Объектная модель данных

Бессарабов Н.В.
bes@fpm.kubsu.ru

2016 г.

Цели лекции

Существует два основных варианта реализации объектной парадигмы в базах данных – объектное расширение модели данных SQL и объектная модель ODMG.

Заметим, что в этом курсе мы будем говорить только о моделях стандартов ODMG 3.0 и SQL:2003.

В настоящей лекции рассмотрим один из возможных вариантов реализации объектной модели ODMG в СУБД Caché . Выбор этой СУБД для курса введения в базы данных сделан потому, что только в Caché можно изучать объектную, реляционную и иерархическую модели данных и, самое главное, без дополнительных усилий наблюдать их взаимодействия.

Сравнения стандартов ODMG 3.0 и SQL:2003 будут проведены после изучения объектной (в Caché) и объектно-реляционной (объектное расширение SQL в Oracle) моделей данных.

© Бессарабов Н.В.2016

Зачем нужны объекты в базах данных?

Реально существующие объекты характеризуются **состояниями**, которые могут изменяться после наступления некоторых **событий.**

Для реляционной и ER-моделей описание **жизненных циклов** (т. е. последовательностей состояний) и самих состояний связаны с определёнными трудностями. Моделирование:

- динамических связей между объектами (связь имеет состояния, может создаваться, изменяться, уничтожаться),
- **сообщений**, которыми могут обмениваться объекты в модели,
- выполнения объектами действий

не предусмотрены базовой семантикой этих моделей и потому затруднительны или невозможны.

Пример: состояния сущности “Заказ”:

- авансирован,
- выполняется,
- приостановлен,
- оплачен,
- завершён,
- принят.

Что делать, если необходимо описывать действия и сообщения?

-- Переходить к объектным моделям

-- Использовать процедурные языки в дополнение к SQL

Но это не решает всех проблем

Особенности архитектуры Caché

Объектная система Caché построена на объектном расширении персистентного языка ObjectScript. Уникальная особенность системы в том, что она позволяет работать с данными одновременно в объектной, реляционной и иерархической моделях, не программируя никаких отображений (mappings). Легко получаемые в Caché дедуктивная, полуструктурированная и другие модели, включая так называемые NoSQL, позволяют считать Caché полимодельной СУБД.

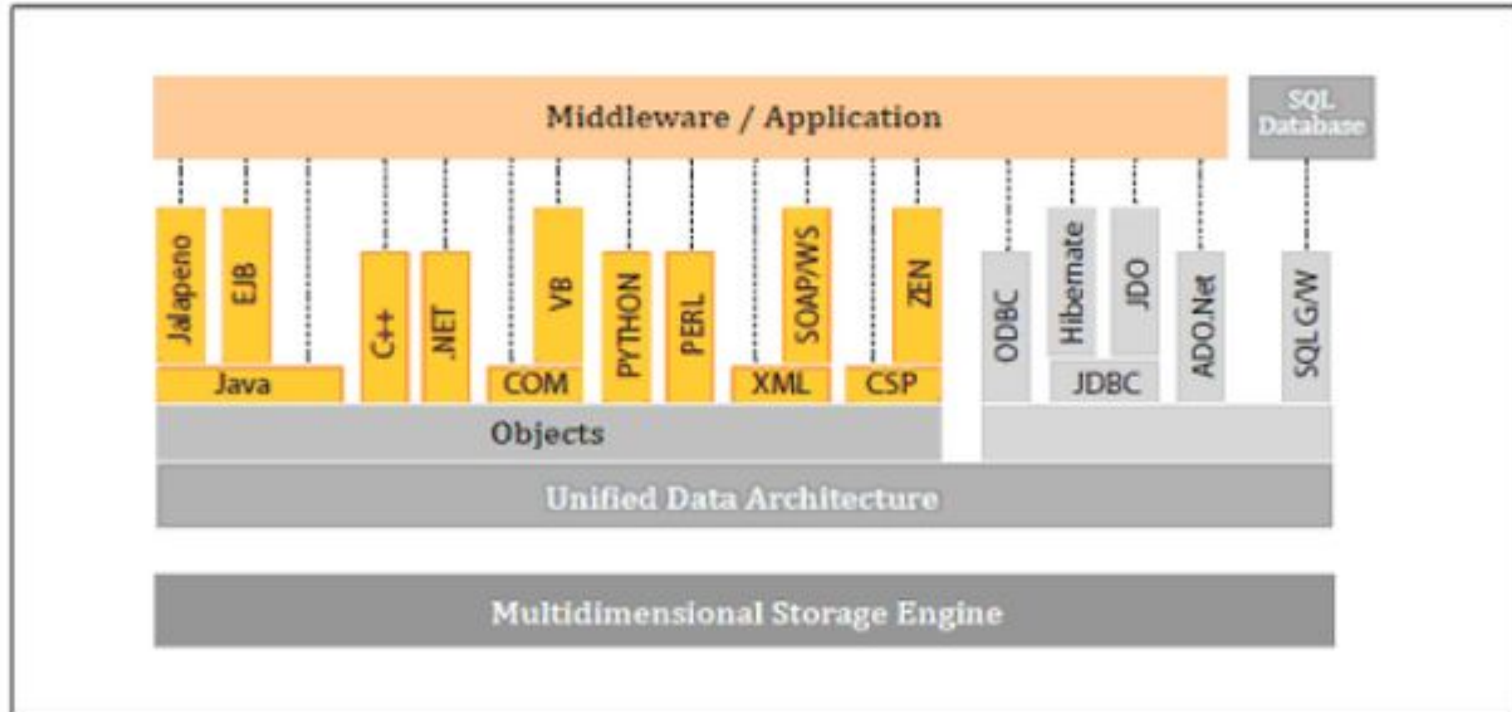
Универсальная архитектура Caché

В универсальной архитектуре Caché (см. след. слайд) данные и объектов и таблиц отображаются в многомерные структуры, хранением которых заведует механизм многомерной памяти.

Унифицированные структуры данных доступны и серверу объектов и SQL-серверу. Программное обеспечение промежуточного уровня может обращаться к одному из этих серверов. Шлюз SQL позволяет обмениваться данными с другими базами реляционного типа.

© Бессарабов Н.В.2016

Универсальная архитектура Caché



Middleware/Application	Промежуточное ПО/Приложение
SQL Database	База данных SQL
Objects	Объекты
Unified Data Arhitecture	Унифицированная архитектура данных
Multidimensional Storage Engine	Средство многомерного хранения

© Бессарабов Н.В.2016

Система классов Caché (1/3)

Как всегда, класс задаёт шаблон, по которому создаются объекты определённого им типа. Объект — это экземпляр класса. Метод представляет собой функции или процедуры класса или объекта, определяющие его поведение. **Свойства класса не используются для идентификации объектов**, как в реляционной модели. Эту роль играют два идентификатора OID и OREF.

- Можно считать понятия класса и типа синонимами. Как и в естественных языках, объёмы понятий-синонимов перекрываются, но не совпадают. Предопределённые типы данных инкапсулированы, то есть их определения не доступны для изменения. Однако, пользовательские типы открыты. Обычно классы выстраиваются в иерархию наследования. У типов этого нет. Типы данных не могут содержать свойств. Для типов данных невозможно создавать экземпляры.

Замечания о странностях терминологии Caché

© Бессарабов Н.В.2016

Система классов Caché (2/3)

В Cache принята следующая **нестандартная терминология.** Классы вообще подразделяются на классы типов данных и классы объектов. Классы типов данных определяют допустимые значения констант (литералов) и позволяют их контролировать. Классы типов данных содержат предопределённые наборы методов проверки и приведения значений атрибутов к другим типам.

Незарегистрированные классы (Non-registered Classes) предназначены для создания пользовательской объектной системы. Мы с ними работать не будем.

© Бессарабов Н.В.2016

Система классов Caché (3/3)

Зарегистрированные классы обладают предопределенным поведением, задаваемым набором встроенных функций, наследуемых из системного класса %RegisteredObject (знак процента определяет системность класса или метода) и отвечающих за создание новых объектов и за управление размещением объектов в памяти.

Зарегистрированные классы делятся на хранимые и встраиваемые. Хранимые классы это потомки класса %Persistent. Они хранятся независимо и потому имеют уникальную и неизменяемую объектную ссылку OID, по которой объект может быть найден на диске, и ссылку OREF для обращения к ним в памяти. Хранимые классы используют весь набор методов класса %Persistent. Здесь конструкторы, методы подкачки объектов, удаления объектов и т.д.

Встраиваемые классы наследуют своё поведение от класса %Library.SerialObject. Они могут попасть на диск только в составе хранимого класса и потому имеют OREF но не имеют OID.

© Бессарабов Н.В.2016

Структура класса Caché

Всегда в ООП

- **Имя класса** понимается в обычном для ООП смысле.
- **Параметры.** Изменяют возможности класса во время его компиляции**.** Обычно для ООП
- **Свойства --** в обычном для ООП смысле.
- **Методы** понимается в обычном для ООП смысле.
- **Запросы** – операции с объектами класса, играющие роль фильтров
- **Индексы** – необычные для ООП элементы, используются для ускорения доступа.
- **Триггеры** – используются только в табл. модели.

Только в Caché

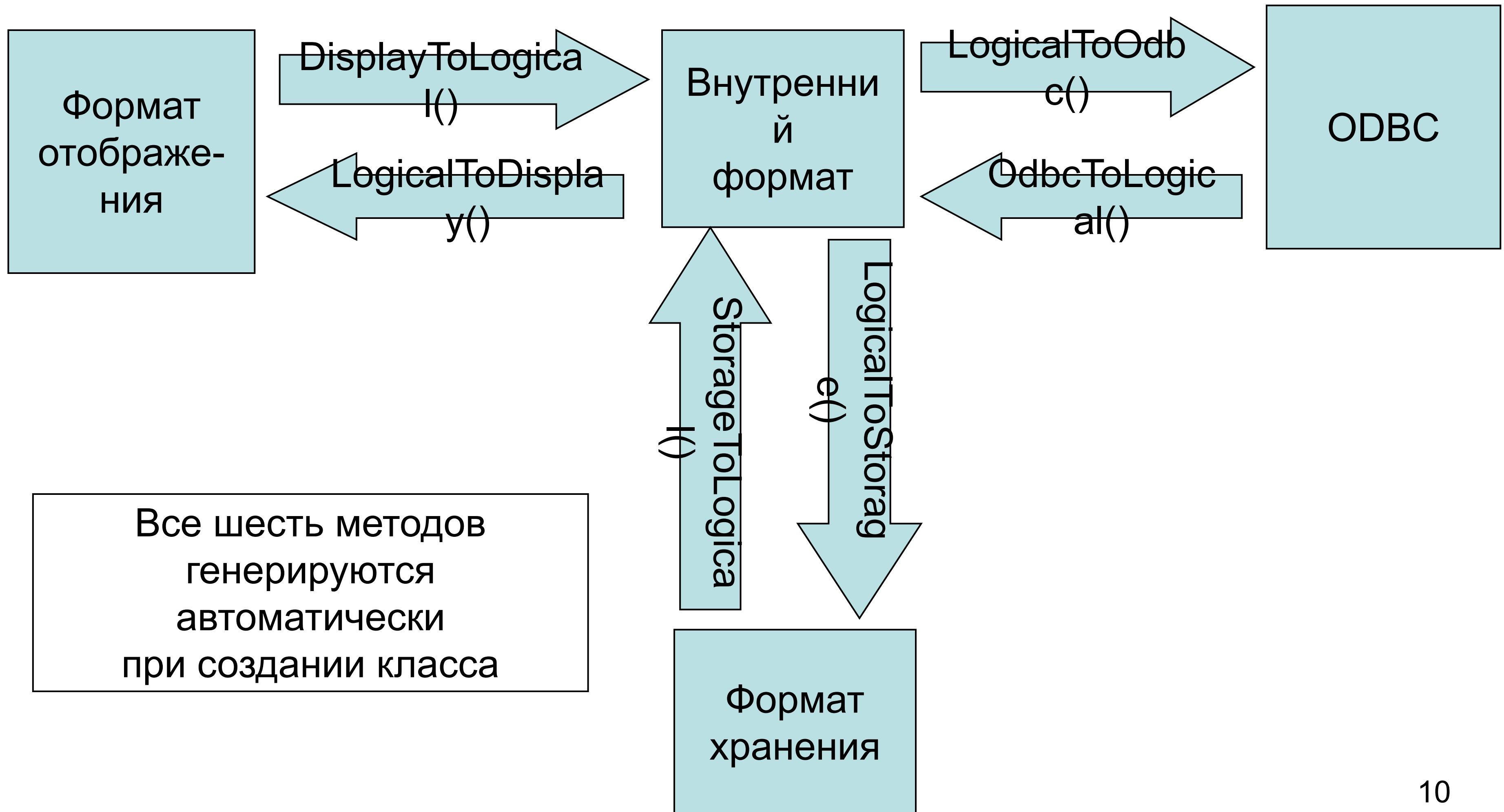
Имя класса
Параметры
Свойства
Методы
Запросы
Индексы
Триггеры

Заметим, что свойства это константы предопределённых типов, ссылки на объекты, встроенные объекты, потоки данных (BLOB и CLOB), коллекции (массивы и списки), отношения.

В языке UML в структуре класса предусмотрено его имя, атрибуты, операции, сигналы.

© Бессарабов Н.В.2016

Форматы данных и их преобразования



© Бессарабов Н. В.2010

Методы преобразования типов

Метод	Назначение
1) DisplayToLogical()	Преобразует отображаемое значение в внутренний формат
2) LogicalToDisplay()	Преобразует значение из внутреннего формата в формат отображения
3) LogicalToOdbc()	Преобразует значение из внутреннего формата в формат ODBC (опциональный метод)
4) OdbcToLogical()	Преобразует значение из формата ODBC во внутренний формат (опциональный)
5) LogicalToStorage()	Преобразует значение из внутреннего формата в формат базы данных (опциональный)
6) StorageToLogical()	Преобразует значение из формата базы данных во внутренний формат (опциональный)

© Бессарабов Н.

Предопределенные типы данных

Тип данных CACHE	CLIENTDATATYPE	ODBCTYPE	SQLCATEGORY
%Library.Binary	BINARY	BINARY	STRING
%Library.Boolean	IINTEGER	IINTEGER	IINTEGER
%Library.Currency	CURRENCY	CURRENCY	CURRENCY
%Library.Date	DATE	DATE	DATE
%Library.Float	DOUBLE	DOUBLE	DOUBLE
%Library.Integer	INTEGER	INTEGER	INTEGER
%Library.List	LIST	VARCHAR	STRING
%Library.Name	VARCHAR	VARCHAR	NAME
%Library.Numeric	NUMERIC	NUMERIC	NUMERIC
%Library.String	VARCHAR	VARCHAR	STRING
%Library.Time	TIME	TIME	TIME
%Library.TimeStamp	TIMESTAMP	TIMESTAMP	TIMESTAMP

© Бессарабов Н.

Свойства

Свойства представляют собой константы предопределенных типов, ссылки на объекты, встроенные объекты, потоки данных (BLOB, CLOB), коллекции, древесные значения и отношения:

- Константы

Пример: Property Name As %String(MAXLEN = 20);

- Ссылки на объекты

Каждый класс это тип данных.

Пример: Пусть имеется хранимый класс Address. Тогда свойство Address можно описать так

Property Addr As Address;

- Встроенные объекты

Пример: Пусть имеется встраиваемый класс Address. Свойство Addr записывается точно так же как в предыдущем варианте,

Property Addr As Address;

но речь идет не о ссылке, а о встраивании объекта.

© Бессарабов Н.

Пять способов создания класса в Cache

- Создание таблицы (нельзя определить методы)
- Использование мастера Studio
- Написание текста в Studio
- Задание из терминала в COS
- Импорт из UML

© Бессарабов Н.

Способ 1: Задание таблицы

- Предварительно проверяем в портале, не существует ли глобал ^QQD и класс ^User.QQ.cls в области имён User. Если существуют, удаляем их. Смысл этих действий будет понятен далее.
- Исполняем в SQL-менеджере портала в области User команду

  create table QQ (c1 char(3), c2 number(4))
- В разделе “Классы” портала обнаруживаем класс ^User.QQ.cls, щелкнув по позиции “Документация” получаем его описание.
- В Studio для той же области User вызываем (Файл – Открыть ..) описание класса User.QQ.cls :

```
Class User.qq Extends %Persistent [ ClassType = persistent, DdlAllowed, Owner =
    "", SqlRowIdPrivate, SqlTableName =QQ, StorageStrategy = ]
    {

    Property c1 As %Library.String(MAXLEN = 3) [
                                    SqlColumnNumber = 2 ];

    Property c2 As %Library.Numeric(MAXVAL = 9999, MINVAL =
                                    -9999, SCALE = 0) [ SqlColumnNumber = 3 ];

    }””
```

- Проверяем, не появился ли глобал ^QQD в области имён User.

<u>Вывод</u>: При создании таблицы появляется соответствующий класс.

© Бессарабов Н.

Способ 2. Использование мастера (1/10)

Шаг 1. Проверяем в портале, не существует ли глобал ^HumanD, класс ^User.Human.cls и таблица Human в области имён User.

В Studio вызываем мастера, задаём область имен User, имя класса и комментарий

Мастер создания Класса

Вас приветствует Мастер создания Класс

Этот мастер поможет Вам добавить Cache Класс. Следуйте инструкциям, нажимайте кнопку "Далее" для перехода на следующую страницу. Вы также можете закончить в любой момент, нажав кнопку "Закончить".

Укажите имя пакета:

User

]росмотр.

Укажите имя Класса:

Human

Укажите описание данного Класса (необязательно):

Это класс первого лекционного примера класса созданного мастером

< Назад | Далее

Что изменится, если не задавать имя пакета?

© Бессарабов Н.

Способ 2. Использование мастера (2/10)

Шаг 2. Выбор вида класса

Мастер Класса

Тип класса

Какого типа класс Вы желаете создать? Сделайте выбор одного из типов:

- (•) Persistent (может храниться в базе данных)
- ( ) Serial (может быть встроен в объекты Persistent)
- ( ) Registered (не могут храниться в базе данных)
- ( ) Абстрактный
- ( ) Тип данных
- ( ) CSP (применяется для обработки HTTP событий)
- ( ) Extends

Имя СуперКласса:

Поиск...

< Назад | Далее > | Готово | Отмена | Справка

Множественное наследование !!

© Бессарабов Н.
В.2010

Способ 2. Использование мастера (3/10)

Шаг 3. Выбор владельца, имени таблицы, отличного от имени класса, поддержки XML и автозаполнения данными

Мастер Класса

Вы можете добавить дополнительные характеристики для этого хранимого класса:

Владелец (необяз.):

Имя таблицы SQL (необяз.):

☐ Этот класс поддерживает XML.

☐ Этот класс поддерживает автоматическую процедуру заполнения данными.

< Назад | Далее > | Готово | Отмена | Справка

© Бессарабов Н.

Способ 2. Использование мастера (4/10)

Шаг 4. Смотрим полученный результат в Studio

```
/// Это класс первого лекционного примера класса созданного мастером
    Class User.Human Extends %Persistent [ ClassType = persistent,
    ProcedureBlock ]
    {

    }
```

Заметим, что описание класса передано вручную введённым комментарием.

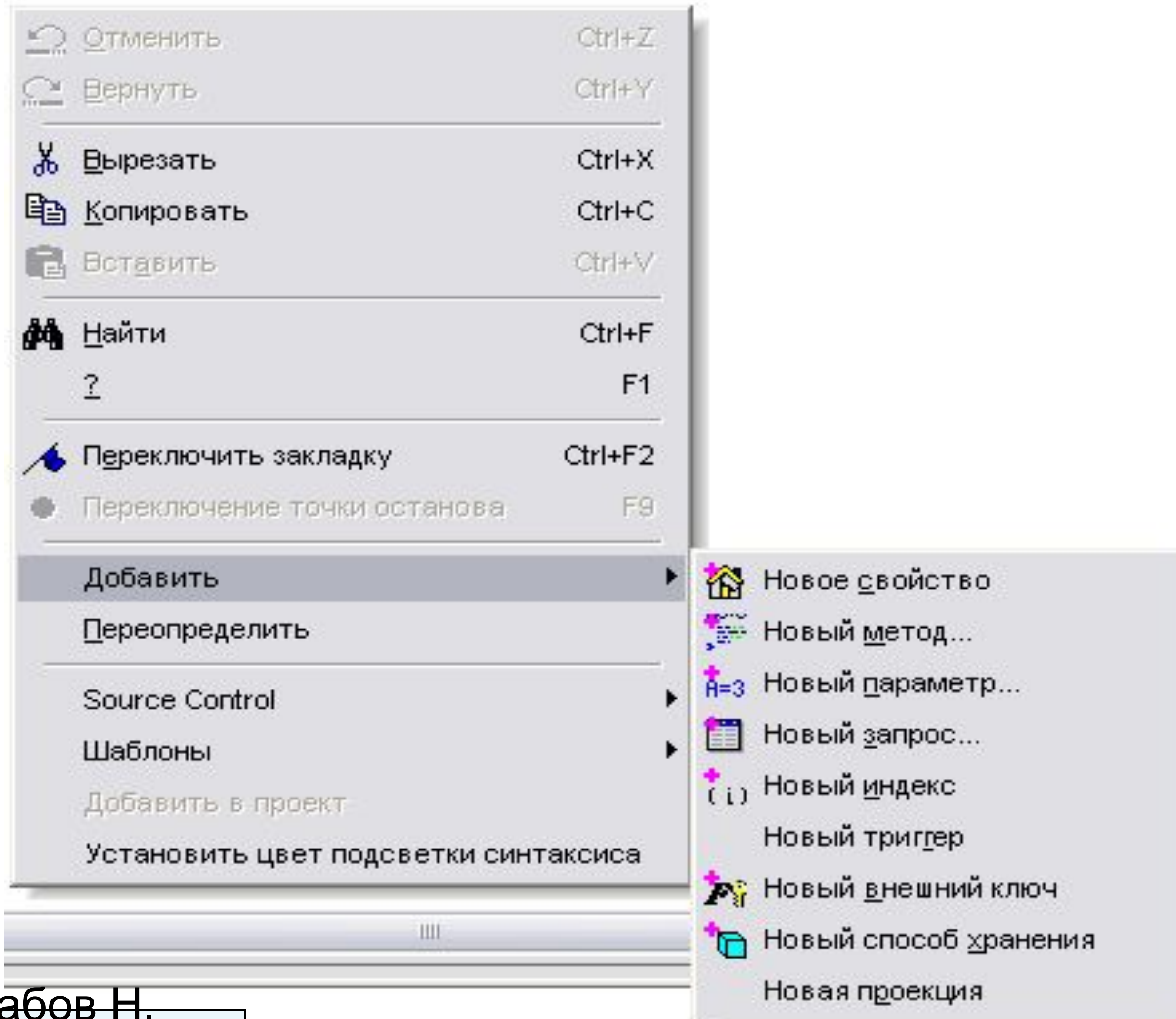
Поскольку создавался хранимый класс, то ^User.Human наследует (Extends) системному классу %Persistent .

Проверьте, появились ли таблица Human? В чём её особенности? А глобал ^HumanD?

© Бессарабов Н. В.2010

Способ 2. Использование мастера (5/10)

Шаг 5. В Studio выбираем «Новое свойство» (Правая кнопка мыши – Добавить)



© Бессарабов Н. В.2010

Способ 2. Использование мастера (6/10)

Шаг 6. Добавляем атрибуты используя появившийся мастер

Мастер создания Свойства

Вас приветствует Мастер Создания Свойства.

Этот мастер поможет Вам добавить новое Свойство в описание Класса. Следуйте инструкциям, нажимайте кнопку "Далее" для перехода на следующую страницу.Вы также можете закончить в любой момент, нажав кнопку "Закончить".

Укажите имя нового Свойства:

Name

Укажите описание нового свойства (необязательно):

Это имя

< Назад | Далее > | Готово | Отмена | Справка

© Бессарабов Н. В 2010

Способ 2. Использование мастера (7/10)

Шаг 7. Выбираем тип данных

Мастер создания Свойства

Тип свойства

Это Свойство:

- (•) Единичное значение типа: %String [Поиск...]
- ( ) Коллекция типа:
  - Содержит элементы типа: [Поиск...]
- ( ) Поток типа:
- ( ) Отношение

Большие типы данных

[< Назад] [Далее >] [Готово] [Отмена] [Справка]

© Бессарабов Н.

Способ 2. Использование мастера (8/109)

Шаг 8. Задаем свойства и переходим к параметрам типа

Мастер создания Свойства

Характеристики свойства

- [ ] Обязательно Это свойство обязательно (не пустое).
- [ ] Индексируемое Создать индекс для этого свойства.
- [ ] Уникально Создать уникальный индекс для этого свойства.
- [ ] Вычисляемое Это свойство не хранится.

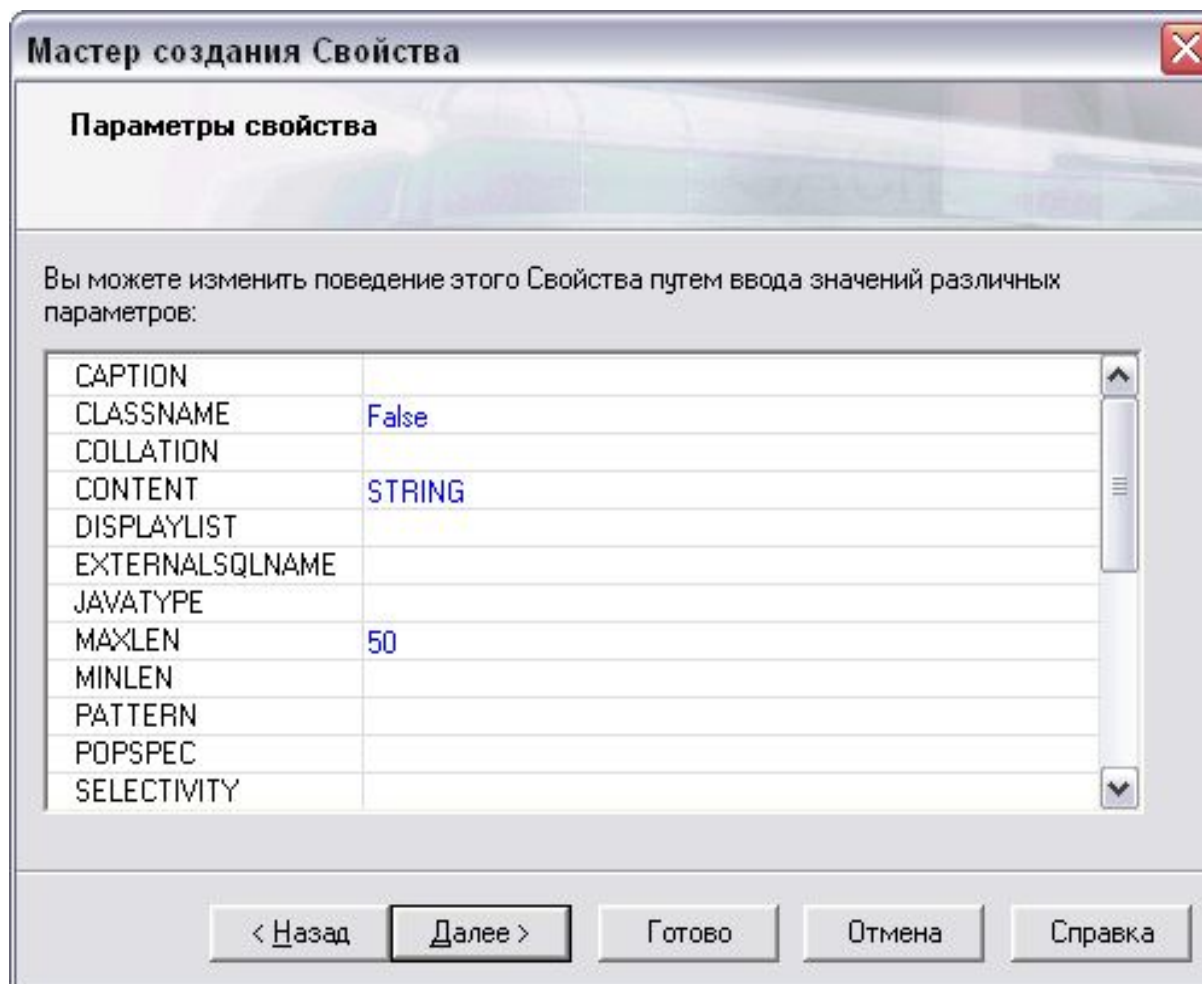
Имя поля для SQL (необяз.):

< Назад | Далее > | Готово | Отмена | Справка

© Бессарабов Н.

Способ 2. Использование мастера (9/10)

Шаг 9. Оставляем параметры типа по умолчанию



© Бессарабов Н. В.2010

Способ 2. Использование мастера (10/10)

Шаг 10. В Studio обнаруживаем текст, определяющий новое свойство:

```
{
        /// Это имя
        Property Name As %String;
        }
```

Просмотрите ещё раз таблицу Human и глобал ^HumanD

Этот же результат мог быть получен непосредственно вводом текста описывающего свойства в Studio. Но это уже способ 3 создания класса.

Способ 4 реализуется из UML-диаграммы при подключении инструмента Rational Rose

© Бессарабов Н.

Методы унаследованые от класса %Persistent

Прежде, чем мы рассмотрим 5-й способ создания класса – из COS – перечислим методы наследуемые от родительского класса %Persistent:

- %New(). Конструктор объекта. Его задача – создать экземпляр класса.
- %Save(). Сохраняет объект на диске.
- %Close(). Закрывает объект, то есть удаляет его из памяти
- %Open(). Метод класса. Если он находит объект существующий в базе данных, то создает в памяти его копию, содержащую значения всех свойств, и возвращает объект. Если объект уже загружен в память, просто возвращается OREF. Вообще у метода три аргумента. Второй аргумент Concurrency определяет особенности параллельной работы и принимает значения 0, 1, 2, 3, 4. По умолчанию установлен в “1”, что означает создание разделяемой блокировки при загрузке объекта в память.
- %OpenID().
- %Delete().
- %IsModified().

© Бессарабов Н.

Работа с объектами в COS (1/3)

- Создадим простейший класс с единственным атрибутом Name.

```
Class User.A Extends %Persistent [ ClassType = persistent,
 ProcedureBlock ]
{
Property Name As %String(MAXLEN = 20);
}
```

- Создадим экземпляр класса с помощью метода %New():

```
s ss=##class(User.A).%New()
```

Макроподстановка ##class создает объектную ссылку OREF. Что же представляет собой эта ссылка?

```
w ss
```

Ответ:

1@User.A

Итак, OREF состоит из двух частей имени класса “User.A” и идентификатора объекта ”1”.

Вторая ссылка OID читается методом %Oid():

```
w ss.%Oid()
User.A
```


© Бессарабов Н.

Работа с объектами в COS (2/3)

OID представляет собой список, состоящий из OID объекта и имени класса. Читаем его компоненты циклом с командой $list:

f i=1:1:$ll(ss.%Oid()) {w !,$li(ss.%Oid(),i)}

1

User.A

- Для того, чтобы завершить создание объекта необходимо назначить значения его атрибутов и сохранить его на диск. Если объект дальше не будет использоваться, необходимо удалить его из памяти.

s ss.Name="John" // параметру Name объекта № 1 присвоено значение.

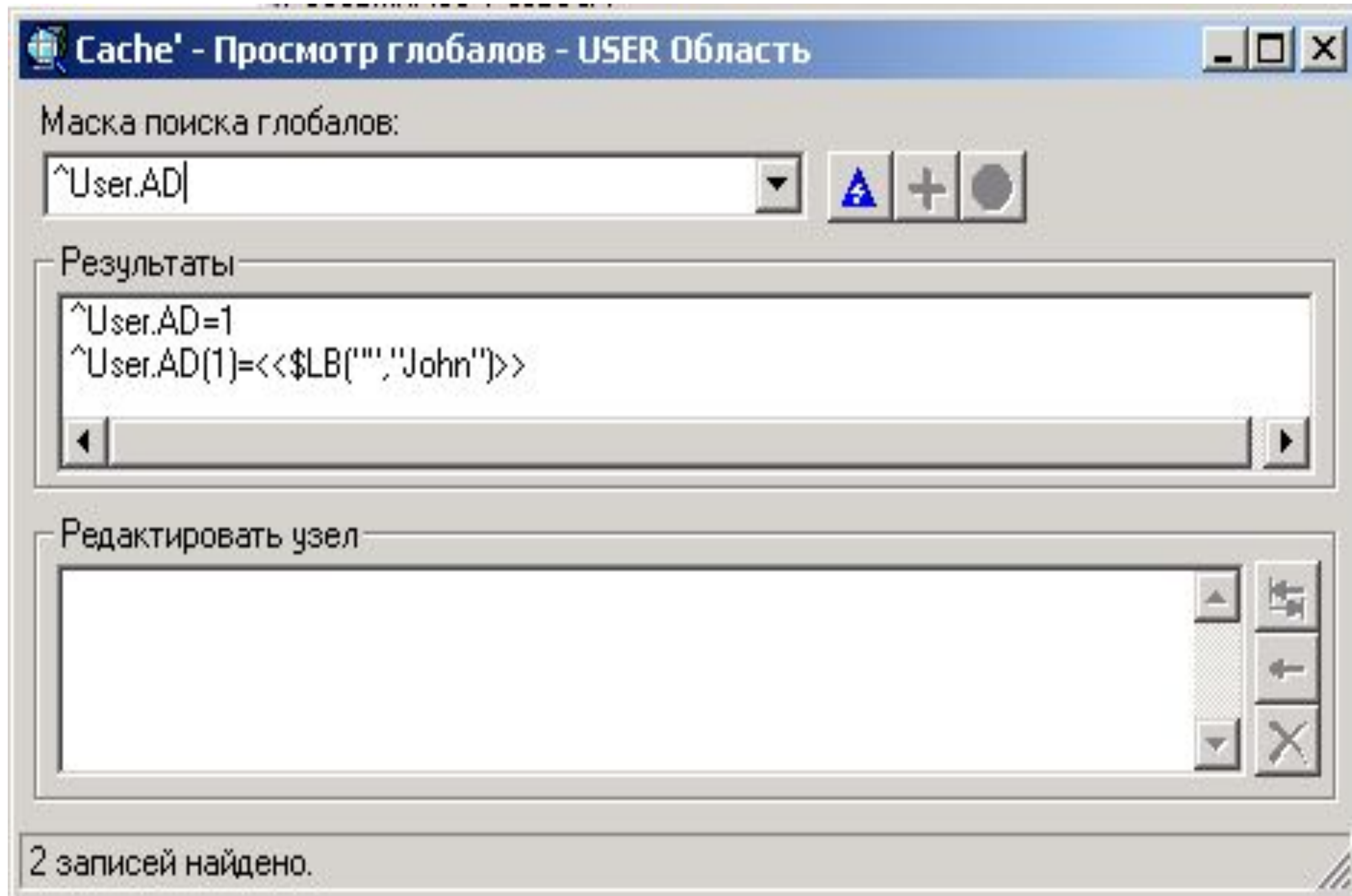
d ss.%Save() // объект № 1 сохранен на диске.

d ss.%Close() // объект № 1 закрыт, то есть удален из памяти.

Вывод: Задание первого объекта класса образует глобал. Проверим проводником образовался ли глобал USER.AD.

© Бессарабов Н. В.2010

Работа с объектами в COS (3/3)



© Бессарабов Н.

Классы, таблицы, объекты, строки и деревья

Теперь понятны связи между иерархической, табличной и объектными моделями в Caché.

Оказывается, таблицы эквивалентны классам без методов, столбец таблицы соответствует атрибуту класса, строка таблицы отображается в объект соответствующего класса.

Как только создаётся строка таблицы или же объект, так сразу же создаётся глобал в виде дерева глубины 1. Работая с его узлами, можно манипулировать строками в табличном представлении или объектами в объектной модели.

Понятно, что основой такого симбиоза трёх моделей может быть только объектная модель. В табличной и иерархических моделях нет места для методов и объектных типов данных.

© Бессарабов Н.

Таблиц в Caché не бывает

В Studio создаём класс T, не смущаясь незнанием языка CDL (Class Define Language) на котором он написан:

```
Class User.T Extends %Persistent [ ClassType = persistent, DdlAllowed, SqlRowIdPrivate, SqlTableName = T ]
{
Property c1 As %Library.Numeric(MAXVAL = 99, MINVAL = -99, SCALE = 0) [ SqlColumnNumber = 2 ];
Property c2 As %Library.String(MAXLEN = 3) [ SqlColumnNumber = 3 ];
}
```

Пока его не компилируем. В разделе SQL портала управления системой проверяем, не существует ли таблица SQLUser.T. Если существует, удалим её.

Теперь компилируем класс не обращая внимания на строки описания добавленные Студией. Появляется таблица SQLUser.T.

© Бессарабов Н. В.2010

Виртуальная таблица SQLUser.T

Портал сообщает об этой таблице следующее:

Таблица: SQLUser.T ◉ сведения таблицы ○ поля ○

Владелец	_SYSTEM
Дата последней компиляции	2014-11-18 14:22:17
Внешний	0
Только для чтения	0
Имя класса	User.A
Размер экстента	100000
%CacheStorage?	Да

Таблица: SQLUser.T ○ сведения таблицы ◉ поля ○ индексы ○ триггеры ○ ограничения ○ кэшированные запросы

Столбец	Тип данных	№ столбца	Обязательный	Уникальный	Сортировка	Hidden	MaxLen	BLOB	Контейнер	Селективность	Тип xDBC	ReferenceTo	Столбец версии
ID	%Library.Integer	1	Yes	Yes		Yes		No		1	INTEGER		No
c1	%Library.Numeric	2	No	No		No		No			NUMERIC		No
c2	%Library.String	3	No	No	SQLUPPER	No	3	No			VARCHAR		No
x_classname	%Library.CacheString	4	No	No		Yes		No			VARCHAR		No

Обратите внимание на два "непрошенных" столбца ID и "x_classname".

© Бессарабов Н.

Сравниваем таблицу SQLUser.T и породивший её класс User.T

Понятно, что в первой строке записано имя класса User.T, а свойства c1 и c2 соответствуют именам столбцов c1 и c2. Понятно, что ширина столбца c2 равна 3. SqlColumnNumber = 2 и 3. Столбец ID играет роль суррогатного ключа, соответствует OID, а столбец x_classname имеет объектный тип %Library.CacheString.

Убедитесь, что созданная таблица пустая.

Проверим на всякий случай, не существует ли глобала с именем T, после которого приписана буква D. Если глобал ^User.TD существует, удалите его. Теперь в SQL-менеджере введём в таблицу T одну строку:

insert into T values (22, “QQ”)

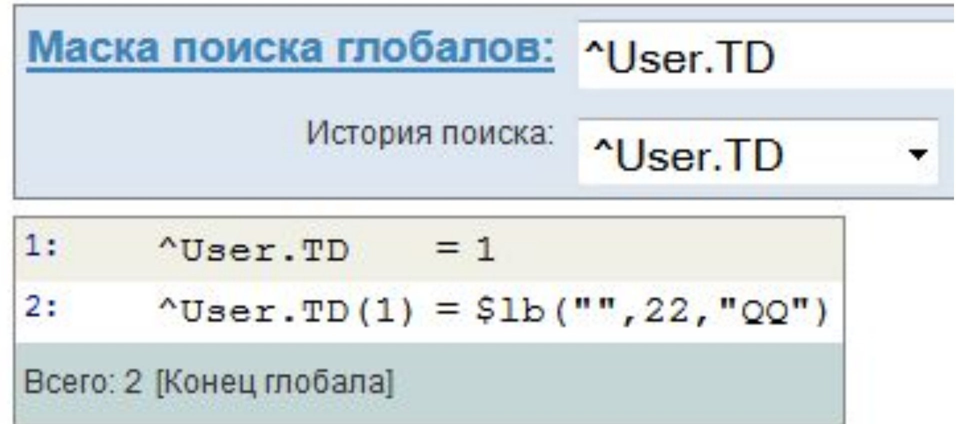
С помощью команды select * from T убеждаемся, что строчка действительно записана.

Переходим в проводник и в папке “Глобалы” обнаруживаем глобал ^User.TD. Если он не появился, нажмите на кнопку F5.

© Бессарабов Н.

Созданный глобал ^User.TD

Интересно, как выглядит вновь созданный глобал. Щёлкаем левой кнопкой мыши по позиции “Просмотр” в строчке ^User.TD и обнаруживаем структуру:



Если добавить ещё одну строку, например 1, “A”, выполнив команду insert into T values (1, "A")

то дерево изменится так:

Корень дерева хранит число строк в таблице

^User.TD=2

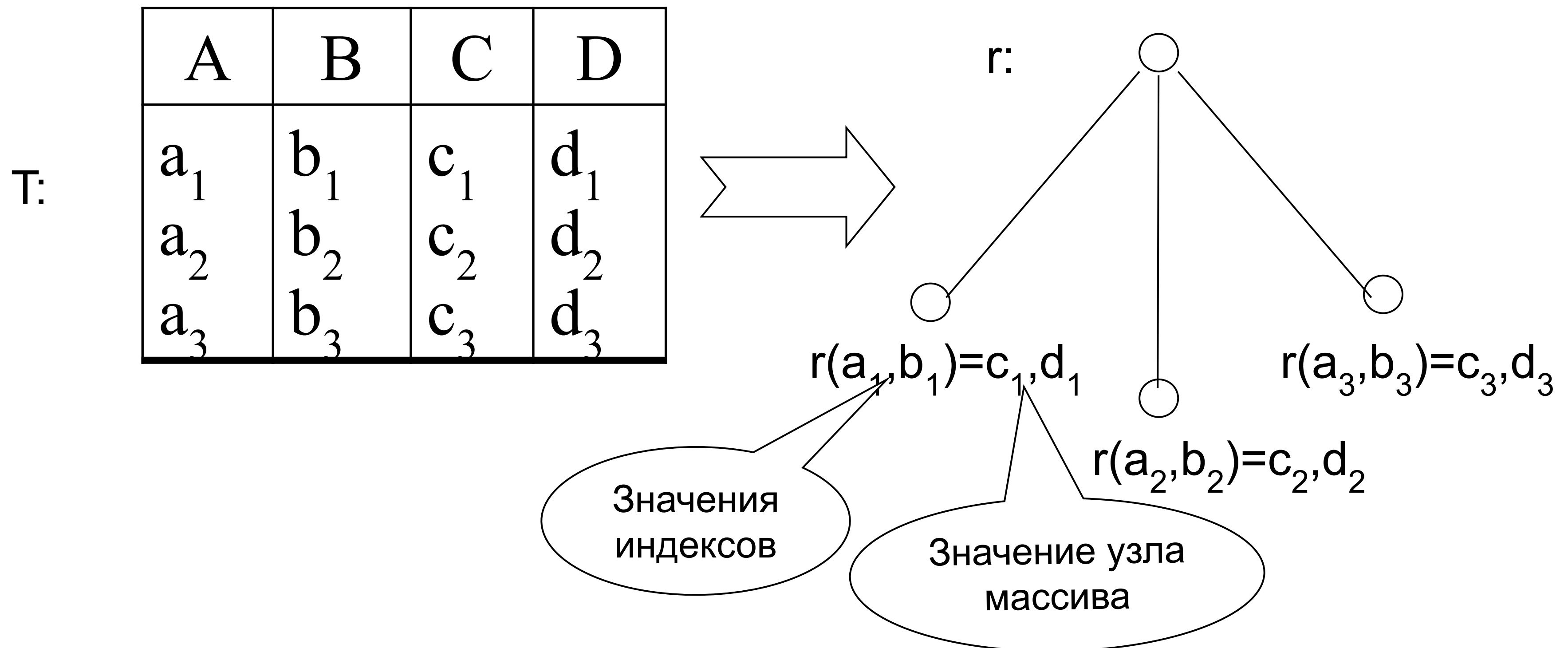
^User.TD(1)=<<$LB("","22","QQ")>>

^User.TD(2)=<<$LB("","1","A")>>

Теперь можно работать непосредственно с глобалом.

© Бессарабов Н.

Представление таблицы деревом



© Бессарабов Н. В.2012

Наследование (1/2)

Классы могут быть наследниками других классов. Для примера создадим класс человек (human) следующей структуры

```
Class User.Human Extends %Persistent
{
Property Pass As %String(MAXLEN = 11);
Property Name As %String(MAXLEN = 20);
}
```

В нём name – имя, pass – номер и серия паспорта.

Класс-наследник student расширяет базовый класс свойством NNZach - номер зачетной книжки:

```
Class TestLib.Student Extends TestLib.Human
{
Property NZach As %String(MAXLEN = 10);
}
```

При этом создаются две таблицы Human и Student причем поля Name и Pass в таблице Student будут виртуальными. То есть при создании новой записи в таблице Student значения этих полей сохраняются в таблице Human и информация о них извлекается по внешней ссылке.

© Бессарабов Н. В 2016

Наследование (2/2)

В классе Human создадим объект Пётр (”0305 855637”, “Петр”), а в классе Student – объект Иван(“0305 163788”, “Иван”, “8765”).

USER>s stud=##Class(User.Student).%New()

USER>s stud.Name= "Иван“

USER>s stud.NZach=" 8765 “

USER>s stud.Pass=" 0305 855637 “ /не забудьте сохранить и закрыть

Данные обоих классов помещаются в один глобал ^User.HumanD:

```
1:   ^User.HumanD                = 2
2:   ^User.HumanD(1)             = $lb("","0305 855637","Петр")
3:   ^User.HumanD(2)             = $lb("~Student~","0305 163788","Иван")
4:   ^User.HumanD(2,"Student")   = $lb("8765")
Всего: 4 [Конец глобала]
```

Запросами SELECT * … просмотрим содержимое этих таблиц

#	ID	Name	Pass
1	1	Петр	0305 855637
2	2	Иван	0305 163788
Завершено			

#	ID	NZach	Name	Pass
1	2	8765	Иван	0305 163788
Завершено				

Один вывод очевиден --“Студент это человек”, а как с наследованием в SQL-представлении данных?

© Бессарабов Н. В.2016

Сериализуемые классы (1/2)

Создадим встроенный класс Addres:

```
Class User.Address Extends %SerialObject
{
Property City As %String;
Property State As %String;
}
```

Используем его в классе Person:

```
Class User.Person Extends %Persistent
{
Property Name As %String;
Property YearOB As %Integer;
Property Home As Address;
}
```

Создаём объект класса Person:

```
S p=##class(User.Person).%New()
S p.Name="Nick", p.YearOB=1984, p.Home.City="NewYork"
S p.Home.State="NY"
D p.%Save()
```


© Бессарабов Н.

Сериализуемые классы (2/2)

Образовался глобал:

^User.PersonD=1

^User.PersonD(1)=$LB("","Nick","1984",$LB("NewYork","NY"))

Объект сериализуемого класса в нём представляется списком: $LB("NewYork","NY").

#	ID	Name	YearOB	Home_City	Home_State
1	1	Nick	1984	NewYork	NY
Завершено					

В SQL-проекции эти столбцы действительно существуют. К ним можно обратиться по имени, например:

SELECT Name, Home_City FROM Person

© Бессарабов Н. В.2010

Отношения

Пусть имеются два класса – Юрист и Клиент:

```
Class User.Lawyer Extends %Persistent
{
Property LawyerName As %String [Required];
}

Class User.Client Extends %Persistent
{
Property ClientName As %String [ Required ];
}
```

В каждый из них добавим атрибут-отношение, определяющий связь один-ко-многим.

Получаем:

© Бессарабов Н. В.2012

Отношения

```
Class User.Lawyer Extends %Persistent
{
Property LawyerName As %String [ Required ];
Relationship MyClients As User.Client [ Cardinality=many, Inverse=MyLawyer ];
}
```

и

```
Class User.Client Extends %Persistent
{
Property ClientName As %String [ Required ];
Relationship MyLawyer As User.Lawyer [ Cardinality = one, Inverse = MyClients ];
}
```

Компилируем их совместно.

Теперь остаётся создать:

- экземпляры (объекты) обоих классов, задавая только свойства (Property)
- экземпляры связей между объектами юристов и клиентов.

Для решения второй задачи необходимо сначала создать ссылку на какой-нибудь объект Lawyer (например,LawyerOref) затем ссылку на объект Client, (например, ClientOref1) который будет с ним связан и, наконец, присвоить атрибуту-ссылке клиента значение атрибута-ссылки юриста, например,

Set ClientOref1.MyLawyer=LawyerOref ;ссылки ClientOref1 и LawyerOref созданы методом %New().

© Бессарабов Н.

Метаданные в Caché (1/3)

Метаданные в Caché хранятся в метаклассах двух видов:

- Defined --представляют определения классов; включают только информацию, о членах класса описанных в нём, но не содержат унаследованных членов классов.
- Compiled -- представляют скомпилированные классы; содержат информацию об унаследованных членов классов.

Метакласс	Описание
ClassDefinition	Хранит общие сведения о других классах
PropertyDefinition	Описания атрибутов класса
IndexDefinition	Определение индекса, в т.ч. перечень атрибутов на которых создан индекс
MethodDefinition	Определение методов, в т.ч. тип возвр. значения, метод класса или экземпляра
ParameterDefinition	Определение параметра класса
QueryDefinition	Определение *SQL*-запроса
TriggerDefinition	Определение триггера, в т.ч. триггерное событие, код .

Метаданные в Caché (2/3)

Рассмотрим структуру двух классов из перечисленных в таблице на предыдущем слайде.

Полное имя класса *ClassDefinition* это *%Dictionary.ClassDefinition*. Его поля:

- Name – имя класса.
- Properties – атрибут (объект метакласса PropertyDefinition).
- ClassType – тип класса (persistent или serial).
- Super – содержит имена базовых классов.
- *Description* – поле описания класса.
- *Abstract* – определяет абстрактность класса.
- *Final* – возможность наследования от класса.
- *Indices* – связи, предназначенные для описания индексов класса.
- *Methods* – связи, предназначенная для описания методов класса.
- *Parameters* – связь, предназначенная для описания параметров класса.

Метаданные в Caché (3/3)

Класс PropertyDefinition, хранящий сведения об атрибутах. Минимальный набор из двух полей, обеспечивающий добавление атрибута в класс:

- Name – поле имени атрибута.
- Type – поле типа атрибута.

Другие поля:

- MultiDimensional – указывает, что атрибут это многомерный массив.
- CollectionAs – указывает, что атрибут это «коллекция».
- Description – поле комментария к атрибуту.
- Calculated – объявление атрибута вычислимым.
- InitialExpression – задание начального значения атрибута.
- NotInheritable – указывает, что атрибут не наследован.
- ParametersAs – поле массива параметров атрибута.
- Private – указания на закрытость атрибута.
- Relationship – указания на то, что атрибут является связью.
- Required – поле указания на обязательность атрибута.
- Transient – указание на то, что атрибут не хранится в базе.
- Parent – поле родительской связи.

Заключение

Итак, изучены основы объектной модели ODMG. Конечно, следовало бы посмотреть на предоставляемые возможности изменения структур хранения данных, изучить возможности индексации, включая bit-slice индексы, вникнуть в интереснейший класс %ResultSet. К сожалению время, выделенное нам на изучение баз данных слишком ограничено.

Несколько расширить свои знания Caché можно проработав первую часть главы 10 книги. Многое может дать участие в конкурсах IT-планета по Caché и DeepSee (это такая интересная реализация многомерной модели данных, используемая в бизнес-аналитике).

© Бессарабов Н.