Объектно-ориентированный транслятор для программируемой реляционной системы.

Евгений Григорьев

2012
Москва

Целевая машина (компьютер)

- **1. *Данные*.** Компьютер должен обрабатывать **различные типы** простейших элементов **данных** и структур данных.
- **2. *Элементарные операции*.** Компьютер должен иметь **набор элементарных операций** для работы с данными.
- ***3. Управление последовательностью действий.*** Компьютер должен обеспечивать **управление последовательностью выполнения** элементарных **операций**.
- **4. *Доступ к данным*.** Компьютер должен предоставлять **механизмы управления данными**, которые необходимы для выполнения любой операции.
- **5. *Управление памятью.*** Компьютер должен предоставлять **механизмы управления распределением памяти** для программ и данных.
- **6. *Операционная среда.*** Компьютер должен предоставлять **механизмы связи с внешней средой**, содержащей программы и данные, подлежащие обработке.

Пратт Т., Зелковиц М.
"Языки программирования: разработка и реализация”

Особенности программируемой реляционной системы

- **Организация данных (описываются реляционная моделью данных)**
- **Персистентность данных**
- **Непроцедурный язык (команды управления памяти, доступа к данным)**

Программируемая реляционная система

Поддерживает существование и функционирование
реляционной базы данных

"The Third Manifecto " by H.Darwen and C.Date

- "…A **database** shall be a named container for **relvars**; the content of a given database at any given time shall be a set of database **relvars**…. "
- "…Database **relvars** shall be either real or virtual…"

"…Each **transaction** shall interact with exactly one **database**…"

Реляционная база данных есть контейнер для множества именованных переменных отношений **R** (хранимых или вычисляемых), с которым взаимодействует транзакции **T**.

DB: (…, R_i, … , T_j, …).

Команды П**Р**С

- Создание relvar **R (a_1:D_i, … a_n:D_j) KEY(…a_i…) FKEY (…) ON (…)**
  Некоторые relvar (виртуальные) могут вычисляться **R AS RValue**

- Выражение (запроса) **RValue(…R…)** возвращает значение отношения
  - Композиция **f(…R…)** операций реляционной алгебры **op** над переменными отношения **op_1(… op_2(R, op_3()…))**. Операции реляционной алгебры могут включать скалярные операции над значениями атрибутов **a** отношений **R**.
    - ***R1*** × ***R2*** – декартово произведение,
    - $R_1 \cup R_2$ – объединение,
    - $R_1 - R_2$ – разность,
    - R_1 **JOIN**$_{criteria}$ R_2 – соединение отношений (в качестве индекса используется критерий соединения). Также спользуется операция **LEFT JOIN…**.
    - ***R*[a_1, a_2, …]** – проекция (где a_i – атрибуты),
    - ***R* WHERE *criteria*** – выборка по критерию,
    - ***R* RENAME *a* AS *b*** – переименование атрибутов.
  - Процедура, возвращающей значение отношения (содержащей оператор **return RValue**)
  - Явно заданное значение отношение (в присваивании).

- Присваивание переменной значения **R := RValue**
  (эквивалентно традиционным операциям INSERT, UPDATE, DELETE)

- Создание транзакции **Tr(…) AS процедура**

- Выполнение транзакции **EXEC Tr(…)**

Команды ПРС

ПРС позволяет определить, сохранить и выполнить **процедуры** - алгоритмические (операторы **if**, **while** и т.д.) последовательности присваиваний $_{pr}$**R' := RValue(… $_{pr}$R …)** **(**где $_{pr}$**R** –переменные, доступные для присваивания в контексте процедуры**)** и вызовов **EXEC Tr(…)** ,

- Процедура может
  - использовать локальные переменные (вместе с relvar **R** базы данных входят в мн-во переменных, доступных для присваивания $_{pr}$**R**).
  - принимать атрибуты.
  - возвращать значение в вычисляемых компонентах (оператор **return RValue**).
- Хранимая процедура связывается с именем транзакции или вычисляемого компонента

```
name AS
  begin
    …
  end
```

Хранимая процедура выполняется по обращению к связанному имени

- транзакции **EXEC Tr(…)**
- виртуальной relvar **R**

- Допускается прямое выполнение последовательности операций

```
EXEC begin
    …
  end
```

Объекты и классы

Объект характеризуется инкапсулированными состоянием (сложное значение) и поведением

1) Структура сложных объектов есть совокупность переменных, типы которых реализуются целевой машиной

2) ПРС как целевая машина реализует тип отношение.

Объектная переменная есть совокупность именованных переменных отношений + связанный функционал	=	Реляционная база есть контейнер для множества именованных переменных отношений (хранимых или вычисляемых) + связанные транзакции.

Объекты и классы

Объект характеризуется инкапсулированными состоянием (сложное значение) и поведением

1) Структура сложных объектов есть совокупность переменных, типы которых реализуются целевой машиной

2) ПРС как целевая машина реализует тип отношение.

Объектная переменная есть совокупность именованных переменных отношений + связанный функционал	=	Реляционная база есть контейнер для множества именованных переменных отношений (хранимых или вычисляемых) + связанные транзакции.

Объект = реляционная БД

Объекты и классы

Вырожденные формы отношений

Могут быть заданы более простыми языковыми конструкциями

Состояние объекта описывается множеством значений, не более сложных, чем значение отношения

Далее:

Рассматриваем только отношения (сложные компоненты) и скаляры (простые компоненты)

Объекты и классы

Класс **D** - множество объектов заданной структуры.
Спецификация класса отделена от реализации.

- Спецификация **D ($^{sc}C_i$, … $^{r}C_j$, … M(…)) KEY (^{sc}C…)..**
  - **D** – имя класса
  - **^{sc}C** – простой компонент (скаляр),
  - **^{r}C** – сложный компонент (отношение),
  - **M(…)** – метод
  - **KEY ($_{sc}C$…)** – необязательные ключи класса

- Реализация
  - компонентов: <u>хранимые</u> **D.C AS STORED**
    <u>вычисляемые</u> **D.C AS RValue(…C…)**
  - методов: связываются с процедурами
    D_n.M(…) AS процедура

    Процедура – алгоритмическая последовательность
    - присваиваний **$_{pr}C$:= RValue(…$_{pr}C$…)**
    - вызовов **call M (…)**

- Возможно множественной наследование.

  D_n : EXTEND D_k, D_l … (…)

  Спецификация класса наследника объединяет спецификации родительских классов и собственных компонентов и методов. Наследуемые компоненты и методы могут менять реализацию.

Объекты и классы

Класс **D** - множество объектов заданной структуры.
Спецификация класса отделена от реализации.

- Спецификация **D ($^{sc}C_i$, … $^{r}C_j$, … M(…)) KEY (^{sc}C…)..**
  - **D** – имя класса
  - **^{sc}C** – простой компонент (скаляр),
  - **^{r}C** – сложный компонент (отношение),
  - **M(…)** – метод
  - **KEY ($_{sc}C$…)** – необязательные ключи класса

- Реализация
  - компонентов: <u>хранимые</u> **D.C AS STORED**
    <u>вычисляемые</u> **D.C AS RValue(…C…)**
  - методов: связываются с процедурами
    D_n.M(…) AS процедура

    Процедура – алгоритмическая последовательность
    - присваиваний **$_{pr}C$:= RValue(…$_{pr}C$…)**
    - вызовов **call M (…)**

- Возможно множественной наследование.

  D_n : EXTEND D_k, D_l … (…)

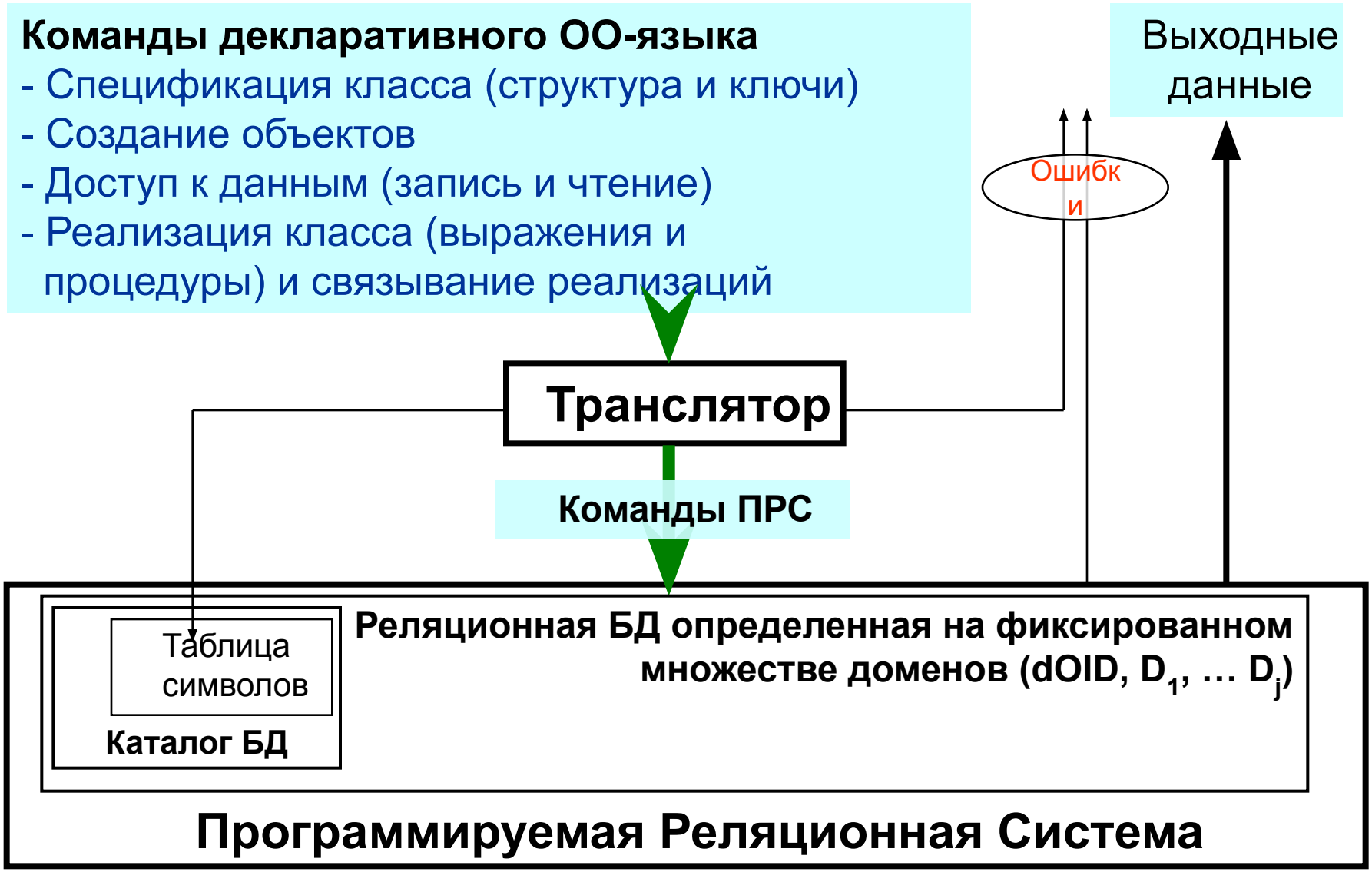
  Спецификация класса наследника объединяет спецификации родительских классов и собственных компонентов и методов. Наследуемые компоненты и методы могут менять реализацию.

Объекты и классы

- **Объектный идентификатор OID** – связанное с объектом уникальное значение из домена **dOID**.
  OID генерируется системой в процессе создания объекта, отделен от его состояния, неизменен на протяжении всего существования объекта.
- **Ссылочный тип D_n** – множество **OID** существующих в системе объектов класса D_n. (имя ссылочного типа = имя класса). Для переменных ссылочного типа определены операции присваивания, сравнивания и неявного разыменования (любая операция, отличная от операций присваивания и сравнивания, выполняется над связанным объектом).
- Создаваемые ссылочные типы D_n входят в **расширяемое множество доменов D**.

Трансляция

Цель: Система, состоящая из множества персистентных объектов разных классов **D**, определенных на расширяемом множестве доменов **(D_1,…, D_n,…)**



Структура и ключи

Структура объектов класса и ключи класса описываются в команде спецификации класса.

D_n(//имя класса

$^{sc}C_i$:D, //простой компонент определенный на скалярном домене **D**,

…

$^{r}C_j$(…a:D…)KEY(…), // сложный компонент (отношение),

…

M(…) //метод

) KEY (^{sc}C…)… //необязательные ключи класса

Структура и ключи

Значения <u>всех простых</u> компонентов $^{sc}\mathbf{C}_i$ всех объектов класса **T** представлены в виде единого <u>отношения скаляров</u> $\mathbf{R}_T$.

$$(_1\mathbf{OID} \times {}_1{}^r\mathbf{C}_1 \times {}_1{}^r\mathbf{C}_2 \times \ldots \times {}_1{}^r\mathbf{C}_n) \cup (_2\mathbf{OID} \times {}_2{}^r\mathbf{C}_1 \times \ldots \times {}_2{}^r\mathbf{C}_n) \cup (\ldots) \cup \ldots \rightarrow \mathbf{R}_T$$

где $_j\mathbf{OID}$ – идентификатор некоторого объекта,
$_j{}^{sc}\mathbf{C}_i$ – простой компонент этого объекта,
n – число простых компонентов класса

Ключи

- Поле OID является ключом.
- Если какое-либо скалярные компоненты указаны в спецификации как ключ класса, они является ключом отношении $\mathbf{R}_T$.

Каждому существующему объекту соответствует кортеж по крайне мере одного (собственного) отношения скаляров. Объектам наследуемых классов также соответствуют по одному кортежу в каждом из отношений скаляров родительских классов.

Структура и ключи

Для <u>каждого из сложных</u> компонентов $^{r}C_i$ класса **T** : значения из всех объектов могут быть объединены в виде <u>отношения сложного компонента</u> $R_{T.C}$.

$(_1OID \times {}_1^{set}C_1) \cup (_2OID \times {}_2^{set}C_1) \cup \ldots \rightarrow R_{T.C1}$
$(_1OID \times {}_1^{set}C_2) \cup \ldots \rightarrow R_{T.C2}$
…
$(_1OID \times {}_1^{set}C_m) \cup \ldots \rightarrow R_{T.Cm}$

где **m** – число сложных компонентов класса,
$_jOID$ – идентификатор некоторого объекта,
$_j^{set}C_i$ –сложный компонент соответствующего объекта.

Ключи

- Ключом отношения является совокупность поля OID и полей, указанных в спецификации как ключи соответствующего сложного компонента.
- Если какие-либо атрибуты сложного компонента указаны в спецификации как уникальные в классе, они является ключами отношения $R_{T.C}$.

Структура и ключи

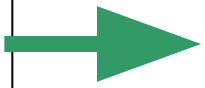
При этом все ссылочные поля (т.е. простые ссылочные компоненты и ссылочные атрибуты сложных компонентов) представлены в соответствующих отношениях как одноименные атрибуты, которые

- определенны на домене объектных идентификаторов **dOID**,
- связаны (внешний ключ) с атрибутом **OID** собственного отношения скаляров того класса, на который указывает ссылка

Структура и ключи

Класс отображается в единственное отношение R_D и множество отношений $R_{D.C}$ (вместе - отношения класса $R_{T...}$). При этом ограничения целостности, заданные для класса **D**, выражаются в ограничениях целостности применяемых к этим отношениям $R_{D...}$.

```
CLASS SHIPMENTS
{
  DocN STRING;
  Cntr CONTRACTORS;
  ...
  Items SET OF
  {
    Art STRING;
    Qty INTEGER;
  }KEY uniqArt (Art);
}KEY uniqDocN (DocN)
...;
```



EXEC begin

$R_{SHIPMENTS}$ (OID:dOID, DocN:String, Cntr:dOID)
KEY(OID), KEY(DocN),
REFERENCE Cntr ON $R_{CONTRACTOR}$.OID;

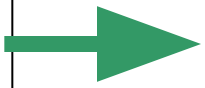
$R_{SHIPMENTS.Items}$ (OID: dOID, Art:Stirng, Qty:Integer)
KEY (OID, Art);

end

Структура и ключи

Класс отображается в единственное отношение R_D и множество отношений $R_{D.C}$ (вместе - отношения класса $R_{T...}$). При этом ограничения целостности, заданные для класса **D**, выражаются в ограничениях целостности применяемых к этим отношениям $R_{D...}$.

```
CLASS SHIPMENTS
{
  DocN STRING;
  Cntr CONTRACTORS;
  ...
  Items SET OF
  {
    Art STRING;
    Qty INTEGER;
  }KEY uniqArt (Art);
}KEY uniqDocN (DocN)
...;
```



EXEC begin

$R_{SHIPMENTS}$ (OID:dOID, DocN:String, Cntr:dOID)
KEY(OID), KEY(DocN),
REFERENCE Cntr ON $R_{CONTRACTOR}$.OID;

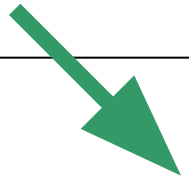
$R_{SHIPMENTS.Items}$ (OID: dOID, Art:Stirng, Qty:Integer)
KEY (OID, Art);

end

Создание объектов

Объект создается командой

new D (*конструирующее_выражение***)**



EXEC begin

- Генерируется новое уникальное значение OID

- В отношение скаляров $\mathbf{R_D}$ добавляется кортеж, содержащий этот OID и значения, определяемые *конструирующим_выражением*

end

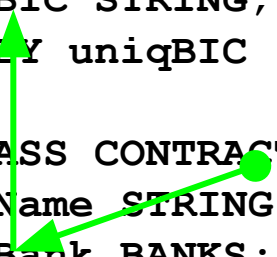
Доступ к данным

Путь - обусловленная структурой классов и ссылками между ними последовательность имен, определенных в спецификации этих классов.

```
CLASS BANKS
{ Name STRING;
  BIC STRING;
}KEY uniqBIC (BIC);

CLASS CONTRACTORS
{ Name STRING;
  Bank BANKS;
  BankAcc STRING;
  INN STRING;
}KEY uniqINN (INN);

CLASS SHIPMENTS
{ DocN STRING;
  Cntr CONTRACTORS;
  ...
  Items SET OF
  {
    Art STRING;
    Qty INTEGER;
  }KEY uniqArt (Art);
}KEY uniqDocN (DocN)...;
```



`CONTRACTORS.Bank.BIK`

Терминальный путь, оканчивается на имя базового типа

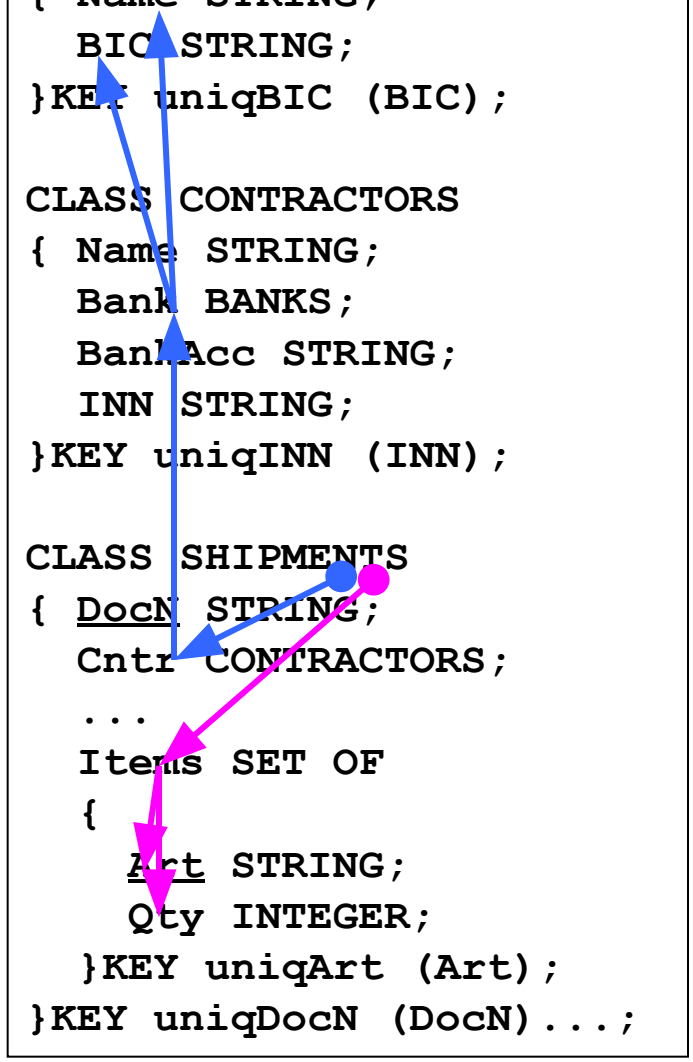
Доступ к данным

Путь - обусловленная структурой классов и ссылками между ними последовательность имен, определенных в спецификации этих классов.

```
CLASS BANKS
{ Name STRING;
  BIC STRING;
}KEY uniqBIC (BIC);

CLASS CONTRACTORS
{ Name STRING;
  Bank BANKS;
  BankAcc STRING;
  INN STRING;
}KEY uniqINN (INN);

CLASS SHIPMENTS
{ DocN STRING;
  Cntr CONTRACTORS;
  ...
  Items SET OF
  {
    Art STRING;
    Qty INTEGER;
  }KEY uniqArt (Art);
}KEY uniqDocN (DocN)...;
```



```
CONTRACTORS.Bank.BIK
```

Терминальный путь, оканчивается на имя базового типа

```
SHIPMENTS.Cntr.Bank
        [.Name, .BIK]
```

```
SHIPMENTS.Items
        [.Art, .Qty]
```

Нетерминальный путь, оканчивается на имя ссылки или сложного компонента. Допускает **путевые продолжения**

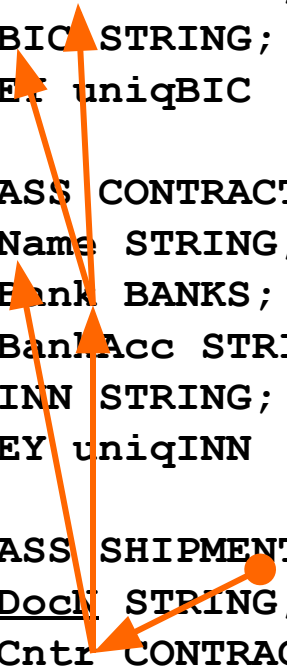
Доступ к данным

Путь - обусловленная структурой классов и ссылками между ними последовательность имен, определенных в спецификации этих классов.

```
CLASS BANKS
{ Name STRING;
  BIC STRING;
}KEY uniqBIC (BIC);

CLASS CONTRACTORS
{ Name STRING;
  Bank BANKS;
  BankAcc STRING;
  INN STRING;
}KEY uniqINN (INN);

CLASS SHIPMENTS
{ DocN STRING;
  Cntr CONTRACTORS;
  ...
  Items SET OF
  {
    Art STRING;
    Qty INTEGER;
  }KEY uniqArt (Art);
}KEY uniqDocN (DocN)...;
```



```
CONTRACTORS.Bank.BIK
```

Терминальный путь, оканчивается на имя базового типа

```
SHIPMENTS.Cntr.Bank
        [.Name, .BIK]
```

```
SHIPMENTS.Items
        [.Art, .Qty]
```

Нетерминальный путь, оканчивается на имя ссылки или сложного компонента. Допускает **путевые продолжения**

```
SHIPMENTS.Cntr
        [.Name, .Bank.Name, .Bank.BIK]
```

Доступ к данным

Путь - обусловленная структурой классов и ссылками между ними последовательность имен, определенных в спецификации этих классов.

```
CLASS BANKS
{ Name STRING;
  BIC STRING;
}KEY uniqBIC (BIC);

CLASS CONTRACTORS
{ Name STRING;
  Bank BANKS;
  BankAcc STRING;
  INN STRING;
}KEY uniqINN (INN);

CLASS SHIPMENTS
{ DocN STRING;
  Cntr CONTRACTORS;
  ...
  Items SET OF
  {
    Art STRING;
    Qty INTEGER;
  }KEY uniqArt (Art);
}KEY uniqDocN (DocN)...;
```

?????

```
CONTRACTORS.Bank.BIK
```

Терминальный путь, оканчивается на имя базового типа

```
SHIPMENTS.Cntr.Bank
        [.Name, .BIK]
```

```
SHIPMENTS.Items
        [.Art, .Qty]
```

Нетерминальный путь, оканчивается на имя ссылки или сложного компонента. Допускает **путевые продолжения**

```
SHIPMENTS.Cntr
        [.Name, .Bank.Name, .Bank.BIK]
```

Выражение отбора объектов
может дополнять любое имя класса или ссылки, содержащееся в пути

```
SHIPMENTS<.DocN LIKE "%100">.Cntr
        [.Name, .Bank.Name, .Bank.BIK]
```

Доступ к данным

Общий принцип:

Любой нетерминальный путь может трактоваться как имя отношения («О-вид»), содержащего атрибуты, имена которых представляют собой скалярные путевые продолжения этого пути

РМД не накладывает какие-либо ограничения на имена, используемые для обозначения отношения и их атрибутов, за исключением требования уникальности.

О-виды являются способом представить данные множества разных объектов в виде отношений, полностью сохранив при этом заданную в описании классов семантику сложных структур

```
SHIPMENTS.Cntr.Bank
        [.Name, .BIK]
```
Имя О-вида
Атрибуты О-вида

```
SHIPMENTS.Items
        [.Art, .Qty]
```
Имя
Атрибуты

```
SHIPMENTS.Cntr
        [.Name, .Bank.Name, .Bank.BIK]
```

Сигнатуры О-видов

```
SHIPMENTS<.DocN LIKE "%100">.Cntr
        [.Name, .Bank.Name, .Bank.BIK]
```

Доступ к данным

Использование О-видов.

О-виды – отношения, представляющие данные об объектах используя имена и последовательностей имен, заданных в описании классов этих объектов. **О-виды могут использоваться в**

- **выражениях запросов (в т.ч. ad-hoc запросах),** основанных на операциях реляционной алгебры.
  В SQL это **SELECT** выражение.

- **командах, изменяющих данные**
  В SQL это команды **INSERT, UPDATE, DELETE**

Доступ к данным

Использование О-видов.

О-виды – отношения, представляющие данные об объектах используя имена и последовательностей имен, заданных в описании классов этих объектов. **О-виды могут использоваться в**

- **выражениях запросов (в т.ч. ad-hoc запросах),** основанных на операциях реляционной алгебры.
  В SQL это **SELECT** выражение.

- **командах, изменяющих данные**
  В SQL это команды **INSERT, UPDATE, DELETE**

Доступ к данным

Вычисление О-видов.

О-виды вычисляются на основании сигнатур, с использований реляционных операций над отношениями $R_{D...}$

- Простая проекция
- Соединение в классе
- Соединение по ссылке в атрибуте О-вида
- Соединение по ссылке в заголовке О-вида
- Отбор объектов

Схема результата операций - (OID:dOID, …)

Доступ к данным

Вычисление О-видов. Простая проекция.

Сигнатура является подмножеством одного из отношений класса -> проекция

```
Shipments
        [.DocNo, .Date]
```

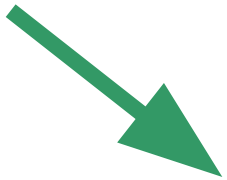
$R_{SHIPMENTS}$[OID, <u>DocN</u>, Cntr]

Доступ к данным

Вычисление О-видов. Соединение в классе.

Сигнатура содержит атрибуты разных отношений класса
-> соединение по OID + переименование атрибутов сложных компонентах

```
Shipments
  [.DocNo,  .Items.Art]
```



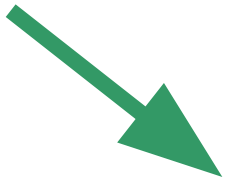
$$(R_{SHIPMENTS} \text{ LEFT JOIN}_{OID} (R_{SHIPMENTS.Items} \text{ RENAME Art AS Items.Art}))$$
$$[OID, DocN, Items.Art]$$

Доступ к данным

Вычисление О-видов. Соединение по ссылке в атрибуте.

Атрибут сигнатуры содержит ссылку
-> соединение (ссылка = OID) + переименование атрибутов «по ссылке»

```
Shipments
  [.DocNo,  .Cntr.Name]
```



$$(R_{SHIPMENTS} \text{ LEFT JOIN}_{Cntr=OID} (R_{CONTRACTOR} \text{ RENAME Name AS Cntr.Name}))\ [OID_{SHIPMENTS}, \text{DocN, Cntr.Name}]$$

Ссылочные конструкции в атрибутах могут иметь любую длину

Доступ к данным

Вычисление О-видов. Соединение по ссылке в заголовке.

Выражение групповой ссылки – выражение, возвращающее унарное отношение, содержащие **множество OID** (групповую ссылку).

…в т.ч. любой путь, оканчивающийся на имя ссылки.

```
Shipments.Cntr //
```
пример выражения групповой ссылки

```
Shipments
      [.Cntr]
```

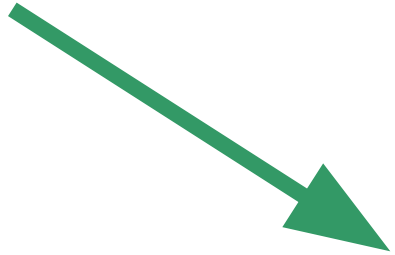
$R_{SHIPMENTS}$**[OID, Cntr]**

Доступ к данным

Вычисление О-видов. Соединение по ссылке в заголовке.

Заголовок сигнатуры содержит ссылку
-> соединение (ссылка = OID)

```
Shipments.Cntr
      [.Name, .INN]
```



$$((R_{SHIPMENTS})[OID, Cntr]\ JOIN_{Cntr=OID}\ R_{CONTRACTORS})\ [OID_{CONTRACTORS}, Name, INN]$$

Ссылочные конструкции в заголовках могут иметь любую длину

Доступ к данным

Вычисление О-видов. Отбор объектов.

Выражение отбора объектов (в путях)

name$_D$<критерий>

Результат: **групповая ссылка** на объекты,
определяемые именем ***name$_D$*** класса **D** или ссылки на него,
которые удовлетворяют WHERE-***критерию*** (покортежному)

Критерий – **logical_exp(cont$_1$, cont $_2$ …)**, где **cont** – продолжение **паме$_D$**
мн-во объектов вычисляется как
(D[cont$_1$, cont$_2$ …] WHERE logical_exp(cont$_1$, cont$_2$ …)) [OID]

```
SHIPMENTS<.Items.Art = "Tie">
                    [.DocN]
```

(((R$_{SHIPMENTS.Items}$ RENAME Art AS Items.Art) WHERE Items.Art = "Tie")[OID]
JOIN$_{OID=OID}$ R$_{SHIPMENTS}$)
[OID, DocN]

Доступ к данным

Вычисление О-видов. Отбор объектов.

Выражение отбора объектов для сложных компонентов

имя_класса_или_ссылки<критерий, критерий…>

Запятая “,” - логическая операция “**межкортежный_AND**”.
Результат: **пересечение множеств объектных идентификаторов**, сформированных по каждому из связанных запятой критериев

```
Shipments< .Items.Art =“Tie”,  .Items.Art = “Axe”>
                                        [.DocN]
```

(((R$_{SHIPMENTS.Items}$ RENAME Art AS Items.Art) WHERE Items.Art = "Tie") [OID]
INTERSEPT
((R$_{SHIPMENTS.Items}$ RENAME Art AS Items.Art) WHERE Items.Art = "Axe") [OID])
JOIN$_{OID}$ R$_{SHIPMENTS}$)
[OID, DocN]

Доступ к данным

Вычисление О-видов. Отбор объектов.

Поиск от условий к данным возможен ….

```
Shipments<.DocN LIKE “1%”>.Cntr.Bank
                  [.Name, .BIC]
```

…. "по ссылке"

```
Shipments<.Cntr.Bank.BIC LIKE “1%”>
                           [.DocN, .Date]
```

…. "против ссылки"

Выражения отбора объектов могут сочетаться и вкладываться

```
Shipments<
    .Cntr.Bank<.Name = “...” OR .BIC = “...”>
     OR .Cntr<.Name = “...”>,
    .No LIKE “1%”>
.Cntr
                              [ .Name, .INN, .Bank.Name]
```

Доступ к данным

Использование О-видов.

О-виды – отношения, представляющие данные об объектах используя имена и последовательностей имен, заданных в описании классов этих объектов. **О-виды могут использоваться в**

- **выражениях запросов (в т.ч. ad-hoc запросах),** основанных на операциях реляционной алгебры.
  В SQL это **SELECT** выражение.

- **командах, изменяющих данные**
  В SQL это команды **INSERT, UPDATE, DELETE**

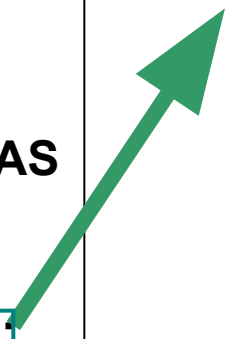
Принцип трансляции этих команд

1) **Анализ** входной команды. Поиск сигнатур используемых О-видов
2) **Построение** выражений, вычисляющего эти О-виды
3) Во входной команде: **Замена** выражений, образующие сигнатуры, на построенные вычисляющие выражения

Трансляция процедур

```
D
(
  a INTEGER;
  b INTEGER;
  ...
  mthd(...)
)

D.mthd(...) AS
begin
    ...;
    ... a + b...;
    ...;
 end
```



Идея

Для одного объекта

(R_D WHERE OID = this)[a] + (R_D WHERE OID = this)[b]

где **this** – объектный идентификатор

Трансляция процедур

```
D
(
  a INTEGER;
  b INTEGER;
  ...
  mthd(...)
)

D.mthd(...) AS
begin
    ...;
    ... a + b...;
    ...;
 end
```

Идея

Для одного объекта

(R_D WHERE OID = this)[a] + (R_D WHERE OID = this)[b]

this – объектный идентификатор

Операцию доступа – за скобки(!)

(R_D WHERE OID = this)[a+b]

Для **множества (!)** объектов

(R_D WHERE OID JOIN these)[OID, a+b]

these – групповая ссылка

Итераторы (явные и неявные) по объектам не нужны!

Трансляция процедур

Утверждение о транслируемости:

Любая процедура *p* над компонентами *C* класса D, может быть транслирована в такую процедуру *p'* над отношениями $R_{D...}$ этого класса, что результат однократного исполнения процедуры *p'* будет аналогичен исполнению исходной процедуры *p* в каждом объекте из заданного множества объектов.

EXEC THESE.p(…) → **EXEC p'(these, …)**

здесь: **THESE** – выражение гр.ссылки

здесь: **these** – групповая ссылка, обязательный атрибут **p'**

Трансляция процедур

Утверждение о транслируемости:

Любая процедура *p* над компонентами *C* класса D, может быть транслирована в такую процедуру *p'* над отношениями $R_{D...}$ этого класса, что результат однократного исполнения процедуры *p'* будет аналогичен исполнению исходной процедуры *p* в каждом объекте из заданного множества объектов.

EXEC THESE.p(…) → **EXEC p'(these, …)**

здесь: **THESE** – выражение гр.ссылки

здесь: **these** – групповая ссылка, обязательный атрибут **p'**

Параметры и локальные переменные существуют в ПРС в relVar с разным временем жизни.

proc (par_1:D1, par_2 D2…) → **R_{par}(OID, par_1:D1, par_2 D2…)**

{

localRelVal (…a:D…) → **$R_{localRelVal}$ (OID, …a:D…)**

}

Трансляция процедур

Трансляция RValue выражений

***f*(C_1, C_2, …) –> C_n**

где ***f*** - суперпозиция примитивных реляционных операций $_{prim}$***op***

$_{prim}$***op1***(C_1, $_{prim}$***op2***(C_2, … (…)))

Любая из $_{prim}$***op*** может содержать скалярные операции над атрибутами отношений C_i

Для любого C_i существует такое отношение классов R_i, что

(R_i WHERE OID = theOID)[!OID] –> C_i

где **[!OID]** операция проекцию, исключающая атрибут **OID**

Атомарные $_{prim}$***op*** над C_i **транслируются** в следующие ***op*** над R_i

Объединение $C_1 \cup C_2$	**($(R_1 \cup R_2)$ WHERE OID = theOID)[!OID]**
Вычитание $C_1 - C_2$	**($(R_1 - R_2)$ WHERE OID = theOID)[!OID]**
Декарт. произв. $C_1 \times C_2$	**($(R_1$ JOIN$_{OID}$ R_2 WHERE OID = theOID)[!OID]**
Выборка **C WHERE cond**	**((R WHERE cond) WHERE OID = theOID)[!OID]**
Проекция **C[a_1, a_2, ...]**	**((R[OID, a_1, a_2, ...]) WHERE OID = theOID) [!OID]**

общая схема трансляции атомарных операций

$_{prim}$***op***(C_1 …) –>C_{res} ⇒ (***op'*** (R_1 …) **WHERE OID = theOID) [!OID]** –>C_{res}

Трансляция процедур

Трансляция RValue выражений

Для любого C_{res} существует такое отношение результатов R_{res}, что
(R_{res} WHERE OID = theOID)[!OID] –> C_i
где **[!OID]** операция проекцию, исключающая атрибут **OID**

Для любого C_i существует такое отношение классовR_i, что
(R_i WHERE OID = theOID)[!OID] –> C_i
где **[!OID]** операция проекцию, исключающая атрибут **OID**

таким образом,

$_{prim}op(C_1 \ldots) \rightarrow C_{res}$ **$op'(R_1 \ldots) \rightarrow R_{res}$, R_{res} – отношение результатов**

В силу замкнутости рел.алгебры, для исходного

$C_n := f(C_1, C_2, \ldots)$, где ***f*** есть суперпозиция $_{prim}op_1(C_1, _{prim}op2(C_2, \ldots (\ldots)))$
существует

$R_n := f'(R_1, R_2, \ldots)$, где ***f*** есть суперпозиция $op'_1(R_1, op'_2(R_2, \ldots (\ldots)))$

$f'(R_1, R_2, \ldots)$ JOIN these -> theseR_n,

R_n – объединяет результаты для множества **these** (гр. ссылка)

f' есть трансляция ***f*** ; скалярные операции в трансляции не меняются

общая схема трансляции атомарных операций

$_{prim}op(C_1 \ldots) \rightarrow C_{res}$ **($op'(R_1 \ldots)$ WHERE OID = theOID) [!OID] –>C_{res}**

Трансляция процедур

Трансляция последовательности операторов

процедура **proc** – алгоритмическая последовательность операторов

- присваивания $C_j := f(\dots C_i \dots)$, R^{temp} - параметры, лок.переменные
- вызова **call p(…)**

присваивания

$C_i := f(C_j, \dots, C_k^{temp}, \dots)$ → **update R_n with (f'(R_1, R_2,…) JOIN these)**

вызовы

EXEC p (…) → **EXEC p'(these …)**

линейная последовательность

$C_i := f_1(C_1 \dots)$
EXEC $_{some}$**p(C …)**
$C_{i+1} := f_2(C_2 \dots)$
…

→

update R_i with $f'_1(R_1 \dots)$ JOIN these
EXEC $_{some}$**proc'(R …)**
update R_{i+1} wirh $f'_2(R_2 \dots)$ JOIN these
…

Трансляция процедур

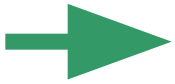
Трансляция алгоритмов

Операторы **if…** и **while…**

групповая ссылка **these** может быть разделена по заданному условию

Оператор **if…**

$C_n := f_1(C \ldots)$
…
if(condition)
then $C_{n+1} := f_2(C \ldots)$

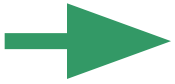


update R_n with (f'_1(R …) JOIN these)
…
$these_{TRUE}$:= these <condition>
update R_{n+1} with (f'_2(R …) JOIN $these_{TRUE}$)

Оператор **while…** (выходим из цикла, если нет объектов, удовлетворяющий условию.)

$C_n := f_1(C \ldots)$
…
while(condition)
begin
…
$C_{n+1} := f_2(C \ldots)$
…

end



update R_n with (f'_1(R …) JOIN these)
…
$these_{TRUE}$:= these <condition>
while(count($these_{TRUE}$)>0) begin
…
update R_{n+1} with (f'_2(R …) JOIN $these_{TRUE}$)
…
$these_{TRUE}$:= these … WHERE condition
end

Трансляция процедур

Утверждение о транслируемости:

Любая процедура *p* над компонентами *C* класса D, может быть транслирована в такую процедуру *p'* над отношениями $R_{D...}$ этого класса, что результат однократного исполнения процедуры *p'* будет аналогичен исполнению исходной процедуры *p* в каждом объекте из заданного множества объектов.

EXEC THESE.p(…)

здесь: **THESE** – выражение гр.ссылки

EXEC p'(these, …)

здесь: **these** – групповая ссылка, обязательный атрибут **p'**

Хранимые компоненты

служат для персистентного хранения данных . Для их реализации создаются реальные переменные отношения ПРС, схема которых в точности соответствует ранее описанным отношениям $\mathbf{R_{D...}}$.

класс
D(…C(a:D…), …)
реализация
D.C AS STORED;

$\mathbf{{}_{real}R_{D.C}}$ **(OID: dOID, a:D, …)**

Связывание

Структура класса-наследника объединяет множества наследуемых и собственных компонентов и методов. Реализации наследуемых компонентов и методов могут быть переопределены.

Связывание полиморфных компонентов

класс
D(…C, …)

реализации
D.C AS $f_1(…)$;

класс- наследник
$_{sub}$D EXTEND D (…)

переопределение реализаций
$_{sub}$D.C AS STORED;

…

Отношение класса **$R_{D...}$** объединяет (связывает) результаты трансляций реализаций.

- для сложного компонента

$R_{D.C}$ AS
$f_1'(…)$
UNION
$_{real}R_{D.C}$

- для простого компонента

R_D AS
$_{real}R_D$ LEFT JOIN$_{OID}$
($f_1'(…)$ [OID, $_{calc}$C])
[OID, …, REPLACE(C, $_{calc}$C), …],
где **REPLACE(C, $_{calc}$C)** заменяет хранимое значение простого компонента **C** на вычисленное значение **$_{calc}$C** (если такое есть).

Связывание

Структура класса-наследника объединяет множества наследуемых и собственных компонентов и методов. Реализации наследуемых компонентов и методов могут быть переопределены.

Связывание полиморфных методов

класс
D(…C, M(…), …)

реализации
D.C AS $f_1(...)$;
D.M(…) AS p_1;

класс- наследник
$_{sub}$D EXTEND D (…)

переопределение реализаций
$_{sub}$D.C AS STORED;
$_{sub}$D.M(…) AS p_2;

…

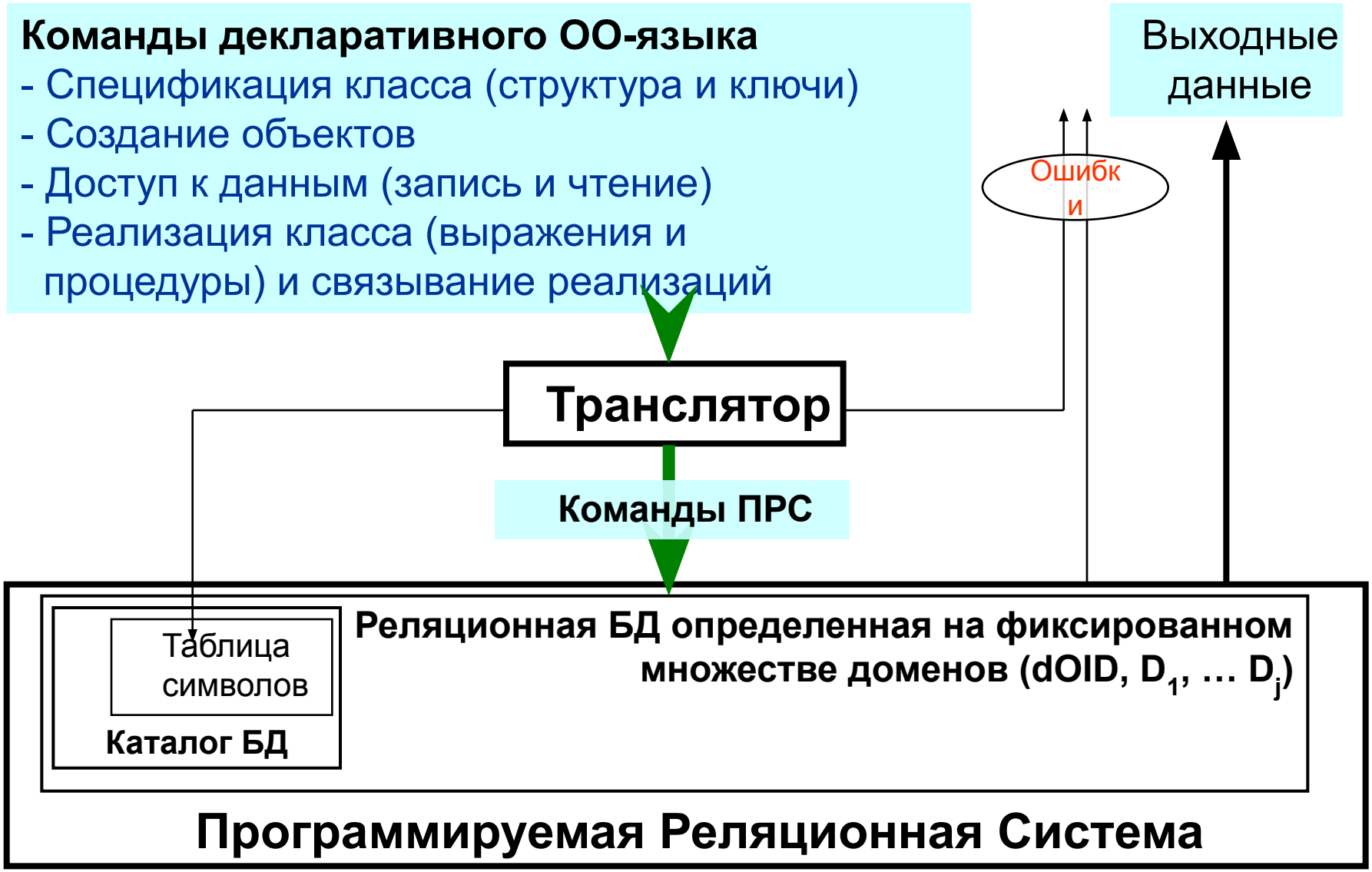
Для метода **M**, определнного в классе **D**, создается процедура **D.M'**, связывающая все существующие реализации этого метода

D.M'(these, …) AS
begin
p'$_1$(these INTERSEPT scope(p$_1$), …)
p'$_2$(these INTERSEPT scope(p$_2$), …)
end

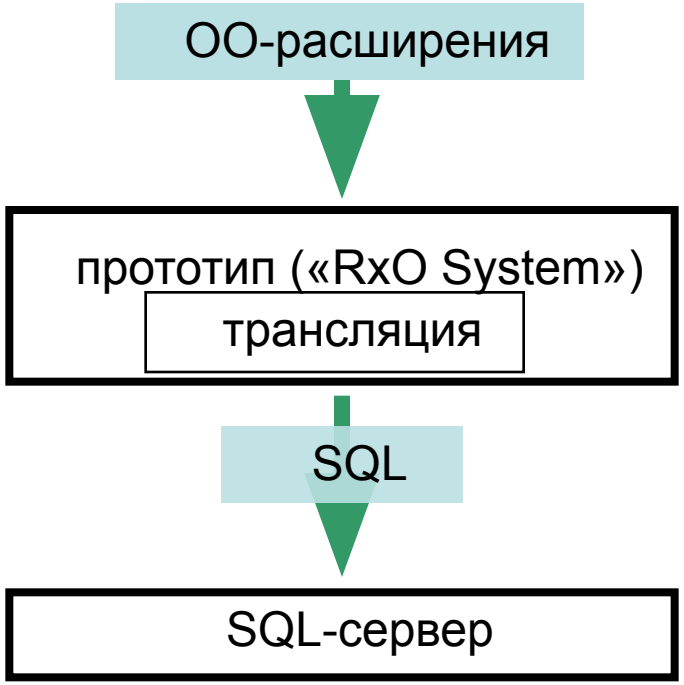
где **scope(p)** определяет множество объектов, с которым связана реализация **p**.

Трансляция

Цель: Система, состоящая из множества персистентных объектов разных классов **D**, определенных на расширяемом множестве доменов **(D_1,…, D_n,…)**



Прототип.



http://theorm.narod.ru/OOtransRUS.pdf

Прототип.

Создание класса. Спецификация класса.

```
CREATE CLASS BANKS
{
  Name STRING;
  BIC STRING;
}KEY uniqBIC (BIC);   //ключ класса - БИК уникален для банка

CREATE CLASS CONTRACTORS  //КОНТРАГЕНТЫ
{
  Name STRING;
  Bank BANKS;     //ссылка на BANKS
  BankAcc STRING;
  INN STRING;
}KEY uniqINN (INN);

CREATE CLASS GOODS    //ТОВАРЫ
{
  Art STRING;
  PricePL FLOAT; //цена
  Turnover SET OF//Оборот (компонент-набор)
  {
    Txt STRING;  //комментарии
    Date DATETIME;    //дата
    DocN STRING;      //номер
    Cntr CONTRACTORS; //ссылка на CONTRACTORS
    Qty INTEGER; //кол-во
  }KEY Key2Turn (DocN, Cntr);  //ключ компонента-набора
  QtyFree INTEGER;    //остаток на складе
}KEY UniqArt (Art);
```

Прототип.

Создание класса. Спецификация класса.

Создание класса

- Спецификация класса (команда **CREATE CLASS** …)
  существование класса и его интерфейс.
- Для каждого компонента и метода задается реализация
  (команда **ALTER CLASS… REALIZE** …)

```
CREATE CLASS SHIPMENTS    //товарные операции
{
  DocN STRING;    //номер
  Cntr CONTRACTORS;   //ссылка на контрагента
  DocDate DATETIME;   //дата заказа
  Txt STRING;         //комментарии
  PostIt(inDate DATETIME);     //метод "учесть"
  PostDate DATETIME;  //да
  Items SET OF    //компон
  {
    Art STRING;   //артику
    Qty INTEGER; //штук
  }KEY uniqArt (Art); //кл
}KEY uniqDocN (DocN)
REFERENCE ToUniqArt   //вн
Items.(Art) ON GOODS.Uniq
```

RxO Query

```
SELECT
  #s.DocN,
  #s.PostDate,
  #s.Txt,
  #s.Cntr.Name,
  #s.Items.Art,
  #s.Items.Qty,
  #g.PricePL
FROM SHIPMENTS #s JOIN GOODS #g ON #s.Items.Art = #g.Art
```

DocN	PostDate	Txt	Cntr.Name	Items.Art

Прототип.

Создание класса. Реализация класса.

Компоненты могут быть реализованы
1) Как хранимые … **AS STORED**;

```
ALTER CLASS BANKS
REALIZE(
  Name STRING,
  BIC STRING
)AS STORED;
```

2) Как вычисляемые с помощью процедуры, возвращающей значение

…**AS**
{
тело_процедуры
};

Методы класса реализуются только как процедуры и не возвращают значения

```
ALTER CLASS GOODS
REALIZE
QtyFree INTEGER AS
{
  DECLARE
  {
    tmpQty INTEGER;
    tmpQtyRes INTEGER;
  }
  tmpQty:=
    SELECT SUM(#g.Items.Qty)
    FROM SHIPMENTS #g
    WHERE #g.PostDate IS NOT NULL
      AND #g.Items.Art = Art;
  IF(tmpQty IS NULL) THEN tmpQty:= 0;
  tmpQtyRes:=
     SELECT SUM(#g.Items.Qty)
     FROM SHIPMENTS #g
     WHERE #g.Items.Art = Art
       AND #g.PostDate IS NULL
       AND #g.Items.Qty < 0;
  IF(tmpQtyRes IS NULL) THEN tmpQtyRes:= 0;
  RETURN tmpQty + tmpQtyRes;
};
```

Прототип.

Создание и доступ к объектам.

Объекты создаются командой NEW

NEW имя_класса [WITH…]

refVar := NEW имя_класса [WITH…]

```
NEW CONTRACTORS
WITH SET
   .Name := "X3",
   .Bank :=
   (NEW BANKS WITH SET
     .BIC := "30602"),
   .BankAcc := "40602",
   .INN := "772";
```

Объекты никогда не "теряются". Относясь к классу, они доступны при групповых операциях с классом.

Объекты уничтожаются командой DESTROY

DESTROY *групповая_ссылка*

```
DESTROY BANKS[.BIC = "I.E."];
```

Прототип.
Изменение данных.

Для изменения данных используются традиционные SQL операции, использующие в качестве аргуменов пути и путевые продолжения

INSERT INTO путь ({ппрод := …}(,n))

UPDATE путь SET {ппрод :=…}(,n)

DELETE FROM путь

```
INSERT INTO SHIPMENTS[.DocN = "F.e."].Items
       (.Art, .Qty)
VALUES("Tie", 10);
```

Методы класса выполняются командой
EXEC путь.имя_метода ([параметры])

```
EXEC SHIPMENTS[.DocDate<'2011.01.01'].PostIt('2011.04.20');
```

В локальных контекстах допускается операция присваивания
$путь_{лок}$:= …;

Прототип.
Запросы.

Для запросов к данным используются традиционное SELECT выражение, использующие для обозначения источников данных пути и путевые продолжения

SELECT ппрод FROM путь …;

```
SELECT
  #gt.DocN,
  #gt.Cntr.Name,
  #gt.Cntr.Bank.Name
FROM GOODS[.Art LIKE "T%"].Turnover #gt
```

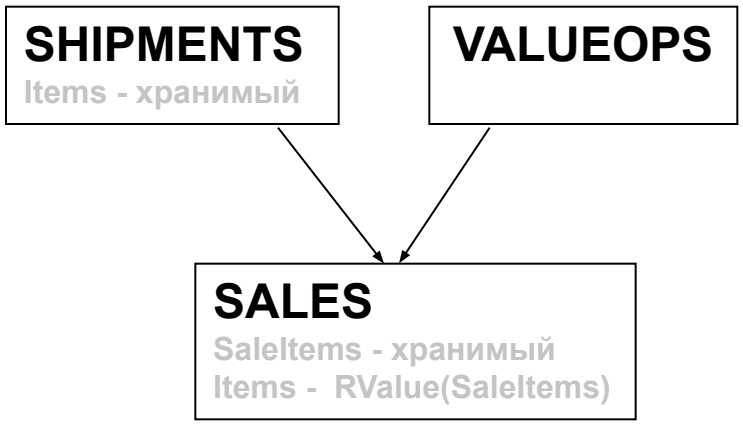
	DocN	Cntr.Name	Cntr.Bank.Name
▶	In01	Levis	CiBank
	In02	Levis	CiBank

Прототип.
Наследование.

Допускается множественное наследование. Структура класса-наследника объединяет множества наследуемых и собственных компонентов и методов. Реализации наследуемых компонентов и методов могут быть переопределены.

```
CREATE CLASS VALUEOPS //фин.документ
{
  VDocN STRING;   //номер
  VDate DATETIME;//дата
  ExpVal FLOAT;   //ожидаемая сумма
  Value FLOAT;    //сумма
};

CREATE CLASS SALES
EXTENDED SHIPMENTS, VALUEOPS
{
  SaleItems SET OF   //компонент-набор: данные о проданных
товарах
  {
    Art STRING;       //артикул
    Price FLOAT;      //цена
    Qty INTEGER;  //штук
  }KEY uniqArtPrice (Art,Price);
}
REFERENCE ToUniqArt SaleItems.(Art) ON GOODS.UniqArt;
```



Прототип.
Полиморфизм

```
ALTER CLASS SHIPMENTS
REALIZE PostIt(inDate DATETIME)
AS
{
IF(PostDate IS NULL) THEN
  {
  PostDate := inDate;
  Txt := "DONE " + DocN;
 IF( DocDate < '2010.01.01') THEN
    Txt := "OLD! " + Txt;
  }
};
```

1

```
EXEC SHIPMENTS[.DocDate<'2011.01.01'].PostIt('2011.
SELECT
  #s.DocN,
  #s.PostDate,
  #s.Txt,
  #s.Cntr.Name,
  #s.Items.Art,
  #s.Items.Qty,
  #g.PricePL
FROM SHIPMENTS #s JOIN GOODS #g ON #s.Items.Art = #g.Art;
```

Для существующих объектов класса SHIPMENT

1)Выполняем метод

2) Вычисляем запрос

```
ALTER CLASS SHIPMENTS
REALIZE
  Items SET OF
  {
    Art STRING;
    Qty INTEGER;
  }KEY uniqArt (Art)
AS STORED;
```

3

DocN	PostDate	Txt	Cntr.Name	Items.Art	Items.Qty	PricePL
In01	20.04.2011 12:00	DONE In01	Levis	Axe	10	
In01	20.04.2011 12:00	DONE In01	Levis	Tie	10	1
In02	20.04.2011 12:00	DONE In02	Levis	Tie	30	1
Sale#1	20.04.2011 12:00	Sale is POSTED!...	X3	Axe	-5	
Sale#1	20.04.2011 12:00	Sale is POSTED!...	X3	Tie	-5	1
Sale#2			X3	Tie	-30	1

1 2 3 4

```
ALTER CLASS SALES
REALIZE PostIt(inDate DATETIME)
AS
{
IF(PostDate IS NULL) THEN
  {
  Txt := "Sale is POSTED! " + Txt;
  PostDate := inDate;
  Value :=
    SELECT SUM (-#pi.Qty*#pi.Price)
    FROM SaleItems #pi;
  }
};
```

2

```
ALTER CLASS SALES REALIZE
  Items SET OF
  {
    Art STRING;
    Qty INTEGER;
  }KEY uniqArt (Art)
AS
{
  RETURN
  SELECT #pi.Art,  SUM(-#pi.Qty)
  FROM SaleItems #pi
  GROUP BY #pi.Art;
};
```

4

Прототип.
Полиморфизм

```
ALTER CLASS SHIPMENTS
REALIZE PostIt(inDate DATETIME)
AS
{
IF(PostDate IS NULL) THEN
  {
  PostDate := inDate;
  Txt := "DONE " + DocN;
 IF( DocDate < '2010.01.01') THEN
    Txt := "OLD! " + Txt;
  }
};
```

1

```
EXEC SHIPMENTS[.DocDate<'2011.01.01'].PostIt('2011.
SELECT
  #s.DocN,
  #s.PostDate,
  #s.Txt,
  #s.Cntr.Name,
  #s.Items.Art,
  #s.Items.Qty,
  #g.PricePL
FROM SHIPMENTS #s JOIN GOODS #g ON #s.Items.Art = #g.Art;
```

Для существующих объектов класса SHIPMENT

1)Выполняем метод

2) Вычисляем запрос

```
ALTER CLASS SHIPMENTS
REALIZE
  Items SET OF
  {
    Art STRING;
    Qty INTEGER;
  }KEY uniqArt (Art)
AS STORED;
```

3

DocN	PostDate	Txt	Cntr.Name	Items.Art	Items.Qty	PricePL
In01	20.04.2011 12:00	DONE In01	Levis	Axe	10	
In01	20.04.2011 12:00	DONE In01	Levis	Tie	10	1
In02	20.04.2011 12:00	DONE In02	Levis	Tie	30	1
Sale#1	20.04.2011 12:00	Sale is POSTED!...	X3	Axe	-5	
Sale#1	20.04.2011 12:00	Sale is POSTED!...	X3	Tie	-5	1
Sale#2			X3	Tie	-30	1

1 2 3 4

```
ALTER CLASS SALES
REALIZE PostIt(inDate DATETIME)
AS
{
IF(PostDate IS NULL) THEN
  {
  Txt := "Sale is POSTED! " + Txt;
  PostDate := inDate;
  Value :=
    SELECT SUM (-#pi.Qty*#pi.Price)
    FROM SaleItems #pi;
  }
};
```

2

```
ALTER CLASS SALES REALIZE
  Items SET OF
  {
    Art STRING;
    Qty INTEGER;
  }KEY uniqArt (Art)
AS
{
  RETURN
  SELECT #pi.Art,  SUM(-#pi.Qty)
  FROM SaleItems #pi
  GROUP BY #pi.Art;
};
```

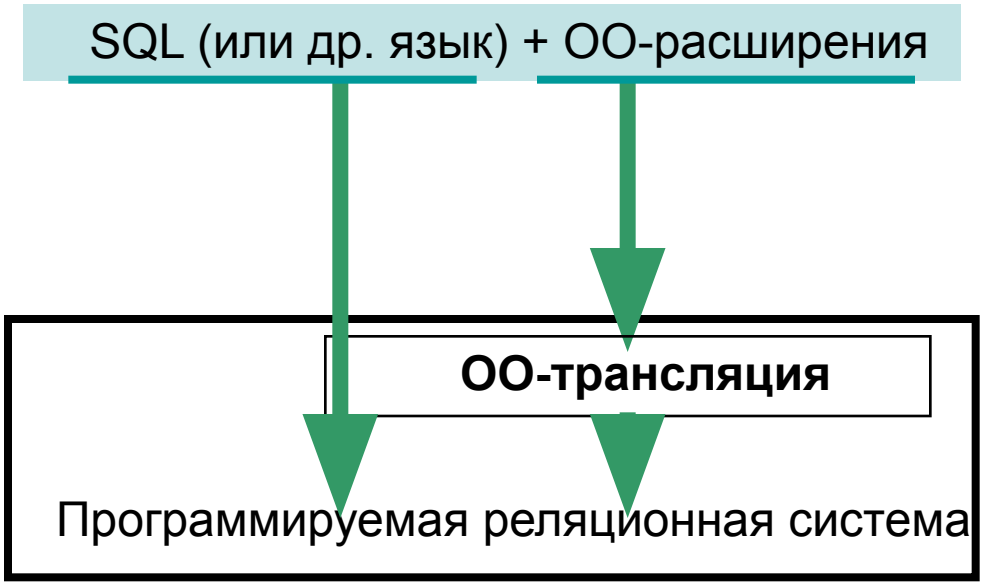
4

Использование

Предложенный подход не противоречит и не препятствует использованию «обычных» реляционных структур. При этом он позволяет:

1) использовать расширяемое множество доменов.
2) задавать внешние ключи между классами и отношениями.
3) комбинировать в запросах данные из отношения и классов.

Развитие существующих реляционных DBMS

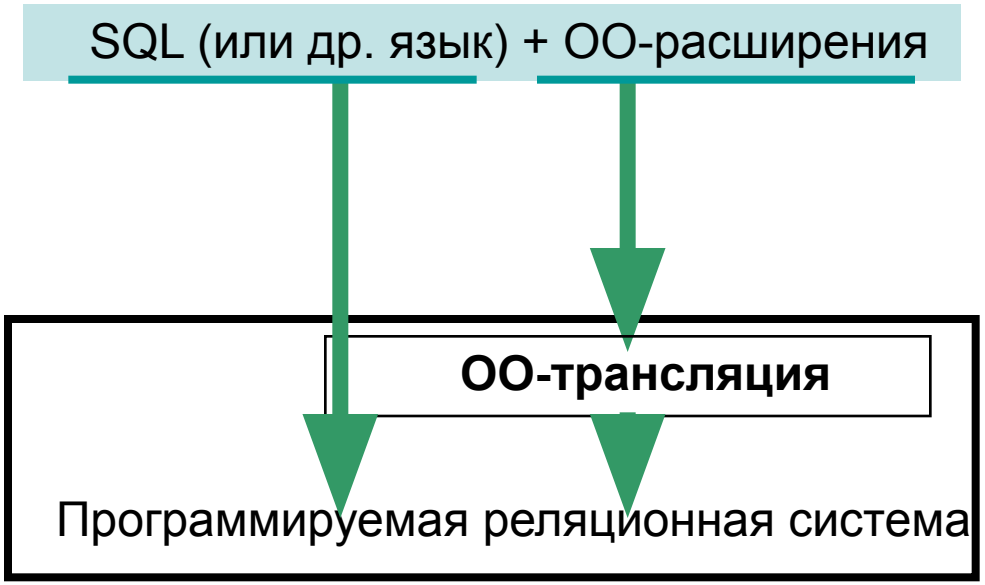


Использование

Предложенный подход не противоречит и не препятствует использованию «обычных» реляционных структур. При этом он позволяет:

1) использовать расширяемое множество доменов.
2) задавать внешние ключи между классами и отношениями.
3) комбинировать в запросах данные из отношения и классов.

Развитие существующих реляционных DBMS



Использование.

Эволюционное развитие существующих реляционных СУБД.

Добавляются функции

- **инструмента создания объектных моделей предметной области.**
- **среды персистентного существования объектов.**
- **инструмента, позволяющего получать данные о состоянии существующих объектов.**

результат: **Сервер модели предметной области**

"Д.002 **Данные**
Представление ***фактов*** *о предметной области* системы баз данных или информационной системы в форме, допускающей их хранение и обработку на компьютере, передачу по каналам связи, а также восприятие человеком."

М.Р.Когаловский
Энциклопедия технологий баз данных.

Использованная литература:

Григорьев Е.А. Объектно-ориентированная трансляция для программируемых реляционных систем. http://theorm.narod.ru/OOtransRUS.pdf (текущая версия)

Григорьев Е.А. RxO - прототип. Объектно-ориентированные SQL-подобные языковые расширения. http://theorm.narod.ru/RxOprototypeRUS.pdf (текущая версия)

Пратт Т., Зелковиц М. Языки программирования: разработка и реализация. 4-е изд. – СПб: Питер, 2002.

Codd, E.F. A Relational Model of Data for Large Shared Data Banks. CACM 13(6), June 1970. Republished in Milestones of Research -- Selected Papers 1958-1982 (CACM 25th Anniversary Issue), CACM 26(1), January 1983.

Мейер Д. Теория реляционных баз данных. – М.: Мир, 1984.

Hugh Darwen and C.J. Date. The Third Manifesto
. http://www.dcs.warwick.ac.uk/~hugh/TTM/TTM-2011-10-30.pdf (текущая версия).

А. Эйзенберг, Дж. Мелтон. SQL:1999, ранее известный как SQL3 (перевод Кузнецова С.Д.). http://citforum.ru/database/digest/sql1999.shtml

Объектно-ориентированный транслятор
для
программируемой реляционной системы.

СПАСИБО!