УДК 004.8

# РАЗРАБОТКА ТРАНСЛЯТОРА, ПЕРЕВОДЯЩЕГО ПРЕДЛОЖЕНИЯ РАСШИРЕНИЯ РЕЛЯЦИОННОЙ АЛГЕБРЫ И РЕЛЯЦИОННОГО ИСЧИСЛЕНИЯ В ЯЗЫК SQL

**И. Ф. Астахова, А. А. Малиев**

*Воронежский государственный университет*

*Поступила в редакцию 13.06.2013 г.*

**Аннотация.** Рассматривается расширение языков реляционной алгебры и реляционного исчисления, позволяющее работать с нечеткими запросами к базе данных, для чего предлагается язык, дающий возможность осуществить «мягкие вычисления». Дается описание транслятора, реализующего перевод предложенного языка в язык SQL, его основных модулей и интерфейса.

**Ключевые слова:** транслятор, реляционная алгебра, реляционное исчисление, мягкое вычисление, интерфейс, язык SQL.

**Annotation.** In article it is considered languages of relational algebra and relational calculation, and also the expansion, allowing to work with fuzzy request , that is some language, allowing to carry out "soft calculations" is offered. The description of basic elements of the translator realizing transfer of described language in the SQL language is considered. Program realization of the offered translator, the interface and the main modules is described.

**Keywords:** translator, relational algebra, relational calculation, soft calculation, interface, SQL language.

Для развития «мягких вычислений» необходимо предложить такой формальный язык, который мог бы реализовать реляционное исчисление и реляционную алгебру в четком и нечетком варианте.

Как известно [1], реляционная алгебра – это набор операций, которые принимают отношения в качестве операндов и возвращают отношение в качестве результата.

В данной работе будут исследованы принципы построения запросов на языке нечеткой реляционной алгебры и исчисления FRACQL (Fuzzy Relational Algebra and Calculus Query Language) с использованием условий с приоритетом. При построении запросов будут применяться коэффициенты пиритизации условий, задания веса, а так же использоваться пороговое значение выбора кортежей по всему условию.

Самой распространенной моделью данных, применяемой в базах данных (БД), является реляционная модель. Эта модель оперирует четкой информацией, такой как, например, метрические размеры какого-либо объекта, численные значения его веса и т.п. Однако у агента (пользователя), который запрашивает нужную ему информацию из БД, требования к данным, характеризующим какой-либо объект, не всегда удовлетворяют четким критериям выбора: критерии могут быть размытыми в силу того, что агенту могут подходить объекты, свойства которых близки.

Язык SQL – самый распространённый в настоящее время язык для составления запросов к БД. Он построен на операциях реляционной алгебры и реляционного исчисления, хотя и не включает их в явном виде. Поэтому его удобно использовать для получения необходимого нам расширения языков реляционной алгебры и реляционного исчисления.

Рассмотрим язык реляционной алгебры и исчисления RACL (Relational Algebra and Calculus Language) [1] и расширим его операциями, позволяющими производить нечеткий поиск по отношениям, содержащим четкие данные.

Реляционное исчисление базируется на понятии правильно построенной формулы WFF (Well Formed Formula). С помощью WFF можно выражать условия, накладываемые на кортежные переменные. WFF состоит из простых сравнений скалярных значений, заданных в

© Астахова И. Ф., Малиев А. А., 2013

виде констант или значений атрибутов. Более сложные варианты WFF строятся с помощью логических операций NOT, AND, OR, IF... THEN и двух кванторов: EXISTS и FORALL (кванторов существования и общности).

Рассмотрим операции языка RACL и определим, как можно ввести оценку нечетких значений.

Пусть A, B – отношения. Тогда операции реляционной алгебры: объединения, пересечения, разности, декартова произведения, сокращения, проекции, соединения, деления можно записать следующим образом: A union B (объединение), A intersect B (пересечение), A minus B (разность), A times B (декартово произведение), A where <логическое выражение> (сокращение), A project <список атрибутов> (проекция), A join B where <логическое выражение> (соединение), A divide by B (деление).

В данной статье мы используем реляционные операции над отношениями с четкими данными и определим нечеткость только в тех операциях, где есть условия, влияющие на число кортежей в результирующем отношении, то есть нечеткость может быть введена в операциях сокращения и соединения, а также в кванторах существования и общности.

Привнесем в язык RACL нечеткость. Используя понятия лингвистической переменной и лингвистического модификатора [2], можно расширить домен атрибута. В итоге, значения атрибута из старого домена могут с дополнительными значениями из нового домена с применением правил нечеткой логики сравниваться. Однако чтобы найти нужные данные, в большинстве случаев приходится проводить поиск по нескольким атрибутам в отношении. При этом возможен случай, когда один атрибут в условии поиска играет более весомую роль, чем другой.

Чтобы решить данную проблему, воспользуемся понятием нечеткого приоритезированного удовлетворения ограничений PFSCP (Prioritized Fuzzy Constraint Satisfaction Problem) [3]. Главной особенностью системы, основанной на PFSCP, являются t-нормы, использованные в анализе условий с приоритетами. Они введены таким образом, что чем выше приоритет, тем большее влияние на результат оказывает значение атрибута.

FSCP определяется на множестве кортежей $\left(X, D, C^f, \rho\right)$, где:

1) $X = \left\{X_i, i = 1,2,..,n\right\}$ – конечное множество атрибутов отношения, $x_i$ – значение атрибута $X_i$;

2) $D = \left\{D_i, i = 1,2,\ldots,n\right\}$ – конечное множество доменов отношения, где $D_i$ – домен атрибута $X_i$;

3) $$C^f = \left\{R^f \mid \mu_{R_i^f} : \left(\prod_{X_j \in var\left(R^f\right)} D_j\right) \rightarrow \rightarrow \left[0,1\right], i = 1,2,\ldots,m\right\}, \quad (1)$$

здесь $C^f$ – множество нечетких ограничений; $R^f$ – нечеткое ограничение; $var\left(R^f\right)$ – множество нечетких атрибутов в $R^f$;

4) $\rho : C^f \rightarrow \left[0,\infty\right]$ – функция приоритета.

Тогда кортеж $v_x = \left(v_1,\ldots,v_n\right) \in X$ удовлетворяет поисковому критерию по атрибутам $X' = \left(X_1',\ldots,X_m'\right) \subseteq X$ со степенью, вычисляемой по формуле:

$$\alpha_\rho\left(v_x\right) = f\left\{g\left(\frac{\rho\left(R^f\right)}{\rho_{max}}, \mu_{R_i^f}\left(v_{var\left(R^f\right)}\right)\right)\right\}, \quad (2)$$

где $f : \left[0,1\right]^n \rightarrow \left[0,1\right]$ и является агрегирующим оператором,

$$g : \left[0,+\infty\right) \times \left[0,1\right] \rightarrow \left[0,1\right],$$

$$\text{если } p_{max} = \{\rho\left(R^f\right) \mid R^f \in C^f\}. \quad (3)$$

$f\left(x,y\right)$ – есть $t$-норма, а $g\left(x,y\right)$ – $s$-норма. Для приоритезации условий достаточно положить $f\left(x,y\right) = \min\left(x,y\right)$, $g\left(x,y\right) = \max\left(x,y\right)$.

Приоритеты в условиях работают следующим образом. Функция $\rho$ противопоставляет некоторый приоритет каждому ограничению. Функция $g$ агрегирует приоритет каждого ограничения со значением самого ограничения, которые впоследствии агрегируются оператором $f\left(x,y\right)$ и дают в результате оценку степени принадлежности кортежа запрашиваемым критериям.

Помимо использования PFSCP для работы с нечеткими условиями, так же можно использовать веса, где каждому ограничению $C_i$ приписывается свой вес $w_i$. Степень выполнения условий может быть найдена по следующей формуле:

$$\alpha_W\left(v_x\right) = T\left(c_1 * w_1,\ldots,c_n * w_n\right), \quad (4)$$

где $\sum_{i=1..n} w_i = 1$, а $c_i = \mu_{C_i}\left(v_x\right)$ – степень выполнения критерия $C_i$.

Используя приоритеты или веса в условиях, можно получить кортежи даже с малой степенью принадлежности запрашиваемым критериям [4–5], то есть получается расширение языка SQL.

Каждому из вышеперечисленных реляционных операторов поставим в соответствие некоторую встроенную функцию, к которой необходимо добавить объявления переменной кортежей – таблицы с указанием набора ее атрибутов.

Транслятор воспринимает строку символов определенного вида (текст программы на исходном языке) и выдает другую строку символов (объектную программу). Как известно, трансляторам присущ ряд общих черт, что упрощает их создание. Так, в состав любого транслятора входят три основных компонента: лексический анализатор (блок сканирования), синтаксический анализатор и генератор кода машинных команд.

Принцип действия анализаторов можно описать с помощью формальных моделей, в то время как для генератора кода пока еще не существует общепринятых четких формальных представлений. На этапе лексического анализа исходный текст программы в виде цепочки не связанных друг с другом символов разбивается на единицы, называемые лексемами. Такими текстовыми единицами являются ключевые слова, используемые в языке, имена переменных, константы и знаки операций. Далее эти слова рассматриваются как неделимые образования, а не как группы отдельных символов. После разбиения программы на лексемы следует фаза синтаксического анализа - грамматического разбора, при котором проверяется правильность следования операторов.

Кроме того, зачастую, к уже рассмотренным составным частям трансляторов добавляют следующие: компонент, производящий оптимизацию результирующего кода и контекстный (семантический) анализатор.

Для разработки лексического и синтаксического анализатора можно воспользоваться «компиляторами компиляторов» – программами, воспринимающими синтаксическое или семантическое описание языка программирования и генерирующими компилятор для этого языка.

Прежде чем приступить к созданию транслятора, необходимо иметь четкое и однозначное определение исходного языка. Можно представить язык состоящим из ряда строк (последовательностей символов). В описании языка определяется, какие строки принадлежат этому языку (синтаксис языка), и значение этих строк (семантика языка).

Будем определять синтаксис языка с помощью грамматики. В нее входит набор правил для получения предложений языка. Грамматика определяется как четверка ($V_t$, $V_n$, P, S), где $V_t$ – алфавит, символы которого называются терминальными (терминалами), из них строятся цепочки, порождаемые грамматикой; $V_n$ – алфавит, символы которого называются нетерминальными (нетерминалами). Алфавиты $V_t$ и $V_n$ не имеют общих символов, т.е. $V_t \cap V_n = \varnothing$. Полный алфавит (словарь) грамматики V определяется как объединение алфавитов $V_t$ и $V_n$. Определим P – набор порождающих правил, каждый элемент которых состоит из пары $(\alpha, \beta)$, где $\alpha$ находится в V+, а $\beta$ в V*, $\alpha$ – левая часть правила, а $\beta$ – правая, т.е. это цепочки, построенные из символов алфавита V. Правило записывается как $\alpha \to \beta$. S принадлежит алфавиту $V_n$ и является начальным символом (аксиомой). Этот символ – отправная точка для получения любого предложения языка.

Для достижения высокой скорости трансляции применяется транслятор с однопроходной структурой. В этом случае синтаксический анализатор выступает в роли основной управляющей программы, вызывая блок сканирования и генератор кода, организованные в виде подпрограмм. Синтаксический анализатор постоянно обращается к блоку сканирования, получая от него лексему за лексемой из просматриваемой программы, до тех пор, пока не построит новый элемент постфиксной записи, после чего он обращается к генератору кода, который создает объектный код для этого фрагмента программы.

Существует полное соответствие между регулярными выражениями, а следовательно, и между грамматиками и конечными автоматами, которые определяются следующим образом. Конечный автомат формально определяется пятью характеристиками: конечным множеством состояний; конечным входным алфавитом; множеством переходов; начальным состоянием; множеством последних состояний. Такой автомат будет детерминированным, т.к. в каждом элементе таблицы переходов содер-

жится лишь одно состояние. В недетерминированном конечном автомате это положение не выполняется. В функции автомата магазинного типа входит: а) чтение входного символа, замещение верхнего символа стека строкой символов (возможно, пустой) и изменение состояния; б) всё то же самое, но без чтения входного символа.

Что же касается конечных автоматов, то мы можем так же определять недетерминированные автоматы магазинного типа, содержащие множество переходов для заданного входа, состояния и содержания стека.

При синтаксическом разборе происходит эффективное моделирование соответствующего автомата магазинного типа.

Для построения синтаксического анализатора создадим универсальную программу грамматического разбора, пригодную для всех возможных грамматик заданного класса. В этом случае конкретные грамматики задаются этой программе в виде данных определенной структуры, которая управляет ее работой. Поэтому такая программа и называется таблично-управляемой.

Рассмотрим основной метод восходящего синтаксического анализа, известный как синтаксический анализ типа “перенос/свертка” (shift-reduce) и называемый далее сокращенно ПС-анализом.

ПС-анализ строит дерево разбора для входной строки, начиная с листьев (снизу) и двигаясь по направлению к корню дерева (вверх). Этот процесс можно рассматривать как свертку строки к стартовому символу грамматики. На каждом шаге свертки некоторая подстрока, соответствующая правой части продукции, заменяется символом из левой части этой продукции, и если на каждом шаге подстроки выбираются корректно, то тогда получается обращенное правое порождение.

Достаточно удобный путь реализации ПС-анализатора состоит в использовании стека для хранения символов грамматики и входного буфера - для хранения анализируемой строки.

Синтаксический анализатор работает путем переноса нуля или нескольких символов в стек до тех пор, пока на вершине стека не окажется основа $\beta$. Затем он свертывает $\beta$ к левой части соответствующей продукции.

Синтаксический анализатор повторяет этот цикл, пока не будет обнаружена ошибка или он не перейдет в конфигурацию, когда в стеке будет находиться только стартовый символ, а входной буфер будет пуст. Попав в эту конфигурацию, синтаксический анализатор прекращает работу и выдает сообщение об успешном разборе входной строки.

Рассмотрим очень важный факт, который поясняет использование стека в ПС-анализаторе: основа всегда находится на вершине стека и никогда – внутри него. Это становится очевидным при рассмотрении возможных видов двух последовательных шагов в любом правом порождении.

Синтаксический анализатор никогда не заглядывает внутрь стека в поисках правого края основы. Все это делает стек особенно удобным для использования для реализации ПС-анализатора.

В таблице символов транслятора может содержаться и другая информация об идентификаторе, необходимая во время трансляции, например, его адрес в ходе прогона или, в случае константы, её значение.

Реализация таблицы символов внутри программ возможна либо в виде массива, либо в виде цепочечной структуры, каждый элемент которой содержит ссылку на последующий и, возможно, предыдущий элемент структуры.

В процессе анализа текста программы транслятор должен выдавать соответствующую диагностику об ошибках и продолжать процесс грамматического разбора, находя возможные ошибки.

Для выполнения функций трансляции программы, по нашему мнению, в данном случае наиболее подходит программа-транслятор, построенная на основе автомата с магазинной памятью с применением LL (1) грамматики, т.к.: а) LL (1) грамматики будет достаточно для распознавания разрабатываемого языка ЯРАИ (язык реляционной алгебры и исчисления); б) автоматы с магазинной памятью расширяются из-за удобства читаемости кода.

При решении данной задачи были разработаны две программы, написанные на языке C++ в среде C#: 1) транслятор и 2) среда для создания программ на ЯРАИ.

Первая программа является консольным приложением и не имеет графического интерфейса пользователя (GUI). Направить ее на выполнение можно, например, из консоли,

указав один или два параметра. Первый параметр является путем к файлу, который необходимо транслировать. Второй параметр - необязательный. Если он указан, то интерпретируется как название результирующего файла с программой на языке SQL, иначе имя данного файла будет сгенерировано автоматически .o второй программе присутствует GUI. Он содержит ряд форм [6]:

• главная форма программы;

• диалоговая форма, предназначенная для получения от пользователя сведений, необходимых для подключения к некоторой БД;

• форма, содержащая сведения о данном программном продукте.

Главное меню программы содержит ряд пунктов:

• Файл
– Новый
– Открыть
– Сохранить
• Вид
– Текст ЯРАИ
– Текст SQL
– Отладка
– Ошибки
• Инструменты
– Установить соединение с БД
– Транслировать
• О программе

Пользователь с помощью главного меню может осуществить следующие действия:

• открыть файл с кодом на языке ЯРАИ. Для этого необходимо выбрать пункт меню «Файл» –> «Открыть» и в диалоговом окне указать необходимый файл;

• сохранить полученный код в конкретный файл, для чего требуется выбрать пункт меню «Файл» –> «Сохранить» и в диалоговом окне выбрать директорию, в которой будет храниться файл и указать его имя;

• создать новый проект – «Файл» –> «Новый», при этом иногда следует указать на необходимость сохранения текущего проекта;

• настроить интерфейс программы, убрав или добавив панели в главном окне:

– «текст ЯРАИ», который содержит код на языке ЯРАИ;

– «текст SQL», отображающий полученную программу на языке SQL;

– «отладка», которая содержит информацию, аналогичную предыдущей, но дополнительно позволяет выполнить полученные запросы;

– «ошибки», отображаемые списком, возникшие при трансляции программы; можно перейти к месту возникновения ошибки в исходном коде;

• установить соединение с некоторой БД, выбрав пункт меню «Инструменты» –> «Установить соединение с БД», и в открывшемся диалоговом окне указать все необходимые параметры соединения;

• транслировать введенный код: пункт меню «Инструменты» –> «Транслировать»;

• просмотреть информацию о данном продукте – пункт «О программе».

Применяя представленное выше программное обеспечение, пользователь может осуществить ввод кода на языке ЯРАИ, просмотреть сгенерированную программу на языке SQL и список ошибок (в случае их возникновения), выполнить полученный код на языке SQL или провести иные необходимые действия.

Таким образом, в настоящей работе представлен вариант транслятора, переводящего расширение ЯРАИ в язык SQL. Особенностью этого языка является его относительная простота и ясность используемых в нем теоретических положений.

## СПИСОК ЛИТЕРАТУРЫ

1. *Кузнецов С.Д.* Основы баз данных: учебное пособие для студентов, обучающихся по специальностям в области информационных технологий. – М.: Интернет-Университет информационных технологий, 2005. – 488 с.

2 *Чулюков В.А.* Системы искусственного интелекта. Практический курс / Под ред. И.Ф. Астаховой // М: ФИЗМАТЛИТ, БИНОМ, 2008. – 324 с.

3. *Luo X., Lee J.H., Jennings N.R.* Prioritised Fuzzy Constraint Satisfaction. Problems: Axioms, Instantiation and Validation // Int. Journal of Fuzzy Sets and Systems. – 2003. – V. 136(10). – P. 155–188.

4. *Zadeh L.A.* Fuzzy Sets: Information and control // 1965. – V. 8.– P. 338–353.

5. *Рыжов А.П.* Модели поиска информации в нечеткой среде. – М.: МГУ, 2004. – 96 с.

6. *Малиёв А.А.* Разработка языка реляционной алгебры и реляционного исчисления и транслятора на язык SQL // Вестник Воронежского государственного технического университета. – 2009. – Т. 5, № 12. – С. 154–155.

**Астахова И. Ф.** – доктор технических наук, профессор кафедры математического обеспечения ЭВМ факультета прикладной математики, информатики и механики Воронежского государственного университета, профессор. Телефон: (473) 2-208-638-314. E-mail: astachova@list.ru

**Малиев А. А.** – аспирант кафедры математического обеспечения ЭВМ факультета прикладной математики, информатики и механики Воронежского государственного университета. Тел.: (473) 2-208-638-314. E-mail: astachova@list.ru

**Astachova I. F.** – Doctor of Technical Science, Professor of the chair mathematical software, Voronezh State University. Phone: (473) 2-208-638-314. E-mail: astachova@list.ru

**Maliev A. A.** – Postgraduate student of the chair mathematical software, Voronezh State University. Phone: (473) 2-208-638-314. E-mail: astachova@list.ru