### Лабораторная работа № 7

Cache Object Script. Три модели данных: иерархическая, объектная и реалиционная. Навигация.

Задание: Провести манипуляции, проанализировать результат выполнения команд (отчетом по работе является протокол). Работа с вебинаром Планета: Введение в Cache <https://www.youtube.com/watch?v=gl8GKHmw9xU> ).

!!!Не забудьте вести протокол работы – это отчет по лабораторной работе!!!

В Caché есть три модели данных: иерархическая, объектная и реляционная.

**Иерархическая:**

На примере создадим небольшую базу данных, в которой будет храниться информация об автомобилях. Для каждого автомобиля будем хранить его номер и один атрибут - фамилию его владельца.

Запускаем терминал. Мы хотим добавить информацию о том, что автомобилем с номером АА100А владеет Иванов. Сделаем командой set.

USER> set ^cars(“AA100A”) =”Иванов”

Мы только что сохранили в переменную cars AA100A значение Иванов. Эта крышка (^) обозначает, что переменная уже хранится в базе и никуда не исчезнет, даже если выключат свет и выключится резко компьютер. Если крышки не будет, то переменная будет храниться в памяти и исчезнет, как только мы закроем терминал.

Set это команда языка, встроенного в Caché, язык называется Cache Object Script, сокращенно COS. Что бы вывести это значение, есть команда write. Пишем эту команду и номер узла

USER>Write ^cars(“AA100A”)

Нажимаем ENTER и терминал выведет фамилию Иванов. Если бы значения этой переменной не было, то вышла бы ошибка.

USER>

Добавим еще две записи, CC100C владеет Сидоров, ББ100Б владеет Петров.

Теперь в БД три записи, можем вывести всю базу командой zwrite ^cars. Команды эти к регистру не чувствительны, в отличии от имен переменных. Если написать CARS большими буквами, то команда zwrite ничего не выведет, потому что cars и CARS это две разные переменные. Переменные с крышками еще называют глобальными переменными (сокращенно глобалы).

Эту БД можно представить в виде дерева. У дерева есть корень и есть под узлы. Корень это ^cars, а от него вниз идут нижние значения (^cars(“СС100С”), ^cars(“AA100A”), ^cars(‘ББ100Б”), их называют индексами. Говорят, что значение Иванов хранится в узле глобала cars с индексом АА100А.

Нарисовать дерево

Когда создавали глобал cars, нигде не определяли его тип и не говорили, что это пространственная структура данных. Просто открыли терминал и написали set ^cars.

Для создания какого-то узла в глобале, не обязательно нужен предок.

Другой пример:

USER>zwrite ^a

ничего не выведет, потому что в глобале а ничего нет

USER>set ^a( 1,”a”,5) =200

USER>write ^a

ничего не выведет

USER>write ^a(1)

тоже не выведет, значений нигде нет. Сработает только

USER>write ^a( 1,”a”,5)

выведет 200

Можно иметь несколько индексов, например, мы узнали марку и цвет автомобиля АА100А

Мы записываем в узел

USER>set ^cars(“AA100A”,”цвет”)= “желтый”

USER>set ^cars(“AA100A”,”марка”)= ‘”BMW”

можем сделать

USER> zwrite ^cars

и выведет

^cars(“AA100A”) =”Иванов”

^cars(“AA100A”,”марка”)=”BMV”

^cars(“AA100A”,”цвет”)=”желтый”

^cars(“BB100B”)=”Петров”

^cars(“CC100C”)=”Сидоров”

Можно представить то же дерево, что у узла ^cars(“AA100A”) появилось два потомка ^cars(“AA100A”,”марка”),^cars(“AA100A”,”цвет”)

Команда zwrite выводит данные не в том порядке, в котором мы их добавляли, а в отсортированном порядке, сортирует по индексам. На диске они записываются в отсортированном порядке. Из за этого мы можем быстро найти определенный автомобиль, такой поиск осуществляется очень быстро и БД (база данных) не тормозит, например write ^cars(“AA100A”).

Физически на диске деревья реализованы в виде структуры звездочка (\*) деревьев.

Нарисовать дерево

Функции в COS начинаются значком $, в противном случае Caché будет интерпретировать запись, как локальную переменную! $Lengh - функция, вызывающая длину строки.

Write $Lengh(^cars(“BB100B”))

Выведет 6

Потому что длина строки Петров 6 символов.

Функция $data() также проверяет - ! есть ли у узла потомки! и возвращает два значения (значение узла и потомка). Функция $Data (коротко $D) – это функция с именем переменной в качестве аргумента. Эта функция может работать с локальной или с глобальной, скалярной или индексированной переменной. Значение функции $Data определяет, существует ли заданная в виде аргумента переменная и какая у неѐ структура. Функция возвращает четыре различных результата:

1.Если еѐ значение равно 0 (10), то переменная не существует;

2.Если еѐ результат 1(11), то переменная существует.

Интересней посмотреть на специфичные функции, например, нужно посмотреть есть ли у нас автомобиль с номером “BB100B”

Write $data(^cars(“BB100B”))

Или убедиться, что в нашей базе автомобиля с номером “Bb100B” - нет

Write $data(^cars(“Bb100B”)) - ! чувствительность к регистру !

Задача – нужно найти все автомобили , которыми владеет Иванов. Есть различные функции для навигации по глобалам, например, $order(). **Функция $Order** является основной функцией, служащей для получения очередного в последовательности сортировки индекса локальной или глобальной индексированной переменной. Применяя цикл, в соответствии с последовательностью сортировки могут быть получены все существующие индексы данного уровня. Для того, чтобы задать самый первый индекс в пределах заданного уровня, используют пустую строку.

USER>Write $order(^cars(“AA100A”))

– возвращает индекс узла на том же уровне “BB100B”, если подставим полученный (возвращаемый параметр) Write $order(^cars(“ВВ100В”)), то получим следующий индекс “CC100C”.

Если мы не знаем первый индекс, то делаем запрос с пустым индексом

USER>Write $order(^cars(“”))

и получаем первый соответственно “AA100A”

USER>zwrite $O(^cars("AA100A"))

получаем "BB100B"

**Открыть Cache Studio**, ознакомиться с его интерфейсом и создать новую программу

1. Set number = “”
2. For {
3. Set number = $order(^cars(number))
4. If number =”” {
5. Quit
6. }
7. Write number, !
8. }

После набора кода в центральном окне, программу следует откомпилировать, путѐм нажатии клавиш Ctrl+F7. Перекомпилируйте программу каждый раз, когда что-то в ней меняете или исправляете ошибки! После первой компиляции программа попросит сохранение назовите, например, **allcars.mac**

**Из Терминала** исполните программу

USER> do ^allcars

zwrite

Cache Studio – это одно из приложений, вызываемых из меню Cache-куба. Для разработки программ предлагается графический редактор, позволяющий комфортно создавать классы, CSP – страницы, программы на макроязыке, включаемые файлы, программы на промежуточном коде, а также компилировать их в выполняемый объектный код.

Cache Studio предоставляет разработчику такое вспомогательное средство, как автоматический синтаксический контроль. Различные элементы языка подсвечиваются различным цветом. И в случае нарушения синтаксических правил выводится сообщение об ошибках. Cache Studio имеет собственный отладчик, который может быть использован для поиска ошибок в приложениях.

**В Cache Studio**, внедрите в программу **код поиска**

Set number = “”

For {

Set number = $order(^cars(number))

If number =”” {

Quit

}

**If ^cars(number)=”Иванов” { Write number, ! }**

**}**

**Из Терминала** исполните программу **allcars.mac** (экспериментируйте на разных фамилиях)

USER> do ^allcars

USER> Zwrite ^cars

С помощью функции $ListBuild создаются списки.

|  |
| --- |
| >Set L1 = $ListBuild("красный", "зеленый", "голубой")  L1 = {красный, зеленый, голубой}  >Set L2 = $ListBuild()  пустой список L2 = {}  >Set L3 = $ListBuild("красный", "зеленый", , "голубой")  L3 = {красный, зеленый, , голубой}  >Set L4 = $ListBuild("красный", "зеленый", "", "голубой")  L4 = {красный, зеленый, "", голубой}  >Set L5 = $ListBuild("красный", $ListBuild("светлокрасный",  "оранжевый", "вишневокрасный"), "зеленый",  "голубой")  Список, содержащий в качестве второго элемента другой список. |

Аргументы $ListBuild могут представлять собой любые выражения:

|  |
| --- |
| Set list = $ListBuild(2 \* x1, 3 \* x2) |

|  |
| --- |
| USER>set ^cars2("AA100A")=”Иванов,BMW,желтый”  USER>set ^cars2("BB100B")=”Петров,Renault,черный”  USER> Zwrite ^cars2  USER>set ^cars3("AA100A")=$listbuild(”Иванов”,”BMW”,”желтый”)  USER>set ^cars3("BB100B")=$listbuild(”Петров”,”Renault”,”черный”)  USER> Zwrite ^cars3  USER>write ^cars3("AA100A")  USER>write $list(^cars3("AA100A",2)  USER>write $list(^cars3("AA100A",3) |

Просмотрите глобалы на Портале.

**Реляционная модель (связь с иерархической):**

**Заходим в режим SQL** (параллельно будем использовать режим Терминала)

Создадим базу данных cars , в которой будет храниться информация об автомобилях:

create table cars (number varchar(6) primary key, owner varchar(30) not null, brand varchar (20), color varchar(20))

выполним команду и уничтожим, ----🡪 перейдем к следующей

Создадим индекс по полю владельца  
create index CarownerInd on cars (owner)

Внесем несколько записей в БД

insert into cars (number, owner, brand, color) values ("AA100A","Иванов","BMW","желтый")

выполним команду и уничтожим, ----🡪 перейдем к следующей

insert into cars (number, owner, brand, color) values ("BB100B","Петров","Renault","черный")

выполним команду и уничтожим, ----🡪 перейдем к следующей

insert into cars (number, owner, brand, color) values ("СС100С","Сидоров","","")

**Перейдем в Терминал, Экземпляр: TRYCACHE**  
  
Имя пользователя: \_system  
Пароль: \*\*\*\*\*\*\*  
  
  
USER>zw ^User.carsD  
  
внесем изменение в БД

USER>set ^User.carsD(2)=$listBuild("---","ИЗМЕНЕНИЯ","","---")  
  
USER>zw ^User.carsD

Для интереса просмотрим, что хранится в индексах

USER>zw ^User.carsI

**Заходим в режим SQL**

Просмотрим изменения в БД

select \* from cars

Просмотрите глобалы на Портале.

**Объектная модель - просмотреть вебинар Планета: Введение в Cache** [**https://www.youtube.com/watch?v=gl8GKHmw9xU**](https://www.youtube.com/watch?v=gl8GKHmw9xU) **)**

Заходим в Студию (создадим класс)  
  
Нажимаем на пиктограмму   
+Создать (Ctrl N)  
+ Класс (Cache)

+ задаем имя пакета - teat  
+ задаем имя класса - car  
+ тип класса Persistent (т.е с с возможностью автосохранения, загрузки и т.п.)

+зададим свойства класса (номер, владелец, брэнд, индекс по номеру /в классах нет понятия первичный ключ/)   
  
Class Test.Car Extends %Persistent  
{  
  
Property Number As %String [ Required ];  
Property Owner As %String [ Required ];  
Property Brand As %String [ Required ];  
Index CarNumberInd On Number [ Unique ];  
  
}

+сохраняем класс и компилируем  (Ctrl + F7), после успешной компиляции с классом можно работать

+ из Терминала создадим экземпляр этого класса и сохраним его на диск

+ заходим в Терминал (для работы с классами есть специальный синтаксис)

Узел: idea-PC, Экземпляр: TRYCACHE  
  
Имя пользователя: \_system  
Пароль: \*\*\*\*\*\*\*

+ используем методы ).%New() %Save() %Id()  
  
USER>set car=##class(Test.Car).%New()  
  
USER>set car.Number="AA100A"  
  
USER>set car.Owner="Иванов"  
  
USER>set car.Brand="BMW"

USER>write car.%Save()  
  
USER>write car.%Id()

USER>set car=##class(test).%OpenId(1)  
  
  
USER>write car.Number  
AA100A  
USER>write car.Owner  
Иванов  
USER>

Для интереса просмотрим, что хранится в индексах

USER>zw ^Test.carD

USER>zw ^User.carsD

USER>zw ^Test.carI

USER>zw ^User.carsI

Эти данные созданные в классе хранятся в глобалах и по структуре напоминают, созданные в SQL

**Заходим в режим SQL**

Просмотрим созданный экземпляр на Портале в SQL

select \* from Test.Car

insert into Test.car (number, owner, brand, color) values ("BB100B","Петров","Renault","Б Е Л Ы Й")

select \* from Test.Car

Просмотрите глобалы на Портале.