Модел. и анализ информ. систем. Т. **18**, № **2** (2011) 39–51

УДК 519.854.2

Организация выполнения объектных запросов в динамической информационной модели DIM

Рублев В.С.

Ярославский государственный университет им. П.Г. Демидова

e-mail: roublev@mail.ru

получена 5 октября 2010

Ключевые слова: объектная СУБД, язык запросов, трудоемкость запросов, оптимизация по трудоемкости

Рассматривается задача организации алгоритма выполнения объектных запросов [1] для новой объектной технологии СУБД DIM [2], при которой трудоемкость выполнения запроса будет возможно меньшей. Решения этой задачи могут быть положены в основу разработки транслятора запросов.

1. Постановка задачи

Описанный в [1] язык объектно-динамических запросов ODQL для динамической информационной модели DIM, концепции которой приведены в [2], обладает полнотой – любая группа объектов DIM (вместе с любыми их свойствами) может быть выделена ODQL-запросом (см. [3]). Оптимизация запросов имеет очень много аспектов (см., например, [4]). Среди них весьма важным для реализации системы DIM является построение алгоритма выполнения ODQL-запроса, обладающего минимальным[1] временем выполнения запроса (*временем отклика*).

Рассмотрим пример ODQL-запроса, приведенный в [1]:

for $c_{p_2}.a_{p_2} = 2,\ c_{p_4}.a_{p_4} = 4,\ c_{i_5}.a_{i_5} = 5,\ c_{ih_1}.a_{ih_1} =$ 'строка'
select $c_0.a_0,\ c_{i_4}.a_{i_4},\ c_{p_3}.a_{p_3},\ c_c.a_c$
from $c_0,\ c_{p_1},\ c_{p_2},\ c_{p_3},\ c_c,\ c_{p_4},\ c_{i_1},\ c_{i_2},\ c_{i_4},\ c_{ih_1},\ c_{i_5}$
links c_{p_1} **parent** $c_0,\ \ c_{p_2}$ **parent** $c_{p_1},\ \ c_{p_3}$ **parent** $c_{p_1},\ \ c_c$ **contains** $c_0,$
c_{p_4} **parent** $c_c,\ \ c_0$ **contains** $c_{i_1},\ \ c_0$ **contains** $c_{i_2},\ \ c_{i_1}$ **contains**$(c_{ih_1})\ c_{i_4},$
c_{i_2} **contains** c_{i_5}

Одна из стратегий вычисления данного запроса – рассмотрение прямого произведения объектов всех классов и для каждого набора из этого произведения проверка выполнения условий запроса. Если каждый класс содержит всего по 10 объектов, то мы должны проверить 10^{11} наборов, что требует огромных временных затрат, а

[1] Имеется в виду минимальность в смысле эвристических алгоритмов выполнения запроса.

потому нерационально. Зависимость времени выполнения запроса от числа объектов классов оценивается трудоемкостью. В примере это

$$\theta(n_0,\ n_{p_1} \cdot n_{p_2} \cdot n_{p_3} \cdot n_n \cdot n_{p_4} \cdot n_{i_1} \cdot n_{i_2} \cdot n_{i_4} \cdot n_{ih_1} \cdot n_{i_5}) \approx \theta(n^{11}),$$

где n – максимальное число объектов класса. Очевидно, нужно построить другую стратегию выполнения запроса, дающую возможно меньшую трудоемкость выполнения запроса.

В реляционной технологии для этого вводятся конструкции индексов таблиц и внешних связей таблиц. Мы также введем в систему DIM подобные конструкции, использование которых делает возможным оптимизацию выполнения запроса по трудоемкости.

2. Индекс-выборка

Подобно индексам реляционных таблиц введем индексы по любому набору свойств класса для множества его объектов, каждый из которых реализуется в виде B^*-дерева. Узел такого дерева содержит указатель на объект класса и ключ, соответствующий значениям идентификационных свойств для этого объекта. Каждый объект, класс объектов и свойство класса снабжается объектным идентификатором – уникальным натуральным ключом, характеризующим эту сущность. Главный индекс объектов класса, создающийся всегда, является индексом по объектным идентификаторам объектов класса.

Помимо B^*-дерева главный индекс содержит указатель на двунаправленный *список выборки объектов класса* и пометку списка: 0 – список выборки пустой, 1 – список выборки содержит не все объекты класса, 2 – список выборки содержит все объекты класса. Этот главный индекс и представляет собой *индекс-выборку* класса. Каждый элемент списка выборки содержит указатель на узел этого дерева, объект которого выбран, а также указатель на предыдущий и последующий элементы списка, а каждый узел дерева содержит указатель на соответствующий элемент списка выборки. Введем следующие соглашения. Будем говорить, что *объект помечен*, если в индекс-выборке его класса есть непустой указатель на элемент списка выборки. *Объект не помечен*, если в индекс-выборке его класса указатель на элемент списка выборки пустой. *Обнулить пометку объекта* означает удалить соответствующий элемент списка индекс-выборки его класса и затем обнулить указатель на этот элемент.

Инициализация списка выборки объектов производится в начале выполнения запроса по фразе **for** для каждого класса, указанного фразой **from**. Если для такого класса фраза **for** содержит ограничение на выбор объектов класса, то выполняется следующий алгоритм начальной установки выборки его объектов:

1. Вычисляются значения всех агрегатных функций, входящих в ограничение для этого класса. Каждое такое вычисление требует одного обхода по всем объектам класса. Поэтому трудоемкость данного шага $\theta(n)$, где n – число объектов класса.

2. Заводится пустой список выборки, выбирается корневой узел дерева индекс-выборки и производится проверка выполнения условия фразы **for** для соответствующего узлу объекта. Если условие выполнено, то к списку выборки добавляется элемент с указателем на этот узел дерева, а указатель узла дерева устанавливается на этот элемент списка. Помечается список индекс-выборки: 0, если объект не помечен, или 2, если объект помечен.

3. Выбирается следующий по обходу дерева узел, если он есть, и производится проверка выполнения условия фразы **for** для соответствующего узлу объекта. Если условие выполнено, то к списку выборки добавляется элемент с указателем на этот узел дерева, а указатель узла дерева устанавливается на этот элемент списка, и если пометка списка равна 0, то она заменяется на 1. Если условие не выполнено, а пометка списка была 2, то она устанавливается на 1. Шаг повторяется, пока дерево не пройдено.

Если для класса из фразы **from** нет в запросе фразы **for**, то пометка списка выборки устанавливается на 2, в список выборки включаются элементы, указывающие на каждый узел дерева, а в такой узел записывается указатель на соответствующий элемент списка. Нетрудно видеть, что трудоемкость инициализации списка объектов индекс-выборки оценивается как $\theta(n)$, а потому трудоемкость начальной установки списков выборки для всех классов запроса оценивается как $\theta(n)$, где n – число объектов всех классов, участвующих в запросе.

3. Индекс-выборки отношений классов

Для каждого отношения классов устанавливаются индексы (B^*-деревья) отношений. Они играют роль взаимных внешних индексов множеств классов находящихся в отношении. Помимо B^*-дерева индекс отношения содержит указатель на двунаправленный *список выборки объектов, находящихся в соответствующем отношении объектов* и пометку списка: 0 – список выборки пустой, 1 – список выборки содержит не все объекты класса, 2 – список выборки содержит все объекты класса. Этот индекс отношения и представляет собой *индекс-выборку* отношения классов.

Каждый элемент списка выборки содержит указатель на узел этого дерева, объекты отношения которого выбраны, а также указатель на предыдущий и последующий элементы списка, а каждый узел дерева содержит указатель на соответствующий элемент списка выборки (пустое значение, если отношение объектов этого узла не входит в выборку) и указатели на соответствующие объектам узлы индекс-выборок их классов. Пусть для классов $A, B, C, \ldots$ соответствующие объектные идентификаторы их объектов обозначены $IdA, IdB, IdC, \ldots$

1. Отношение **наследования** (внутреннего наследования). IdA, IdB – объектные идентификаторы объектов родительского и дочернего классов (для внутреннего наследования классы совпадают, но объектные идентификаторы объектов в каждом узле дерева в паре различны). Вводятся 2 индекса, ключ каждого из которых является парой объектных идентификаторов объектов обоих классов:

1) IdA, IdB
2) IdB, IdA

В каждом узле каждого индексного дерева индекс-выборки отношения 2 указателя на соответствующие узлы (по объектам) индекс-выборок обоих классов и указатель на элемент списка выборки отношения.

2. Отношение **включения** (внутреннего включения). IdA, IdB, IdC – объектные идентификаторы объектов классов включающего, включенного и связи (для внутреннего включения классы A и B совпадают, но их объекты – различны). Для включения с классом связи вводятся 2 индекса, ключ каждого из которых является тройкой объектных идентификаторов объектов трех классов:
   1) IdA, IdB, IdC
   2) IdB, IdA, IdC

   В каждом узле индексного дерева 3 указателя на соответствующие узлы (по объектам) индекс-выборок соответствующих классов и указатель на элемент списка выборки отношения.

   Для простого включения вводятся 2 индекса, ключ каждого из которых является парой объектных идентификаторов объектов включающего и включенного классов:
   1) IdA, IdB
   2) IdB, IdA

   В каждом узле каждого индексного дерева 2 указателя на соответствующие узлы (по объектам) индекс-выборок обоих классов и указатель на элемент списка выборки отношения.

3. Отношение **истории** (см. [2]). IdA, IdB – объектные идентификаторы объекта-предшественника и объекта-последователя. Вводятся 2 индекса, ключ каждого из которых является парой объектных идентификаторов объектов. Классы объектов могут совпадать или быть различными.
   1) IdA, IdB
   2) IdB, IdA

   В каждом узле каждого индексного дерева 2 указателя на соответствующие узлы (по объектам) индекс-выборок их классов и указатель на элемент списка выборки отношения.

4. Отношение **взаимодействия** (см. [2]). IdF, IdT, IdW, IdH – объектные идентификаторы объектов классов F (Откуда), T (Куда), W (Что), H (Как). Вводятся 6 индексов, ключ каждого из которых является четверкой объектных идентификаторов объектов 4 классов взаимодействия:
   1) IdF, IdT, IdW, IdH
   2) IdF, IdW, IdT, IdH
   3) IdT, IdF, IdW, IdH
   4) IdT, IdW, IdF, IdH
   5) IdW, IdF, IdT, IdH
   6) IdW, IdT, IdF, IdH

   В каждом узле индексного дерева 4 указателя на соответствующие узлы (по

объектам) индекс-выборок их классов и указатель на элемент списка выборки отношения.

4. Алгоритм выполнения запроса

Ведущую роль в алгоритме играет схема слоев классов запроса, описанная в [1].

Идея алгоритма состоит в том, что для каждого класса, участвующего в запросе, создается индекс-выборка. Затем, двигаясь по схеме слоев от слоя с максимальным уровнем к нулевому (к базовому классу), мы устанавливаем индекс-выборки каждого из отношений и корректируем индекс-выборки классов отношения. В результате по индекс-выборке базового класса получаем выборку его объектов, каждый из которых удовлетворяет условиям запроса. И, наконец, двигаясь по схеме слоев от нулевого слоя к внешнему слою, определяем как результат наборы свойств запроса, удовлетворяющие его условиям.

Заметим, что схема слоев соответствует фразе **links** запроса, а при получении ODQL-запроса по алгоритму, описанному в [1], порядок отношений классов в этой фразе соответствует движению по слоям от нулевого до максимального слоя. Поэтому схему слоев для выполнения алгоритма слоев не надо строить.

При описании алгоритма мы будем оценивать трудоемкость отдельных его частей, чтобы затем в качестве итога получить оценку трудоемкости всего запроса.

Общий алгоритм

1. Для каждого класса фразы **from** проводится начальная установка его индекс-выборки, как это описано выше. При этом трудоемкость начальной установки оценивается как $\theta(n)$, где n – число объектов всех классов запроса.

2. По фразе **links** запроса проводится коррекция всех индекс-выборок классов, участвующих в запросе и установка индекс-выборок отношений этих классов. Для этого выбираются по очереди отношения классов, начиная с конца фразы. Пусть A – класс внешнего слоя отношения, B – класс внутреннего слоя, C – класс связи, участвующий в отношении включения, или классы C, H – классы, участвующие в отношении взаимодействия (класс H в роли *Как*, а остальные в других ролях). В зависимости от вида отношения выполняем описанный ниже *алгоритм коррекции индекс-выборок*. В результате выполнения этого шага для каждого помеченного объекта опорного класса и для любого множества связанных с ним объектов запроса, отвечающих ограничениям **for** и **links**, списки индекс-выборок классов этих объектов содержат пометки этих объектов, а списки выборки индекс-выборок классов отношений этих объектов содержат указатели на соответствующие их отношения.

3. Для каждого объекта индекс-выборки базового класса применяем описанный ниже *алгоритм обхода с возвратом схемы слоев* для получения всех деревьев связанных объектов запроса, и для каждого такого дерева выбираем по объектам, входящим в него, значения свойств фразы **select**.

Алгоритм коррекции индекс-выборок

1. **Отношение наследования: A – дочерний, B – родительский классы**
   Выбираем индексное дерево с ключом (IdB, IdA) и обходим его узлы по индексу:

   1.1 Если для текущего узла объект IdB не помечен, то пропускаем все узлы до изменения значения IdB. Выполняем п. 1.1.

   1.2 Если объект IdB помечен, а IdA не помечен, то переходим к следующему узлу с тем же IdB и выполняем п. 1.2.

   1.3 Если в следующем узле IdB изменился, то обнуляем пометку предыдущего значения IdB в индекс-выборке и выполняем п. 1.1.

   1.4 Если объекты IdB и IdA помечены в своих индекс-выборках, то включаем в список отношения указатель на этот узел, переходим к следующему узлу и выполняем п. 1.5.

   1.5 Если в следующем узле IdB не изменился, а IdA помечен,то выполняем п. 1.4.

   1.6 Если в следующем узле IdB не изменился, а IdA не помечен,то переходим к следующему узлу и выполняем п. 1.5.

   1.7 Если в следующем узле IdB изменился, то выполняем п. 1.1.

   Так как индексное дерево содержит n_A узлов, то трудоемкость этого алгоритма оценивается как $\theta(n_A)$.

2. **Отношение наследования: A – родительский, B – дочерний классы**
   Выбираем индексное дерево с ключом (IdB, IdA) и обходим его узлы по индексу:

   2.1 Если для текущего узла объект IdB не помечен, то переходим к следующему узлу дерева. Выполняем п. 2.1.

   2.2 Если объект IdB помечен, а IdA не помечен, то обнуляем пометку IdB в индекс-выборке, переходим к следующему узлу и выполняем п. 2.1.

   2.3 Если объекты IdB и IdA помечены в своих индекс-выборках, то переходим к следующему узлу дерева и выполняем п. 2.1.

   Так как индексное дерево содержит n_B узлов, то трудоемкость этого алгоритма оценивается как $\theta(n_B)$.

3. **Отношение включения: простое включение или все объекты класса C помечены**
   Выбираем индексное дерево с ключом (IdB, IdA) или (IdB, IdA, IdC) и применяем к нему алгоритм 1 (как для отношения наследования).
   Пусть класс C имеет k непосредственных родительских классов (отношение включения связывает $2 + k$ классов) $P_1, \ldots, P_k$. Так как индексное дерево отношения в худшем случае (каждый объект одного класса включен в каждый

объект другого класса с каждым отличным от другого наборов родительских объектов классов $P_1, \ldots, P_k$) содержит $n_A \cdot n_B \cdot n_{P_1} \cdot \ldots \cdot n_{P_k}$ узлов, то трудоемкость этого алгоритма оценивается как $\theta(n_A \cdot n_B \cdot n_{P_1} \cdot \ldots \cdot n_{P_k})$.

4. **Отношение включения: с классом связи и не все объекты класса C помечены**
   Выбираем индексное дерево с ключом (IdB, IdA, IdC) и обходим его узлы по индексу:

   4.1 Если для текущего узла объект IdB не помечен, то пропускаем все узлы до изменения значения IdB. Выполняем п. 4.1.

   4.2 Если объект IdB помечен, а IdA или IdC не помечен, то переходим к следующему узлу с тем же IdB и выполняем п. 4.2.

   4.3 Если в следующем узле IdB изменился, то обнуляем пометку предыдущего значения IdB в индекс-выборке и выполняем п. 4.1.

   4.4 Если объекты IdB, IdA, IdC помечены в своих индекс-выборках, то включаем в список отношения указатель на этот узел, переходим к следующему узлу и выполняем п. 4.5.

   4.5 Если в следующем узле IdB не изменился, а IdA и IdC помечены, то выполняем п. 4.4.

   4.6 Если в следующем узле IdB не изменился, а IdA или IdC не помечен, то переходим к следующему узлу и выполняем п. 4.5.

   4.7 Если в следующем узле IdB изменился, то выполняем п. 4.1.

   Индексное дерево в худшем случае (каждый объект одного класса включен в каждый объект другого класса для каждого объекта каждого родительского класса для класса связи) содержит $n_A \cdot n_B \cdot n_{P_1} \cdot \ldots \cdot n_{P_k}$ узлов, а потому трудоемкость этого алгоритма оценивается как $\theta(n_A \cdot n_B \cdot n_{P_1} \cdot \ldots \cdot n_{P_k})$.

5. **Отношение истории**
   Выбираем индексное дерево с ключом (IdB, IdA) и применяем к нему алгоритм 1 (как для отношения наследования).
   Так как индексное дерево в худшем случае (каждый объект одного класса предшествует каждому объекту другого класса) содержит $n_A \cdot n_B$ узлов, то трудоемкость этого алгоритма оценивается как $\theta(n_A \cdot n_B)$.

6. **Отношение взаимодействия: A, B, C – классы объектов с разными ролями *Откуда, Куда, Что* и H – класс *Как***
   Выбираем индексное дерево с ключом (IdB, IdA, IdC, IdH) и обходим его узлы по индексу:

   6.1 Если для текущего узла объект IdB не помечен, то пропускаем все узлы до изменения значения IdB. Выполняем п. 6.1.

   6.2 Если объект IdB помечен, а один из объектов IdA, IdC, IdH не помечен, то переходим к следующему узлу с тем же IdB и выполняем п. 6.2.

6.3 Если в следующем узле IdB изменился, то обнуляем пометку предыдущего значения IdB в индекс-выборке и выполняем п. 6.1.

6.4 Если объекты IdB, IdA, IdC, IdH помечены в своих индекс-выборках, то включаем в список отношения указатель на этот узел, переходим к следующему узлу и выполняем п.6.5.

6.5 Если в следующем узле IdB не изменился, а IdA, IdC, IdH помечены, то выполняем п. 6.4.

6.6 Если в следующем узле IdB не изменился, а один из объектов IdA, IdC, IdH не помечен,то переходим к следующему узлу и выполняем п. 6.5.

6.7 Если в следующем узле IdB изменился, то выполняем п. 6.1.

Пусть класс H имеет k непосредственных родительских классов (отношение взаимодействия связывает $3 + k$ классов) $P_1, \ldots, P_k$. Индексное дерево в худшем случае (любая комбинация объектов классов A, B, C находится во взаимодействии для каждого объекта каждого класса родительского класса H) содержит $n_A \cdot n_B \cdot n_C \cdot n_{P_1} \cdot \ldots \cdot n_{P_k}$ узлов, а потому трудоемкость этого алгоритма оценивается как $\theta(n_A \cdot n_B \cdot n_C \cdot n_{P_1} \cdot \ldots \cdot n_{P_k})$.

Замечание 1. Алгоритмы коррекции индекс-выборок классов по индекс-выборке отношения наследования различны по той причине, что в одном случае корректируется выборка объектов родительского класса по выборке объектов дочернего класса, а в другом случае наоборот, и это требует различных действий.

Замечание 2. Алгоритм коррекции выборки объектов класса по выборке объектов другого класса, между которыми имеется отношение простого включения, такой же как и для коррекции выборки объектов родительского класса по выборке объектов дочернего класса, хотя трудоемкости в том и другом случае различны. Это объясняется различием в величине деревьев индекс-выборок отношений. В случае отношения истории индекс-выборка отношения организована так же, как и для отношения простого включения. Поэтому и алгоритм тот же.

Алгоритм обхода с возвратом схемы слоев

Полученные индекс-выборки для каждого класса из схемы слоев запроса позволяют получить любой набор связных объектов запроса. Для этого с каждой индекс-выборкой класса схемы слоев и с каждой индекс-выборкой отношений классов схемы слоев связывается указатель на элемент списка. В каждом слое схемы классы упорядочиваются: сначала в порядке отношений с классами предшествующего слоя, а затем по типу отношений (наследования, включения, истории и взаимодействия); причем классы одного взаимодействия идут подряд с последним классом связи *Как* и классы связи идут сразу за соответствующим классом включения в этом слое. Вводится текущий класс схемы (ТК), текущий объект этого класса (ТО), текущий связанный класс с текущим классом (ТС). Определяется стек для откатов при обходе схемы слоев.

1. Указатели всех списков отношений обнуляются. В качестве ТК выбирается опорный класс схемы, указатель ТК устанавливается на 1-й элемент списка ТК, которым и становится ТО. В стек записывается указатель на ТК.

2. В качестве ТС выбирается первый класс слоя с номером на 1 больше номера слоя ТК, если он есть. Если его нет, то происходит переход на п. 7.

3. По схеме слоев определяется отношение между ТК и ТС. Если его нет, происходит переход на п.4. Если это отношение содержит также другие классы, то они также входят в рассмотрение (обозначим их TC_1, TC_2). Определяется индекс-выборка отношения

$$(Id_{TK}, Id_{TC}[, Id_{TC_1}[, Id_{TC_2}]])$$

и указатель отношения устанавливается на следующий (если указатель отношения был пустой, то на 1-й) элемент списка отношения с указателем на Id_{TK} текущего класса, т. е. на ТО (это имеет место, так как в противном случае пометка ТО должна была быть обнулена при выполнении алгоритма коррекции для данного отношения классов ТК и ТС). Указатель на объект класса ТС устанавливается также по этому элементу. В стек записывается указатель на класс ТС.

4. В качестве ТС выбирается следующий класс слоя, связанный отношением с ТК, если он есть, и происходит переход на п. 3.

5. Если в слое нет следующего класса ТС, связанного отношением с ТК, то в качестве ТК выбирается следующий класс слоя ТК, если он есть, а в качестве ТО – объект, который определяется по указателю списка ТК. Происходит переход на п. 4.

6. Если нет следующего класса слоя ТК, то в качестве ТК выбирается первый класс слоя ТС, а в качестве ТО – объект, который определяется по указателю списка ТК, и происходит переход на п. 2.

7. Набор связанных объектов запроса закончен (они определяются по значениям указателей на элементы списков индекс-выборок классов. Для него проверяется условие фразы **where** и, если оно выполнено, по нему вычисляется кортеж значений фразы **select**.

8. Производится откат для выбора следующего набора объектов. Для этого выбирается вершина стека и по указателю в ней выбирается ТС. По отношению, в которое входит этот класс с классом в предыдущем слое, если он есть, выбирается ТК. Если такого класса нет, то происходит переход к п. 11. По индекс-выборке отношения между ТК и ТС находится указатель элемента списка отношения. Он устанавливается на следующий элемент списка, если он есть и объект ТК не меняется, и происходит переход на п. 10.

9. Если нет следующего элемента списка или объект ТК меняется, то указатель элемента списка индекс-выборки отношения обнуляется и происходит переход на п. 8.

10. Указатель элемента списка отношения определяет указатели на объекты ТС и ТО. В стек записывается указатель на класс ТС и происходит переход на п. 4.

11. В качестве ТК выбирается опорный класс схемы, указатель ТК устанавливается на следующий элемент списка ТК (если его нет, то алгоритм завершен), которым и становится ТО. В стек записывается указатель на ТК и происходит переход на п. 2.

Трудоемкость алгоритма обхода с возвратом схемы слоев является в общем случае экспоненциальной, так как алгоритм обхода дерева слоев является переборным. Например, если схема слоев содержит m вершин-классов, а индекс-выборка каждого класса после использования алгоритма коррекции индекс-выборок содержит не более p объектов, то трудоемкость алгоритма обхода схемы слоев можно оценить как $\theta(m^p)$. Однако, так как сложность выбора каждого кортежа значений запроса в данном алгоритме оценивается как $\theta(m)$, то при выборе запросом k кортежей значений и предположении, что ограничение **where** отсекает число кортежей порядка $\theta(k)$, трудоемкость такого запроса оценивается как $\theta(m \cdot k)$.

Покажем на некотором примере отношений объектов, что проектирование сложного запроса для модели DIM может быть проще, чем для реляционной модели, а трудоемкость запроса при этом не больше трудоемкости запроса для реляционной модели.

5. Пример

Рассмотрим следующий пример:
Корабли из множества $Ships$, число которых n_s, перевозят грузы из множества $Cargoes$, число которых n_c, в порты из множества $Ports$, число которых n_p. При этом корабль s везет груз c в порт p в количестве $k(s, p)$.

Необходимо определить схему базы данных (БД) и написать следующий запрос к ней:
Список наименований грузов, которые везут наиболее нагруженные корабли и при этом в наименьшее число портов среди таких кораблей.
Имеется в виду, что из всех кораблей необходимо выделить имеющие максимальный суммарный груз, а среди них взять те корабли, которые идут в наименьшее число портов среди таких кораблей, а затем для таких кораблей нужно взять общий список всех грузов, перевозимых ими, выделив в нем неповторяющиеся наименования.

Сначала определим схему реляционной БД и запрос к ней. В этом случае выделим таблицы:

$Ships$ с полями $(IdShip, ShipName)$, главным индексом $IdShip$,

$Cargoes$ с полями $(IdCargo, CargoName)$, главным индексом $IdCargo$ и

$Ports$ с полями $(IdPort, PortName)$, главным индексом $IdPort$, а также таблицу связи

Ships_Cargoes с полями $(IdShip, IdCargo, IdPort, Quantity)$, главным индексом $(IdShip, IdCargoes, IdPort)$ и тремя внешними индексами $IdShip$, $IdCargoes$, $dPort$.

Запрос, оптимальный по трудоемкости, может выглядеть следующим образом:

```
Select Distinct c.CargoName
  From ( Select IdShip
         From Ships s,
           ( Select s.IdShip,
             ( Select Sum (Quantity) as SumC – сумма грузов корабля
               From Ships_ Cargoes
               Where IdShip=s.IdShip
             ),
             ( Select Count (IdPort) as CountP – кол-во портов груз. кораб.
               From ( Select IdPort
                      From Ships_ Cargoes
                      Where IdShip=s.Id
                      Group by IdPort
                    )
             )
           )
         Where SumC = ( Select Max (SummC) – max сумма грузов корабля
                        From ( Select Sum (Quantity) as SummC
                               From Ships_ Cargoes
                               Where IdShip=s.IdShip
                             )
                      ) And
               CountP = ( Select Min (CntP) – min кол-во порт. груз. корабля
                          From ( Select Count (IdPort) as CntP
                                 From ( Select IdPort
                                        From Ships_Cargoes
                                        Where IdShip=s.Id
                                        Group by IdPort
                                      )
                               )
                        ),
               Ships_ Cargoes sc,
               Cargoes c
         Where sc.IdShip = IdShip And
               c.Id = sc.IdCargo
```

Отметим, что такой запрос непросто написать. Анализ запроса показывает, что его трудоемкость оценивается как $\theta(n_c(n_s n_p + \log n_c))$.

Определим теперь БД DIM и запрос к ней. Она описывается тремя классами (подобно таблицам реляционной БД):

Ships с параметрами $(IdShip, ShipName)$,

$Cargoes$ с параметрами $(IdCargo, CargoName)$ и

Ports с параметрами $(IdPort, PortName)$, а также классом связи *Ships_Cargoes* с параметрами $(IdShip\_Cargo, IdPort, Quantity)$, который имеет родителем класс *Ports*.
Между классами *Ships* и *Cargoes* устанавливается отношение включения с классом связи *Ships_Cargoes*.

Запрос на SODQL имеет достаточно понятную (с точки зрения задачи) структуру:

select *CargoName* **from** *Cargoes*
where *Ships.obj* **in** (**select objmincount** (*Ports.obj*) **on** *Ships* **from** *Ships*
where *Ships.obj* **in**
(**select objmaxsum** (*Quantity*) **on** *Ships* **from** *Ships*))

Он выделяет список имен грузов (главный запрос) для объектов (*Ships.obj*) класса *Ships* (кораблей), которые везут максимальный груз (второй подзапрос) и при этом направляются в минимальное (среди кораблей уже выделенных вторым подзапросом) число портов (первый подзапрос).

Но для более рационального выполнения запроса лучше избавиться от фразы **where**, перенеся подзапросы в фразу **from**. При этом ODQL-запрос будет выглядеть следующим образом:

select *c.CargoName* **from** *Cargoes c, Ships_ Cargoes sc,*
((**select objmaxsum** (*sc.Quantity*) **on** *s* **from** *Ships s* **links** *s* **contains**(*sc*) *c*) *s1*
intersection
(**select objmincount** (*p.obj*) **on** *s1* **from** *s1, Ports p* **links** *s1* **contains**(*sc*) *c*,
p **parent** *sc*)
) *s2*
links *s2* **contains**(*sc*) *c*

Теперь сначала выполняется первый подзапрос, который выделяет корабли (объекты класса s), которые наиболее нагружены (по индекс выборке отношения классов s, c через класс связи *sc* и агрегатной функции **select objmaxsum** (*sc.Quantity*) определяет индекс-выборку класса s). Его трудоемкость $\theta(n_s n_c)$.

Вслед за ним выполняется второй подзапрос, который из уже выделенных наиболее нагруженных кораблей (индекс-выборки $s1$ класса s) выделяет те из них, которые перевозят грузы в минимальное для этой группы число портов (по индекс-выборке отношения классов s, c через класс связи *sc* и индекс-выборке отношения классов *sc* и p, а также агрегатной функции **select objmincount** (*p.obj*) корректирует индекс-выборку класса s). Его трудоемкость $\theta(n_s n_c n_p)$.

Наконец, выполняется главный запрос, который определяет сначала объекты класса c (индекс-выборку класса c) (по индекс выборке отношения классов s, c через класс связи *sc*), а затем список имен грузов *c.CargoName*, которые везут эти корабли. Его трудоемкость $\theta(n_s n_c + n_c) = \theta(n_s n_c)$.

Трудоемкость запроса, равная сумме трудоемкостей этих подзапросов и главного запроса определяется как $\theta(n_s n_c n_p)$, что даже лучше, чем для аналогичного запроса реляционной БД, где условие **Distinct** требует устранения повторяющихся имен.

6. Заключение

Таким образом, наглядность запросов для языка объектных запросов ODQL лучше, чем для языка SQL реляционных запросов. Что касается трудоемкости запроса, то заметим, что для k-арного отношения таблиц РСУБД, участвующих в запросе, трудоемкость такого запроса не ниже, чем $\theta(n^k)$, где n – средняя длина таблиц. А для ODQL-запроса трудоемкость может быть выше за счет обхода дерева объектов, удовлетворяющих запросу. Но, как мы показали выше, при участии в запросе k классов (соответствующих таблицам РСУБД) и ожидании ответа на запрос из m кортежей трудоемкость такого обхода дерева составляет $\theta(k \cdot m)$. Поэтому в целом трудоемкость ODQL-запросов в основном близка к трудоемкости SQL-запросов. Это дает основание для построения эффективной реализации запросов ODQL.

Список литературы

1. Рублев В.С. Язык объектных запросов динамической информационной модели DIM // Моделирование и анализ информационных систем. 2010. Т. 17, №3. С. 144–161.

2. Писаренко Д.С., Рублев В.С. Объектная СУБД Динамическая информационная модель DIM и ее основные концепции // Моделирование и анализ информационных систем. 2009. Т. 16, №1. С. 62–91.

3. Рублев В.С. Запросная полнота языка ODQL динамической информационной модели DIM // Ярославский педагогический вестник. Серия "Физико-математические и естественные науки". Ярославль, 2011. Вып. 1.

4. Jarke M., Koch J. Query Optimization in Database Systems // Computing Surveys. June 1984. Vol. 16, №2.

Object Query Computing Optimization in the Dynamic Information Model DIM

Roublev V.S.

Keywords: object DBMS, query language, query computational complexity, optimization of computational complexity

The problem of algorithm for organization of the object queries execution [1] for a new object DBMS technology DIM [2] is considered such that the computational complexity of the queries execution is as little as possible. Solutions of this problem may be taken as a basis for the development of a query compiler.

Сведения об авторе: Рублев Вадим Сергеевич,
Ярославский государственный университет им. П.Г. Демидова,
профессор кафедры теоретической информатики