Станишич Предраг

Метод трансляции реляционных SQL-запросов в эквивалентные запросы к трансформированной объектно-ориентированной базе данных.

1. Введение

В настоящее время актуальной задачей является трансформация реляционной базы данных в объектно-ориентированную. Эта задача возникает при использовании гетерогенных баз данных и при переходе от реляционных систем управления базами данных к объектно-ориентированным. Первым шагом в процессе трансформации реляционной базы данных в объектно-ориентированную является процесс трансформации схемы (см.[7]). В этом шаге в исходной реляционной базе данных идентифицируются объектно-ориентированные концепты и создается схема объектно-ориентированной базы данных. В данной статье рассматривается процесс трансляции запросов, который основан на процессе трансформации схемы и предлагается алгоритм для выполнения этой задачи.

В качестве целевого объектно-ориентированного языка запросов, принимаем язык описанный в п.2 (см.[5]), так как он достаточно хорошо использует концепт выражений пути, который является важным преимуществом объектно-ориентированной модели. В данной статье реляционные SQL-запросы без выражений пути транслируются в запросы объектно-ориентированной системы, которые содержат выражения пути.

Основной проблемой при трансляции запросов является тот факт, что атрибуты классов, которые создаются в процессе трансформации схемы, могут иметь домены, отличающиеся от доменов реляционной схемы. Это касается атрибутов, которые являются объектными идентификаторами (OID) экземпляров классов. С помощью этих атрибутов представляются объектно-ориентированные концепты наследования и ссылок (сложных атрибутов). Нижеследующие примеры демонстрируют возможности появления в полученном классе OID атрибутов и иллюстрируют некоторые аспекты процесса трансформации схемы.

Пример 1: Пусть даны отношения (подчеркнуты возможные ключи): *student(id, name)* и *grad_student(gsid, thesis)*. Домен атрибута *id* и *gsid* есть *integer*, а домен остальных атрибутов есть *string*. В отношении *grad_student* атрибут *gsid* является внешним ключом, указывающим на возможный ключ *id* отношения *student*. Трансформированная схема здесь такова:

```
class student     class grad_student
    id : integer;          inherits student;
    name : string;         thesis : string;
end;                   end;
```

В этом примере создается отношение наследования между двумя классами. В этом отношении, каждом экземпляру подкласса соответствует один

экземпляр надкласса. Эти два экземпляра надо связать. Предполагаем, что подкласс содержит один (косвенный) атрибут с названием *название_надкласса-OID* для каждого его прямого надкласса. Этот атрибут, следовательно, является объектным идентификатором экземпляров класса с названием *название_надкласса*. Так, например, класс *grad_student* содержит косвенный атрибут *student-OID*. Информация о возможных ключах, создающих отношение наследования, сохраняется во множестве *CK-Inheritance*. (см. [7])

Пример 2: Пусть даны отношения *author(author#, name), book(book#, publisher)* и *author_book(authorNo, bookNo, date)*. Домен атрибутов *author#, authorNo, book#* и *bookNo* есть *integer*, домен атрибутов *name* и *publisher* есть *string*, а домен атрибута *date* есть *datetime*. Атрибуты *authorNo* и *bookNo* являются внешними ключами отношения *author_book* на отношения *author* и *book*. Трансформированная схема здесь такова:

```
class author                    class book
        author# : integer;              book# : integer;
        name : string;                  publisher : string;
end;                            end;


class author_book
    author : ref author;
    book   : ref book;
    date   : datetime;
end;
```

Класс *author_book* создается из отношения *author_book*, возможный ключ которого содержал внешние ключи, указывающие на отношения *author* и *book*. Атрибуты внешних ключей трансформированы в ссылки (сложные атрибуты), которые, в принципе, являются объектными идентификаторами экземпляров классов, созданных из отношений, на которые указывали эти внешние ключи. Эта ситуация обрабатывается Случаем 6 (см.[7]) в процессе трансформации схемы. Возможные ключи, описанные в этом случае, иногда будем обозначать как CK-ref ключи. Информация о возможных ключах, создающих отношение этого типа, сохраняется во множестве *CK-Reference* ([7]). Информация об обычных атрибутах таких возможных ключей (например об атрибуте *date* возможного ключа *authorNo, bookNo, date*) сохраняется во множестве *CK-Ordinary* ([7]).

Пример 3: Пусть даны отношения *office(office#, building)* и *employee(name, adress, office_no)*. Домен атрибутов *office#* и *office_no* есть *integer*, домен атрибутов *name, adress* и *building* есть *string*. Атрибут *office_no* являются внешним ключом отношения *employee* указывающим на отношение *office*. Трансформированная схема здесь такова:

```
class office                    class employee
```

office# : integer; *name : string;*

building : string; *address : string;*

end; *emp_office: ref office;*

end;

Атрибуты внешнего ключа, не содержащегося в возможных ключах, трансформированы в сложный атрибут, который является объектным идентификатором экземпляров класса, созданного из отношения, на которые указывал этот внешний ключ. Эта ситуация обрабатывается Случаем 7 (см.[7]) в процессе трансформации схемы. Внешние ключи, рассматриваемого вида, иногда будем обозначать как FK-ref ключи. Информация о внешних ключах, создающих отношение этого типа, сохраняется в множестве *FK-Reference* ([7]).

Информация об обычных возможных ключах, т.е. о возможных ключах, которые не использованы при создании отношений, как в примерах 1 и 2, сохраняется во множестве *Ordinary_keys* ([7]). Информация об обычных атрибутах, т.е. об атрибутах, которые находятся вне всех возможных и внешних ключей, сохраняется во множестве *Ordinary* ([7]).

2. Описание процедуры трансляции запросов

Предполагаем, что и реляционные и объектно-ориентированные запросы имеют вид SELECT-FROM-WHERE.

SELECT часть запроса специфицирует названия атрибутов отношений/классов, которые являются результатом выполнения запроса.

FROM часть запроса специфицирует кортежные переменные отношений (КПО), в случае реляционного запроса, и экземплярные переменные классов (ЭПК), в случае объектно-ориентированного запроса. С помощью R(КПО) (С(ЭПК)), будем обозначать отношение (класс), для которого определена КПО (ЭПК). Например, в FROM части некоторого SQL-запроса "FROM *student, person as X*" определены две кортежные переменные отношения, student и X, для которых выполняется R(*student*)=*student* и R(X)=*person*. Во FROM-части объектно-ориентированного запроса можно определить переменные, связанные с так называемыми выражениями пути (анг. *path expressions*). Например, выражение "*X student.science_adviser.name*" связывает переменную *X* с выражением пути *student.science_adviser.name*. Это означает, что X пробегает все имена научных руководителей некоторого студента. Такие переменные будем называть *ссылочным*. Выражения пути могут содержать и селекции, например, *(student: course= computer_science).science_adviser.name* .

WHERE-часть запроса специфицирует квалификационное условие запроса, т.е. условие, которое должны выполнять текущие значения КПО (ЭПК), чтобы эти значения вошли в результат. Предполагаем, что WHERE-часть реляционного SQL-запроса задана в конъюнктивном виде и что не существуют вложенности, так как каждой реляционный запрос можно преобразовать такому виду.([6])

Поскольку реляционный WHERE-предикат имеет конъюнктивный вид, то и транслированный объектно-ориентированный предикат оказывается в конъюнктивном виде. Опишем синтаксис предиката. Предикаты имеют вид -

ЛеваяЧасть Компаратор ПраваяЧасть. Причем :

1. *ЛеваяЧасть* есть выражение вида $Term_1.Term_2...Term_n$. Здесь $Term_1$ - выражение вида *(ЭПК$_1$: Селекции-на-классе$_1$)*, $Term_i$ есть выражение вида $(ATR_{i-1}$: *Селекции-на-dom*$(ATR_{i-1}))$, причем ATR_{i-1} есть сложный атрибут класса, соответствующего $ЭПК_{i-1}$ и $dom(ATR_{i-1})=Класс_i$, i=2,...,n. $Term_n$ может быть и обычным (простым) атрибутом. Выражение *Селекции-на-классе$_i$* состоит из конъюнкции селекций вида, *А Компаратор Значение*, где А обычный атрибут класса *Класс$_i$* а *Компаратор* есть один из операторов <,>,=, ≠,<= или >=.
2. *ПраваяЧасть* есть выражение одного из следующих видов:
   - отсутствует (*Компаратор* тоже отсутствует)
   - константа. В этом случае *Компаратор* также является одним из операторов <,>,=, ≠,<= или >=.
   - то же самое, что и *ЛеваяЧасть* (это, по существу, эксплицитное соединение)

Пример. Запрос: "Найти всех студентов, средняя оценка которых больше чем 9 и научный руководитель которых имеет офис в здании номер 3" можно выразить как

'SELECT Student.Adviser.Name
FROM Student
WHERE (Student: Average>9).(Adviser:).(Office:).(Building: Number=3)'.

Здесь, *Adviser* есть сложный атрибут класса *Student*, домен которого есть класс *Professor*.

В предположении, что реляционный и объектно-ориентированный запросы имеют вид SELECT-FROM-WHERE, процесс трансляции можно разделить на три части: трансляцию SELECT-части, трансляцию FROM-части и трансляцию WHERE-части запроса. Главной проблемой трансляции реляционных SQL-запросов в эквивалентные объектно-ориентированные запросы является трансляция WHERE-части запроса потому, что реляционный и объектно-ориентированный WHERE-предикат сильно отличаются друг от друга. Реляционные предикаты, которые, по существу, состоят из выражений соединения и селекции и не содержат выражений пути, транслируются в объектно-ориентированные предикаты, содержащие выражения пути.

Трансляцию WHERE-предиката будем реализовать в три шага. Сначала, из реляционного предиката создается граф реляционного предиката. На втором шаге этот граф трансформируется в соответствующий граф объектно-ориентированного предиката. Наконец, из графа объектно-ориентированного предиката получается сам объектно-ориентированный предикат. Некоторые идеи этого процесса, со значительными модификациями, взяты из [5]. В отличие от [5], в ходе трансляции реляционного WHERE-предиката, производится трансляция и SELECT и FROM-частей запроса в вид, который является вполне эквивалентным начальному реляционному запросу.

Как и в [7], класс возникший из отношения r будем обозначать с C(r). Отношение, из которого возникает класс t обозначаем с R(t).

3. Трансляция реляционного SQL-запроса в эквивалентный SQL-запрос в соответствии с изменениями реляционной схемы

В этом пункте описывается подготовительный шаг, в котором реляционный SQL-запрос преобразуется в эквивалентный SQL-запрос. Это преобразование нужно, так как в некоторых ситуациях в процессе трансформации схемы, меняется схема исходной реляционной базы данных.

В процессе трансформации схемы, в случаях 1, 2, 4 и 5, (см.[7]) добавляются новые отношения в реляционную схему, некоторые атрибуты *удаляются из отношений и переходят в другие отношения а также, могут* измениться и некоторые внешние ключи. Это может осложнить процесс трансляции запросов, так как этот процесс основан на сравнивании атрибутов возможных и внешних ключей отношений.

Перечислим случаи, где возникают эти проблемы и способы их разрешения. При описании этих случаев, и далее в тексте, используем обозначения вида $КПО_1.CK=КПО_2.FK$ (или $КПО_2.CK=КПО_1.FK$) и $КПО_1.CK=КПО_2.CK$, которые обозначают что в WHERE-части реляционного запроса существуют выражения $КПО_1.A_1=КПО_2.B_1$, $КПО_1.A_2=КПО_2.B_2$, $КПО_1.A_n=КПО_2.B_n$ (некоторые из них могут быть заданы и в оборотном порядке, например $КПО_2.B_i=КПО_1.A_i$). Здесь предполагается, что атрибуты $A_1,A_2,...,A_n$ и $B_1,B_2,...,B_n$ создают возможные (или внешние) ключи отношений, которым соответствуют переменные $КПО_1$ и $КПО_2$.

Случаи описаны в порядке их сложности.

3.1. Случай 1 (соответствует случаю 4 процесса трансформации схемы)

В случае 4 процесса трансформации схемы идентифицируется наследование в отношении, которое содержит внешний ключ на его возможный ключ.

Пример: Пусть дано отношение *person(id_no, name, age, parent_id)*. Домен атрибутов *id_no, parent_id* и *age* есть *integer* а домен атрибута *name* есть *string*. Атрибут *parent_id* есть внешний ключ указывающий на возможный ключ K отношения *person*. Применением процедуры описанной для случая 4, получае новое отношение *parent* с единственным атрибутом *parent_id*, который одновременно является первичным ключом этого отношения и внешним ключом этого отношения, указывающим на возможный ключ *id_no* отношения *person*. Трансформированная объектно-ориентированная схема здесь такова:

```
class person                class parent
    id_no : integer;            inherits person;
    name : string;          end;
    age : integer;
    parent_id : integer;
end;
```

Пусть дан запрос, на имена родителей рожденных точно 50 лет тому назад:

SELECT *father.name*
FROM *person AS father, person*
WHERE (*person.parent_id=father.id_no*) AND (*father.age=50*)

Этот запрос транслируется в объектно-ориентированный запрос, который содержит эксплицитное соединение, т.е. который мало отличается от реляционного запроса.

Этот случай возникает когда в FROM-части реляционного SQL-запроса имеется две кортежные переменные отношений, $КПО_1$ и $КПО_2$, соответствующие одному отношению r, ($r=R(КПО_1)=R(КПО_2)$), причем r обработано в случае 4 ([7]) процесса трансформации схемы, а в WHERE-части имеется сравнение возможного ключа с внешним ключом, т.е. имеется выражение $КПО_1.FK=КПО_2.CK$. В этом случае надо создать новую КПО, с единственным названием, соответствующую отношению, возникшему из отношения r в случае 4. После того, надо в выражении $КПО_1.FK=КПО_2.CK$ заменить $КПО_2$ на КПО. В результате получаем выражение $КПО_1.FK=КПО.CK$, и в WHERE-часть SQL-запроса надо добавить выражение $КПО.CK=КПО_2.CK$.

Новую КПО необходимо, соответствующим способом, добавить в FROM-часть SQL-запроса. Информация о том, какие отношения в процессе трансформации схемы, были обработаны случаем 4, находится в множестве *case_4,* созданном в процессе трансформации.([7])

Применением описанной процедуры к нашему примеру получаем запрос:

SELECT *father.name*

FROM *person AS father, person , parent*

WHERE (*person.parent_id=parent.id_no*) AND (*parent.id_no= father.id_no*) AND (*father.age=50*)

Этот запрос транслируется в объектно-ориентированный запрос без эксплицитного соединения *SELECT osoba.parent.ime FROM osoba WHERE osoba.(parent: age=50)* который лучше соответствует семантике базы данных. Похожее замечание можно сделать и в следующем случае.

3.2. Случай 2 (соответствует случаю 5 процесса трансформации схемы)

В случае 5 процесса трансформации схемы обрабатывается отношение, возможный ключ которого содержит внешних ключей.

Пример: Пусть данные отношения *person(id_no, name)* и *marriage(male#, female#, marriage_date*). Домен атрибутов *id_no, male#* и *female#* есть *integer,* домен атрибута *name* есть *string,* а домен атрибута *marriage_date* есть *date.* Атрибуты *male#* и *female#* являются внешними ключами отношения *marriage,* указывающим на возможный ключ *id_no* отношения *person.*

Применением процедуры, описанной для случая 5, получаются новые отношения *husband(male#)* и *wife(female#).* Удаляются внешние ключи *male#* и *female#* отношения *marriage,* указывающие на возможный ключ *id_no* отношения *person* и определяются новые внешние ключи, которые указывают на отношения *husband* и *wife,* соответственно. Определяются и классы *husband* и *wife,* которые являются подклассами класса *person.*

Этот случай возникает когда в FROM-части реляционного SQL-запроса имеем две кортежные переменные отношений $КПО_1$ и $КПО_2$, соответствующие отношениям r_i и r_j, соответственно, т.е. когда $r_i = R(КПО_1)$ и $r_j=R(КПО_2)$, причем r_i и r_j обработаны в случае 5 процесса трансформации схемы, а в WHERE-части имеется сравнение внешнего ключа отношения r_i с возможным

ключом отношения r_j, т.е. когда имеется выражение $КПО_1.FK=КПО_2.CK$. Предположим что, пользуясь этими ключами создается новое отношение t_k. В этом случае, надо эту часть WHERE-предиката удалить и добавить в WHERE-часть запроса выражения $КПО_1.FK=КПО.CK$ и $КПО.CK= КПО_2.CK$, где КПО новая кортежная переменная отношения t_k (t_k=R(КПО)), которую надо *добавить в FROM-часть запроса. Информация об отношениях, обработанных* случаем 5 в процессе трансформации схемы, находится в множестве *case_5,* созданном в этом процессе([7]).

Пусть дан запрос об именах мужчин, которые поженились 1 января 1970. года:

SELECT *person.name*
FROM *person, mariage*
WHERE (*mariage.male#=person.id_no*) AND (*mariage.date='*1 January 1970*'*)

Применением описанной процедуры на этот запрос получаем :

SELECT *person.name*
FROM *person, mariage, husband*
WHERE(*mariage.male#=husband.male#*)
AND(*husband.male#=person.id_no*)
AND (*mariage.date='*1 January 1970*'*)

3.3. Случай 3 (соответствует случаю 2 процесса трансформации схемы)

Случай 2 процесса трансформации схемы идентифицирует иерархию классов, которая была в реляционной схеме представлена так, что определено одно отношение для каждого подкласса некоторого класса, но не определено отношение для самого этого класса. Тогда все атрибуты надкласса находятся в каждом отношении, которое представляет подклассы. Здесь обрабатываются отношения с одинаковыми возможными ключами.

Пример: Пусть данные отношения $r_1(\underline{A},B,C,D,E)$, $r_2(\underline{A},B,C,F,G)$ и $r_3(\underline{A},B,H,I)$. Атрибут A является возможным ключом в всех трех отношений. Применением процедуры описанной для случая 2, создаются отношения/классы $t_1(A,B,C)$ и $t_2(A,B)$ и создается иерархия, иллюстрированная на Рис. 1:

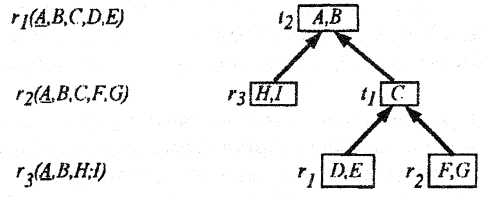

Рис. 1. Случай 2 процесса трансформации схемы

Этим процессом обработки некоторые атрибуты удаляются из отношений и переходят в другие отношения. Текущий случай относится к запросам, в FROM-части которых имеется КПО отношения r (r=R(КПО)), обработанного

случаем 2, и в WHERE или SELECT-части запроса имеется атрибут КПО.А. Если в процессе трансформации схемы атрибут А остается в отношении r, то запрос не надо трансформировать.

В противоположном случае, надо определить путь атрибута А по иерархии классов от класса C(r) до класса k_s, в котором находится атрибут А по окончании процесса трансформации схемы. Под путем здесь подразумеваем последовательность упорядоченных пар $(C(r), k_1), (k_1, k_2),...,(k_{s-1}, k_s)$, причем (k_i, k_{i+1}) означает что класс k_i является подклассом класса k_{i+1}. Информация такого вида, вместе с названием атрибута, который переходит из класса в класс, находится во множестве *Move* ([7]), созданном в процессе трансформации схемы и, поэтому, этот 'путь' легко вычислить.

В данном случае надо применить следующую процедуру. Пусть $КПО_1$, $КПО_2,...,КПО_s$ кортежные переменные отношений r_{k_1}, $r_{k_2},...,r_{k_s}$, из которых возникли классы k_1, $k_2,...,k_s$, т.е. для которых выполняется $k_v=C(r_{k_v})$ и $r_{k_v}=R(КПО_v)$, для v=1,...,s . Запрос трансформируется так что в его FROM-часть добавляются $КПО_1$, $КПО_2$, ... , $КПО_s$ а в WHERE-часть добавляются выражения $КПО.СК=КПО_1.СК$, $КПО_1.СК=КПО_2.СК$, ... $КПО_{s-1}.СК=КПО_s.СК$, причем СК есть любой из одинаковых возможных ключей отношений r_{k_1}, $r_{k_2},...,r_{k_s}$. В SELECT и WHERE-частях запроса каждое появление атрибута А, т.е. выражения вида КПО.А, заменяем на $КПО_s.А$.

Пуст, например, дан запрос:

SELECT $r_2.B$

FROM r_2

WHERE $(r_2.B > 5)$

Применением описанной процедуры на этот запрос получаем:

SELECT $t_2.B$

FROM r_2, t_1, t_2

WHERE $(t_2.B > 5)$ AND $(r_2.A = t_1.A)$ AND $(t_1.A = t_2.A)$

3.4. Случай 4 (соответствует случаю 1 процесса трансформации схемы)

Случай 1 процесса трансформации схемы идентифицирует иерархию классов с помощью значений кортежей отношения. Алгоритмы для открытия знания (анг. knowledge discovery) дают множество правил, на основе которых создается иерархия.

Пример: Пусть дано отношение *project(project_number, project_name, project_type, vendor, programming_language)*. Применением некоторого алгоритма для открытия знания на это отношение получаем множество правил, в данном случае содержащие два правила: *(project_type='S')* $\Rightarrow$ *vendor=NULL* и *(project_type ='H')* $\Rightarrow$ *programming_language =NULL*. Здесь создаются два отношения, *hardware_project* и *software_project*, а соответствующие классы становятся подклассами класса *project*.

Этот случай относится к запросам, в FROM-части которых находятся КПО отношения r (r=R(КПО)) причем отношение r обработано случаем 1 процесса трансформации схемы, а в WHERE или SELECT-частях запроса

имеем атрибут КПО.А . В процессе трансформации схемы, атрибут А может не оказаться в отношении r, т.е. может не находится в соответствующем классе C(r), но в некотором из его непосредственных подклассов.

В данном случае надо найти класс c_k, в который перешел атрибут А. Информацию можно получить, как и в предшествующем случае, из множества *Move* где сохранены тройки вида (A, r_i, r_j), которые обозначают, что атрибут А перешел из отношения r_i в отношение r_j. Тройки, относящейся к этому случаю легко распознать, т.к. только они содержат r_i и r_j для которых класс $C(r_i)$ является надклассом класса $C(r_j)$. В предшествующем случае, класс $C(r_j)$ является надклассом класса $C(r_i)$.

В FROM-часть запроса надо добавить новую $КПО_k$, соответствующую отношению r_k (r_k=R($КПО_k$)), в SELECT и WHERE-части заменить КПО.А на $КПО_k$.А и, наконец, в WHERE-часть добавить выражение КПО.СК=$КПО_k$.СК, где СК есть один из общих возможных ключей отношений r и r_k.

Пусть, например, дан запрос:

SELECT *project.vendor*

FROM *project*

WHERE (*project.project_number=123*)

Применением описанной процедуры получаем запрос:

SELECT *hardware_project.vendor*

FROM *project , hardware_project*

WHERE (*project. project_number = 123*) AND

(*hardware_project.project_number = project. project_number)*

4. Конструирование графа реляционного предиката из WHERE-части реляционного SQL-запроса

После выполнения этого подготовительного шага (п.3), можем перейти к собственно процессу трансляции SQL-запроса в объектно-ориентированный запрос.

Для данного реляционного предиката RW в WHERE-части реляционного SQL-запроса определяем его граф: G(RW) = (RV, RE), где каждая вершина из множества вершин RV представляет одну КПО, которая появляется в предикате RW (т.е. во FROM-части SQL-запроса), а каждое ребро между двумя вершинами, которые находятся в RE, представляет некоторый предикат соединения между двумя КПО в RW. Каждая вершина помечена всеми предикатами селекции соответствующей КПО, а каждое ребро между двум вершинами, например, ребро между $КПО_1$ и $КПО_2$, помечено некоторым предикатом соединения вида '$КПО_1.A_1$ *компаратор* $КПО_2.A_2$' где A_1 и A_2 являются атрибутами отношений r_1=R($КПО_1$) и r_2=R($КПО_2$), соответственно.

Чтобы реализовать этот шаг, необходимо для каждой КПО из FROM-

части реляционного SQL-запроса создать одну вершину графа G с таким же именем, как и КПО. Для каждой части WHERE-предиката, которая является предикатом соединения, т.е. для каждой фразы вида '$КПО_1.A_1$ *компаратор* $КПО_2.A_2$', надо создать ребро между вершинами $КПО_1$ и $КПО_2$ и пометить его выражением '$КПО_1.A_1$ *компаратор* $КПО_2.A_2$'. Для остальных частей WHERE-предиката, которые имеют вид 'КПО.А *компаратор константа*' или '*константа компаратор* КПО.А', т.е. для предикатов селекции, надо добавить выражения 'А *компаратор константа*' или '*константа компаратор* А', соответственно, как метки соответствующих вершин графа.

Очевидно, что таким образом сконструированный граф реляционного предиката WHERE-части SQL-запроса содержит ту же самую информацию, как и сам предикат.

Пример: Пусть задана следующая схема, где CK обозначает возможные ключи, а FK обозначает внешние ключи:

person(id_card_no, name, birth_year, parent_id_card_no)
CK: id_card_no
FK: person.parent_id_card_no ⊂ person.id_card_no
employee(work_id, years_at_work)
CK: work_id
student(id_card_no, student_id, department, average)
CK: id_card_no
CK: student_id, department
FK: student.id_card_no ⊂ person.id_card_no
course(course_name, book)
CK: course_name
graduated_student(student_id, department, scientific_adviser, course_1, course_2)
CK: student_id, department
FK: graduated_student.{student_id, department} ⊂ student. {student_id, department}
FK: graduated_student.scientific_adviser ⊂ teacher.work_id
FK: graduated_student.course_1 ⊂ course. course_name
FK: graduated_student.course_2 ⊂ course. course_name
exam(student_no, department, course_name, teacher, date, mark)
CK: student_no, department, course_name, teacher, date
FK: exam.{student_id,department} ⊂ graduated_student.{student_id, department}
FK: exam.course_name ⊂ course.course_name
FK: exam.teacher ⊂ teacher. work_id

Пусть дан запрос: "Найти среднюю оценку всех дипломированных студентов, которые 15. июня сдавали экзамен по математике". Соответствующий SQL-запрос имеет вид:

```
SELECT student.average
FROM exam, graduated_student, student
WHERE (student.student_id=graduated_student.student_id) AND
```

(student.department=graduated_student.department) AND
(exam.student_no=graduated_student.student_id) AND
(exam.department=graduated_student.department) AND
(exam.date='15 jun') AND (exam.course_name=mathematics)

Согласно приведенному алгоритму, из WHERE предиката этого запроса *получаем граф, показанный на Рис. 2.*

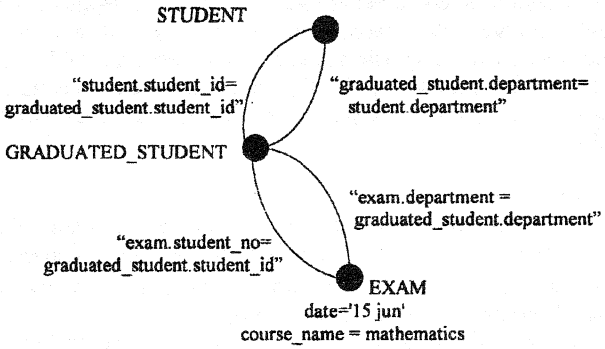

Рис 2. Пример графа реляционного WHERE-предиката

5. Конструирование графа объектно-ориентированного предиката из графа реляционного предиката

Пусть RW будет WHERE-частью реляционного SQL-запроса реляционной схемы. Граф этого реляционного предиката G(RW)=(RV, RE) трансформируется в помеченный граф OG(RW)=(OV, OE_1, OE_2) причем OV есть множество вершин, OE_1 множество направленных ребер, а OE_2 множество ненаправленных ребер.

Каждая вершина из OV соответствует одной экземплярной переменной класса (ЭПК), соответствующей некоторому классу в объектно-ориентированной схеме. Каждой вершине будем обращаться по переменной ЭПК, которая соответствует этой вершине. В графе существуют три вида ребер: направленные ребра с меткой, направленные ребра без метки, и ненаправленные ребра с меткой.

Направленное ребро от $ЭПК_1$ к $ЭПК_2$, с меткой *METKA*, соответствует косвенному соединению классов $C(ЭПК_1)$ и $C(ЭПК_2)$ с помощью сложного атрибута с именем *METKA* класса $C(ЭПК_1)$, домен которого есть класс $C(ЭПК_2)$. Направленное ребро от $ЭПК_1$ к $ЭПК_2$ без метки соответствует косвенному соединению экземпляра класса $C(ЭПК_1)$ и соответствующего экземпляра его надкласса $C(ЭПК_2)$. Ненаправленное ребро от $ЭПК_1$ до $ЭПК_2$ с меткой *METKA*, где *METKA* есть выражение вида '$ЭПК_1.A_1$ *Компаратор*

$ЭПК_2.A_2$', означает эксплицитное соединение между экземплярами классов $C(ЭПК_1)$ и $C(ЭПК_2)$ по атрибутами A_1 и A_2.

Процедура трансформации графа G(RW) в граф OG(RW) состоит из трех частей: трансформации вершин, трансформации ребер и трансформации меток на вершинах.

5.1. Трансформация вершин

Трансформация вершин проводится следующим способом. Сначала, для каждой вершины с названием КПО во множестве GV вершин графа G(RW), надо создать вершину в OV с таким же именем. Эта вершина соответствует экземплярной переменной класса, возникшего из отношения, которому соответствовала вершина из множества GV. Новая вершина в OV получает те же самые метки, что и вершина графа G, из которой она возникла. В нашем примере, получаем граф с тремя вершинами: EXAM, GRADUATED_STUDENT и STUDENT.

5.2. Трансформация ребер

Трансформация ребер проводится следующим способом. Сначала, для каждой пары вершин $КПО_1$ и $КПО_2$ из RV, определяется множество E ребер между этими вершинами. Это множество разбиваем на части, из которых создаем ребра объектно-ориентированного графа.

Сначала трансформируются ребра, которые соответствуют косвенному соединению класса и надкласса. Если во множестве E существуют ребра, помеченные '$КПО_1.A_1$=$КПО_2.B_1$', ... , '$КПО_1.A_s$ =$КПО_2.B_s$', и атрибуты $A_1,A_2,...,A_s$ и $B_1,B_2,...,B_s$ создают возможные ключи отношений $R(КПО_1)$ и $R(КПО_2)$, которые участвовали в создании отношения наследования между соответствующими классами, то все эти ребра удаляются и в объектно-ориентированный граф, в множество OE_1, включается *направленное ребро без метки* между соответствующими вершинами объектно-ориентированного графа. При этом устанавливается направление от подкласса к надклассу. Информация, необходимая для этого случая, находится в множестве *CK-Inheritance*. В нашем примере создается направленное ребро без метки от вершины GRADUATED_STUDENT до вершины STUDENT.

Далее, трансформируются ребра, которые соответствуют косвенному соединению двух классов, причем один класс является доменом сложного атрибута другого класса. Если во множестве E существуют ребра, помеченные выражениями '$КПО_1.A_1$=$КПО_2.B_1$',...,'$КПО_1.A_s$=$КПО_2.B_s$', где атрибуты $A_1,A_2,...,A_s$ создают внешний ключ отношения $R(КПО_1)$, а атрибуты $B_1,B_2,...,B_s$ создают возможный ключ отношения $R(КПО_2)$, и если эти внешний и возможный ключи участвовали в создании сложного атрибута *ATR* класса, возникшего из первого отношения, и домен которого есть класс, возникший из второго отношения, то все эти ребра удаляются, и в объектно-ориентированный граф, т.е. во множество направленных ребер OE_1, включается *направленное ребро с меткой ATR* от вершины объектно-ориентированного графа, возникшей из $КПО_1$ до вершины, возникшей из $КПО_2$. Информация, необходима для этого случая находится во множествах *FK-Reference* и *CK-Reference*. В нашем

примере создается направленное ребро с меткой *graduated_student* от вершины EXAM к вершине GRADUATED_STUDENT.

Все остальные ребра прямо трансформируются в *ненаправленные ребра с меткой* объектно-ориентированного графа, т.е. включаются во множество OE_2. Эти ребра, по существу, соответствуют эксплицитному соединению.

5.3. Трансформация меток вершин

Наконец, надо трансформировать метки вершин. Метки вершин реляционного графа соответствуют селекциям в реляционном WHERE-предикате. Их трансформация проводится в зависимости от того, удален ли атрибут A из класса, возникшего в процессе трансформации схемы из отношения, которому принадлежит тот атрибут, на котором делается селекция. Процедура является рекурсивной. Если атрибут не удален из класса, ту же самую метку вершины графа реляционного предиката добавляем соответствующей вершине графа объектно-ориентированного предиката. В нашем примере это выполняется для селекции Date='15 jun' вершины EXAM. Противоположный случай означает, что роль атрибута A в классе, возникшем в процессе трансформации схемы из отношения, которому принадлежит этот атрибут, играет атрибут некоторого другого класса. Информацию об этом другом атрибуте и классе, которому он принадлежит, можно получить из множеств *CK-Inheritance* (если A был атрибутом некоторого возможного ключа, который участвовал в создании отношения наследования) и *FK-Reference* и *CK-Reference* (если A был атрибутом некоторого внешнего ключа, из которого, в процессе трансформации схемы, был порожден сложный атрибут, например *ATR).*

В первом случае, если существует направленное ребро без метки между соответствующими вершинами, то надо повторить всю процедуру для другого класса и для атрибута возможного ключа, соответствующего атрибуту A. Если ребро с этим признаком не существует, надо создать новую ЭПК, которая соответствует этому другому классу, создать новую вершину графа и связать текущую вершину с новой вершиной с помощью *направленного ребра без метки.* Далее следует повторить всю процедуру для этой новой вершины и для атрибута, соответствующего атрибуту A. Похожую процедуру надо выполнить и во втором случае, когда A является атрибутом некоторого внешнего ключа, из которого в процессе трансформации схемы был создан некоторый сложный атрибут. Разница только в том, что теперь создается *направленное ребро с меткой*, в качестве которой используется имя этого сложного атрибута.

В нашем примере для селекции 'course_name = mathematics' сначала создается новая ЭПК для класса *course*, которую можем назвать COURSE, вершина COURSE и направленное ребро с меткой *exam_course* от вершины EXAM к вершине COURSE. Теперь процедура повторяется для вершины COURSE и атрибута course_name. Атрибут course_name остается в классе COURSE, а вершина COURSE получает метку 'course_name =mathematics'.

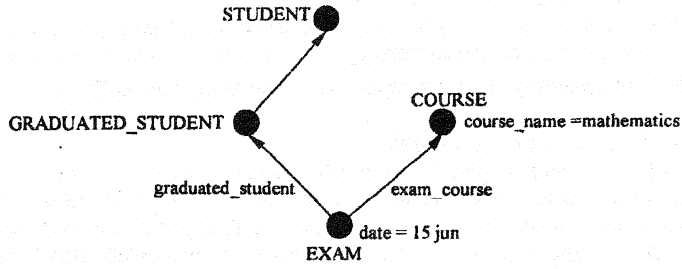

Рис. 3. Пример графа объектно-ориентированного WHERE-предиката

6. Конструирование объектно-ориентированного предиката из его графа

Сначала перечислим все основные конструкции, которые могут оказаться в графе объектного предиката, и объясним их семантику. Основные конструкции показаны на Рис. 4. В каждой конструкции большими буквами обозначены экземплярные переменные классов (ЭПК), а малыми буквами обозначены сложные атрибуты, доменами которых являются классы, соответствующие экземплярным переменным класса, которые находятся на концах направленных ребер.

Рис. 4 Основные конструкции графа объектно-ориентированного WHERE предиката

Конструкция 1: На рис. 4(a) предполагается: (1) не существует направленного ребра, которое заканчивается вершиной A; (2) кроме вершины A, ни одна вершина не связана ни с одной вершиной, кроме ее предшественника в направленном пути; (3) ни одна вершина не появляется больше одного раза в направленном пути. Семантика этой конструкции говорит, что нужен 'переход' от A до E. В определении объектно-ориентированного WHERE-предиката это

соответствует выражению:

(A: Селекция-на-A).(b: Селекция-на-B)....(e: Селекция-на-E).

Конструкция 2: На рис. 4(b) - два направленных пути, связанные с помощью ненаправленного ребра. Возможно, что один или оба пути имеют длину нуль. Предположим что *"E.Атрибут1 Компаратор T.Атрибут2"* есть метка ненаправленного ребра. Эта конструкция транслируется в эксплицитное соединение по простым атрибутам.

(A: Селекция-на-A).(b: Селекция-на-B)....(e: Селекция-на-E).Атрибут1 Компаратор
(F: Селекция-на-F).(g: Селекция-на-G)....(t: Селекция-на-T).Атрибут2

Конструкция 3: Эта конструкция представлена на рис. 4(c). Если существуют три или более ребер, связанных с вершиной, то все эти ребра ссылают на тот же самый экземпляр класса, которому соответствует ЭПК, соответствующий этой вершине. Например, d1, d2, d3, e, f ссылаются на один и тот же самый экземпляр класса C(D). Для этого надо специфицировать ссылочную переменную, которая соответствует одному из входных ребер. Например, в ситуации на рис. 4(c), можно определить, что ссылочная переменная X соответствует выражению пути (*A: Селекция-на-A).(d1: Селекция-на-D).* Эту информацию, конечно, надо добавить в FROM-часть запроса. Получаем следующие объектно-ориентированные предикаты:

(B: Селекция-на-B).d2=X AND (c: Селекция-на-C).d3=X
AND X.(e: Селекция-на-E) AND X.(f: Селекция-на-F)

Конструкция 4: Эта конструкция представлена на рис. 4(d). Если начнем вычисление, например, от вершины A, тогда сложный атрибут *a* класса, соответствующего ЭПК T, т.е. класса C(T), ссыляется на тот же самый экземпляр класса C(A). Поэтому эту конструкцию можно транслировать в следующий объектно-ориентированный предикат:

(A: Селекция-на-A).(b: Селекция-на-B).... (d: Селекция-на-D).(e: Селекция-на-E).(F: Селекция-на-F)...(g: Селекция-на-G).(h: Селекция-на-H).(a:)= A.OID

Очевидно, если начнать из разных вершин, получим различные объектно-ориентированные предикаты. Конечно, все эти предикаты эквивалентны.

Теперь можно описать и алгоритм, выполняющий конверсию. Для реализации алгоритма, нужны два множества упорядоченных пар, *Эксплицитное_соединение* и *Reference*.

Когда первый раз рассматривается ненаправленное ребро с меткой *"C_1.Атрибут1 Компаратор C_2.Атрибут2"*, во множество *Эксплицитное_соединение* добавляем упорядоченную пару, первый элемент которой есть метка, т.е. выражение *"C_1.Атрибут1 Компаратор C_2.Атрибут2"*, а второй элемент есть *выражение пути* для вершины, через которую пришли к этому ребру (C_1 или C_2). Следующий раз, когда рассматривается это ребро в связи со второй вершиной, с помощью выражения пути для этой вершины, раньше сохраненного выражения пути второй вершины и метки, можно создать

выражение эксплицитного соединения.

Во множестве *Reference* находятся упорядоченные пары, которые соответствуют ссылочным переменным. Первый элемент упорядоченной пары есть собственно переменная, а второй элемент - вершина, которой эта переменная соответствует.

В каждый момент работы алгоритма *активная вершина* назначается. Это значит, что алгоритм будет исследовать ребра, связанные с ней. Прежде чем полностью будет создан предикат, существует возможность создания и некоторых промежуточных результатов. Частный предикат, который рассматривается в данный момент, хранится в переменной *Частный_предикат*. Кроме частного предиката будем хранить и переменную *Путь*, обозначающую путь, который в данный момент пройден, так же как и для *Частного_предиката*, с тем различием, что *Путь* не содержит селекции. Это используется при генерации SELECT-части объектно-ориентированного предиката. Для конструкции SELECT-части объектно-ориентированного предиката нужна и информация о соответствии атрибутов. По существу, эта информация получается способом, описанным при обсуждении процесса трансформации селекции на вершинах. Различие только в том, что пройденный 'путь' от начального до конечного класса надо сохранять. При переходе от подкласса к надклассу не делается ничего, а при переходе от класса к классу с помощью сложного атрибута надо добавить в путь название этого атрибута. Для остальных атрибутов, они сами являются путями. Например, атрибуту *course_name* отношения *exam* соответствует выражение пути *exam_course.course_name* а атрибуту *date* соответствует выражение пути *date*. Предполагаем, что вся эта информация сохранена во множестве *Пути*.

Надо отметить, что информацию, содержащуюся во множестве *Пути*, можно получить и в процессе конверсии данных ([8]) тем же самым способом, которым создаются так называемые запрос-множества. Различие только в том что здесь надо добавить и информацию, содержащуюся во множествах *Ordinary_keys*, *Ordinary* и *CK-Ordinary*, а при конструировании правых частей упорядоченных пар не добавляются скобки и получаются 'чистые' выражения пути.

Degree(V) обозначает число ребер (направленных и ненаправленных) связанных с V. Это значение меняется, в соответствии с тем, как алгоритм удаляет ребра из графа.

6.1. Алгоритм

Исходные данные: (a) Граф объектно-ориентированного WHERE-предиката OG(OV, OE1, OE2)

(b) SELECT-часть реляционного SQL-запроса

Результат: Объектно-ориентированный запрос.

Шаг 1. Повторять этот шаг, пока множество OG не станет пустым:

Шаг 1.1:

1. Если уже существует активная вершина перейти на Шаг 1.2.
2. Иначе, выбрать активную вершину. Выбор активной вершины делается в соответствии с текущим OG следующим образом:

(a) выбрать любую вершину, которая не имеет входных направленных ребер

(b) если вершины со свойством (a) не существует, выбрать любую вершину V для которой *Degree*(V)>2

(c) если вершины со свойством (b) не существует, выбрать любую вершину

Здесь надо отметить, что выбор активной вершины делается так, что если существует более одной возможности, то за активную вершину выбирается вершина, которая выбралась за меньшее число итераций. Это надо делать, чтобы *избежать бесконечных циклов, которые возможны, если постоянно выбираем* одну и ту же вершину (на пример, в случае б).

Если случай (a) не выполняется, то каждая вершина должна иметь хотя бы одно входное направленное ребро. Потому, в случаях (b) и (c) выбранная вершина должна быть в направленном цикле. Чтобы формировать предикат для этой конструкции, надо сохранить *начальную вершину.* В случаях (b) и (c) обозначаем выбранную вершину как *начальную.* Возможно, что существует более чем одна начальная вершина. Если выбранная вершина связанна со ссылочной переменной X (информация об этом находится во множестве *Reference*), то *Частный_предикат*="X". Иначе, *Частный_предикат*=" (V: P_V) ", где P_V конъюнкция селекций вершины V. Далее, если V еще не находится во FROM-части объектно-ориентированного запроса, добавить его. Пусть *Путь*="V". Для каждого атрибута вида V.A из SELECT-части SQL-запроса, для которого это еще не сделанно, добавить в SELECT часть объектно-ориентированного запроса выражение V.B, где B есть выражение пути (из множества *Пути*), соответствующее атрибуту A.

Шаг 1.2:

Здесь имеет место хотя бы один из нижеуказанных 6 случаев. Их надо тестировать и выполнять в указанном порядке:

Случай 1: Активная вершина имеет выходное направленное ребро с меткой *METKA* к вершине, которая связана со ссылочной переменной X (которая создана в случае 5). Этот случай соответствует Конструкции 3 по созданию ссылочной переменной. Он может также соответствовать и Конструкции 4. Тогда:

(1) Если *METKA* является пустой строкой, создать предикат "*Частный_предикат*= X", иначе создать предикат "*Частный_предикат.METKA* = X ".

(2) Удалить выходное ребро и все вершины, которые с удалением этой вершины станут изолированными.

(3) Пусть *Путь*=" ".

(4) Дезактивировать активную вершину и перейти к Шагу 1.

Случай 2: Активная вершина имеет выходное направленное ребро с меткой *METKA*, к начальной вершине SV. Это значит, что обнаружен направленный цикл, т.к. нам дважды встретились одна и та же вершина. Этот случай соответствует Конструкции 4. Тогда:

(1) Если *METKA* является пустой строкой, создать предикат "*Частный_предикат*= SV-OID", иначе создать предикат "*Частный_предикат.METKA* = SV-OID ".

(2) Удалить выходное ребро и все вершины, которые с удалением этой вершины станут изолированными.

(3) Пусть *Путь*=" ".

(4) Дезактивировать активную вершину и перейти к Шагу 1.

Случай 3: Активная вершина имеет выходное направленное ребро с меткой *METKA* к вершине V_1, для которой выполняется условие *Degree*(V_1)=1. Этот случай соответствует последней вершине Конструкции 1. Тогда:

(1) Если *METKA* отличается от пустой строки, создать предикат *"Частный_предикат.*(*METKA* : P_{v_1})", где P_{v_1} конъюнкция селекций вершины V_1. Иначе, если *Частный_предикат* заканчивается скобкой, добавить конъюнкцию селекций P_{v1} перед этой скобкой. Если *Частный_предикат* не содержит скобки, т.е. имеет вид "X", создать предикат " (X: P_{v_1}) ".

(2) Удалить выходное ребро и все вершины, которые с удалением этой вершины станут изолированными.

(3) Если *METKA* отличается от пустой строки, для каждого атрибута вида V_1.A из SELECT-части SQL-запроса, добавить в SELECT-часть объектно-ориентированного запроса выражение *Путь.METKA*.B, где B есть выражение пути (из множества *Пути*), которое соответствует атрибуту A.

(4) Пусть *Путь*=" ".

(5) Дезактивировать активную вершину и перейти к Шагу 1.

Случай 4: Активная вершина имеет выходное направленное ребро с меткой *METKA*, к вершине V_1, для которой выполняется условие *Degree*(V_1)=2, причем существуют точно одно направленное входное ребро и одно направленное выходное ребро для этой вершины. Этот случай соответствует средней вершине Конструкции 1. Тогда:

(1) Если *METKA* отличается от пустой строки, то *Частный_предикат* принимает значение *"Частный_предикат.*(*METKA* : P_{v_1}) ", где P_{v_1} конъюнкция селекций вершины V_1. Иначе, если *Частный_предикат* заканчивается скобкой, добавить конъюнкцию селекций P_{v_1} перед этой скобкой, и это будет новый *Частный_предикат.* Если *Частный_предикат* не содержит скобки, т.е. имеет вид "X", тогда *Частный_предикат* = " (X: P_{v_1}) "

(2) Удалить выходное ребро и все вершины, которые с удалением этой вершины станут изолированными.

(3) Если *METKA* отличается от пустой строки, для каждого атрибута вида V_1.A из SELECT-части SQL запроса, добавить в SELECT-часть объектно-ориентированного запроса выражение *Путь.METKA*.B, причем B есть выражение пути (из множества *Пути*), которое соответствует атрибуту A.

(4) Если *METKA* отличается от пустой строки, то вводим *Путь* =" *Путь.METKA* "

(5) Дезактивировать активную вершину и перейти к Шагу 1.

Случай 5: Активная вершина имеет выходное направленное ребро с меткой *METKA*, к вершине V_1, для которой выполняется *Degree*(V_1) ≥ 2. Этот

случай соответствует Конструкции 3 до момента, когда будет создана ссылочная переменная. Тогда :

(1) Создать единственную ссылочную переменную, например Y.

(2) Если *Частный_предикат* заканчивается скобкой, добавить конъюнкцию селекций P_{v_1} перед этой скобой и пусть это будет новый *Частный_предикат*. Если *Частный_предикат* не содержит скобки, т.е. имеет вид "X", то *Частный_предикат* = "(X: P_{v_1})". Добавить выражение "*Частный_предикат.METKA* AS Y" во FROM-часть объектно-ориентированного запроса, если *METKA* отличается от пустой строки. В противоположном случае, добавить выражение "*Частный_предикат* AS Y ".

(3) *Частный_предикат* =" Y ".

(4) Если *METKA* отличается от пустой строки, для каждого атрибута вида V_1.A из SELECT части SQL-запроса, добавить в SELECT-часть объектно-ориентированного запроса выражение Y.*METKA*.B, где B есть выражение пути (из множества *Пути*), которое соответствует атрибуту A.

(5) *Путь* = " Y "

(6) Удалить выходное ребро и все вершины, которые с удалением этой вершины станут изолированными Добавить упорядоченную пару (Y, V_1) во множество *Reference*.

(7) Пусть активная вершина будет V_1 и перейти на Шаг 1.

Случай 6: Активная вершина V связана с ненаправленным ребром *e*, которое помечено как *METKA*. Пусть*Путь* =" ". Предположим, что *METKA*=" *C_1.Атрибут1 Компаратор C_2.Атрибут2* ". Этот случай соответствует Конструкции 2. Тогда,

(1) Если *METKA* отличается от первых элементов каждой упорядоченной пары из множества *Эксплицитное_соединение*, тогда:

(a) Добавить упорядоченную пару (*METKA*, *Частный_предикат*) во множество *Эксплицитное_соединение*.

(b) Дезактивировать активную вершину и перейти к Шагу 1.

(2) Если *METKA* совпадает с первым элементом упорядоченной пары из множества *Эксплицитное_соединение*, а второй элемент этой пары есть предикат *Частный_предикат2*, тогда:

(a) Сформировать эксплицитное соединение, т.е. предикат "*Частный_предикат.Атрибут1 Компаратор Частный_предикат2.Атрибут2* ", если *C_1= V*, или "*Частный_предикат2.Атрибут1 Компаратор. Частный_предикат.Атрибут2*", если *C_2= V*.

(b) Удалить найденную упорядоченную пару из множества *Эксплицитное_соединение*

(c) Удалить ребро *e* и все вершины, которые с удалением этой вершины станут изолированными

(d) Дезактивировать активную вершину и перейти к Шагу 1.

ШАГ 2: Все предикаты, полученные на Шаге 1, связать с помощью "AND". Получим WHERE-часть объектно-ориентированного запроса.

Пример. В нашем примере получаем объектно-ориентированный запрос:

```
SELECT exam.graduated_student.average
FROM exam
WHERE (exam: Date='15 jun').(Exam_course: Course_name=mathematics)
```

Сложность приведенных алгоритмов зависит от большого числа взаимосвязанных параметров. Например, сложность зависит от схем реляционной и объектно-ориентированной баз данных, числа кортежных переменных отношений, числа экземплярных переменных классов, числа соединений в запросе, числа селекций в запросе, от того на каких атрибутах делается селекция или соединение ...

Литература.

1..Joseph Fong, "Converting Relational to Object-Oriented Databases", SIGMOD Record, Vol.26, No. 1, March 1997

2. C.J. Date, "An Introduction to Database Systems", (6th edition) Addison-Wesley Systems Programming Series, (1995)

3. Shekar Ramanathan, Julia Hodges, "Extraction of Object-Oriented Structures from Existing Relational Databases", SIGMOD Record, Vol.26, No. 1, March 1997

4. Gregory Piatetsky-Shapiro, William Frawley "Knowledge Discovery in Databases", AAAI Press / MIT Press, California, 1991

5. Gordana Pavlovic-Lazetic, "Osnove relacionih baza podataka", Vesta-Matematicki fakultet, Beograd, 1996

6. Won Kim, "Introduction to Object Oriented Databases", The MIT Press, 1990

7. Predrag Stanisic, "Schema Transformation From Relational to Object-Oriented Database as a Part of Database Reverse Engineering Process", Mathematica Montisnigri, No. 9, 1998

8. Предраг Станишич, "Конверсия данных из реляционной в объектно-ориентированную базу данных", в печати.