### Лабораторная работа №8

### Cache Object Script. Строки, строки с разделителями, списки. Косвенность и XECUTE.

Задание: Провести манипуляции, проанализировать результат выполнения команд (отчетом по работе является протокол).

Примеры числовой интерпретации.

|  |  |
| --- | --- |
| **Выражение** | **Значение** |
| +"123abc" | 123 |
| "314.159E - 2xyz" | 3.14159 |
| +"Это число = 1265" | 0 |
| +$Horolog | 57123 |
| "3 Яблока" + "4 Груши" | 7 |
| 1 > "Большой" | 1 (ИСТИНА) |
| ".3" = "ЗЕ-1" | 0 (ЛОЖЬ) |
| +".3" = +"ЗЕ-1" | 1 (ИСТИНА) |

Объясните результат выполнения следующих команд

USER> s x=+ “ abc” w x

USER> s x= -“.7e2cccw” w x

USER> s x= “.7e2cccw” w x

Логические операторы (&, !, &&, ||)

Результатом оператора логическое И (&) является ИСТИНА тогда, когда истинны оба операнда, а оператора ИЛИ (!) — когда по меньшей мере один операнд содержит значение ИСТИНА.

Проведите манипуляции с логическими операторами

Пусть m = 0, n = 1.

Выражение Значение

(m = 1) & (n = 1) 0

(m = 1) ! (n = 1) 1

'(m = 1) & (n = 1) 1

m = 0 '& (n = 1) 1

m = 0 & 'n '= 1 1

Строки. Над строками определены следующие бинарные операции: операторы строкового сравнения (=, [, ], ]])

Оператор *сравнения*«=»проверяет две строки символов на равенство. Если хотят осуществить проверку на числовое равенство, то прежде необходимо произвести числовую интерпретацию с помощью знака плюс (например, + число1 = + число2).

Оператор *содержит* «[» определяет, содержится ли правая строка в левой, а оператор «]» определяет, является ли первая строка большей, чем вторая с лексикографической точки зрения, то есть будет ли вторая строка при сортировке идти после первой. Оператор сортировки «]]» является аналогом оператора «]», но, в отличие от него, производит проверку не с лексикографической точки зрения, а интерпретируя значения операндов, например, числа при этом всегда больше любых других строковых значений.

Пусть х = 3, у = "А", z = "3.0"

|  |  |
| --- | --- |
| Выражение | Значение |
| х = "а" | 0 |
| x = z | 0 |
| "ABC" [ у | 1 |
| х ] у | 0 |
| у ] х | 1 |
| z ] (х = у) | 1 |
| 2 ]] 19 | 0 |

USER> w “conca”\_”te”\_”nation”

USER> w “a””b”

USER> s x= “abcd”[“bc” w x

USER> s x= “abcd”[“aa” w x

USER> s x= “abc”]“” w x

USER> s x= “cd”]“axz” w x

USER> s x= “cd”]“exz” w x

USER> s x= “aacd”]“aabxz” w x

USER> s x= “aacd”]“aa” w x

USER> s x= “10”]]“9” w x

Функции Cache ObjectScript, которые служат для обработки строк.

|  |  |
| --- | --- |
| $E[xtract] |  |

|  |  |
| --- | --- |
| >Write $Extract ("ABC") // выводит «A»  >Write $Extract("XYZ", $L("XYZ")) // выводит «Z»  >Write $Extract("AABB", 2, 3) // выводит «AB»  >Write $Extract( "лето", 3, 255) // выводит «тo»  >Write $Extract ("abc", 5) // выводит пустую строку | |
| $F[ind] | |  | |

|  |  |
| --- | --- |
| >Write $ Find("ABC", "A") // выводит 2  >Write $Find("XYZ", "T") // выводит 0  >Write $Find("ABABAB", "AB", 3) // выводит 5 | |
| $J[ustify] | |  | |

|  |  |
| --- | --- |
| >Write $Justify(12, 3)  12  >Write $Justify("Text", 10)  Text  >Write $Justify(12, 3, 2)  12.00  >Write $Justify(3.14, 1, 0)  3  >Write $Justify(0.414, 6, 3)  0.414 | |
| $L[ength] | |  | |

|  |  |
| --- | --- |
| >Write $Length("ABCD")  4  >Write $Length("")  0  >Write $Length ("AB/CD/EF", "/")  3 | |
| $P[iece] | |  | |

|  |  |
| --- | --- |
| >Set v = "ABC/XYZ/123"  >Write $Piece(v, "/")  ABC  >Write $Piece(v, "/", 2)  XYZ  >Write $Piece(v, "/", 2, 3)  XYZ/123  >Set $P(v, "/", 2) = "\*\*\*" Write v  ABC/\*\*\*/123 | |
| $R[everse] | |  | |

|  |  |
| --- | --- |
| >Write $Reverse("Rail")  liaR | |
| $TR[anslate] | |  | |

|  |
| --- |
| >Set u = "EPUR", l = "epur"  >Write $TR("UPPER", u, l)  upper  >Write $TR("train station", "aeiou")  trn sttn |

USER> w $l( “12345678”)

USER> w $l( “”)

USER> w $e( “12345678”,3,5)

USER> w $f( “12345678”,”45”)

Строки с разделителями, элементы строк с разделителями называют полями, одной строкой можно записать, например, Ф.И.О. Иванов^Иван^Иванович. Для строк с разделителями должны поддерживаться: определение количества полей в строке, выделение поля вставка значения в нужное поле.

USER> s str1=”A^77^BD

USER> w $p(srt1,”^”,2,3)

USER> s str1=”A^77^BD

USER> w $p(srt1,”^”,2)=”QQ” w str1

USER> w $p(srt1,”^”,4)=”PP” w str1

Важнейшие команды Cache ObjectScript.

|  |  |
| --- | --- |
| Группа команд | Типичные представители |
| Управление последовательностью выполнения программы | If, Else, For, Quit, Do, Goto, Break, Xecute |
| Операции с переменными | Set, Merge, Kill, New, Lock |
| Ввод-вывод | Read, Write, Open, Use, Close |
| Различные другие | Job, Hang, Halt, View |

Командные конструкции For, While, Do/While обеспечивают многократное выполнение заданного сегмента кода. Это называется циклом.

**Общая форма команды For:**

***For <for parameter> {code}***

Здесь блок ***{code}***выполняется столько раз, сколько задано в параметре команды

***<for parameter>.***Команды, следующие в той же строке программы после закрывающей скобки, уже не относятся к команде «For» и выполняются только один раз.

Интересной особенностью команды «For» в Cache является разнообразие вариантов параметра команды. Здесь возможны:

Список различных выражений

***For level=<value> [, <value>…] {code}***

Задание числового диапазона

***For level=<num value> : <num value> : <num value> {code}***

Здесь первое выражение рассматривается как начальное значение, второе как величина инкремента и третье как конечное значение переменной цикла.

Числовой диапазон, как показано выше, но без конечного значения.

***For level=<num value> : <num value> {code}***

В этом случае в теле цикла наращивается некоторая переменная, а для выхода из цикла служит функция Quit.

Безаргументная форма. В этой форме нет ни переменной цикла, ни начального, ни конечного значений.

***For {code} //как минимум два пробела после «For»!***

**Основные правила циклических выражений:**

Начальное значение, инкремент и конечное значение (если оно задано)

вычисляются до первого прохода тела цикла и сохраняются внутри цикла.

Изменение этих значений внутри цикла не влияет на число итераций.

После окончания цикла переменная цикла имеет то значение, которое она имела при выполнении последней итерации.

В конфликтных случаях (например, если конечное значение при положительном инкременте меньше начального) тело цикла не выполняется

ни разу.

Необходимо отметить, что различные формы конструкции «For» могут

свободно комбинироваться.

**Общая форма команд While и Do/While**

Две управляющие конструкции:

***While <value> [, <value>…] {code}***

и

***Do {code} While <value> [, <value>…]***

Также являются циклами. Они отличаются методом проверки условия окончания цикла. В первой конструкции условие проверяется до выполнения блока программы, а во второй – после. Кроме того, чтобы блок программы выполнился, все выражения в списке должны быть логически истинны.

Важное отличие между этими конструкциями заключается в том, что в случае **Do/While**блок кода выполняется хотя бы один раз, чего нельзя сказать о**While.**

For j = -9, 5, 7, 36, -100 w j

For i = 1:1:10 w i

For k = "A", "B", 1:2:11 w k

S x=8

F { W " x=",x,! S x=x-1 Q:x=3 } W "Цикл завершен"

S p=1 F{s p=$F(“1111”,”1”,p) W “p=”,p, ! Q:p=0 }

Косвенность ( @ ) и Xecute

Косвенное имя

Предположим, переменная v1 содержит в качестве значения имя локальной переменной, например, «v1 = "locvar"». Тип данных значения v1 очевидно представляет собой строку.

Разумеется, присваивание «Set "locvar" = 4» приведет к сообщению об ошибке, так как команда Set ожидает имя переменной, а не строку символов. Однако имея оператор косвенности "@", программист может преобразовать тип данных в имя: «Set @v1 = 4» присваивает переменной locvar значение 4. Эта форма косвенности называется косвенным именем.

Косвенное имя может применяться всюду, где в Cache ожидается имя переменной (локальной или глобальной, неиндексированной или индексированной) или точка входа в программу (метка или метка плюс имя программы).

|  |
| --- |
| Set pname = "^Prog1" Do @pname |

Здесь вызывается программа ^Prog1.

Косвенный аргумент

В отличие от косвенного имени, в случае косвенного аргумента осуществляется преобразование строк в аргументы команд. За исключением команды For, в которой косвенность аргумента не допускается, в остальных командах можно косвенно ссылаться на полный аргумент или на список аргументов.

|  |
| --- |
| Set isetarg = "x = 1", @isetarg |

Переменная isetarg содержит синтаксически корректный аргумент команды Set в виде строки, которая обрабатывается оператором косвенности. Отчетливо видно отличие от косвенного имени, ведь «"х = 1"» — не разрешенное имя. Разумеется, во многих командах (Read, Write, Kill, New — их далеко не полный перечень) одиночный аргумент также может быть именем. В этом случае обе формы косвенности неотличимы друг от друга.

А вот несколько примеров, в которых обе эти формы косвенности имеют различия:

|  |
| --- |
| Set ikill = "(e, f, g)" Kill @kill  Set inew = "(a, b, c)" New @inew |

Индексная косвенность

Несколько реже применяемой формой косвенности является индексная косвенность, которая служит для расширения имен локальных и глобальных переменных. Ее синтаксический признак — дважды встречающийся оператор косвенности. Первый из них преобразует идущую вслед за ним строку в имя переменной, второй — добавляет к полученному имени заданный в скобках индекс (или индексы).

Пример иллюстрирует основную идею:

|  |
| --- |
| >Set x(2, 5, 3) = "IndInd" Set feld = "x(2, 5)", d1 = 3  >Write @feld@(d1)  IndInd |

Мы видим расширение ссылки х (2, 5) индексом третьего уровня со значением 3.

USER> S a=”b”, b=77, w a,!,@a

USER> S a1=1, a2=2, a3=3 F i=1:1:3 w @(“a”\_i)

USER> S a1=1, a2=2, a3=3, fb=”ab” F i=1:1:3, “b” w @(“a”\_i)

|  |  |
| --- | --- |
| Xecute (x) | Команда Xecute обеспечивает выполнение строки символов в виде однострочной подпрограммы (без метки). Допускается вложенная конструкция Xecute. На выполнение всей команды, как и каждого ее отдельно взятого аргумента, может быть наложено постусловие. |

|  |
| --- |
| Xecute а, ^В  Xecute "S x = 1"  Xecute:t["a" t(i, 2):true  Xecute @a\_"X @b" |

USER> S z=”W 22,”, v=”!,1+7 “

USER> X z\_v

**Задание**:

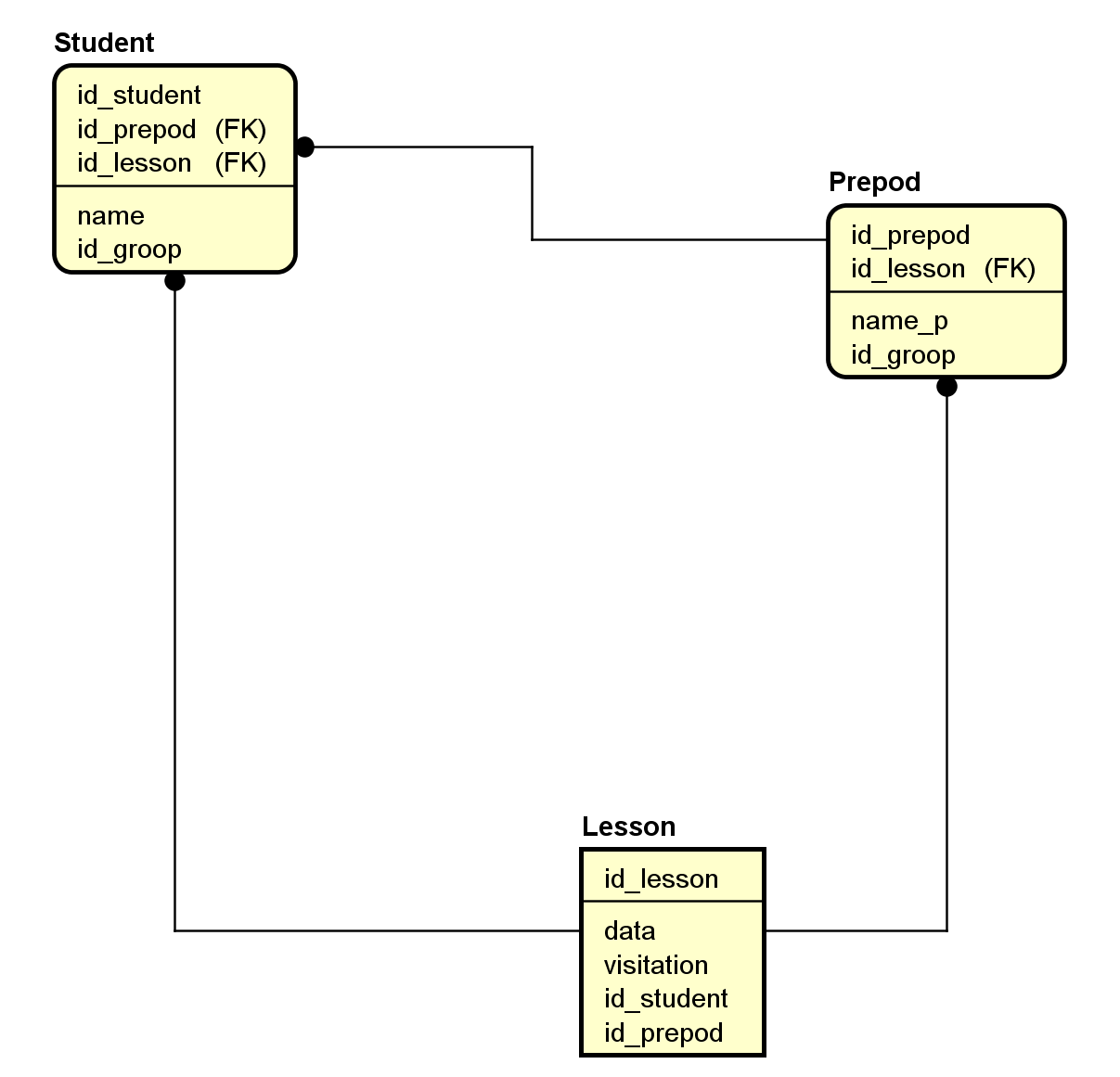
Создать глобал, который хранит информацию о посещаемости занятий студентами Вашей группы (см. таблицы 1-ой лабораторной работы)

Ex.: «Индивидуальный журнал преподавателя»

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Факультет |  | Дисциплина |  | Группа |  |
| Кафедра |  | Курс |  | Преподаватель |  |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Ф.И.О.  студента | Занятие | | | Занятие | | | Занятие | | |
| Содержание | | | Содержание | | | Содержание | | |
| Дата | | | Дата | | | Дата | | |
| посещ. |  | выполн. | посещ. |  | выполн. | посещ. |  | выполн. |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |

!!! (УСТНО) УСТРАНИТЕ НЕКОРРЕКТНУЮ СВЯЗЬ И СООТВЕТСТВУЮЩИЕ ЕЙ КОМАНДЫ СКРИПТА.



CREATE TABLE Lesson (

id\_lesson CHAR(10) NOT NULL,

data CHAR(10),

visitation CHAR(10),

id\_student CHAR(10),

id\_prepod CHAR(10)

);

ALTER TABLE Lesson ADD CONSTRAINT PK\_Lesson PRIMARY KEY (id\_lesson);

CREATE TABLE Prepod (

id\_prepod CHAR(10) NOT NULL,

id\_lesson CHAR(10) NOT NULL,

name\_p CHAR(10),

id\_groop CHAR(10)

);

ALTER TABLE Prepod ADD CONSTRAINT PK\_Prepod PRIMARY KEY (id\_prepod,id\_lesson);

CREATE TABLE Student (

id\_student CHAR(10) NOT NULL,

id\_prepod CHAR(10) NOT NULL,

id\_lesson CHAR(10) NOT NULL,

name CHAR(10),

id\_groop CHAR(10)

);

ALTER TABLE Student ADD CONSTRAINT PK\_Student PRIMARY KEY (id\_student,id\_prepod,id\_lesson);

ALTER TABLE Prepod ADD CONSTRAINT FK\_Prepod\_0 FOREIGN KEY (id\_lesson) REFERENCES Lesson (id\_lesson);

ALTER TABLE Student ADD CONSTRAINT FK\_Student\_0 FOREIGN KEY (id\_prepod,id\_lesson) REFERENCES Prepod (id\_prepod,id\_lesson);

ALTER TABLE Student ADD CONSTRAINT FK\_Student\_1 FOREIGN KEY (id\_lesson) REFERENCES Lesson (id\_lesson);