УДК 519.7: 007 + 06

ОПЕРАТОР SELECT В ЯЗЫКЕ N-DECLARATIVE LANGUAGE

О.В. ОЛЬХОВИК
(Донской государственный технический университет)

Описаны синтаксис и семантика оператора SELECT в языке N-Declarative Language. Показано, что для каждого класса в смысле N-модели данных может быть построено отношение, содержащее значения атрибутов, информативных на этом классе, а оператор SELECT реализует операции реляционной алгебры и другие специальные операции над подмножествами этого отношения.
Ключевые слова: *Model-driven engineering, информационные системы, базы данных, языки запросов данных, оператор SELECT.*

Введение. Современные Model-driven engineering (MDE) технологии предназначены для улучшения коммуникаций между участниками проекта и для генерации программного кода на основе проектных диаграмм. Однако они повышают стоимость разработки и сопровождения программ из-за необходимости тратить усилия отдельно и на поддержку проектных диаграмм, и на создание программного кода. Источником проблемы здесь является невозможность полной автоматической генерации программного кода, а нотации, предназначенные для кодогенерации, настолько усложняются, что их коммуникативная ценность сводится к нулю, и их разработка не менее сложна, чем построение кода.

В статье [1] предложена N-модель данных, на основе которой разработаны декларативный язык запросов N-Declarative Language (NDL) [2] и визуально-декларативный язык проектирования N-Visulal Language (NVL) [3]. Принципы, заложенные в NVL, позволяют сократить цикл производства программного обеспечения информационных систем (ПО ИС) посредством полной автоматической кодогенерации структур данных и бизнес-логики и повысить коммуникативный эффект за счет более простого и понятного визуально-декларативного языка построения проектных диаграмм.

С другой стороны, стоимость сопровождения ПО ИС может быть существенно снижена путем уменьшения количества обращений пользователей за счет самостоятельного решения ими информационных задач. Этого можно добиться посредством увеличения числа запросов «ad-hoc», выполняемых пользователями без помощи программистов (возможно, с помощью соответствующих интерфейсов, например, интерактивных отчетов). Перспектива увеличения числа запросов «ad-hoc», выполняемых пользователями, видится в замене Structured Query Language (SQL) на более простой SQL-подобный язык без потери выразительной мощности. В качестве такого языка предлагается NDL.

Поскольку основным оператором любого языка, оперирующего структурированными данными, является оператор выборки SELECT, рассмотрению его синтаксиса и семантики в языке NDL и посвящена данная работа.

Синтаксис оператора SELECT в NDL [2]. SELECT в NDL состоит из предложений аналогичных предложениям оператором выборки в SQL: SELECT, FROM, WHERE, GROUP BY, HAVING и ORDER BY. Предложение SELECT определяет множество атрибутов, формирующих результат выборки. Предложение FROM определяет источник выборки. Здесь принципиальное отличие NDL состоит в том, что источником выборки может служить только один класс (или категория), а не набор таблиц, как в SQL. Предложение WHERE определяет предикат, ограничивающий результат выборки. Предложение GROUP BY задает группировку, а предложение HAVING ограничивает результат группировки предикатом, заданным на ее результате. Предложение ORDER BY реализует сорти-

ровку результата. При этом сортировка может выполняться только по именованным атрибутам. В общем случае синтаксис оператора SELECT в NDL таков:

<выборка> ::=
SELECT [DISTINCT] [*] [<операционное задание> [AS "<имя>"]
(, <операционное задание> [AS <имя>]..)]
FROM <имя>
[WHERE <условие>]
[GROUP BY <имя> (,<имя>..)]
[HAVING <условие>]
[ORDER BY <имя> [DESC] (,<имя> [DESC]..)]
<операционное задание>::= {(<операционное задание>)| <операнд>}
[<операция> <операционное задание>]
*<операция> ::= {INTERSECT| MINUS| UNION| +| -| *| //| '||'}*
<операнд> ::= {<атрибут-операнд>| <значение>|
<унарная функция>| <бинарная функция>}
<атрибут-операнд>::={<имя>|{<имя>|PARENT}.<имя>| <имя>:<имя>}
<значение>::={<целое число>| <вещественное число>| <литерал>}
<бинарная функция>::= <имя> (<операнд>,<операнд>)
<унарная функция>::= {INV| COUNT| MIN| MAX| SUM|
AVG| ALL|<имя>} (<операнд>)
<условие> ::= {(<условие>)| NOT <условие> |<сравнение>}
[{AND|OR} <условие>]
<сравнение> ::= <операционное задание> {<rel-op> <операционное задание>
| BETWEEN <значение> AND <значение>
| IN <операционное задание>| '('<значение> (,<значение>..)')'
| STARTING <значение>
| CONTAINING <значение>}
<rel-op>::= {=|<>|>|<|=>|<=}
<имя> ::= <буква>(<буква>|<цифра>)
<цифра>::= {0,1,2,3,4,5,6,7,8,9}
<буква>::= {a..z, A..Z, а..я, А..Я, _}

Семантика оператора SELECT в NDL. Семантику оператора SELECT определим на основе базовых конструкций N-модели данных, описанных в [1]. Базовым понятием N-модели является объект, который рассматривается как множество образов. Объектам в реальном мире могут соответствовать конкретные предметы и явления, а также абстрактные понятия предметной области, причем не обязательно вербализованные в естественных языках. Поскольку объекты суть множества, на них определены базовые теоретико-множественные отношения и операции. Теоретико-множественное отношение нестрогого включения при этом трактуется как отношение наследования. Все объекты различимы и индексируются натуральными числами идентифицирующей функцией Id: $X \rightarrow N$.

Свойства объектов определяются атрибутами, которые представляют собой бинарные функции, определенные на множестве объектов. Значения каждого атрибута лежат в области определенного simple-типа данных (целые, вещественные числа, литералы и т. д.). На произвольном объекте атрибут в общем случае возвращает некоторое множество значений одного типа данных. Атрибут также может ссылаться на объекты. При этом его значение на объекте в общем случае представляет собой множество идентификаторов объектов.

Поскольку атрибуты – это функции, они могут задаваться различным способом. Если атрибут задается перечислением пар <объект, значение>, то он называется исходным. Если атрибут задается формулой, то он считается расчетным. Формулы представляют собой инфиксную запись последовательности операций над атрибутами, определенных в [1].

Существует требование к аналитической различимости объектов. Для любой пары различных объектов должен существовать хотя бы один исходный атрибут, принимающий на них различные значения неравные неопределенности.

Важным является определение терминальности. Атрибут терминален на объекте, если его значения на любом подмножестве объекта равны с точностью до перестановки. Если атрибут не терминален на объекте, то его значение на нем представляет собой объединение значений этого атрибута на всех его подмножествах.

Классом называется объект, в котором можно выделить непересекающиеся подмножества такие, что существует некоторое непустое множество исходных атрибутов, каждый из которых терминален на любом из этих подмножеств. Данные подмножества класса называются его экземплярами, а множество атрибутов называется множеством атрибутов терминальных на классе.

Класс наследует от другого класса, если является его подмножеством. При этом каждый экземпляр класса-потомка является наследником (подмножеством) строго одного экземпляра класса-предка. Наследование может быть простым или кратным. В случае простого наследования (будем рассматривать именно такой случай) существует функция Parent: $Id(X) \rightarrow Id(X)$, которая по идентификатору экземпляра возвращает идентификатор его непосредственного предка.

Множество терминальных атрибутов класса-предка является подмножеством терминальных атрибутов класса-потомка. Атрибут определен на классе, если терминален на нем, но не терминален на любом из его предков.

В иерархии классов существует верхняя грань – класс x_0, являющийся предком любого класса. На нем определены все константы (атрибуты, значения которых одинаковы для всех объектов), в том числе внешние переменные, которые можно считать константами в любой отдельно взятый момент существования базы данных.

Категория – это подмножество класса, состоящее из его экземпляров, на которых истинен некоторый предикат. Все, что далее будет говориться о классах, может быть обобщено и для категорий.

База данных представляет собой схему, включающую в себя классы и категории с определенными на них атрибутами, и множество экземпляров классов данной схемы.

Теперь перейдем непосредственно к семантике оператора SELECT. Оператор SELECT в применении к некоторому классу оперирует атрибутами потока этого класса.

Пусть $C = \{c_1, c_2, \ldots, c_n\}$ $n \in N$ – конечное множество классов, определенное в заданной схеме. На произвольном классе c_i $(1 \le i \le n)$ определено некоторое множество атрибутов Υ_i с тем же индексом. Множество атрибутов, определенных в классе c_i или в его ветви наследования обозначим:

$$\Omega_i = \{f / f \in \Upsilon j,\ c_j = c_i \vee c_j \subset c_i \vee c_j \supset c_i\}.$$

Множество экземпляров класса c_i обозначим $Ext(c_i)$. Тогда рекурсивно поток s_i класса c_i – это $S_i = \bigcup_j \Upsilon_j$ таких, что $\Upsilon_j \subseteq \Omega_i$ или $\exists$ g:

$$X \rightarrow B(Ext(c_k)) \ \&\ Ext(c_i) \subseteq X \ \&\ \Upsilon_j \subseteq s_k.$$

Иными словами, в поток класса входят атрибуты, определенные на нем, или на его предках, или на его потомках, или входящие в потоки классов, на которые он ссылается. Поток класса позволяет определить имена атрибутов, которые доступны в операторе SELECT, построенном на заданном классе.

Помимо значений атрибутов, определенных в ветви наследования класса, или полученных из них операцией композиции, оператор SELECT может возвращать значения атрибутов, полученных в результате применения над атрибутами потока любых других операций, описанных в [1]. Определим множество информативных на произвольном классе c_i атрибутов ϑ_i как множество атрибутов, соответствующих правилам ***P1 – P4***:

P1. $f \in \Omega_i \Rightarrow f \in \vartheta_i$.

P2. $f = h(g) \Rightarrow f \in \vartheta_i$, где $g \in \vartheta_i$, $Im_g \cap Def_h \neq \varnothing$, $h \in s_i$.

P3. $f = \bullet\, g \Rightarrow f \in \vartheta_i$, где $\bullet$ – унарная операция или агрегация над атрибутом в смысле [1], $g \in \vartheta_i$.

P4. $f = g \circ h \Rightarrow f \in \vartheta_i$, где $\circ$ – бинарная или теоретико-множественная операция над атрибутами в смысле [1], $g, h \in \vartheta_i$.

Пусть c_i – произвольный класс в базе данных, $Ext(c_i) = \{x_1,x_2,...,x_q\}$, $q \in N$ – множество его экземпляров в определенный момент времени, и $\vartheta_i = \{f_1,f_2,...,f_m\}$, $m \in N$ – множество информативных на нем атрибутов. Тогда можно построить отношение $\Re_i$, в котором каждый экземпляр x класса c_i порождает ровно один кортеж (Id(x), Parent(x), $f_1(x)$, $f_2(x)$,..., $f_m(x)$) и никаких других кортежей не содержится:

Id	**Parent**	**f_1**	**f_2**	**...**	**f_m**
$Id(x_1)$	Parent (x_1)	$f_1(x_1)$	$f_2(x_1)$	**...**	$f_m(x_1)$
$Id(x_2)$	Parent (x_2)	$f_1(x_2)$	$f_2(x_2)$	**...**	$f_m(x_2)$
...	...	**...**	**...**	**...**	**...**
$Id(x_q)$	Parent (x_q)	$f_1(x_q)$	$f_2(x_q)$		$f_m(x_q)$

Можно было бы сказать, что результат оператора SELECT на произвольном классе c_i – это результат применения операций реляционной алгебры и/или операций группировки и сортировки над отношением $\Re_i$. Однако с отношением $\Re_i$ связана проблема практической интерпретации. Возникает она из-за того, что информативный на классе атрибут в общем случае может принимать на его экземплярах значения, по сути являющиеся множествами значений simple-типов данных. Если информативный атрибут на каждом экземпляре класса принимает строго одно значение simple-типа, то, согласно [1], он называется ординарным, в противном случае – множественным. Причем, если атрибут определен на классе как ординарный, то он является множественным на его классах-предках. Проблема собственно заключается в интерпретации множественных атрибутов, если они входят в $\Re_i$.

Как вариант решения проблемы можно предложить ограничить множество атрибутов, из которых строится $\Re_i$, только ординарными на классе c_i. Обозначим $\vartheta'_i=\{f/f\in\vartheta_i, \forall x\in Ext(c_i)\ |f(x)|=1\}$. Допустим, что для произвольного класса c_i существует $\vartheta'_i = \{f_1,f_2,...,f_k\}$, $k\in N$. Тогда в отношении $\Re_i$ каждый экземпляр x класса c_i порождает ровно один кортеж (Id(x), Parent(x), $f_1(x)$, $f_2(x)$,..., $f_k(x)$) и никаких других кортежей не содержится:

Id	**Parent**	**f_1**	**f_2**	**...**	**f_k**
$Id(x_1)$	Parent (x_1)	$f_1(x_1)$	$f_2(x_1)$	**...**	$f_k(x_1)$
$Id(x_2)$	Parent (x_2)	$f_1(x_2)$	$f_2(x_2)$	**...**	$f_k(x_2)$
...	...	**...**	**...**	**...**	**...**
$Id(x_q)$	Parent (x_q)	$f_1(x_q)$	$f_2(x_q)$		$f_k(x_q)$

Недостаток данного решения – ограничение выразительной мощности оператора SELECT.

На практике множественные атрибуты могут интерпретироваться как коллекции. В этом случае отношение $\Re_i$ можно переопределить посредством функции кодирования Code: $B(P)\rightarrow N$, определенной в булеане множества P значений simple-типов данных. Тогда в отношении $\Re_i$ каждый экземпляр x класса c_i порождает ровно один кортеж (Id(x), Parent(x), $Code(f_1(x))$, $Code(f_2(x))$,..., $Code(f_m(x))$) и никаких других кортежей не содержится:

Id	**Parent**	**f_1**	**f_2**	**...**	**f_m**
$Id(x_1)$	Parent (x_1)	$Code(f_1(x_1))$	$Code(f_2(x_1))$	**...**	$Code(f_m(x_1))$
$Id(x_2)$	Parent (x_2)	$Code(f_1(x_2))$	$Code(f_2(x_2))$	**...**	$Code(f_m(x_2))$
...	...	**...**	**...**	**...**	**...**
$Id(x_q)$	Parent (x_q)	$Code(f_1(x_q))$	$Code(f_2(x_q))$		$Code(f_m(x_q))$

Проблематичность этого решения заключается в том, что для обработки результата SELECT реляционных операторов будет недостаточно. Потребуются раскодирование и перебор-

ные операции, которые могут быть реализованы только на императивном или функциональном языке и, возможно, только процедурно. Это приведет к необходимости совместного использования различных парадигм программирования, что усложнит как семантику, так и практическую реализацию оператора SELECT.

Последний вариант решения проблемы множественных атрибутов заключается в снятии неявного требования функциональной зависимости атрибутов $f_1,f_2,\ldots,f_m$ от Id в отношении $\Re_i$. Отношение $\Re_i$ ограничим таким образом, что каждый экземпляр x_j класса c_i продуцирует в отношении $\Re_i$ множество кортежей равное декартову произведению значений функций Id, Parent, f_1, $f_2,\ldots$, f_m на этом экземпляре. Пусть c_i – произвольный класс в базе данных, $Ext(c_i) = \{x_1,x_2,\ldots,x_q\}$, $q \in N$ – множество его экземпляров в определенный момент времени, $\Im_i = \{f_1,f_2,\ldots,f_m\}$, $m \in N$ – множество информативных на нем атрибутов. Тогда можно построить отношение:

$$R_i = \bigcup_{j=1}^{j=q} \mathrm{Id}\left(x_j\right) \times \mathrm{Parent}\left(x_j\right) \times f_1\left(x_j\right) \times f_2\left(x_j\right) \times \ldots \times f_m\left(x_j\right). \quad (1)$$

Продемонстрируем построение $\Re_i$ на примерах. Предварительно заметим, что число информативных на классе атрибутов бесконечно, поскольку бесконечно число атрибутов, порождаемых операциями. Поэтому в рассматриваемых примерах (и только в них) ограничим множество $\Im_i$ атрибутами потока класса, который в любой момент существования базы данных конечен. Допустим, имеется класс c_1, содержащий экземпляры x_1 и x_2, и на нем определены атрибуты f_1 и f_2. При этом $Id(x_1) = 91$, $Id(x_2) = 92$, $f_1(x_1) = \{1,0\}$, $f_2(x_1) = 10$, $f_1(x_2) = 1$, $f_2(x_2) = 20$. Тогда отношение $\Re_1$, построенное на классе c_1, будет выглядеть так:

Id	**f_1**	**f_2**
91	1	10
91	0	10
92	1	20.

Теперь добавим в предыдущий пример класс c_2, который наследует от класса c_1. Допустим, что c_2 содержит экземпляры x_3, x_4 и x_5 и $Id(x_3)=93$, $Id(x_4)=94$, $Id(x_5)=95$. При этом x_3 и x_4 наследуют от экземпляра x_1 класса c_1, а x_5 наследует от x_2. На c_2 определен атрибут f_3: $f_3(x_3)=300$, $f_3(x_4)=400$, $f_3(x_5)=500$. Тогда отношение $\Re_1$, построенное на классе c_1, будет таким:

Id	**f_1**	**f_2**	**f_3**
91	1	10	300
91	1	10	400
91	0	10	300
91	0	10	400
92	1	20	500.

Отношение $\Re_2$, построенное на классе c_2, будет выглядеть следующим образом:

Id	**Parent**	**f_1**	**f_2**	**f_3**
93	91	1	10	300
93	91	0	10	300
94	91	1	10	400
94	91	0	10	400
95	92	1	20	500.

В последнем примере добавим к классам c_1 и c_2 класс c_3 с экземплярами x_6 и x_7. Допустим, что $Id(x_6)=10$, а $Id(x_7)=20$. На c_3 определен атрибут f_4: $f_4(x_6)=1000$, $f_4(x_7)=2000$. Кроме того, атрибут f_2, определенный на c_1, будем трактовать как ссылку на класс c_3. Тогда на c_3 определен расчетный атрибут $f_5=Inv(c_1,f_2)$, что, согласно [1], означает обратную ссылку на класс c_2, полученную в результате операции инволюции. Тогда отношение $\Re_1$, построенное на классе c_1, таково:

Id	f_1	f_2	f_3	$f_2.f_4$
91	1	10	300	1000
91	1	10	400	1000
91	0	10	300	1000
91	0	10	400	1000
92	1	20	500	2000.

Отношение $\Re_2$, построенное на классе c_2:

Id	Parent	f_1	f_2	f_3	$f_2.f_4$
93	91	1	10	300	1000
93	91	0	10	300	1000
94	91	1	10	400	1000
94	91	0	10	400	1000
95	92	1	20	500	2000.

Отношение $\Re_3$, построенное на классе c_3:

Id	f_4	f_5	$f_5.f_1$	$f_5.f_3$
10	1000	91	1	300
10	1000	91	0	300
10	1000	91	1	400
10	1000	91	0	400
20	2000	92	1	500.

Главным недостатком последнего из предложенных здесь вариантов решения проблемы множественных атрибутов является «скрытая угроза» экспоненциального возрастания мощности результирующего отношения при увеличении числа таковых в запросе. Угроза является скрытой, поскольку декартово произведение при формировании результатов запроса выполняется неявно в зависимости от свойств атрибутов. Запрос, выдающий практически неприемлемый по мощности результат, в языке NDL выглядит довольно «невинно», в отличие от SQL, где аналогичный запрос будет иметь более громоздкую конструкцию с явным указанием операции соединения, продуцирующей декартово произведение.

Тем не менее, мы предпочитаем предоставить программистам возможность ошибаться, чем делает невозможным решение некоторых задач, как в первом варианте, или же совмещать различные парадигмы программирования с непредсказуемым эффектом, как во втором. Необходимо только предупредить программистов о «скрытой угрозе», и количество ошибок не будет слишком большим.

Таким образом, здесь и далее мы будем использовать для $\Re_i$ последнее определение. Это подразумевает, что в любой нашей реализации NDL оператор SELECT будет иметь следующую семантику:

S1. Результат применения оператора SELECT к произвольному классу c_i – это всегда некоторое отношение, полученное путем последовательного применения операций реляционной алгебры и/или операций группировки и сортировки над отношением $\Re_i$, которое определяется выражением (1).

Семантика предложений оператора SELECT в NDL. В соответствии с правилом ***S1*** результат оператора SELECT на классе можно представить как результат применения операций реляционной алгебры и/или операций группировки и упорядочивания к отношению, содержащему значения информативных на этом классе атрибутов. Здесь и далее мы будем использовать операции реляционной алгебры Кодда.

Еще в «Третьем манифесте» [4] были четко сформулированы признаки нереляционности SQL. И хотя авторы манифеста посчитали это недостатком SQL, практика показала обратную картину: для выражения многих важных классов информационных запросов одной реляционной алгебры недостаточно. Хотя следующее высказывание может показаться противоречащим преды-

дущему, мы поддерживаем мнение Дейта и Дарвена, что нереляционность SQL – это «зло». Но, хотим мы этого или нет, от этого «зла» никуда не деться. Во-первых, мы не можем уйти от упорядоченности результата запроса – порядок как по столбцам, так и по строкам часто критически важен не только для представления, но и для программной обработки результата запроса. Во-вторых, устранение кортежей-дубликатов в некоторых случаях приводит к невыразимости информационной потребности в запросе. В-третьих, многие аналитические запросы оперируют сгруппированными таблицами, которые нельзя получить посредством операций реляционной алгебры над исходным отношением. Поэтому мы сознательно вводим операции группировки и упорядочивания, семантика которых определена в стандарте SQL ISO/IEC-9075-01:2008 [5].

Кроме того, условимся обозначать name(z) – имя объекта z (атрибута или класса) в базе данных в произвольный момент времени.

S2. Предложение FROM оператора SELECT определяет класс, на котором выполняется этот оператор.

S3. Предложение SELECT оператора SELECT на классе c_i реализует операцию проекции реляционной алгебры на отношении $\Re_i$ (при этом явно задается порядок столбцов в результате):

SELECT name(A), name(B),.., name(C)
FROM name(c_i)→
PROJECT $\Re_i${A,B,…,C},

где A,B,..,C – атрибуты, информативные на c_i.

Примечания:

1. В операторе "SELECT * FROM name(c_i)" символ "*" – это сокращенная запись имен атрибутов, определенных на c_i.

2. В операторе "SELECT DISTINCT name(A), name(B),.., name(C) FROM name(c_i)" из результата выборки устраняются кортежи-дубликаты.

3. В операторе "SELECT DISTINCT name(A) AS "<имя>" FROM name(c_i)" имени атрибута A явно задается значение <имя>.

4. Предложение WHERE может содержать функцию EXT(name(c_j)), возвращающую множество идентификаторов класса или категории c_j.

S4. Предложение WHERE оператора SELECT на классе c_i реализует операции ограничения, пересечения, объединения и вычитания реляционной алгебры на подмножествах отношении $\Re_i$:

SELECT name(A), name(B),.., name(C)
FROM name(c_i)
WHERE $\wp$→
PROJECT $\Re_i${A,B,…,C} WHERE $\wp$,

где $\wp$ – предикат, определенный на $\Re_i$, синтаксис которого описывается нетерминальным символом <условие>.

SELECT name(A), name(B),.., name(C)
FROM name(c_i)
WHERE $\wp_1$ AND $\wp_2$ →
PROJECT $\Re_i${A,B,…,C} WHERE $\wp_1$ INTERSECT
PROJECT $\Re_i${A,B,…,C} WHERE $\wp_2$,
где $\wp_1$, $\wp_2$ – предикаты, определенные на $\Re_i$.
SELECT name(A), name(B),.., name(C)
FROM name(c_i)
WHERE $\wp_1$ OR $\wp_2$ →
PROJECT $\Re_i${A,B,…,C} WHERE $\wp_1$ UNION
PROJECT $\Re_i${A,B,…,C} WHERE $\wp_2$.
SELECT name(A), name(B),.., name(C)
FROM name(c_i)

WHERE NOT $\wp \rightarrow$

$\Re_i$ MINUS PROJECT $\Re_i${A,B,…,C} WHERE $\wp$.

Примечание.

В операторе" SELECT name(A), name(B),.., name(C) FROM name(c_i)

WHERE name(A) IN name(B) " запись " name(A) IN name(B)" выражает предикат, который истинен на кортеже, если значение атрибута A является нестрогим подмножеством значения атрибута B в этом кортеже.

S5. Предложение GROUP BY оператора SELECT на классе c_i реализует нереляционную операцию группировки (выделение реляционного множителя со сверткой на нем значений атрибутов агрегатными функциями) на отношении $\Re_i$ или его подмножестве:

SELECT name(A),…, name(B),.., AGR(name(C)),…, AGR(name(D))

FROM name(c_i)

GROUP BY name(A),…, name(B) $\rightarrow$

GROUPING $\Re_i${A,…,B) WITH (AGR(C),…, AGR(D)),

где AGR – агрегатная функция (COUNT| MIN| MAX| AVG| SUM).

Примечание.

Оператор "SELECT AGR(name(C)),…, AGR(name(D)) FROM name(c_i)" неявно формирует сгруппированную таблицу, содержащую ровно один кортеж, в которой происходит свертка значений атрибутов C,…,D всех кортежей исходной таблицы.

S6. Предложение HAVING оператора SELECT на классе c_i реализует операции ограничения, объединения, пересечения и дополнения реляционной алгебры на отношении, полученном в результате группировки $\Re_i$ или его подмножества.

SELECT name(A),…, name(B),.., AGR(name(C)),…, AGR(name(D))

FROM name(c_i)

GROUP BY name(A),…, name(B)

HAVING $\wp \rightarrow$

GROUPING $\Re_i${A,…,B) WITH (AGR(C),…, AGR(D))

WHERE $\wp$,

где $\wp$ – предикат, который определен на результате группировки.

S7. Предложение ORDER BY оператора SELECT на классе c_i задает отношение алгебраического порядка на кортежах подмножества отношения $\Re_i$. Сортировка осуществляется по возрастанию значений атрибутов, перечисленных в предложении ORDER BY в том порядке, в котором они перечислены. Если перед именем атрибута указано слово DESC, то сортировка по этому атрибуту осуществляется по убыванию.

О полноте оператора SELECT в NDL. Язык запросов, представленный в NDL оператором SELECT, вычислительно не полон. Здесь можно выполнить транзитивное замыкание, но некоторые частично рекурсивные функции, в которых последующие значения вычисляются по предыдущим, неизвестным на момент начала вычислений, средствами NDL описать нельзя. Мы расцениваем это, скорее, как достоинство, поскольку, с одной стороны, подобные функции, как и любые другие, можно реализовать на вычислительно полном языке, императивном или функциональном, и получать их значения в операторе SELECT с помощью вызова. С другой стороны, вычислительная неполнота языка позволяет надеяться на разрешимость его синтаксически-правильных конструкций.

Язык запросов в NDL реляционно не полон. Все базовые операции реляционной алгебры Кодда, за исключением декартова произведения, и производным от нее операциям соединения и деления, реализуются правилами S3 и S4.

Что же касается декартова произведения, то, согласно (1), оно уже частично реализовано в отношении $\Re$, содержащем информативные на классе атрибуты. Можно показать, что произвольный класс c_i в смысле N-модели данных представим в виде одного отношения r_i или, в случае

если на нем определены множественные атрибуты одного основного r_i и нескольких дочерних отношений. При этом атрибут-ссылка в классе c_i может быть представлен как внешний ключ, определенный на r_i или на дочернем отношении. Следовательно, любую схему базы данных Sh в смысле N-модели, не содержащую категорий, можно представить в виде некоторой схемы реляционной базы данных ShR. Таким образом, из (1) и свойств операции композиции следует, что отношение $\Re_i$ произвольного класса c_i схемы Sh содержит любое естественное соединение соответствующего ему отношения r_i в соответствующей реляционной схеме ShR. Однако в базе данных может возникнуть потребность в соединении, не являющемся естественным. Или же могут возникнуть запросы, реализующие реляционный оператор деления. И эти запросы не будут выразимы на NDL.

Можно, оставаясь в рамках N-модели данных, сделать язык NDL реляционно полным. Для этого достаточно позволить использование в операторе SELECT функции ALL(name(c_j)), возвращающей множество идентификаторов экземпляров класса или категории c_j и являющейся изоморфизмом функции Ext. В рамках описанной в данной работе концепции, когда $\Re$ строится на множестве информативных на классе атрибутов, на функцию ALL(name(c_j)) полезно наложить следующие прагматические ограничения:

M1. В предложении SELECT оператора SELECT функция ALL(name(c_j)) может использоваться только как операнд теоретико-множественных операций INTERSECT, MINUS и UNION и агрегатной операции COUNT, определенных в [1].

M2. В предложении WHERE оператора SELECT в качестве операнда сравнения функция ALL(name(c_j)) может использоваться только как операнд теоретико-множественных операций INTERSECT, MINUS и UNION, агрегатной операции COUNT, определенных в [1], или же самостоятельно.

M3. Если функция ALL(name(c_j)) является операндом операций INTERSECT, MINUS или UNION в операторе SELECT, определенном на классе c_i, то другим операндом должен быть атрибут, информативный на c_i, чья область значений содержит ALL(name(c_j)), или, иначе говоря, другой операнд должен быть ссылкой на класс c_j (если же c_j – категория, то ссылкой на ее родительский класс).

Введенные здесь ограничения не делает NDL реляционно полным. Для достижения реляционной полноты необходимо их снять. В этом случае отношения $\Re$ различных классов совпадают и содержат все атрибуты, определенные в базе данных, и все атрибуты, производные от них, т.е. в операторе SELECT, построенном на любом классе, будут доступны значения атрибутов всех классов базы данных вне зависимости от того, насколько такая доступность вообще имеет смысл. Реляционная полнота в данном случае дает возможность порождать множество бессмысленных запросов. Такая же ситуация, благодаря конструкции подзапросов, наблюдается и в SQL. Однако мы считаем ее проблематичной и поэтому в настоящее время полагаем, что ограничений ***M1 – M3*** необходимо придерживаться. Тем не менее, мы понимаем всю дискуссионность данного вопроса и необходимость его дальнейшего исследования.

Теперь перейдем к проблеме невыразимости транзитивного замыкания в SQL (транзитивное замыкание не реализовано в стандарте SQL, но реализуется в промышленных системах управления базами данных с помощью конструкций, определенных в расширениях SQL, например, в PL/SQL или в T-SQL). В NDL она решена, поскольку транзитивное замыкание выражается через рекурсивно вычисляемые расчетные атрибуты. Покажем это на хрестоматийном примере. В базе данных содержатся сведения о сотрудниках: табельный номер, имя, начальник. Под последним понимается непосредственный начальник сотрудника. Необходимо выбрать всех руководителей (транзитивных начальников) для каждого сотрудника.

В базе данных сведения о сотруднике будут содержаться в классе (на NDL):

```
create class entity СОТРУДНИК
attributes ТАБНОМЕР: varchar(32),
           ИМЯ: varchar(128),
           НАЧАЛЬНИК: ext(СОТРУДНИК),
           РУКОВОДИТЕЛЬ = НАЧАЛЬНИК union
                                        НАЧАЛЬНИК.РУКОВОДИТЕЛЬ
unique     (ТАБНОМЕР);
```

Запрос на NDL, возвращающий для каждого сотрудника всех его руководителей, будет выглядеть так:

```
select ТАБНОМЕР, ИМЯ, РУКОВОДИТЕЛЬ
from СОТРУДНИК.
```

Заключение. Описаны синтаксис и семантика оператора SELECT языка NDL и оценена его полнота. NDL базируется не на реляционной, а на объектной N-модели, и переход в данной работе к реляционной алгебре обусловлен только желанием авторов объяснить семантику оператора SELECT в NDL с помощью формализмов, привычных и понятных большинству специалистов в области баз данных.

При соблюдении введенных здесь прагматических ограничений ***M1 – M3*** язык NDL не является реляционно полным, поскольку декартово произведение в этом случае ограничено. Снятие этих ограничений является дискуссионным вопросом, но позволяет сделать NDL реляционно полным и даже более выразительным, чем стандарт SQL, поскольку в нем выразимо транзитивное замыкание.

Библиографический список

1. Ольховик О.В. N-модель данных / О.В. Ольховик, А.В. Белых // Изв. ЮФУ. Сер. Техн. науки. Тем. вып. «Интеллектуальные САПР». – 2009. – № 8.

2. Белых А.В. Декларативный язык для N-модели данных / А.В. Белых, С.М. Ковалев, О.В. Ольховик // Вестн. РГУПС. – 2009. – № 4.

3. Белых А.В. Визуально-декларативный язык для проектирования программного обеспечения информационных систем / А.В. Белых, С.М. Ковалев, О.В. Ольховик // Вестн. Донс. гос. техн. ун-та. – 2009. – № 4.

4. Date, C.J. Foundation for Future Database Systems: The Third Manifesto / C.J. Date, Hugh Darwen. – 2nd edition. – Addison-Wesley Pub Co, 2000.

5. ISO/IEC-9075-01:2008. Information technology – Database languages – SQL – Part 1: Framework (SQL/Framework): http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=45498 (дата обращения: 10.03.2011).

Материал поступил в редакцию 24.06.2011.

References

1. Ol`xovik O.V. N-model` danny`x / O.V. Ol`xovik, A.V. Bely`x // Izv. YUFU. Ser. Texn. nauki. Tem. vy`p. «Intellektual`ny`e SAPR». – 2009. – # 8. – In Russian.

2. Bely`x A.V. Deklarativny`j yazy`k dlya N-modeli danny`x / A.V. Bely`x, S.M. Kovalyov, O.V. Ol`xovik // Vestn. RGUPS. – 2009. – # 4. – In Russian.

3. Bely`x A.V. Vizual`no-deklarativny`j yazy`k dlya proektirovaniya programmnogo obespecheniya informacionny`x sistem / A.V. Bely`x, S.M. Kovalyov, O.V. Ol`xovik // Vestn. Dons. gos. texn. un-ta. – 2009. – # 4. – In Russian.

4. Date, C.J. Foundation for Future Database Systems: The Third Manifesto / C.J. Date, Hugh Darwen. – 2nd edition. – Addison-Wesley Pub Co, 2000.

5. ISO/IEC-9075-01:2008. Information technology – Database languages – SQL – Part 1: Framework (SQL/Framework): http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=45498 (date of access: 10.03.2011).

SELECT STATEMENT IN N-DECLARATIVE LANGUAGE

O.V. OLKHOVIK
(Don State Technical University)

Syntax and semantics of the SELECT statement in N-Declarative Language are described. It is shown that for each class in the sense of N-data model the relation containing the attribute values in this informative class can be built up, and the SELECT statement implements the relational algebra operations and other special operations on the subsets of this relationship.

Keywords: *Model-driven engineering, information systems, databases, query languages, SELECT statement.*