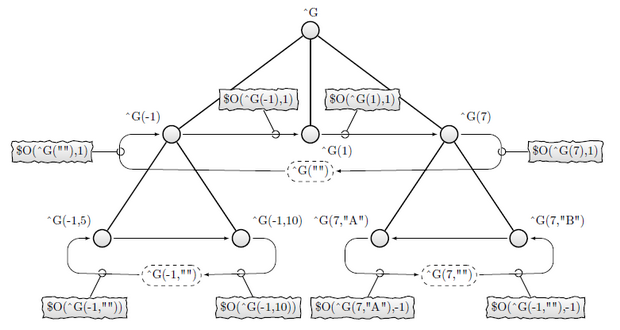
Создадим глобал, изображённый на рисунке 3.26.

s ^G="",^G(-1)=-1,^G(1)=1,^G(7)=7,^G(-1,5)=-15

s ^G(-1,10)=-110,^G(7,"A")="7A",^G(7,"B")="7B"

Обратите внимание, что, например, команда W $O(^G(""),1) вернёт индекс -1 крайнего левого узла на глубине 1. Команда $O(^G(7),1) вернёт пустой индекс.

Можно считать, что цепочка индексов каждого горизонтального подуровня замыкается на фиктивный пустой индекс, что изображено на рисунке 3.26.

[](https://www.intuit.ru/EDI/06_03_17_3/1488752433-23564/tutorial/959/objects/3/files/03_26.png)

**Рис. 3.26.** Функция $ORDER

Начнём движение со второго пустого индекса. Команда W $O^G(7, ""),-1) вернёт индекс крайнего правого узла в правом горизонтальном слое уровня 2. После выполнения команд W $O^G(7,"B"),-1) и $O^G(7,"A"),-1) получим пустой индекс. Цепочка замкнулась.

Для поиска самого левого индекса воспользуйтесь следующими функциями:

1. W $O^G(""),1) даёт крайний левый узел на первом уровне дерева
2. W $O(^G(""),-1) даёт крайний правый узел на первом уровне дерева
3. W $O(^G(-1,""),1) даёт крайний левый узел на втором уровне поддерева, образованного узлом ^G (-1).

Обойти все узлы поддерева на одном уровне можно с помощью циклов, учитывающих имя узла, образующего поддерево, и замкнутость горизонтальных подуровней на узел с пустым индексом (пример 3.43)

USER>S x="" F S x=$O(^G(x)) Q:x="" W !, x, ?30, ^G(x)

-1 -1

1 1

7 7

USER>S x="" F S x=$O(^G(7,x)) Q:x="" W !,x,?30,^G(7,x)

A 7A

B 7B

Пример 3.43. Обход горизонтального подуровня

Не забывайте, что после ключевого слова F в цикле без параметров и перед командой W стоят два пробела.

$ORDER пробегает все узлы уровня, включая виртуальные. Однако, приведенный цикл использовать без переделки нельзя, потому что, встретив виртуальный узел, не имеющий значения, он не сможет распечатать его.

Уточним детали. Сортируемые элементы располагают слева направо в порядке возрастания.

Вспомним естественный порядок сортировки. Сначала канонические числа, не содержащие избыточных нулей и знака "+", в порядке возрастания. Затем неканонические числа. За ними идут слова, начинающиеся с букв латиницы. После них слова, начинающиеся с букв кириллицы.

Для уточнения порядка сортировки одиночных символов наберите следующие две строки

F i=1:1:255 S a($CHAR(i))=$CHAR(i)

S x="" F S x=$O(a(x)) Q:x="" W !,x,?30,a(x)

и сами рассмотрите полученную последовательность. Обратите внимание на то, что буквы кириллицы расположены не подряд.

*Замечание*. Функция $CHAR(i) выдаёт символ с кодом i.

###### Поиск в глубину (функция $QUERY)

Поиск в глубину производится с помощью функции $Q[ERY], возвращающей в отличие от $ORDER, не следующий индекс, а имя следующего узла. Однако, возвращаются лишь узлы имеющие значение, то есть виртуальные узлы игнорируются. Движение по ним производится, но результаты не выдаются.

При обходе всего глобала начинаем движение с корня. Правила перемещения:

* делаем шаг по крайней левой ветви вниз;
* если это невозможно, смещаемся на один шаг вправо;
* если и это невозможно, смещаемся на один шаг вверх.

При этом необходимо помнить, что движение по горизонтали, как и для $ORDER возможно только по узлам одного подуровня, имеющим общего непосредственного предка.

Для проверки правила несколько изменим глобал ^G, добавив в него узел ^G(1,1,1) так, чтобы образовался виртуальный узел ^G(1,1) (пример 3.44)

USER>S ^G="", ^G(-1)=-1, ^G(1)=1, ^G(7)=7, ^G(-1, 5)=-15

USER>S ^G(-1, 10)=-110, ^G(7, "A")="7A"

USER>S ^G(7, "B")="7B", ^G(1,1,1)=111

USER>S x="^G" F S x=$Q(@x) Q:x="" W x, "=", @x, !

^G(-1)=-1

^G(-1, 5)=-15

^G(-1, 10)=-110

^G(1)=1

^G(1,1,1)=111

^G(7)=7

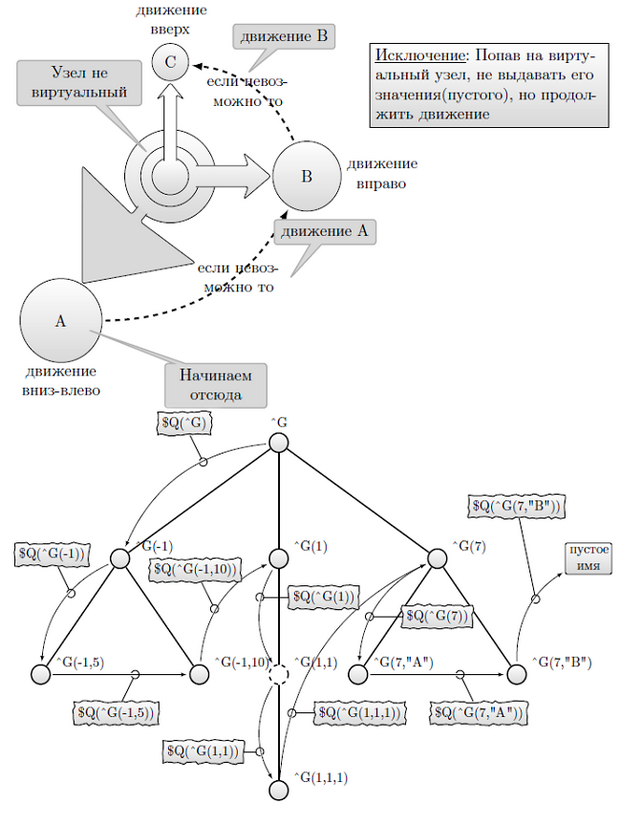
^G(7, "A")=7A

^G(7, "B")=7BF

Пример 3.44. Цикл с функцией $QUERY

После узла ^G(1) был пройден виртуальный узел ^G(1,1), но поскольку он не имеет значения, в выходных данных сведений о нём нет. При следующем повторе цикла отработано движение вниз к узлу ^G(1,1,1).

Мнемоническое изображение правил поиска в глубину с использованием функции $QUERY и подробное описание обхода дерева ^G из предыдущего примера приведены на рисунке 3.27.

[](https://www.intuit.ru/EDI/06_03_17_3/1488752433-23564/tutorial/959/objects/3/files/03_27.png)

**Рис. 3.27.** Функция $QUERY

###### Копирование индексированных переменных (команда MERGE)

Команда MERGE позволяет вклеивать в индексированные переменные копии других индексированных переменных, независимо от того, являются ли они локальными или глобальными. Синтаксис похож на синтаксис присваивания:

M[ERGE]\verb*| |переменная\_результат = вставляемая\_переменная

Для проверки результатов этой операции удобно воспользоваться программой просмотра глобала ^%G (пример 3.45). При первом её исполнении в ответ на предложение ввести устройство и ширину строки нажимайте на Enter, вставив значения по умолчанию, затем введите имя глобала. После ввода пустого имени программа прекращает работу.

USER>D ^%G

Device:

Right margin: 80 =>

Screen size for padding (0=nopadding) ? 24 =>

For help on global specifications DO HELP^%G

Global ^G

^G

""

^G(-1)=-1

^G(-1,5)=-15

10)=-110

^G(1)=1

^G(1,1,1)=111

^G(7)=7

^G(7,"A")=7A

"B")=7B

Global ^

Пример 3.45. Пример использования программы ^%G

Повторы значений программа не печатает. Так что запись " 10)=-110" после записи <^G(\_l,5)=-15" следует читать как "^G(\_1,10)=-110".

Создадим глобал с узлами ^a(l), ^a(2), ^a(l,l) и локал b("a"), b("b"). Вставим локал b в узел ^a(1) глобала и просмотрим результат операции (пример 3.46)

Приведённые в "Иерархические модели данных. Деревья в СУБД Cache" сведения о языке Cache ObjectScript достаточны для создания иерархических баз данных, манипуляций данными и выполнения запросов.

В практической работе должны использоваться средства для администрирования базы данных, которые мы, к сожалению, не изучаем.

USER>S ^a="", ^a(1)="", ^a(2)="", ^a(1, 1)=""

USER>S b("a")="", b("b")=""

USER>MERGE ^a(1)=b

USER>D ^%G

For help on global specifications DO HELP^%G

Global ^a

^a

""

^a(1)=""

^a(1,1)=""

"a")=""

"b")=""

^a(2)=""

Пример 3.46. Пример использования команды MERGE