НадРеляционный Манифест.v1.09

Евгений Григорьев © (egrigor@mail.ru)

НадРеляционный Манифест (далее НРМ) предлагает возможный подход к созданию систем хранения данных следующего поколения. Потребность в таких системах обуславливается тем, что возможности существующих СУБД для разработки сложных информационных систем не являются удовлетворительными. Во многом речь идет о возможностях для адекватного описания сложной предметной области.

Важнейшей частью системы хранения данных, созданной на основе предлагаемого подхода, является объектно-ориентированный транслятор, выполняющийся в среде реляционной СУБД. Такая система сравнима в своих выразительных возможностях с объектно-ориентированными языками программирования. С другой стороны, предлагаемый подход признает фундаментальность реляционной модели данных[2,3] - в том числе и возможность использования этой модели в качестве формального фундамента системы хранения данных следующего поколения.

Введение

Тема объединения свойств, присущих объектно-ориентированным и реляционным системам, в рамках единой системы в течение длительного времени будоражит умы специалистов в области баз данных. Около десяти лет назад за сравнительно короткий промежуток времени появилось три работы, в которых разные группы авторов декларировали набор необходимых свойств, которыми должна обладать подобная система. Требования и идеи, изложенные в этих работах, к настоящему времени имеют практическую реализацию в виде стандартов и систем хранения данных [1].

Манифест систем объектно-ориентированных баз данных"[M1] (в хронологическом порядке появившейся первым) создавался сторонниками объектно-ориентированных баз данных. Делая свой выбор, они фактически отвергают реляционную модель как пережиток прошлого. По их мнению, объектная база данных есть ни что иное, как объектно-ориентированная среда программирования, реализующая свойство долговременного хранения объектов и дополненная средствами, позволяющими выполнять поиск необходимой информации, т.е. системой построения и выполнения запросов.

Манифест систем баз данных третьего поколения"[M2] (второй), наоборот, предлагает эволюционный подход, когда системы хранения данных следующего поколения должны "вырасти" из уже существующих систем хранения данных, унаследовав все их преимущества. Авторы перечисляют свойства, которыми, по их мнению, должны обладать базы данных нового поколения, фактически объединяя сильные стороны объектных и реляционных систем, и утверждают, что полезные свойства должны найти свое воплощение в языке программирования БД, основой для которого должен служить SQL.

Авторы "Третьего Манифеста"[M3] не соглашаются с первым манифестом и считают, что основой систем управления базами данных третьего поколения должна являться математически строгая реляционная модель, предполагая, что она должна быть дополнена расширяемой системой базовых (скалярных типов). Если в современных (на момент написания работы) реляционных системах существует весьма ограниченная

система базовых типов, то в СУБД нового поколения она должна быть расширяема. Таким образом, атрибут кортежа отношения может содержать не только простое (например, число или строка фиксированной длины), но любое произвольно-сложное значение. Авторы "Третьего Манифеста" не соглашается и со вторым манифестом, поскольку крайне негативно относятся к языку SQL, обоснованно утверждая, что этот язык извращает реляционную модель. Серьезной критике подвергаются и другие звучащие во втором манифесте идеи, в частности, идея о соответствии типа отношению. "НадРеляционный Манифест" считает эту критику обоснованной и справедливой.

В отличие от предыдущих, "Третий Манифест" является формальным и логичным, и в этом, без сомнения, заключается его сила. Однако НРМ не может безоговорочно принять предложения "Третьего Манифеста", поскольку считает, что лежащие в их основе исходные посылки, являются, по крайней мере, неполными. Напомним, что, отвечая на вопрос "какая концепция в реляционном мире является двойником концепции класса в мире объектном?", "Третий Манифест" рассматривает два возможных варианта

1) объектный класс = домен,
2) объектный класс = отношение.

Третий манифест убедительно показывает, что второй вариант является ошибочным (НРМ полностью согласен с этим), и, далее, исходит в своих рассуждениях именно из первого варианта.

Отметим, что НРМ не утверждает, что предложения, высказанные в "Третьем Манифесте", являются ошибочными. Однако НРМ не сомневается в том, что ответ (даже правильный!) на процитированный в предыдущем абзаце вопрос, не является полным ответом на вопрос, как можно соотнести "мир объектный" и "мир реляционный". Существует еще один подход, который не может быть описан ни одним из предложенных в "Третьем Манифесте" вариантов ответа, и, тем не менее, позволяет объединить свойства объектных и реляционных систем в рамках единой системы. Этот подход рассматривается далее.

Основное требование НРМ.

Известно, что данные, существующие в реляционной БД, представлены как набор значений разных отношений[3]. Предполагается, что речь идет о данных, описывающих некую предметную область, которая представляет собой множество сущностей. Нельзя не отметить, что между моделируемыми сущностями и описывающим их множеством отношений существует сложная связь: данные о любой сущности могут входить во многие отношения и любое отношение может содержать данные о многих разных сущностях. Возможные варианты (например, когда сущности соответствует только один кортеж одного из отношения) можно рассматривать лишь как частный случай описанной ситуации.

Единственное правило выполняется в любом случае. Заметим, что вся предметная область может рассматриваться как сложная сущность, состоящая из многих сущностей. Вся эта предметная область описывается в реляционной БД как набор отношений. Можно представить ситуацию, когда в какой-то момент времени база данных хранит информацию о состоянии только одной из этих более простых сущностей, однако и в этом случае значение, хранящиеся в реляционной БД, будет представлять собой набор отношений. Речь идет о любой предметной области и о любой ее сущности – в любом случае данные, описывающие их состояние, должны быть представлены в реляционной

БД как множество значений отношений, поскольку это является основным требованием[3] реляционных БД.

Таким образом, значение, описывающее состояние любой сущности предметной области, представляет собой множество отношений (говоря более определённо – значений отношений), являющееся подмножеством реляционной БД, описывающей состояние предметной области в целом.

НРМ утверждает, что система, позволяющая явно определять такие подмножества и манипулировать ими, и будет искомой системой, обладающей свойствами как объектно-ориентированных, так и реляционных систем. В соответствии с этим, **основное требование** НРМ звучит следующим образом:

Значение, описывающее состояние сущности предметной области, должно представлять собой множество значений отношений

Будем называть систему, выполняющую основное требование, R*O-системой.

Замечание. Тем самым, соотнося "мир объектный" и "мир реляционный", НРМ ставит в соответствие объекту множество значений отношений. Отметим, что существующее в РМД понятие "база данных" определятся тоже как множество значений отношений. По сути, НРМ рассматривает "базу данных" как набор подмножеств (возможно пересекающихся и, даже, вложенных), каждое из которых по определению так же может быть названо "базой данных".

Часть 1. R*O система снаружи. Свойства и возможный язык управления.

Описывается система типов, ограничения целостности данных и применимые к типам операций. Показано, что определение сложных структур может интерпретироваться как определения множества реляционных переменных (R-переменные). Вводится правило, определяющее существование и именование таких переменных и их атрибутов. Рассматриваются основные команды управления системой.

Типы R*O системы.

Данные в R*O системе представлены в виде набора значений, принадлежащих одному из предопределенных или конструируемых типов. Типы делятся на значимые и объектные. Значимые типы описывают значения, объектные – объекты.

Единственный способ, позволяющий различить значения, заключается в их прямом и полном сравнении. Другими словами, значения идентифицирует себя само. В этом заключается принципиальное отличие значений от объектов, которые снабжаются уникальными объектными идентификаторами, используемыми для того, что бы различать объекты, а также для организации доступа к этим объектам.

К значимым типам относятся

1) скалярные – включая базовые (числовые, символьные, булевские и т.п. типы) и ссылочные типы (о них - далее). Значение скалярного типа будем называть далее скалярами.
2) конструируемые кортежные типы. Значения этого типа (далее - кортежи) представляют собой множество пар "имя атрибута, значение атрибута скалярного типа". Соответственно, кортежный тип определяется как множество пар "имя атрибута, скалярный тип атрибута".

3) конструируемые типы-множества. Значения этого типа (далее - множества) представляют собой множества скалярных или кортежных значений. Соответственно этому определяется и переменная типа-множества (*имя_переменной* AS SET OF *имя_скалярного_или_кортежного_типа*).

Замечание. Мы опускает возможность существования других способов определения скалярных типов, однако допускаем, что такие способы могут существовать. Например, тип можно определить путем явного перечисления его значений. Также мы опускаем возможность существования типов-коллекций, отличных от типов множеств, однако допускаем существование таких типов при условии, что значения этих типов могут быть однозначно преобразованы в значения типов множеств и, из них, обратно в те же самые значения.

Пример. *В качестве сквозного примера рассмотрим простую схему данных, позволяющих описать движение товаров и их хранение на множестве складов.* `INTEGER`, `FLOAT`, `DATE`, `STRING` *являются базовыми скалярными типами. Оговоримся, что этот пример служит исключительно для демонстрации некоторых особенностей R*O систем и, поэтому, не претендует на полноту и точность.*

Описываем тип `ArtQty` *как кортежный тип. Атрибут* `Art` *имеет скалярный ссылочный тип* `Article` *(разъяснения см. далее).*

```
DESCRIBE TUPLE ArtQty
{
  Art Article;
  Quantity INTEGER;
}
```

Элемент системы, служащий для хранения скаляров, будем называть *полем*. Соответственно, элемент, служащий для хранения кортежей, представляет собой неупорядоченный набор полей, а элемент, служащий для хранения значения типа-множества, представляет собой множество таких наборов.

Объектные типы описывают объекты. Объект имеет уникальный идентификатор (OID), который выражает присущее ему свойство уникальности и идентифицируемости, а также используется для организации доступа к этому объекту. Уникальный идентификатор объекта отделен от значений его компонентов.

Пример. *Объектный тип* `Brand` *описывает уникальные торговые марки товаров.*

```
CREATE CLASS Brand
{
  Name STRING
    CONSTRAIN GLOBALKEY Name;
}
```

(Ограничение целостности `GLOBALKEY` *будет рассматриваться далее)*

Объектный тип `Article` *описывает товары. Объекты этого типа имеют уникальное поле* `No`*. Каждый артикул принадлежит одному из брендов.*

```
CREATE CLASS Article
{
  No STRING
    CONSTRAIN GLOBALKEY No;
  BrandName STRING
    CONSTRAIN FOREIGNKEY BrandName ON Brand.Name;
}
```

(Ограничение целостности `FOREIGNKEY` *будет рассматриваться далее)*

Объектные типы являются конструируемыми. Определение объектного типа стоит из спецификации и реализации[6]. *Спецификацией* называется декларативный перечень внешних свойств (атрибутов и методов), который можно рассматривать как интерфейс, по которому можно организовывать взаимодействия с объектом. *Реализацией* называется скрытая от внешнего доступа совокупность структур данных и программного кода, воплощающих заданную для этого типа спецификацию на основании существующего в данной системе набора типов и операций.

Мы рассматриваем и атрибуты, и методы объектных типов как компоненты, содержащие или возвращающие значения - т.е. имеющие значимый тип. Спецификация метода может рассматриваться как спецификация компонента, имя которого совпадает с именем метода, а тип - с типом значения, возвращаемого методом (спецификация метода включает также описание параметров значимых типов).

Таким образом, спецификация объектного типа представляет собой набор спецификаций значимых компонентов. Совокупность значений компонентов объекта определяет состояние объекта. Любой компонент может быть реализован либо как хранящий значение, либо как вычисляющий его, однако важно понимать, что в спецификации компонента **не** определяется, является ли это значение хранимым или вычисляемым.

Замечание. Конечно, спецификация в определённом смысле определяет реализацию. Например, наличие параметров у метода позволяет предположить, что значение, возвращаемое методом, должно быть вычисляемым. Однако эта зависимость все же не очевидна. Вполне возможна такая реализация метода, что возвращаемое им значение не требует вычислений (например, при наследовании одна из реализаций метода использует параметры, а другая может их не использовать). Более того, это значение может быть реализовано вообще как хранимое. С другой стороны, реализации атрибутов, не имеющих параметров, может содержать вычисляющие выражения.

Для объектных типов могут быть заданы локальные, глобальные и внешние ключи, определяемые как набор принадлежащих объекту скалярных полей. Ключи могут быть простыми (содержащие одно скалярное поле) или сложными (содержащие несколько скалярных полей). Ключи являются ограничением целостности данных.

Локальный ключ может задаваться для компонентов-множеств. В него входят поля, определяющие уникальность входящих в множество скаляров или кортежей в пределах компонента объекта (при этом в разных объектах эти скаляры или кортежи, конечно же, могут повторяться). В случае отсутствия явно определенного локального ключа подразумевается, что элементы множества различаются по своему полному значению, или, другими словами, что структура ключа совпадает со структурой элементов множества.

Замечание. В связи с этим отметим, что для множеств скаляров, когда структура ключа не может отличаться от структуры элемента множества, его явное определение не имеет смысла.

Глобальный ключ явным образом определяет, что объект будет отличаться своим состоянием от других объектов этого же класса. Глобальные ключи не являются обязательными, поскольку уникальность объекта в любом случае определяется объектным идентификатором. В глобальный ключ могут входить либо компоненты-скаляры, либо скалярные поля одного из компонентов-кортежей или компонентов-множеств. Для объектного типа может быть задано несколько глобальных ключей.

Глобальный ключ, заданный как набор полей компонента-множества, определяет, что элементы этого множества уникальны во всех существующих в системе объектах данного типа - в том числе и в пределах каждого объекта. Отметим, что если для типа

задан глобальный ключ, определенный на полях компонента-множества, то для каждого объекта этого типа одновременно может существовать множество уникальных значений этого ключа.

Во внешние ключи входят либо компоненты-скаляры, либо скалярные поля одного из компонентов-кортежей или компонентов-множеств, аналогичных полям, входящих в один из существующих глобальных ключей. Смысл и цели внешних ключей аналогичны смысл и целям внешних ключей РСУБД.

Пример. *Объектный тип* `Warehouse` *описывает склады. Для объектов этого типа не определены глобальные ключи – таким образом, в системе могут существовать неразличимые по значению объекты этого типа. Компонент* `ResourceItems` *содержит данные о товаре, хранящемся на складе. Значение компонента* `ResourceItems` *определено как множество кортежных значений типа* `ArtQty`*, при этом определено, что атрибут* `Art` *является ключом.*

```
CREATE CLASS Warehouse
{
  Address STRING;
  ResourceItems SET OF ArtQty
    CONSTRAIN
    LOCALKEY Art;
}
```

Объектный тип `GoodsMotion` *описывает движение товара. Существующие в системе объекты этого типа уникальны по атрибуту* `No`*. Компоненты* `FromWarehouse` *и* `ToWarehouse` *могут ссылаться на существующие в системе объекты типа* `Warehouse` *(значение этих полей может быть не определено -* `FromWarehouse` *в случае поставок,* `ToWarehouse` *в случае продажи). Компонент* `MovedItems` *содержит информацию о количестве перевозимого товара.*

```
CREATE CLASS GoodsMotion
{
  No INTEGER
    CONSTRAIN GLOBALKEY No;
  DateOfAction DATE;
  FromWarehouse Warehouse;
  ToWarehouse Warehouse;
  MovedItems SET OF ArtQty
    CONSTRAIN
    LOCALKEY Art;
}
```

Рассматриваемый здесь набор значимых типов позволяет однозначно выполнить основное требование R*O системы. В самом деле

- Значение компонента-множества можно рассматривать как значение отношения, схема которого соответствует схеме элементов этого множества.
  Замечание. Говоря о множества значений скалярного типа, будем предполагать, что соответствующая схема отношения содержит единственный аргумент **value** этого типа.
- Значение компонента-кортежа можно рассматривать как значение задаваемого схемой этого кортежа отношения с одним-единственным кортежем.
  Замечание. Мы считаем, что для отношений, однозначно состоящих из одного-единственного кортежа, ключ задавать не нужно. Подразумевается, что ключи, определяющие уникальность кортежей в пределах отношения и позволяющие организовать доступа к каждому среди этих кортежей, в случае, когда отношение имеет один кортеж, попросту не нужны.
- Совокупность входящих в объект компонентов-скаляров можно рассматривать как значение отношения с одним-единственным кортежем. Будем называть эту

совокупность *собственным кортежем* объекта. Схема соответствующего отношения задается в процессе описания объектного типа и содержит все его собственные (т.е. неунаследованные) скалярные компоненты.
Замечание. В принципе, ничто не мешает рассматривать каждый из компонентов-скаляров как значение унарного и однокортежного отношения. Объединение этих атрибутов в единое собственное отношение существенно упрощает наше построение.

Рассматриваемый набор значимых типов позволяет описывать структуры данных, сравнимые своей в сложностью со структурами данных, существующими в традиционных ОО-языках. В самом деле, компоненты объектов могут быть простыми значениями (скаляры), записями (кортежи), повторяющимися группами (множества). Существование ссылочного типа, относящегося к базовым скалярным типам, позволяет описывать вложенные сложные структуры.

Объектные типы образуют иерархию наследования (невозможно наследование объектных типов от значимых, значимые типы не могут наследоваться от объектных). Наследование объектных типов подразумевает, что спецификация типа наследника включает спецификацию родительского типа (или типа предка). Допускается множественное наследование. Существует предопределенный фиктивный объектный тип **Object**, по умолчанию являющийся предком для любого объектного типа.

Замечание. При наследовании от типов, имеющих общий базовый тип, спецификация последнего не дублируется (используя термины C++ можно сказать, что спецификации объектных типов наследуются *виртуально*). В системе должен существовать механизм, позволяющий разрешать возможные в случае множественного наследования конфликты реализаций компонентов.

Пример. *Опишем кортежный тип* `SaleQty`*., служащий для описания количества определённого артикула.*

```
DESCRIBE TUPLE SaleQty
{
  Art Article;
  Quantity INTEGER;
  Price FLOAT;
}
```

Объектный тип `Sales` *описывает факты продаж. Поскольку продажи можно рассматривать как частный случай отгрузки со склада, объектный тип* `Sales` *является наследником типа* `GoodsMotion`*. Компонент* `SaleItems` *содержит данные о продаваемом товаре. Его значение определено как множество кортежных значений типа* `SaleQty`*, где атрибуты* `ArticleNo` *и* `Price` *составляют ключ (одни и тот же артикул может продаваться по разным ценам). Компонент* `IsPayed` *указывает, была ли произведена оплата. Компонент* `DoSale` *является методом, принимающим в качестве атрибута дату отгрузки и возвращающим результат, свидетельствующий об успехе выполнения операции отгрузки.*

```
CREATE CLASS Sales EXTENDED GoodsMotion
{
  IsPayed BOOLEAN;
  SalesManager Manager;
  SaleItems SET OF SaleQty
    CONSTRAIN
    LOCALKEY (Art, Price);
  DoSale (DateOfSale) BOOLEAN;
}
```

Возможно изменение схемы данных. Например, пусть в объектный тип `Brand` *нужно добавить компонент, содержащий информацию о продажах артикулов, принадлежащих данной(*`this`*) торговой марки. Отметим, что поскольку каждый артикул принадлежит только одной торговой марке, поле* `Art` *описано как глобальный ключ. Таким образом, любой кортеж компонента* `SaledItems` *любого объекта типа* `Brand` *будет уникальным во всех этих объектах.*

```
ALTER CLASS Brand
  ADD SaledItems SET OF ArtQty
    CONSTRAIN
    GLOBALKEY Art;
```

В общем случае спецификация объектного типа включает в себя

1) имя типа
2) перечень родительских типов (если не определено иначе, родительским типом является тип Object)
3) набор спецификаций компонентов, включающих a) имя компонента, b) значимый тип компонента, и c), возможно, набор параметров, каждый из которых описывается как пара <имя параметра, значимый тип параметра>.
4) набор ограничений целостности данных – ключи.

Замечание. Мы не рассматриваем иные характерные для языков программирования возможности спецификации объектных типов (например, модификаторы видимости private и public, модификаторы обновляемости readonly, и т.д.), однако допускаем, что такие возможности могут существовать и быть полезными.

Реализация типа представляет собой набор реализаций его компонентов. Реализация любого компонента определяет источник значения этого компонента, указывая, является ли он хранимым или вычисляемым, и, в последнем случае, содержит вычисляющее выражение или вычисляющую функцию. Реализация компонента может содержать предикат компонента – определяемые предметной областью условия на его возможные значения. Кроме того, в реализации компонента может быть определен триггер, выполняемый системой при определенных манипуляциях с этим компонентом. Вычисляющие выражения и функции могут принимать аргументы значимых типов.

Пример. *Реализуем объектный тип* `GoodsMotion` *. Все его компоненты являются хранимыми.*

```
ALTER CLASS GoodsMotion
REALIZE No, Date, FromWarehouse, ToWarehouse, MovedItems AS STORED;

Точно так же реализуются тип Article
ALTER CLASS Article
REALIZE * AS STORED;
```

При наследовании компонента его реализация может меняться. Таким образом, компоненты типа (а, следовательно, и сам тип) являются полиморфными в том смысле, что одной спецификации может соответствовать несколько реализаций.

Замечание. Фактически, НРМ утверждает, что не только методы, но и атрибуты объектов могут и должны являются средством реализации таких ОО-концепций, как инкапсуляция, наследование и полиморфизм. Данное утверждение основывается на том факте, что теоретико-множественные и специальные операторы реляционной алгебры могут применяться к значениям отношения вне зависимости от того, являются ли эти значения хранимыми или вычисляемыми, что позволяет разделить задаваемое в описании объектного типа описание атрибута, на спецификацию и реализацию. В спецификации задается сигнатура атрибута – его имя и имя его значимого типа. Реализация определяет источник значения этого

атрибута, указывая, является ли он хранимым или вычисляемым, и, в последнем случае, содержит вычисляющее выражение. Таким образом
- Атрибуты объектов инкапсулируются в объектном типе. В общедоступной спецификации этого типа определяется только сигнатура атрибута. Описание реализации элемента, содержащее информация об источнике значения этого компонента, скрыто.
- Атрибуты объектов наследуются. Спецификация типа-наследника включает спецификацию базового типа - в том числе и определенные в ней спецификации атрибутов.
- Атрибуты объектных типов могут быть полиморфными. Реализация типов может меняться в процессе наследования, из чего, например, следует, что атрибут, определённый в родительском типе как хранимый, в типе-наследнике может стать вычисляемым (и наоборот), или что в процессе наследования может измениться вычисляющее выражение.
Таким образом, перечисленные ОО концепции могут реализовываться всеми компонентами объектов - и атрибутами, и методами.

Операции над объектами.

Объекты могут создаваться и уничтожаться. Для того чтобы изменить состояние объекта, необходимо обратиться к определенным в соответствующем объектном типе компонентам этого объекта. Состояние объекта может изменяться как явным (явно заданная операция, изменяющая значение компонента), так и неявным (при выполнении методов) образом.

Все компоненты допускают операцию чтения. Компоненты, не имеющие параметров (т.е. атрибуты), допускают также операцию присваивания. Естественно, что значение, присваиваемое атрибуту, должно иметь тип этого компонента. Для атрибутов-множеств возможны так же операции, изменяющие число кортежей (INSERT, DELETE) и операции, изменяющие состояние существующих кортежей (UPDATE).

Замечание. Мы осознаем трудности, которые могут возникнуть при реализации операций, явно изменяющих значение атрибута, в случае, когда эти атрибуты реализованы как вычисляемые. Отметим, что существующие СУБД позволят решать такие проблемы (например, с помощью триггеров).
Замечание. Допуская для всех без исключения атрибутов операцию присваивания, НРМ не утверждает, что эта операция должна обязательно *изменять* значение этого атрибута. Это замечание относится к вычисляемым атрибутам. Например, атрибут, содержащий информацию об остатках на складе, вычисляется исходя из цифр поставок и отгрузок. Исходя из этого, система не должна, говоря образно, обращать внимание на попытки явного присваивания значений этому атрибуту – если, конечно, иное не определяется реализацией.
Замечание. Говоря о соответствии типов операндов в операции присваивания, мы опускаем существующие во многих языках возможности по неявному приведению типов (например, арифметических), считая их возможными и полезными.

Методы, изменяющие состояние объекта, представляют собой последовательность вышеописанных операций, выполняемых над компонентами, определенными в объектном типе, а также над видимыми в теле метода переменными (к которым относятся описываемые далее глобальные R-переменные). Методы могут иметь локальные переменные значимых типов. Время жизни локальных переменных ограничено временем выполнения метода или триггера.

У объектных типов могут существовать конструкторы и деструкторы – методы, вызываемые, соответственно, при создании и уничтожении объектов.

Пример. *Реализуем метод* `DoSale` *типа* `Sales`

```
ALTER CLASS Sales
  REALIZE DoSale
AS BEGIN
IF DateOfAction NOT IS NULL THEN //если отгрузка уже сделана
  IF DateOfAction = DateOfSale Then Return TRUE;
```

```
    //и сделана тем же числом – OK!
  ELSE RETURN FALSE;
    //отгрузка сделана другим числом - ошибка
ELSE//отгрузка еще не сделана
  IF IsPayed THEN
    BEGIN
      DateOfAction := Date Of Sale;
      RETURN TRUE; //продажа уже оплачена - OK
    END
  ELSE
    RETURN FALSE ;//отгрузка невозможна (нет оплаты) - Ошибка
END;
```

Для объектов определены операции, позволяющие определить тип этих объектов. В связи с тем, что в R*O системе поддерживается наследование объектных типов, данное требование требует некоторых разъяснений. Существуют две операции, позволяющие определить тип объекта. Первая операция **o IS t** (где **o** - ссылка на объект, а **t** – имя типа), возвращает истину, если объект, определяемый ссылкой **o**, является объектом данного типа. Подразумевается, что, в случае наследования, объект любого типа-наследника является также объектом базового типа. Из этого, в частности, следует, что операция **o IS Object** (где **Object** – предопределенный базовый тип) вернет истину для любого объекта. Вторая операция, **o OF t,** возвращает истину, когда объект, определяемый ссылкой **o**, был создан как объект класса **t** (т.е. этот объект был создан операцией **new t**). Соответственно, **o OF Object** всегда будет ложью.

OID и ссылки.

Повторим, что каждый существующий в системе объект идентифицируется своим уникальным объектным идентификатором (OID), который присваивается ему системой при создании и отличает его от любого другого объекта любого объектного типа. Именно OID обеспечивает доступ к определенным для соответствующего объектного типа компонентам конкретных объектов.

Операция сравнения объектов (один и тот же объект – разные объекты) основывается на непосредственном сравнении их объектных идентификаторов. Именно поэтому объектные идентификаторы (сами по себе) рассматриваются как генерируемые системой значения ссылочного скалярного типа (домена) **DOID**. Входящие в компоненты объектов поля ссылочного типа позволяют описывать связи, существующие между объектами моделируемой предметной области. Для переменных ссылочного типа определены операции присваивания, сравнивания и неявного разыменования. Последнее подразумевает, что любая операция, отличная от операций присваивания и сравнивания, выполняется не над ссылкой, а над объектом, на который эта ссылка указывает.

В любой момент времени множество действующих в системе значений ссылочных типов ограничиваются множеством значений объектных идентификаторов объектов, существующих в системе на данный момент времени. Каждому объектному типу соответствует свой ссылочный тип. Это ссылочный тип создается одновременно с объектным типом. Имена этих типов совпадают. Ссылочные типы образуют иерархию наследования аналогичную иерархии наследования объектных типов. В случае ссылочных типов наследование подразумевает, что ссылка на объект некого типа, может содержать OID любого из объектов этого типа – в том числе OID объектов

любого из типов-наследников. В системе предопределен ссылочный тип **Object**. Поле этого типа может ссылаться на любой существующий в системе объект.

В системе могут существовать переменные, представляющие собой группу ссылок на объекты заданного типа (переменные типов-множеств определенных на ссылочном типе или, по другому, групповые ссылки). Отметим, что значение групповой ссылки можно рассматривать как значение отношения. Одиночную ссылку (т.е. ссылку на один-единственный объект) можно рассматривать как частный случай групповой.

R-переменные.

R*O-система позволяет организовать основанный на реляционной модели данных групповой ассоциативный доступ к объектам и к данным этих объектов. Возможность такого доступа основывается на том, что объявление типа можно рассматривать также как объявление набора переменных отношений, содержащих данные обо всех существующих в системе объектах данного типа. Будем называть их R-переменными. Рассмотрим эти переменные подробнее.

1) R-переменные компонентов типа. Как мы уже сказали, состояние объекта описывается набором значений отношений, определённых на множестве скалярных типов. Каждому объекту соответствует уникальный объектный идентификатор, который также представляет собой значение скалярного типа. Исходя их этого, объявление объектный типа **t**, содержащего компонент **a** со схемой **(x_1:D_1, …, x_n:D_n)** можно рассматривать также как объявление переменной **t.a** отношения со схемой **(OID: DOID , x_1:D_1, …, x_n:D_n)** (отметим, что это отношение определено на том же множестве скалярных типов). Как видно, имя этой переменной определяется как комбинация имени типа и имени компонента этого типа (мы используем точечную нотацию). Уместно заметить, что реляционная модель не накладывает каких-либо ограничений на имена переменных отношений и атрибутов отношений, за исключением требования их уникальности.

Переменная **t.a** содержит совокупность значений компонента **a** всех существующих в системе объектов объектного типа **t**.так, что каждому кортежу компонента **a** объекта типа **t** поставлен в соответствие объектный идентификатор этого объекта. Этот идентификатор содержится в атрибуте OID, который, тем самым, является ссылкой на этот объект.

Отметим, что, охарактеризовав содержимое переменой **t.a** как "совокупность значений" мы ни в коем случае не имели в виде, что оно есть результат простого объединения этих значений. Например, кортежи компонента-множества **a** объектов класса **t**, будучи уникальными внутри каждого объекта, могут повторяться среди разных объектов этого класса. Таким образом, простое объединение **v_1 UNION v_2 UNION …** где **v**- значение компонента-множества **a** одного из объектов, может повлечь потерю данных, которая выразится в потере таких повторяющихся кортежей. Имеющийся в переменной **t.a** атрибут **OID**, уникально идентифицирующий каждый объект,

(OID_1 × v_1) UNION (OID_2 × v_2) UNION … (где × – символ декартова произведения)

гарантирует, что такой потери данных не произойдет.

Следствием такого представления данных является следующее. Привычный способ доступа к данным в традиционных ОО-системах начинается с так или иначе

хранящегося OID (т.е. с ссылки на объект), используя который можно получить доступ к атрибутам и методам объектов (если OID не сохранен, объект считается потерянным). Переменная же **t.a** позволяет выполнить обратное действие - получить OID (т.е. ссылку на объект) на основании данных компонента **a** объектов типа **t**. Из этого, в частности, следует, что для того, что бы получить доступ к тому или иному объекту, нет нужды хранить его OID. Таким образом, переменная **t.a** представляет данные, говоря образно, в *однородном* виде.

Будем называть переменные, подобные описанной переменной **t.a**, R-переменными компонентов объектного типа. Каждому объектному типу **t** может соответствовать множество таких R-переменных - по одной на каждый компонент кортежного типа или типа-множества, и еще одна для собственного кортежа объекта. Другими словами, число R-переменных компонентов типа определяется числом отношения, описывающих состояние объектов этого типа.

Замечание. Важно понимать, что значения, хранящиеся в R-переменных, всегда являются значениями отношений. Выражение "R-переменная … типа **t**" означает лишь то, что указанная R-переменная ассоциирована с типом **t**.

Заметим, что значение имеющегося в R-переменных атрибута OID является системным. В связи с этим для доступа к атрибуту OID вместо выражения **Rvar.OID** далее будет использоваться функциональное выражение **Object(Rvar)**. Здесь **Rvar** имя R-переменной, которая содержит атрибут OID. Выражение **Object(expr)**, где **expr** – выражение, вычисляющее значение отношения с атрибутом OID, будет использоваться вместо операции проекции **expr[OID]**. В обеих случаях выражение **Object(…)** возвращает групповую ссылку. В частности, групповую ссылочную переменную мы будем рассматривать как переменную унарного отношения с единственным атрибутом **Object(ref)**, где **ref** – имя этой ссылочной переменной.

Замечание. Более того, очевидно, что для пользователя не представляет интереса собственно значение OID (оно генерируется системой и зависит от реализации). Соответственно, возможное представление значений ссылочного типа может вообще не зависеть от самих этих значений. Например, любое значение ссылочного типа может быть представлено для пользователя строкой "Object". Можно предположить, что возможные представления могут быть изменены в процессе наследования. Например, для документов, у которых определен номер, являющийся глобальным ключом, возможное представление может быть реализовано как строка "Документ номер ….".

Пример. *Определение объектного типа* `GoodsMotion` *можно также рассматривать как объявление R-переменной компонента типа* `GoodsMotion.MovedItems` *со схемой (OID, Article, Quantity). Соответственно, проекция* `Object(GoodsMotion.MovedItems WHERE Article = "art1")` *вернет значение, представляющее собой множество OID объектов, которые описывают движения товаров с артикулом "art1".*

Как мы сказали, схема R-переменной компонента типа **t.a** представляет собой схему компонента **a**, дополненную атрибутом OID. Ключи переменной **t.a** также однозначно определяются ключами, заданными для этого компонента. Возможны три случая:

- если для компонента **a** определен глобальный ключ, ключ соответствующей переменной **t.a** содержит в точности те же поля, что и этот глобальный ключ (это относится и к внешним ключам);
- если для компонента **a** определен локальный ключ, ключ соответствующей переменной **t.a** содержит поля, входящие в указанный локальный ключ, а также поле OID;
- если для компонента **a** ключ не определен, ключ соответствующей переменной **t.a** содержит единственное поле OID.

2) R-переменная типа. Еще раз повторим, что в соответствии с основным требованием R*O системы состояние объекта описывается набором значений отношений, определённых на множестве скалярных типов. Декартово произведение этих значений представляет собой значение 1НФ отношения, полностью описывающее состояние этого объекта (схема этого отношения включает все *скалярные* поля, существующие в компонентах этого объекта). Каждому объекту и, следовательно, каждому такому значению, соответствует уникальный объектный идентификатор, также представляющий собой значение скалярного типа. Сказанное позволяет рассматривать объявление объектного типа **t** также и как объявление переменной **t,** содержащей данные всех существующих в системе объектов этого типа (это отношение определено также на множестве скалярных типов).

Оговоримся, что операция, производящая значение 1НФ отношения, полностью описывающее состояние объекта, должна быть несколько более сложной, чем простое декартово произведение. Дело в том, что имена полей, входящих в различные компоненты объекта, могут совпадать между собой. Для того чтобы избежать подобных конфликтов имен, предлагается в процессе создания декартова произведения уточнять имена полей именами компонентов. Например, кортежные типы $\mathbf{R_1}$ и $\mathbf{R_2}$, на которых определены компоненты $\mathbf{a_1}$ и $\mathbf{a_2}$, могут иметь поля с одинаковым именем **x**. (Еще раз отметим, что реляционная модель не накладывает каких-либо ограничений на имена, за исключением требования их уникальности.) Если использовать точечную нотацию, то уточненные имена будут выглядеть как $\mathbf{a_1.x}$ и $\mathbf{a_2.x}$. По нашему мнению, уточнение имен сохраняет семантику и позволяет выразить сложность структуры объекта в сложном имени атрибута R-переменной. Например, множество значений атрибута **x** компонента **a** типа **t** может быть получено как с помощью операции выборки из R-переменной компонента **t.a[x]**, так и из R-переменной типа **t[a.x]**.

Таким образом, значение переменной **t** представляет собой совокупность значений декартовых произведений семантически уточненных компонентов объектов типа **t**, существующих в системе. Каждому объектному типу **t** соответствует единственная переменная **t**. Будем называть такие переменные R-переменными типов.

Пример. *Объектному типу* `GoodsMotion` *соответствует R-переменная этого типа* `GoodsMotion` *со схемой (OID:DOID, No:INTEGER, DateOfAction:DATE, FromWarehouse:Warehouse, ToWarehouse:Warehouse, MovedItems.Article:STRING, MovedItems.Quantity:INTEGER). Отметим, что это отношение определено на множестве скалярных типов, включающих тип объектных идентификаторов DOID и ссылочный тип Warehouse, который был определен в процессе объявления соответствующего объектного типа. Операция* `Object(GoodsMotion WHERE MovedItems.Article = "art1" AND DateOfAction = '31.05.2005')` *вернет значение, представляющее собой множество OID объектов, описывающих движения товаров с артикулом "art1", произведенных 31мая 2005 года.*

Свойства R-переменных.

Предполагается, что данные, хранящиеся в этих переменных, всегда актуальны – любое изменение состояний объектов приводит к изменению значений соответствующих R-переменных. Другими словами, данные, представленные в виде значений компонентов объектов, и данные, представленные в виде значений R-переменных - это одни и те же данные (в дальнейшем мы будем говорить о двояком представлении данных).

Рассматривая свойства R-переменных относящихся к типу **t** (т.е. переменных типа **t** и переменных компонентов этого типа **t.a**), надо отметить, что эти переменные существуют вне зависимости от существования объектов данного объектного типа и, тем более, от числа этих объектов. Существование этих переменных определяется в момент создания типа. Таким образом, эти переменные являются глобальными и могут использоваться как операнды непроцедурных команд управления системой, а также в любых определяемых в системе процедурах и функциях.

Такой подход освобождает программиста-пользователя от необходимости производить какие–либо действия по организации группового доступа к данным. Объявление типа **t** одновременно является объявлением соответствующих R-переменных, данные в которых всегда актуальны. При этом для обозначения R-переменных используются имена, вводимые в процессе объявления и описания типа. Будем называть такие имена многозначными. Необходимая интерпретация многозначных имён определяется операцией, в которой эти имена используются. Например, в операции создания нового объекта **new t** имя **t** интерпретируется как имя типа. В операциях же группового доступа к данным, имя **t** должно интерпретироваться как имя R-переменной.

Замечание. Этот подход освобождает от неоднозначностей, присущих термину "класс", описывающему нечто, что является одновременно и фабрикой объектов (~тип), и хранилищем объектов (~переменная) В нашем изложении этот термин вообще не используется.

Пример. *Реализуем объектный тип* `Warehouse`. *Он содержит и хранимые, и вычисляемые компоненты. Так, компонент* `ResourceItems` *вычисляется как разница между поставленными на данный(*`this`*) склад и отгруженными с данного склада количествами. Отметим, что в вычисляющем выражении используется глобальная R-переменная типа* `GoodsMotion`.

```
ALTER CLASS Warehouse
REALIZE Address As STORED
REALIZE ResourceItems AS
SUMMARIZE (
  SUMMARIZE
  (GoodsMotion WHERE ToWarehouse = this)
  BY Art ADD Sum(MovedItems.Pieces) AS SumPieces
  UNION
  SUMMARIZE (GoodsMotion WHERE ToWarehouse = this)
  BY Art ADD Sum(0-MovedItems.Pieces) AS SumPieces)
BY Art ADD Sum(SumPieces) AS Pieces;
```

Как мы уже сказали, R-переменные позволяют получить ссылку на объект типа **t** на основании данных компонентов объектов этого типа. Из этого, в частности, следует, что для того, чтобы получить доступ к тому или иному объекту нет нужды хранить его OID. В свою очередь это означает, что при создании объекта не нужно сохранять его OID в ссылочной переменной, что позволяет использовать оператор `new`, создающий объект, как непроцедурную команду.

Замечание. Из этого, в частности, следует, что особенно важным становится использование конструкторов, позволяющих инициализировать компоненты объекта в процессе его создания (объект, созданный командой new без использования конструктора не будет содержать данных, позволяющих отличить его от других объектов того же класса).

Пример. *Предположим, что у типа Article создан конструктор, принимающий в качестве параметра наименование артикула.*

```
ALTER CLASS Article
```

```
  ADD Article( InArticle As STRING);

ALTER CLASS Article
  REALIZE Article AS
  BEGIN
    No := InArticle;
  END;
```

Создадим командой
`new Article("art1");`
новый объект класса `Article`*. Отметим, что при этом мы не сохранили ссылку на этот объект. Однако она всегда может быть получена путем выборки из R-переменной типа* `Article`*. Например, объявленная в теле метода локальная переменная* `refArt` *ссылочного типа* `Article` *…*

```
BEGIN
…
refArt Article;
refArt := Object(Article WHERE No = "art1")
…
END
```

… инициализируется ссылкой на созданный ранее объект. Ограничение целостности GLOBALKEY, установленное ранее для поля `No`*, гарантирует, что выборка по этому полю вернет ссылку на один единственный объект.*

Как указывалось ранее, в спецификации объектного типа не определяется, являются ли значения его компонентов хранимыми или вычисляемыми. Компонент, реализованный в родительском типе как хранимый, может быть переопределен в классе наследнике как вычисляемый (и наоборот). Соответственно, когда речь идет о полиморфном наследуемом объектном типе, любая из R-переменных его компонентов может содержать одновременно как хранимые, так и вычисляемые разными способами значения.

Говоря строго, значение R-переменной компонента, конечно же, вычисляется и представляет собой объединение **UNION** нескольких значений, одни из которых могут быть реализованы как хранимые, а другие являются вычисляемыми. Для R-переменных типов картина еще более сложна. Поскольку значение R-переменной типа определяется как декартово произведения значений компонентов, возможен случай, когда в некоторых кортежах только несколько атрибутов являются хранимыми. А из-за того, что в процессе наследования типа реализация компонентов, содержащих эти атрибуты, может измениться, в других кортежах указанные атрибуты могут вычисляться (т.е. R-переменная типа, говоря образно, хранится *мозаично*). Но в любом случае вычисления значений любых R-переменных должны выполняется системой неявно для пользователя (на основании информации о наследовании типов и реализации компонентов этих типов). Для использования R-переменных необходима только спецификация типа.

Можно провести следующую аналогию. Полиморфные ОО-языки программирования делают ненужными использование громоздких и чувствительных к изменениям программы селекторных конструкций, необходимых для выполнения близких по смыслу действий для структур, хранящих близкие по смыслу данные, например

```
if s.f=1 then function1(s)
elseif s.f =2 then function2(s)
```

, заменяя их вызовом полиморфного метода `s.function()`..

Точно так же полиморфные компоненты делают ненужным явное использование оператора **UNION** для получения значения отношения, объединяющего близкие по смыслу данные в том случае, когда эти данные хранятся и/или вычисляются разными способами. При этом подразумевается, что объектный тип является наследуемым и что полиморфный компонент может менять свою реализацию. Таким образом, если имеется объектный тип **t**, в котором определён компонент **a**, и соответствующие этому типу R-переменные используются в запросах и методах, то создание типа **t***, наследующего тип **t** и переопределяющего реализацию компонента **a**, не ведет к необходимости менять эти запросы и методы.

Пример. *Реализуем объектный тип* `Sales`*. Его компонент* `SaleQty` *является хранимым*

```
ALTER CLASS Sales
  REALIZE SaleItems AS STORED;
```

Реализуем компонент `MovedItems`*, описывающий отгружаемый со склада товар. Поскольку его значение напрямую связано с количеством продаваемого товара (равно ему), необходимо переопределить его как вычисляемый.*

```
ALTER CLASS Sales
  REALIZE MovedItems AS
  SUMMARIZE SaleItems
  BY Art ADD Sum(Pieces) AS Pieces;
```

Теперь количество находящегося на складе товара будет вычисляться на основании данных не только об отгруженном (`GoodsMotion.MovedItems`*), но и о проданном (*`Sales.SaleItems`*) товаре. Это достигнуто за счет изменения реализации полиморфного компонента* `MovedItems` *в процессе наследования. Заметим, что при этом ранее определённые схемы и вычисляющие выражения никак не изменились. Например, выражение, возвращающее общее количество штук на всех складах*

```
SELECT SUM(pieces) FROM Warehouse.ResorceItems;
```

будет возвращать правильные значение вне зависимости от наличия у типов `Warehouse` *и* `GoodsMotion` *типов-наследников (конечно же, при условии, что участвующие в вычислении компоненты этих типов переопределены корректно). Выражение не нужно менять, даже если необходимость в таком типе-наследнике возникнет в процессе функционирования системы.*

Операции с R-переменными.

Итак, R-переменные позволяют получить ссылку на объект (или ссылки на объекты) используя значения компонентов и, таким образом, позволяют организовать ассоциативный доступ к объектам заданного типа. Значения этих переменных представляют собой значения отношения, доступ к которым может осуществляться при помощи операций, традиционных для существующих реляционных БД. Тот факт, что структура этих переменных определяется структурой соответствующих объектных типов, позволяет ввести две операции, обладающие объектной семантикой.

Замечание. Эти операции не выходят за рамки реляционной модели, поскольку представляют собой суперпозицию базовых реляционных операций. Из этого, в частности, следует, что их результат представляет собой значение отношение.

Выборка объектов по значениям.

Отметим, что оператор выборки **WHERE** представляется явно недостаточным, если целью выборки является выборка сложных объектов, отвечающих определенным

условиям. Это связано с тем, что условие отбора оператора **WHERE** применяется к кортежам, но данные каждого объекта могут быть представлены в R-переменной *множеством* кортежей. Предположим, например, что речь идет об объектах типа **t**, у которых в компоненте **a** хранится значение отношения **R(…, x, …),** и необходимо найти объекты (т.е. получить ссылки на объекты), компонент **a** которых содержит, по меньшей мере, один кортеж со значением атрибута **x = 1** и, по меньшей мере, один кортеж со значением **x=2**. Поскольку атрибут **x** в одном кортежа не может одновременно равняться и единице и двойке, выражение **Object(t.a WHERE x = 1 AND x =2)** является попросту бессмысленным.

Для решения подобных задач предлагается операцию выборки объектов типа **t** по значениям. Эта операция имеет вид **Rvar<cond$_1$, cond$_2$,…>**, где **Rvar** – выражение, определяющее R-переменную (это может быть, например, имя объектного типа, или имя ссылки на объекты этого типа), а каждое **cond$_i$** представляет собой условие, применимое в операции **WHERE**.

Значение. Выражения **<cond$_1$, cond$_2$,…>** вычисляется как **(Object (t WHERE cond$_1$) INTERSEPT Object (t WHERE cond$_2$) INTERSEPT …)** и, следовательно, представляет собой групповую ссылку на те существующие в системе объекты типа **t**, которые удовлетворяют заданным условиям. Конечным этапом выборки по значениям является выборка из R-переменной **Rvar** по атрибуту, содержащему объектный идентификатор**.** Например значение **t<cond$_1$, …>.a** вычисляется как **t WHERE EXIST (Object(t) JOIN (Object (t WHERE cond$_1$) INTERSEPT …))** и представляет собой подмножество значения R-переменной компонента **t.a**, которое содержит информацию только о тех объектах, для которых выполняются все перечисленные условия.

Раскрытие ссылки.

Еще одним следствием того, что данные в R-переменных представлены в *однородном* виде, является возможность использования групповых операций при работе со ссылками. В основе данного предложения лежит то, что, если объекты типа **t** содержат поле **x$_{ref}$**, ссылающееся на объекты типа **t***, то R-переменная типа **t** может быть соединена с R-переменной типа **t*** с условием эквивалентности значений атрибутов **x$_{ref}$** и **Object(t*)**. Назовем эту операцию раскрытием ссылки. В общем случае она выглядит как **RVar EXPAND RefAttr**, где **RefAttr** является сслочным атрибутом R-переменной **RVar**.

Предположим, что нами определен объектный тип **t**, содержащий компонент **a$_n$**, типа-множества **SET OF R**. В свою очередь, кортежный тип **R** содержит атрибут **x$_{ref}$**, являющейся ссылкой на объект типа **t***. Таким образом, объектный тип **t** связан по ссылке с объектным типом **t***.Рассмотрим операцию раскрытия ссылки **t EXPAND(a.x$_{ref}$).** Применив ее к отношениям типов **t(Object(t), a$_1$.x$_1$, … , a$_n$.x$_{ref}$, … , a$_z$.x$_n$)** и **t*(Object(t*), a*$_1$.x$_1$*, …, a*$_z$.x$_m$*)**, мы получим отношение со схемой **(Object(t), a$_1$.x$_1$, …, a$_n$.x$_i$.a*$_1$.x$_1$*, …, a$_n$.x$_i$.a*$_z$.x$_m$*, … , a$_z$.x$_n$)**.

Операция раскрытия ссылки **t EXPAND(a.x$_{ref}$)**, вычисляется как
t JOIN $_{a_n.x_{ref}}$ (t* RENAME Object(t*), a*.x$_1$*, … , a*.x$_n$* AS a.x$_{ref}$,a.x$_i$.a*x$_1$*, … , a.x$_i$.a*x$_n$*)
, где **a*.x*** - атрибуты R-переменной объектного типа **t*.**

Как видно, операция раскрытия ссылки является быть более сложной, чем просто эквисоединение. Дело в том, что типы **t** и **t*** могут иметь компоненты и атрибуты компонентов с одинаковыми именами. Более того – возможен случай, когда типы **t** и **t*** являются одним и тем же типом (например, объекты, описывающие людей, содержат ссылки на объекты, описывающие родителей – тех же людей). Для того, что бы избежать подобных конфликтов, операция раскрытия ссылки *уточняет* имена

компонентов того типа, на который указывает ссылочное поле, именем этого ссылочного поля.

Например, если объектный тип **t** содержит компонент **a** с атрибутом $\mathbf{x_{ref}}$, который ссылается на объект, содержащий компонент **a*** с атрибутом **x***, то операция раскрытия ссылки, использующая точечную нотацию, уточнит имя последнего атрибута как $\mathbf{a.x_{ref}.a^*.x^*}$. Повторим, что реляционная модель не накладывает каких-либо ограничений на имена, за исключением требования их уникальности. В данном случае сложное имя $\mathbf{a.x_{ref}.a^*.x^*}$ является гарантировано уникальным. Важно, что сложное имя атрибута, возникающее при раскрытия ссылки, является корректным путевым выражением описанной ссылочной структуры. Таким образом, операция раскрытия ссылки сохраняет семантику данных, выражая сложность структуры объектов и связей между ними в сложном имени атрибута R-переменной.

Операция раскрытия ссылки позволяет организовывать ассоциативный доступ к данным объектов любого типа, связанного с объектами данного типа по ссылке. При этом доступ к данным возможен как по ссылке, так и в противоположном направлении (конечно, на самом деле используемое при этом отношение, являющееся результатом операции **EXPAND**, не подразумевает каких-либо направлений, поскольку поля, содержащие OID объектов, ссылочные поля, содержащие OID связанных объектов, и поля данных в них абсолютно *равнозначны*). Например, можно получить ссылки на объекты типа **t**, связанные по ссылке $\mathbf{x_{ref}}$ с теми объектами типа **t***, у которых атрибут **x*** компонента **a** равен определённому значению, например
$\mathbf{Object((t\ EXPAND\ a.x_{ref})WHERE\ a.x_{ref}.a^*.x^* =1)}$

Операции раскрытия ссылки может применяться к вложенным ссылкам
$\mathbf{(t\ EXPAND\ a.x_{ref})EXPAND\ a.x_{ref}\ .a^*.x_{ref}^*}$

Заметим, что существование атрибутов с уточненными именами подразумевает обязательное выполнение операции раскрытия ссылки. Из этого следует, что эта операция может вызываться неявно. С учетом этого, предыдущее выражение может быть записано как
$\mathbf{Object(\ t\ WHERE\ a.x_{ref}.a^*.x^* =1)}$

Замечание: В данном конкретном случае аналогичный результат может быть получен используя операцию выборки объектов по значениям, примененную к ссылке
$\mathbf{Object(t\ WHERE\ a.x_{ref}< a^*.x^* =1>)}$
Эти варианты неравноценны как по своему смыслу, так и по способу вычисления.
В первом случае мы строим раскрытое по ссылке $\mathbf{a.x_{ref}}$ отношение типа **t** и затем из кортежей, содержащих атрибут $\mathbf{a.x_{ref}.a^*.x^*}$, выбираем OID объектов **.**
$\mathbf{Object(}$
$\mathbf{(t\ JOIN_{an.xref}\ (t^*\ RENAME\ Object(t^*), a^*.x_1^*, \dots, a^*.x_n^*\ AS\ a.x_{ref}, a.x_i.a^*x_1^*, \dots, a.x_i.a^*x_n^*))}$
$\mathbf{WHERE\ a.x_{ref}.a^*.x^* =1\ )}$
Во втором случае мы сначала выбираем из отношения типа кортежи тех объектов, которые отвечают требуемым условиям, и затем используем этот результат при построении раскрытого по ссылке $\mathbf{a.x_{ref}}$ отношения типа **t.**
$\mathbf{Object(}$
$\mathbf{t\ WHERE\ EXIST\ a.x_{ref}\ JOIN\ (\ Object(t^*\ WHERE\ a^*.x^* = 1))}$
$\mathbf{)}$
По нашему мнению, второй вариант является более мощным. Например, трудно найти аналог в стиле первого варианта следующему выражению в стиле второго варианта
$\mathbf{Object\ (t\ EXPAND\ a.x_{ref}< a^*.x^*=1, a^*.x^*=2 >))}$

Необходимо отметить, что операция раскрытия ссылки может применяться как к R-переменной типа **t**, так и к R-переменной компонента типа **t.a**. Выборка **t [a.x_{ref}.a*.x*]** по результату будет эквивалентна выборке **t.a[x_{ref}.a*.x*]**

R-переменные и ссылки.

Рассмотрим введенную ранее групповую операцию **Object(x)**, возвращающую OID всех объектов, данные о которых содержатся в R-переменной x. Например, выражение **Object(t)** вернет OID всех существующих в системе объектов класса **t**.

Пример. Предположим, что существует групповая переменная `someSales` *ссылочного типа* `Sales`**.** *После выполнения операции*

```
someSales := Object(Sales WHERE IsPayed = TRUE);
```

эта переменная будет содержать ссылки на объекты типа `Sales`*, описывающие уже оплаченные продажи.*

Вспомним, что ссылки разыменовываются, то есть любая операция, отличная от операций присваивания и сравнения, выполняется не над ссылкой, а над объектом, на который эта ссылка указывает. Это замечание справедливо и для операции обращения к компоненту объекта. Если существует ссылка **o** на объект класса **t**, то выражение **o.a** описывает обращение к компоненту **a** объекта (или объектов), на который указывает ссылка o. Соответственно, выражение **ref_t.a** можно рассматривать как имя R-переменной, значение которой представляет собой совокупность значений компонента **a** тех объектов типа **t**, ссылки на которые содержатся в переменной **ref_t**. Значение R-переменной **ref_t.a** вычисляется как **t.a JOIN ref_t**.

Отметим явную аналогию, которая существует между именем R-переменной **t.a**, содержащей совокупность значений компонентов **a** всех существующих в системе объектов типа **t**, и именем R-переменной **ref_t.a,** описывающим такую же совокупность для группы объектов, определяемой ссылкой **ref_t** (эта группа, возможно, состоит из одного объекта). Схемы этих переменных полностью совпадают. И в том, и другом случае подразумевается, что речь идет о совокупности значений компонента **a** некоторой группы объектов. Такая аналогия подразумевает, что эти переменные могут использоваться в одних и тех же операциях.

Пример. Уже описанная переменная `someSales` *ссылочного типа* `Sales` *после операции*

```
someSales := Object(someSales.SaleItems WHERE Price > 100);
```

будет содержать ссылки на объекты типа `Sales`*, описывающие уже оплаченные продажи, содержащие строки продаж с ценой больше, чем 100.*

Такая же аналогия существует непосредственно между именем типа **t** и именем ссылки **ref_t** на объекты этого типа. Подобно имени типа, имя ссылки может использоваться как имя R-переменной, содержащей полную информацию (в 1НФ) о тех объектах типа **t**, на которых указывает ссылка **ref_t**. Схемы этих переменных полностью совпадают, а значение R-переменной **ref_t** вычисляется как **t JOIN ref_t**. Соответственно, к этим переменным применимы одни и те же операции.

Пример. После операции

```
someSales := Object(someSales WHERE DateOfAction = #01.04.2005#);
```

будет содержать ссылки на объекты типа `Sales`*, описывающие уже оплаченные продажи, которые содержат строки продаж с ценой больше, чем 100, и сделаны 1-го апреля 2005 года .*

Таким образом, имя ссылки, подобно имени объектного типа, является многозначным именем. Определение ссылочной переменной можно интерпретировать как определение R-переменной, значение которой представляет собой выборку из R-переменной соответствующего типа (в дальнейшем R-переменная ссылки) или компонента типа (в дальнейшем R-переменная компонента ссылки). По большому счету единственное отличие между R-переменной **t** и R-переменной $\mathbf{ref_t}$ заключается в том, что первая определена глобально, а вторая – только там, где определена соответствующая ссылочная переменная.

Общее правило, определяющее существование и именование R-переменных и атрибутов.

Вернемся к примеру с содержащим ссылку объектным типом (см. описание операции раскрытия ссылки). Как мы ранее сказали, результат выборка $\mathbf{t[a.x_{ref}]}$ из R-переменной типа **t** эквивалентен результату выборке $\mathbf{t.a[x_{ref}]}$ из R-переменной компонента **t.a** этого типа. Этот результат представляет собой множество OID объектов типа **t***. Фактически, существование этого множества предопределено путевым выражением $\mathbf{t.a.x_{ref}}$. Таким образом, это выражение можно рассматривать как имя предопределенной ссылочной переменной, хранящей значение унарного отношения, вычисляемое любыми из приведенных способов.

Подобно любым именам ссылочных переменных, такое предопределенное имя можно рассматривать как имя предопределенной R-переменной (см. пред. раздел). Таким образом, определение объектного типа **t**, содержащего компонент **a** с атрибутом $\mathbf{x_{ref}}$, который является ссылкой на объект класса **t***, содержащего компонент **a*** с атрибутом **x*** можно также рассматривать как определение следующих R-переменных
переменной **t** содержащей, среди прочих, скалярный атрибут $\mathbf{a.x_{ref}.a^*.x^*}$ (раскрытие ссылки $\mathbf{a.x_{ref}}$ R-переменной типа **t**)
переменной **t.a** содержащей, среди прочих, скалярный атрибут $\mathbf{x_{ref}.a^*.x^*}$ (раскрытие ссылки $\mathbf{x_{ref}}$ R-переменной компонента типа **t.a**)
переменной **t. a.**$\mathbf{x_{ref}}$ содержащей, среди прочих, скалярный атрибут **a*.x*** (R-переменная ссылки **t. a.**$\mathbf{x_{ref}}$)
переменной **t. a.**$\mathbf{x_{ref}.a^*}$ содержащей, среди прочих, скалярный атрибут **x*** (R-переменная компонента ссылки **t. a.**$\mathbf{x_{ref}.a^*}$)

Рассуждения такого рода применимы к структурам с любым числом вложенных ссылок. Напомним, что мы исходим из того, что объектные типы, образующие эти структуры, соответствуют основному требованию НРМ. Операция раскрытия ссылки позволяет рассматривать в качестве имени предопределенной ссылочной переменной путевое выражение вида $\mathbf{n_1.*.n_n}$ (где $\mathbf{n_i}$ – любые, не обязательные разные, имена, а знак * означает произвольную, возможно пустую, последовательность таких имен) для любого числа вложенных ссылок. Соответствующая R-переменная может содержать ссылочный атрибут, для которого также может быть выполнена вложенная операция раскрытия ссылке.

Таким образом, можно утверждать, что определение сложной ссылочной структуры, в которой корректно путевое выражение $\mathbf{n_1.*.*.n_z}$ можно рассматривать как определение переменной отношения с именем $\mathbf{n_1.*}$, в котором определен скалярный атрибут с

именем ***.n$_z$**. Данное правило является универсальным правилом, определяющим существования и именования R-переменных и их атрибутов в R*O системе. Важно, что имена, имеющие семантику сложных вложенных структур, используются в операциях, определённых в реляционной модели данных.

Глобальные переменные значимых типов.

В связи с тем, что в непроцедурных командах доступа к данным могут использоваться только имена переменных, определенные глобально, НРМ считает полезной и необходимой возможность определять и использовать глобальные переменные значимых типов. Эти переменные могут реализовываться и как хранимые, и как вычисляемые.

Пример. Так, переменная `someSales`*, используемая в примере предыдущего раздела, может быть определена как глобальная следующей непроцедурной командой*

```
CREATE someSales AS SET OF Sales
      REALIZE AS STORED;
```

После этого, имя `someSales` *может использоваться в непроцедурных командах группового доступа к данным.*

Опишем кортеж, содержащий ссылку на объект типа `Article` *и ссылку на объект типа* `Warehouse`

```
DESCRIBE TUPLE Art2Ware
{
  Art Article;
  Ware Warehouse;
}
```

Создадим глобальную переменную, содержащую множество таких кортежей

```
CREATE ArticleOnWarehouse AS SET OF Art2Ware
      CONSTRAIN Art AS GLOBAL KEY
      REALIZE AS STORED;
```

Эта глобальная переменная может содержать информацию, например, о распределении артикулов по складам. Ключ этой переменной, содержащий единственный атрибут `Art`*, указывает, что каждый артикул может храниться на одном единственном складе. Таким образом, переменная* `ArticleOnWarehouse` *содержит информацию о связи типа один-ко-многим между множеством артикулов и множеством складов.*

Замечание. Хранимые и вычисляемые переменные значимых типов могут использоваться для эмуляции таблиц и видов, существующих в современных реляционных СУБД.

Групповые вызовы методов.

Групповые операции с объектами представляют собой, так или иначе, операции с компонентами этих объектов, причем под компонентами подразумеваются и атрибуты, которые содержат значения, и методы, которые возвращают их (говоря о содержащих значения атрибутах, мы имеем в виде исключительно спецификацию – любой атрибут может быть реализован и как хранимый, и как вычисляемый). Из этого, в частности, следует возможность группового вызова методов
expr*.f(…)**, где ***expr – выражение, определяющее группу объектов, например имя типа или имя ссылки.

Конечно, возможна суперпозиция

- традиционных реляционных операций,
- операций выборки объектов по значениям и раскрытия ссылок, применяемые к R-переменным полиморфных типов
- группового вызова методов (которые также являются полиморфными).

Например, следующее выражение

((t EXPAND a.x_{ref}< a*.x* =1>).method())[x]

получает проекцию атрибута результата группового вызова метода для объектов типа **t**, которые ссылаются на объекты, отвечающие определённым условиям. Отметим, что, несмотря на выраженную объектную семантику, данное выражение является полностью реляционным.

Пример. *Реализуем компонент* `SaledItems` *типа* `Brand`*, содержащий информацию о продажах артикулов, принадлежащих данной(*`this`*) торговой марки. Используя операцию раскрытия ссылки* `SaleItems.Art`*, мы получаем доступ к атрибуту* `brandname` *тех объектов типа* `Article`*, ссылка на которые имеется в объектах, описывающих продажи.*

```
ALTER CLASS Brand
  REALIZE SaledItems AS
    SUMMARIZE (Sales EXPAND SaleItems.Art<brandname = this.name>)
    BY Art, ADD Sum(Pieces) AS Pieces;
```

Следующий код представляет собой транзакцию, в которой отгружаются все не отгруженные продажи. Если это возможно для всех продаж, выполненные изменения принимаются. Если не все продажи могут быть отгружены, транзакция откатывается.

```
BEGIN TRANSACTIOM
IF EXIST
  ((SALES <DateOfAction IS NULL>.DoSale(GetTodayDate()) = FALSE)
THEN ROLLBACK
ELSE COMMIT
```

Краткий обзор команд управления R*O-системой.

Типы данных определяются на языке определения данных (DDL). Типы делятся на значимые и объектные. Существует набор базовых скалярных типов. Среди команд, манипулирующих значимыми типами, необходимо отметить команду, позволяющую определить новый кортежный тип.

```
DESCRIBE TUPLE tuplename.
{
      scalar_attribute_definition;
      [scalar_attribute_definition;]
}
```

где `scalar_attribute_definition` – выражение, описывающее атрибут скалярного типа. Должны существовать команды, манипулирующие кортежными типами и позволяющие уничтожать эти типы.

Компонент или переменная типа-множества определяется с использованием конструктора SET OF с указанием используемого скалярного или кортежного типа.

```
rvarname SET OF tuplename CONSTRAIN [local_keys_definition];
```

где `local_keys_definition` перечисляет поля, входящие в необязательный локальный ключ**.**

В команде, создающей новый объектный тип (и соответствующий ссылочный тип), необходимо указывать имя этого типа, перечислить базовые типы, спецификацию компонентов и определить ключи.

```
CREATE CLASS otypename [EXTENDED parenttypename[,parenttypename] ]
{
  value_signature  [CONSTRAIN keys_definition];
  [value_signature [CONSTRAIN keys_definition];]
} [CONSTRAIN keys_definition]
```

Здесь `value_signature` – выражение, описывающее компонент значимого типа. Выражение `keys_definition` определяет тип ключа и перечисляет поля, входящие в ключ**.**

Операция изменения определения типа подразумевает добавление, изменение и удаление спецификаций собственных атрибутов и методов типа (т.е. изменение спецификации),

```
ALTER CLASS otypename
      ADD| DROP|ALTER value_signature[CONSTRAIN keys_definition];
```

…а также изменение реализаций собственных и наследуемых атрибутов и методов типа (в соответствии с существующей спецификацией).

```
ALTER CLASS otypename REALIZE value_signature AS realize_expr;
```

…где `realize_expr` определяет реализацию компонента – является ли он хранимым (`AS STORED`) либо вычисляемым; в последнем случае для компонента должно быть определено вычисляющее выражение (`AS valueexpr`) либо вычисляющая функция (`AS BEGIN …END`)

Операция удаления объектного типа

```
DROP otypename
```

выполняет действия, обратные действиям, выполняемым в процессе добавления, а также удаляет записи о реализации атрибутов и методов этого типа.

Непроцедурные команды, определяющие реализацию компонентов или триггеров, содержат вычисляющие выражения, а также процедурные выражения, описывающие действия, выполняемые триггерами и методами (в свою очередь, в этих процедурных выражениях могут вызываться непроцедурные команды управления системой). Таким образом, процедурные расширения языка управления можно рассматривать как важную часть языка определения данных (DDL).

Подъязык манипулирования данными (DML) должен включать в себя

1) команды, позволяющие создавать и уничтожать объекты заданного объектного типа.
   - `NEW objecttype(constructor_parameters)`
   - `DESTROY objectgroup`
2) команды, позволяющие изменить значение обновляемых атрибутов группы объектов заданного объектного типа
   - операция присваивания
   - `INSERT … INTO objectgroup.a`
   - `UPDATE objectgroup.a`
   - `DELETE FROM objectgroup.a`
3) команды, позволяющие выполнить метод у заданной группы объектов, `EXECUTE objectgroup.methodname(parameters)`
4) выражения, основанные на известных операциях реляционной алгебры, дополненных операцией поиска объекта по значениям и операцией раскрытия ссылки.

Группа объектов `objectgroup`, над которыми выполняется то или иное действие может определяться явным указанием содержащихся в них данных `t< cond1,…>`, ссылками и др. выражениями. Возможен вариант, когда в группу будет входить всего один объект (например, в случае выборки по глобальному ключу):
`EXECUTE t<GlobalKeyField=1>.method(…)`

Команды управления должны включать команды позволяющие создавать и уничтожать глобальные переменные значимых типов
- `CREATE value_signature[CONSTRAIN keys_definition]`
- `DROP global_value_name`

а так же манипулировать значениями этих переменных.
- `INSERT … INTO global_value_name`
- `UPDATE global_value_name`
- `DELETE FROM global_value_name`

Доступ к данным, содержащимся в этих переменных должен осуществ*-+ляться с использованием выражения, основанные на известных операциях реляционной алгебры.

Заканчивая первую часть, отметим ее ключевые моменты. НРМ исходит из известного требования - любая информация в реляционной БД представлена множеством значений отношений. Соответственно, предполагается, что информация о любой сущности предметной области так же должна быть представлена как множество значений отношений (основное требование НРМ).

Вводится система типов, позволяющая выполнить основное требование. Данные представляются в виде множества сложных объектов, при этом состояние любого объекта описывается как множество значений отношений. Отметим, что типы, описывающие объекты, являются инкапсулированными, наследуемыми и полиморфными. Далее показано, что данные, представленные в виде множества таких объектов могут быть представлены также в виде множества значений отношений, определённых на множестве скалярных доменов (двоякое представление данных). При этом, в общем случае, каждому классу соответствует множество переменных отношений (R-переменных), каждая из которых содержит данные обо всех существующих в системе объекта этого класса. Использование сложных (с точки зрения пользователя) имен R-переменных и их атрибутов позволяет сохранить семантику сложных структур для данных, представленных в виде множества значений отношений.

Часть 2. R*O система изнутри. Возможная реализация.

Использование n-арных отношений как единственно возможной структуры данных, является основой формальной реляционной модели данных. Однако эта структура не является достаточно выразительной для того, что бы создать адекватную модель предметной области. Предложенный в первой части подход разрешает это противоречие, позволяя представить данные, описанные как множество идентифицируемых, взаимосвязанных, сложных объектов наследуемых, полиморфных типов, в виде множества отношений.

Во второй части мы хотим показать, что R*O система может быть создана на базе существующих реляционных СУБД. Тем самым, НРМ утверждает, что системы управления реляционными БД могут эволюционно развиваться в направлении, определяемым в первую очередь необходимостью адекватно описывать сложные предметные области и предлагает путь этого развития. В этом утверждении мы исходим из описанного в первой части подхода, который подразумевает *двоякое* представление данных, когда одни и те же данные представлены одновременно и как значения компонентов объектов данных, и как значения R-переменных. В общих словах, предлагаемая далее реализация системы исходит из того, что, поскольку R-переменные есть не что иное, как переменные отношения, то в качестве основы можно использовать систему, в которой такие переменные уже так или иначе реализованы.
Замечание. Рассматривая принципиальную возможность предлагаемой реализации, мы не задаемся вопросами ее производительности и эффективности.

Говоря о существующих реляционных СУБД, мы имеем в виду в первую очередь SQL СУБД. Мы осознаем, что некоторые свойства SQL СУБД расходятся с реляционной моделью данных[M3, 5], однако этот факт с практической точки зрения, по нашему мнению, не важен. Мы исходим из того, что существующие РСУБД ….

- Несмотря на определенные несоответствие с реляционной моделью данных, позволяют, так или иначе, реализовать ее основные положения. Подразумевается, что существует команды, позволяющие манипулировать данными, представленными как набор значений отношений (или, в терминах SQL, таблиц), в том числе определять схему этих отношений (заголовки таблиц), ограничения целостности данных (ключи) и выполнять определенные в реляционной модели данных операции над значениями отношений.
- Являются системами долговременного хранения данных. Подразумевается, что существует команды, которые позволяют создавать и уничтожать таблицы, служащие для долговременного хранения данных, а также манипулировать хранящимися в них данными. В системе существует набор предопределенных таблиц, служащих для хранения метаданных, которые описываю хранящиеся данные.
- Являются программируемыми системами. Подразумевается, что существуют команды, позволяющие определить, сохранить и выполнить последовательность команд. Фактически, речь идет о процедурных расширениях SQL.

Отметим, что некоторые возможности реализуемой R*O системы целиком и полностью определяются свойствами используемой РСУБД; важнейшим из этих свойств, по нашему мнению, является свойство постоянства хранимых данных.

Как было сказано в первой части статьи, данные в реализуемой R*O системе, представлены значениями компонентов объектов и, одновременно, значениями R-переменных. Будем называть реализуемую совокупность объектов и R- переменных

уровнем представления данных. Надо понимать, что уровень представления данных реализуется и, следовательно, является виртуальным. О существовании объектов и R-переменных уровня представления можно говорить лишь постольку, поскольку существует набор команд, которые манипулируют ими (в том числе хранящимися в них значениями), и программа, выполняющая эти команды. Получив команду, эта программа преобразует (транслирует) ее в команду или последовательность команд используемой РСУБД, выполняя которые последняя манипулирует данными, хранящимися в таблицах. Будем называть реализуемый РСУБД набор переменных отношений (т.е. таблиц) уровнем хранения данных. Отметим, что данные на уровне хранения представляют собой не что иное, как реляционную базу данных.

Замечание. Слово "реализация" в данном изложении становиться явно перегруженным. Во-первых, мы используем это слово, когда говорим о реализации типов R*O системы. Во-вторых, мы используем его, когда говорим о реализации системы на базе существующих РСУБД. Естественно, это не одно и тоже. Говоря образно, понятие "реализация типов" принадлежит модели данных и, следовательно, относится только к уровню представления данных. Реализация же системы определяет взаимосвязи между уровнем представления и уровнем хранения.

Таким образом, важнейшей частью R*O системы является транслятор, выполняющий преобразование команд R*O системы в команды используемой РСУБД.

Замечание. Демонстрируя возможность трансляции команд уровня представления в команды РСУБД, мы, так или иначе, подразумеваем возможность записи результатов трансляции, используя SQL. Однако, с практической точки зрения, это не является обязательным. Подразумевается, что транслятор может генерировать некое внутреннее, используемое в конкретной РСУБД представление команд или последовательностей команд, минуя, таким образом, создание SQL-скрипта.

Понятно, что система должна контролировать структуру объектов на всем протяжении их жизни таким образом, что эта структура не противоречила описанию объекта. Выполняя команду, манипулирующую объектом (т.е. транслируя ее), система должна обладать информацией о структуре этого объекта. Исходя из этого,

1) в системе необходимо хранить описание структуры объектных типов на всем протяжении их существования (возможная схема используемых для этого таблиц каталога будет предложена далее).
2) необходимо, что бы для любого объекта существовала возможность определить тип этого объекта. Реализация выполняющих это требование операций **o IS t** и **o OF t** будет рассмотрена далее.

Повторим, что основная идея, на которой основывается предлагаемая реализация R*O-системы, заключается в том, что R-переменные (уровень представления данных) являются переменными отношений. Важно, однако, понимать, что R-переменные уровня представления и переменные отношений уровня хранения не эквивалентны. Детальная организация данных на уровне хранения скрыта от пользователя. В частности, значение R-переменной компонента типа **t.a** (уровень представления) в общем случае реализуется системой как объединение нескольких значений, из которых только одно является значением базовой переменной $_{st}$**t.a'** (эта переменная используется для хранения значений тех компонентов объектов, которые на уровне представления реализованы как хранимые).

Замечание. Схема базовой переменной уровня хранения $_{st}$**t.a'** абсолютно совпадает со схемой R-переменной уровня представления, и представляет собой схему соответствующего компонента, к которой добавлен атрибут OID.

Остальные значения можно рассматривать как промежуточный результат выражения **f'(BASE_VALUE)**, соответствующего тем компонентам объектов, которые на уровне представления реализованы как вычисляемые

t.a = $_{st}$t.a' UNION $_{st}$f'(BASE_VALUE) UNION $_{st}$f'$_2$(BASE_VALUE) UNION …

здесь **$_{st}$f'(BASE_VALUE)** – выражение, вычисляющее некое значение, используя как параметр (в общем случае) текущее состояние(значение) БД на уровне хранения . Как мы уже сказали, эти вычисления должны выполняться системой неявно для пользователя на основании информации о наследовании типов и о реализации компонентов этих типов.

Трансляция выражений.

Интересным является вопрос о том, откуда берется вычисляющее выражение **$_{st}$f'**, что оно собой представляет. Не вызывает сомнения, что выражение **$_{st}$f'** определяется существующим на уровне представления вычисляющим выражением **f** – то есть, речь идет о трансляции определенного на уровне представления выражения **f** в выполняемое на уровне хранения выражение **$_{st}$f'**.
Замечание. Значение операнда **$_{st}$t.a'** можно рассматривать как результат простейшей операции, возвращающей значение переменной **$_{st}$t.a'**. Эту операция определяется задаваемым на уровне представления выражением **AS STORED**, указывающим, что компонент реализуется как хранимый.

В чем же заключается трансляция? Обратим внимание на то, что выражение **f** представляет собой последовательность операций, возможно, использующих в качестве операндов компоненты некого, вполне *конкретного* объекта типа **t**. Выражение же **$_{st}$f'**, исходя из свойств используемой РСУБД, представляет собой последовательность операций, определенных в реляционной модели данных и использующих в качестве операндов переменные отношений уровня хранения, каждая из которых содержат данные о *множестве объектов*.

Обратимся к традиционным ОО-языкам программирования. Описывая метод, манипулирующий атрибутами объекта, мы подразумеваем, что этот метод будет вызван у вполне конкретного объекта. В C++ , например, такой конкретный объект в теле метода определяется необязательным для использования ключевым словом **this** (по сути, **this** есть не что иное как ссылка на данный *конкретный* объект). Оттранслировав этот метод, мы получим процедуру на машинном языке. Обязательным параметром этой процедуры будет значение, представляющее собой адрес участка машинной памяти - подразумевается, что в этом участке памяти хранятся данные некоторого конкретного объекта из всего *множества* объектов, данные о которых хранятся в памяти.

В самом первом приближении R*O транслятор работает по схожему принципу, что подразумевает, что, передав ему на вход метод объекта, на выходе мы получим процедуру, получающую в качестве обязательного аргумента значение OID. При этом, на наш взгляд, значительный интерес представляет тот факт, что используемая РСУБД реализует определяемые реляционной моделью *групповые* операции. Основываясь на этом, можно показать, что определенное на уровне представление выражение **f** может быть транслировано в такое выражение **$_{st}$f'** уровня хранения, что однократное (никаких итераторов!) исполнение последнего приведет к таким изменениям в системе, как будто исходное выражение **f** было выполнено для каждого из существующих в системе объектов типа **t**.

Эта идея может быть продемонстрирована на следующем простом примере. Предположим, что в теле метода **method** типа **t** происходит присваивание значения

одного хранимого компонента $\mathbf{a_1}$ другому хранимому компоненту $\mathbf{a_2}$ (предполагается, что это компоненты одного и того же типа)

ALTER CLASS t REALIZE method(…)… AS
BEGIN
…;
this.a_2 := this.a_1;
…;
END

Рассмотрим данную операцию в терминах R-переменных **t.a_1** и **t.a_2**. После ее выполнения переменная **t.a_2** будет содержать кортежи, значение которых в точности равны результату выборки из отношения **t.a_1** по значению ссылки **this**. Исходя из этого, соответствующая этому методу процедура **$_{st}$method'** уровня хранения должна выглядеть приблизительно следующим образом.

CREATE PROCEDURE $_{st}$method'(this' …, …)…
BEGIN
…;
INSERT INTO $_{st}$t.a'$_2$ VALUE (SELECT * FROM $_{st}$t.a'$_1$ JOIN this');
…;
END

Предполагается, что значение параметра **this'**, служащего для передачи в процедуру OID того объекта, в котором выполняется указанное действие, представляет собой унарное отношение с единственным атрибутом **OID**.

В данном случае не вызывает сомнения, что параметр **this'** может содержать OID множества объектов - в этом случае описываемое действие будет *одновременно* выполнено для всех этих объектов. Именно этот факт делает возможным групповой вызов метода. Например, в ходе вызова

t<cond>.method(…)

описываемое действие будет выполнено для всех объектов типа **t**, для которых выполняется условие **cond**.

Данный случай является наиболее простым. Рассмотрим принципы трансляции в общем виде.

Утверждения о транслируемости.

НРМ утверждает, что любая заданная на уровне представления реляционная операция **f** (где **f** - суперпозиция теоретико-множественных и специальных операторов реляционной алгебры [3, 4]) над компонентами **a** объекта **o** типа **t**, может быть преобразована к операции **$_{st}$f**, применив которую к значениям базовых переменных уровня хранения данных, мы получим значение отношения, представляющее собой объединение результатов применения операции **f** ко всем существующим в системе объектам данного типа (утверждение о R-транслируемости).

Для любой операции f, res = f(o.a_1 ,…, o.a_n), существует такая операция $_{st}$f, res' = $_{st}$f'($_{st}$t.a'$_1$, …, $_{st}$t.a'$_n$), что res = (res' WHERE OID = o)[!OID]

Сначала докажем, что для любой операции **f**, **res = f(o.a_1, …, o.a_n)**, существует такая операция **f'**, **res' = f'(t.a_1, …,t.a_n)**, что **res = (res' WHERE OID = o)[!OID]**. Другими словами, покажем, что операция **f** может быть преобразована в операцию **f'**, использующую в качестве операндов значения R-переменных компонентов типа **t.a**

уровня представления данных (операндами операции $_{st}$**f'** являются значения базовых переменных $_{st}$**t.a'** уровня хранения).

Вспомним, что значение **o.a** рассчитывается как **(t.a WHERE OID = o) [!OID]**. Можно показать, что для следующих примитивных операций верно, что

1) объединение (операнды совместимы по схеме) **o.a$_i$ UNION o.a$_j$**
   эквивалентно **((t.a$_i$ UNION t.a$_i$) WHERE OID = o) [!OID]**
2) вычитание (операнды совместимы по схеме) **o.a$_i$ MINUS o.a$_j$**
   эквивалентно **((t.a$_i$ MINUS t.a$_j$) WHERE OID = o) [!OID]**
3) декартово произведения **o.a$_i$ TIMES o.a$_j$**
   эквивалентно **((t.a$_i$ JOIN$_{OID}$ t.a$_j$) WHERE OID = o) [!OID]**
   (здесь **JOIN$_{OID}$** есть соединение по атрибуту **OID**)
4) выборка **o.a$_i$ WHERE *condition***
   эквивалентна **((t.a$_i$ WHERE *condition*) WHERE OID = o) [!OID]**
   (здесь ***condition*** означает некое условие**)**
5) проекция **o.a$_i$[r$_{a1}$, …, r$_{an}$]**
   эквивалентна **((t.a$_i$[r$_{a1}$, …, r$_{an}$]) WHERE OID = o) [!OID]**

Таким образом, доказываемое утверждение верно в том случае, если **f** является одной из перечисленных примитивных операций. Однако, поскольку реляционная алгебра является замкнутой (т.е. результат одной операции может являться операндом другой), и поскольку любой оператор можно представить как сложную суперпозицию примитивных операторов, то индуктивно можно показать, что доказываемое утверждение верно для любых реляционных операций.

Теперь покажем, что операция **f'**, использующая в качестве операндов значения R-переменных компонентов типа **t.a** уровня представления данных, может быть преобразована в операцию $_{st}$**f'**,операндами которой являются значения базовых переменных(таблиц) $_{st}$**t.a'** уровня хранения. Мы основываемся на том, что значение R-переменой реализуется системой как объединение нескольких значений, из которых одно является значением базовой переменной $_{st}$**t.a'** уровня хранения, используемое для хранения значений тех компонентов объектов, которые на уровне представления реализованы как хранимые. Остальные значения можно рассматривать как промежуточный результат выражения $_{st}$**f'(BASE_VALUE)**, соответствующего тем компонентам объектов, которые на уровне представления реализованы как вычисляемые

t.a = $_{st}$t.a' UNION $_{st}$f'$_1$(…) UNION …

здесь $_{st}$**f'$_i$(…)** есть не что иное, как R-трансляция определенного на уровне представления данных операции **f$_i$**.Если предположить, что эта операция использует в качестве операндов значения компонентов объекта **f$_i$(…, o.a, …)**, то можно утверждать, что, в соответствии с первой частью доказательства,

t.a = $_{st}$t.a' UNION f'$_1$(t.a$_1$, …, t.a$_n$) UNION …

Исходя из этого, опять же индуктивно можно показать, что значение любой R-переменной **t.a** действительно может быть рассчитано на основании значений базовых переменных отношений уровня хранения данных.

t.a = $_{st}$t.a' UNION $_{st}$f'$_1$($_{st}$t.a'$_1$ UNION f'$_{11}$($_{st}$t.a'$_{11}$, …) UNION …) UNION …

Следствия из утверждения о транслируемости.

Предварительное замечание 1. Как мы уже сказали, изменение состояния объектов заключается в изменении значений их компонентов. Изменение значения компонента подразумевает присвоение этому компоненту нового значения (сами значения неизменны). Значение компонента **a** объекта **o** изменится вследствие операции присваивания
o.a := rval
где **rval** является значением отношения, схема которого совпадает со схемой, заданной для компонента **a**.

Отметим, что традиционные для существующих реляционных СУБД операции, изменяющие значения переменных отношения (вставки и удаления кортежей, а также операция обновления, изменяющая значения атрибутов существующих в переменной кортежей) могут быть записаны, используя операцию присваивания:
- операция вставки **INSERT o.a_i VALUE(rval)** может быть представлена как **o.a_i := o.a_i UNION rval**,
- операция удаления кортежей **DEL o.a_i WHERE (condition)** может быть представлена как **o.a_i := o.a_i WHERE (NOT condition)**,
- операция, изменяющая значения атрибутов **UPDATE o.a_i SET x_j=s WHERE(condition)** может быть представлена как **o.a_i := (o.a_i WHERE (NOT condition)) UNION (o.a_i WHERE (condition)[…, x_{j-1}, s, x_{j+1},…]).**

И наоборот, оператор присваивания
relvar:=relvalue
может быть реализован с использованием традиционных для существующих реляционных СУБД операций
DELETE FROM relvar;
INSERT INTO relvar VALUE relvalue;

Предварительное замечание 2. Как мы сказали ранее, в методы классов возможны локальными переменными значимых типов. Рассмотрим возможную реализацию этих переменных в используемой РСУБД.

Локальные переменные служат для хранения значений, используемых в методе и/или возникающих в процессе выполнения этого метода. Время жизни локальной переменной ограничено временем выполнения метода. В том случае, когда действия, описываемые неким методом **method()**, производятся одновременно для множества объектов (никаких итераторов), можно говорить о множестве таких значений и, соответственно, о множестве переменных **localvar**, служащих для их хранения, причем каждому объекту из этого множества, соответствует одна переменная, и, в каждый момент времени, одно значение этой переменной.

Исходя из этого можно сказать, что локальной переменной **localvar** со схемой **(…, x_k:D_k, …)** на уровне хранения должна соответствовать одна-единственная переменная отношения **$_{st}$localvat'** со схемой **(OID: DOID , …, x_k:D_k,…)**. Таким образом, реализация локальных переменных на уровне хранения не отличается от таковой для хранимых компонентов. Единственное различие между ними заключается в том, что соответствующая локальной переменной переменная уровня хранения **$_{st}$localvar'**, является временной переменной, время жизни которой ограничено временем выполнения R-транслированной процедуры **$_{st}$method'()**. Это позволяет утверждать, что последующие следствия верны как для компонентов, так и для локальных переменных методов.

Следствие 1 (о R-транслируемости операции присваивания). Предположим, что значение компонента $\mathbf{a_k}$ объекта типа **t** заданного ссылкой **o** представляет результат операции **f** над другими компонентами $\mathbf{a_i}$ этого объекта. Другим словами речь идет об операторе присваивания, изменяющем состояние компонента $\mathbf{a_k}$
$\mathbf{o.a_k := f(o.a_1,\dots , o.a_n, \dots)}$. (1)
Тогда, следуя из утверждения о R-транслируемости операций, такому оператору соответствует оператор
$\mathbf{t.a_k := f'(\dots , t.a_i, \dots)}$ (1')
изменяющий значение соответствующей компоненту $\mathbf{a_k}$ R-переменной $\mathbf{t.a_k}$ так, как будто оператор (1) был выполнен для каждого существующего в системе объекта типа **t**. Как мы сказали ранее, в существующих РСУБД такой оператор может быть реализован связкой **DELETE …, INSERT …**

Следствие 2 (о R-транслируемости последовательности операций присваивания).

Предположим, что для объекта типа **t** заданного ссылкой **o** определена простая последовательность операторов вида (1)
$\mathbf{o.a_j := f_1(\dots , o.a_i, \dots);}$
$\mathbf{o.a_k := f_2(\dots , o.a_j, \dots);}$.
$\mathbf{o.a_l := f_3(\dots , o.a_k, \dots);}$

В R- представлении такой последовательности операторов будет соответствовать абсолютно идентичная последовательность R-транслированных операторов вида (1')
$\mathbf{t.a_j := f_1(\dots , t.a_i, \dots);}$.
$\mathbf{t.a_k := f_2(\dots , t.a_j, \dots);}$.
$\mathbf{t.a_l := f_3(\dots , t.a_k, \dots);}$
изменяющая состояние системы таким образом, как будто последовательность (2) была выполнена для каждого существующего в системе объекта типа **t**.

НРМ утверждает, что любая последовательность операций над компонентами **a** объекта типа **t**, заданного ссылкой **o,** может быть R-транслирована в такую последовательность операций над соответствующими R-переменными **t.a**, что ее единственное исполнение изменит состояние системы так, как будто исходная последовательность была выполнена для каждого существующего в системе объекта этого типа. В качестве примера, подтверждающего данное утверждение, рассмотрим R-трансляцию условного оператора и оператора цикла.

Отметим, что любая операция $\mathbf{f_i}$ возвращает значение $\mathbf{p_i}$, которое, в случае его дальнейшего использования, должно быть сохранено в соответствующей локальной переменной (возможно временной и/или не определяемой явно). Предположим, что операция возвращает значение булевского типа, для хранения которого используется соответствующая переменная **b**. На уровне хранения этой переменной будет соответствовать переменная отношения $\mathbf{_{st}t.b'}$ со схемой **(OID, b)**.

Условный оператор. Предположим, что для объекта типа **t** заданного ссылкой **o** определена последовательность операций

$\mathbf{IF\ f_1(\dots , o.a_i, \dots)\ THEN\ o.a_k := f_2(\dots , o.a_j, \dots);}$

Предполагается, что результатом выполнения $\mathbf{f_1}$ является значение типа BOOLEAN. Данная последовательность может быть переписана как.

b := f_1(… , o.a_i, …);
IF b THEN o.a_k := f_2(… , o.a_j, …);

что может быть R-транслировано, например, в следующую последовательность действий над R-переменными

t.b := f'_1(… , t.a_i, …);
t.a_k := (f'_2(… , t.a_j, …) $JOIN_{OID}$ ($_{st}$t.b' WHERE b = TRUE)) UNION
(t.a_k $JOIN_{OID}$ ($_{st}$t.b' WHERE b = FALSE));

Оператор цикла. Предположим, что для объекта типа **t** заданного ссылкой **o** определена последовательность операций

DO …
o.a_k := f_2(… , o.a_j, …);
…
WHILE f_1(… , o.a_i, …);

Так же, к и в предыдущем случае, результатом выполнения f_1 является значение типа BOOLEAN. Данная последовательность может быть так же переписана как.

DO …
o.a_k := f_2(… , o.a_j, …);
…
b := f_1(… , o.a_i, …);
WHILE b;

что может быть R-транслировано, например, в следующую последовательность действий над R-переменными

DO …
t.a_k := (f'_2(… , p.a_j, …) $JOIN_{OID}$ ($_{st}$t.b' WHERE b = TRUE)) UNION
(t.a_k $JOIN_{OID}$ (t.b WHERE b = FALSE));
…
$_{st}$t.b' := f'_1(… , t.a_i, …);
WHILE EXIST $_{st}$t.b' WHERE b = TRUE;

Таким образом, возможен групповой вызов операций, представляющих собой последовательность действий над компонентами объекта типа **t**, для всех объектов этого типа, например.

```
EXECUTE t.somemethod();
```

Подразумевается, что такое выражение должно быть транслировано в единственный вызов R-транслированной последовательности `somemethod'`.

Следствие 3 (о R-транслируемости последовательности операций присваивания для явно определенной группы объектов).

Можно показать, что все утверждения о транслируемости реляционных операций и последовательности таких операций могут быть применены и к явно заданной группе

объектов. Предполагается, что группа объектов задается групповой ссылкой **g**. В самом деле, любую R-транслированную операцию

$t.a_j$:= f'(… , $t.a_i$, …)

можно рассматривать как такой частный случай операции

$t.a_j$:= (f'(… , $t.a_i$, …) JOIN g) UNION ($t.a_j$ JOIN (objects(t) MINUS g)),

когда **g** содержит ссылки на все существующие в системе объекты типа **t**, т.е. **g = objects('t')**. Отметим, что если **g** определяет несобственное подмножество существующих в системе объектов типа **t**, то однократное выполнение последней операции изменит значение R-переменной **$t.a_j$** так, как будто исходная операция **f(… , $o.a_i$, …)** была выполнена для всех объектов, которые входят в это подмножество.

Исходя из этого, можно утверждать, что любая последовательность операций над компонентами **a** объекта типа **t** заданного ссылкой **o** , может быть R-транслирована в последовательность операций над соответствующими R-переменными **t.a**, которая изменит состояние системы таким образом, как будто исходная последовательность была выполнена для каждого объекта из множества, определяемого групповой ссылкой **g**. Как уже было сказано, это множество представляет собой собственное подмножество всех существующих в системе объектов типа **t**.

Таким образом, возможен групповой вызов операций, представляющих собой последовательность действий над компонентами объекта типа **t**, для множества объектов, заданного групповой ссылкой **g**, например

```
EXECUTE g.somemethod();
```

или для группы, определяемой, например, ссылкой `xref` из выборки объектов типа **t** по значению

```
EXECUTE t<cond1,...>.xref.somemethod();.
```

Замечание (о триггерах). Отметим, что определённый на уровне представления для компонента a_i объектов типа t триггер можно рассматривать как последовательность операторов вида
$o.a_k$:= f(… , $o.a_i$, …)
выполняемых в ответ на изменение компонента **a_i** объекта, заданного ссылкой **o**. Из этого следует, что к триггерам также можно применять утверждения о транслируемости. В самом деле, предыдущий оператор может быть R-транслирован в оператор
$t.a_k$:= (f'(… , $t.a_i$, …) JOIN g) UNION ($t.a_k$ JOIN (objects(t) MINUS g))
где **g** определяет существования явно заданной группы объектов (возможно, содержащей один единственный объект). Выполнение R-транслированной последовательности таких операторов изменит состояние системы таким образом, как будто исходный триггер был выполнен для каждого объекта из группы **g**.

Каталог.

Описывая принципы организации каталога, НРМ предполагает, что используемая РСУБД уже содержит каталог, позволяющий описать схему своей "табличной" памяти (то есть схему реляционной БД). Эта схема включает описание отношений уровня хранения, схемы которых определяются и однозначно соответствует схемам компонентов, которые определяются и описываются на на уровне представлении. Поэтому мы не будем останавливаться на части каталога, содержащего информацию о структуре этих отношений, ограничившись утверждением о том, что каждой такой

переменной следует поставить в соответствие некий идентификатор $_{\mathbf{STORED}}\mathbf{R}$, определенный на домене идентификаторов отношений $_{\mathbf{STORED}}\mathbf{RVarIDs}$ (здесь и далее идентификаторами могут являться, например, уникальные имена).

В самых общих чертах, каталог, описывающий структуру типов, должны включать нижеперечисленные таблицы (подчеркнуты поля, входящие в первичный ключ таблицы).

1) Таблица **valTYPES(vT:vTypes …)** , перечисляет существующие в системе значимых типах. (**vTypes** - домен идентификаторов значимых типов, например имена типов). Детальное описание этих типов может содержаться в других (не описываемых здесь) таблицах.
2) Таблица объектных типов **objTYPES (oT:oTypes …)** перечисляет все существующие в системе объектные типы (**oTypes** - домен идентификаторов объектных типов, например имена типов).
3) Таблица **IS_T(oT:oTypes, IS_oT: oTypes …)** полностью описывает существующее между объектными типами отношение наследования. Для каждого типа **t** в таблице существует запись, где **oT = IS oT** . Также, для каждого типа **t** в таблице перечисляются все его прямые и непрямые базовые типы **IS_oT**.
4) Таблица **SPEC(A:Comps, oT:oTypes, vT: vTypes, signature …)** перечисляет собственные (т.е. неунаследованные) компоненты (поле **A**) объектных типов (поле **oT**), указывает значимый тип этих компонентов (поле **vT**) и иную информацию определяющую их сигнатуру (**CompID -** домен идентификаторов компонентов, например, имена компонентов). Фактически, эта таблица содержит полную информацию о спецификации собственных компонентов объектных типов.
5) Таблица **REAL(A:Comps, OF_oT:oTypes, isSTORED:BOOLEAN, RealExpr…)** содержит информацию о реализации компонентов (поле **A**) объектных типов (поле **OF_oT**) и, в частности информацию о том, является ли атрибут хранимым (поле **isSTORED = TRUE**) или вычисляемым (поле **isSTORED = FALSE**). В первом случае поле **RealExpr** содержит имя используемой для хранения переменной уровня хранения $_{\mathbf{STORED}}\mathbf{R}$, во втором – транслированные вычисляющее выражение или функцию. (В таблице **REAL** может содержаться информация только о собственных и переопределенных в процессе наследования компонентах типа. Для компонентов, которые не переопределяются в процессе наследования, должен существовать механизм, возвращающий информацию о реализации этого компонента в ближайшем базовом типе, где эта реализация была явно задана.)

Естественно, что все перечисленные таблицы могут содержать другие поля, содержащие информацию о спецификации и о реализации объектных типов.

Операции уровня представления данных, манипулирующие схемой данных, транслируются в операции, манипулирующие таблицами каталога.

Команда, создающая новый объектный тип

```
CREATE CLASS otypename [EXTENDED parenttypename[,parenttypename] ]
{
      signature
      [;signature]
}
```

должна выполняться следующим образом.

1) В таблицу **objTYPES** добавляется единственная запись о новом типе, содержащая имя (**otypename**) этого типа.
2) В таблицу **SPEC** добавляются записи, содержащие спецификации сигнатур компонентов и методов типа по числу определенных в типе собственных компонентов и методов.
3) В таблицу **ISt** добавляется не менее одной записи. Одна запись добавляется для типа, у которого отсутствует базовый тип (для простоты здесь мы опускаем обязательное наследование от фиктивного типа Object). Для типов, наследующих уже существующие в системе типы (**EXTENDED parentclassname**), в таблице **ISt** необходимо также перечислить все (прямые и непрямые) базовые типы. Заметим, что информацию о непрямых базовых типах можно получить из содержащихся в этой же таблице **ISt** записей, описывающих прямые базовые типы.

Операция изменения типа подразумевает добавление, изменение и удаление *спецификаций* собственных атрибутов и методов типа (т.е. изменение спецификации),

```
ALTER CLASS otypename ADD| DROP|ALTER signature_name;
```

а также добавление, изменение и удаление *реализаций* собственных и наследуемых атрибутов и методов типа (в соответствии с существующей спецификацией).

```
ALTER CLASS otypename REALIZE signature AS realize_expr;
```

Указанные операции реализуются в виде добавления, изменения и удаления записей в таблицах **SPEC** и **REAL** соответственно.

Операция удаления объектного типа

```
DROP otypename;
```

выполняет действия, обратные действиям, выполняемым в процессе добавления, а также удаляет записи о реализации атрибутов и методов этого типа.

Таблица идентификаторов

Таблица идентификаторов **OIDS(OID: tOID: OF_oT:oTypeIDs)** перечисляет уникальные идентификаторы существующих в системе объектов и ставит в соответствие каждому идентификатору объекта (поле **OID**) существующий в каталоге идентификатор (поле **OF_oT**) объектного типа этого объекта.

Заметим, что имеющееся в таблицах уровня хранения поле **OID**, а также любые поля, определяющие существование связи по ссылке, должны быть объявлены как внешний ключ, ссылающийся на поле **OID** таблицы идентификаторов. Это позволяет
1) контролировать целостность связей "по ссылкам", используя для этого существующие в используемой реляционной БД механизмы контроля ссылочной целостности.
2) утверждать, что в системе отсутствуют записи таблиц данных, для которых не определен объектный идентификатор (т.е. любая запись таблиц данных действительно описывает какой-либо объект). Отметим, в существующих РСУБД, реализующих каскадное удаление данных, удаление из таблицы идентификаторов записи, содержащей OID некого объекта, вызовет удаление связанных с ней записей, содержащие данные этого объекта.

В свою очередь, поле **oT** должно быть объявлено как внешний ключ, связанный с полем **oT** таблицы каталога **objTYPES**. Это гарантирует, что для любого объекта в системе определен тип этого объекта. Существование такой связи позволяет также использовать существующие в используемой РСУБД механизмы, позволяющие при выполнении команды **DROP otypename** удалять все существующие в системе объекты этого типа.

Описанный ранее оператор **o OF t** может быть реализован как

EXIST OIDS (WHERE OF_oT = t AND OID = o).

Для реализации оператора **o IS t** необходимо использовать информацию о наследовании типов (см. описание таблицы каталога **IS_T**)

EXIST (OIDS JOIN$_{OF_oT = oT}$ IS_T(WHERE OID = o AND IS_oT = t))

Реализации команд управления.

Совокупность команд, служащих для управления R*O-системой, представляет собой непроцедурный язык высокого уровня, который необходимо рассматривать как основной (возможно единственный) способ доступа к данным, хранящимся в системе. Входящие в этот язык команды можно разделить на две группы, составляющие соответственно подъязык определения данных (DDL) и подъязык манипулирования данными (DML). Предполагается, что команды этого языка транслируются в команды используемой РСУБД.

Действия, которые должны выполнять основные команды DDL, уже освещены нами как операции над каталогом системы. Как мы уже сказали, непроцедурные команды, определяющие реализацию компонентов или триггеров, содержат вычисляющие выражения. Указанные процедурные выражения транслируются в процедуры используемой РСУБД, которые получают в качестве обязательного параметра объектные идентификаторы, определяющие объект или группу объектов. Эти процедуры сохраняются в таблице каталога, откуда они могут быть загружены для выполнения. Таким образом, при обработке процедурных расширений DDL транслятор выступает в роли компилятора, преобразующего R*O код в код используемой РСУБД.

Команды подъязыка манипулирования данными (DML) служат для создания уничтожения объектов данных, а также для управления состоянием этих объектов и для получения информации о данных, хранящихся в системе. Отметим, что, обрабатывая и выполняя команды DML, транслятор выступает в роли уже интерпретатора.

Команда `NEW t(constructor_parameters)` служит для создания новых объектов типа **t**. Получив эту команду, система генерирует новый OID и заносит его в таблицу идентификатор **OIDS** вместе с идентификатором типа **t**. Затем, исходя из содержащегося в каталоге типов описания структуры объекта, система добавляет к базовым отношениям уровня хранения кортежи, предназначенные для хранимых компонентов нового объекта. При этом атрибут **OID** этих кортежей инициализируется объектным идентификатором создаваемого объекта. Далее, при необходимости, вызывается конструктор.

Получив команду `DESTROY objectgroup`, система должна выполнить действия, обратные действиям, выполняемым при обработке команды `NEW`. Повторим, что система позволяет контролировать целостность ссылок (см. гл. Таблица идентификаторов) – в

частности, невозможно удалить объект, если в других частях системы существуют ссылки на него. .

Состояние объектов может быть изменено путем непосредственного изменения значений компонентов этих объектов.

```
INSERT … INTO objectgroup.a;
UPDATE objectgroup.a;
DELETE FROM objectgroup.a;
```

или путем вызова методов,

```
EXECUTE objectgroup.methodname(parameters);
```

Для получения данных, хранящихся в системе, используются применяемые к R-переменным команды группового доступа к данным, основанные на

- известных операциях реляционной алгебры,
- операциях выборки объектов по значениям и раскрытия ссылок
- групповом вызове методов
- суперпозицией всего вышеперечисленного

Трансляция команд, изменяющих состояние объекта, определяется утверждением о трансляции и следствиями из него.

Заключение.

"НадРеляционный Манифест" подтверждает важнейшие положения своих предшественников. Подобно "Манифесту систем объектно-ориентированных баз данных", он поддерживает идею долговременно хранимых сложных объектов. Подобно "Манифесту систем баз данных третьего поколения", он предполагает возможность использования и развития существующих систем хранения данных. Наконец, подобно "Третьему Манифесту" он стремится сохранить чистоту идей реляционной модели данных.

Предлагаемый HPM подход может служить основой для создания системы, которую можно рассматривать в первую очередь как систему позволяющую создать адекватную, активную и долговременно существующую модель предметной области, управляемую пользователем и предоставляющую пользователю данные о своем состоянии.

Автор надеется, что предложенный работа может оказаться полезной специалистам и программистам, работающим на стыке областей, касающихся систем хранения данных и моделирования данных. Конечно, в ограниченных размерами статьи рамках невозможно раскрыть тему более подробно – некоторые вопросы освещены крайне схематично, некоторые не освещены вообще. Не вызывает сомнения и то, что многие вопросы, касающиеся предложенного подхода, еще даже не поставлены. Однако тот факт, что предложенный подход опирается на имеющую формальное математическое обоснование реляционную модель данных, позволяет надеяться, что на эти вопросы может быть получен однозначный ответ.

Используемая литература:

M1. М. Аткинсон и др. “Манифест систем объектно-ориентированных баз данных”, *СУБД,* No. 4/1995, http:// www. osp. ru/ dbms/1995/04/23.htm

M2. Стоунбрейкер М. и др. “Системы баз данных третьего поколения: Манифест”, *СУБД,* No. 2/1996, http://www. osp. ru/ dbms/1995/02/23. htm

M3. Х. Дарвин, К. Дейт. “Третий манифест”, *СУБД,* No. 1/1996, http:// www. osp. ru/ dbms/1996/01/23. htm

1. С. Д. Кузнецов, Три манифеста баз данных: ретроспектива и перспективы http://www.citforum.ru/database/articles/manifests/
2. Codd E.F. “A relational model for large shared data banks”. Commun. ACM 13, 6, June, 1970, 377-387X
3. К. Дж. Дейт. Введение в базы данных. Изд. 6-е. Киев, Диалектика, 1998.
4. С. Д. Кузнецов, СУБД и файловые системы, 2001, Майор
5. С. Д. Кузнецов, Дубликаты, неопределенные значения, первичные и возможные ключи и другие экзотические прелести языка SQL. http://www.citforum.ru/database/articles/art_5.shtml
6. А. В. Замулин. Системы программирования баз данных и знаний, 1990, Наука, (Новосибирск)