Правительство Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«Санкт-Петербургский государственный университет»

Кафедра системного программирования

Савельев Николай Геннадьевич

Язык высокого уровня для алгебры нечетких запросов в неоднородной среде

Дипломная работа

Допущена к защите.
Зав. кафедрой:
д. ф.-м. н., профессор Терехов А. Н

Научный руководитель:
д. ф.-м. н., профессор Новиков Б. А

Рецензент:
ассистент Чернышев Г. А.

Санкт-Петербург
2015

SAINT-PETERSBURG STATE UNIVERSITY

Software Engineering Chair

Savelev Nikolai

High-level language for fuzzy query algebra in heterogeneous environment

Graduation Thesis

Admitted for defence.
Head of the chair:
professor Andrey Terekhov

Scientific supervisor:
professor Boris Novikov

Reviewer:
assistant George Chernishev

Saint-Petersburg
2015

Оглавление

1. Введение 4
2. Постановка задачи 6
3. Обзор существующих решений 7
   3.1. Интерфейсы и решения для взаимодействия с разнородными источниками 7
   3.2. Параметризация запросов 8
   3.3. Поддержка нечетких запросов 10
4. Модель алгебры нечетких запросов 11
5. Язык высокого уровня для представления алгебры нечетких запросов 13
   5.1. Требования к языку высокого уровня 13
   5.2. Грамматика языка высокого уровня 14
6. Транслятор языка высокого уровня в модель алгебры 18
   6.1. Трансляция арифметических выражения 18
   6.2. Трансляция операторов алгебры нечетких запросов 18
      6.2.1. Ключевое слово FROM 19
      6.2.2. Ключевое слово WHERE 20
      6.2.3. Ключевое слово GROUP BY 21
      6.2.4. Ключевое слово HAVING 22
      6.2.5. Ключевое слово ORDER BY 22
      6.2.6. Ключевое слово LIMIT 22
      6.2.7. Ключевое слово SELECT 23
      6.2.8. Ключевое слово FUSION 23
      6.2.9. Особые случаи трансляции 24
   6.3. Особенности реализации 25
      6.3.1. Архитектура 25
      6.3.2. Технологии 26
7. Апробация 28
8. Заключение 32
Список литературы 34
Приложение 1. Грамматика языка высокого уровня 36

1. Введение

Объем разнородных данных, требующих обработки, растет с каждым годом: показания приборов, сенсоров, трафик социальных сетей, новостных аггрегаторов, видео-порталов, данные научных исследований и т.д. Разнообразие таких источников информации создает трудности в их централизованной обработке, потребности в которой также увеличиваются.

Несмотря на различную структуру информации, часто ее обработка основана на схожих принципах и техниках. Большинство современных систем не предназначены для анализа существенно разнородных источников информации, хотя такие задачи появляются все чаще. В связи с этим возникает потребность в системах, способных решать такие задачи.

Данные системы не должны ограничиваться малыми объемами данных, так как рост количества информации влечет за собой и сложность ее конвертации в удобные для обработки форматы в существующих продуктах. По этой же причине становится актуальным приближенный анализ данных, который предлагает возможность выбора между временем работы и точностью решения задач. Эти задачи могут представлять собой и составление общих отчетов с различных источников данных, и приближенный анализ научных данных с использованием удаленных исполнителей, и обработку видеопотоков и т.д. Все это предъявляет особые требования к языку запросов, с помощью которого данные задачи могут быть описаны и впоследствии выполнены.

На текущий момент большинство языков запросов являются декларативными. Такой подход позволяет описать результат запроса, абстрагировавшись от деталей исполнения. В соответствии с запросом производится поиск плана его исполнения в терминах специализированной алгебры, а также оптимизация запроса [1]. Из этого следует, что возможности языка пропорциональны возможностям используемой алгебры. Иными словами, имея сильную алгебраическую структуру возможно определить

широкий спектр высокоуровневых операций, доступных при составлении запросов.

Данная работа выполняется в рамках проекта кафедры информационно-аналитических систем СПбГУ по созданию системы обработки неоднородных данных, которая основывается на расширенной реляционной алгебре [2]. Эта алгебра отличается новым подходом к различным источникам данных, а также совмещает в себе работу как с точными запросами, так и с нечеткими, что делает ее удобной для использования в системах обработки неоднородных данных. В настоящее время в данном проекте нет языка высокого уровня для задания запросов системе, что, безусловно, усложняет как ее использование, так и разработку, тестирование.

2. Постановка задачи

Целью данной работы является создание языка высокого уровня для представления алгебры нечетких запросов в неоднородной среде и создание транслятора в эту алгебру. Для достижения этой цели были сформулированы следующие задачи.

1. Ознакомиться с существующими инструментами для обработки неоднородных данных и языками нечетких запросов.
2. Реализовать представление модели алгебры нечетких запросов для последующего ее использования в процессе трансляции.
3. Создать язык высокого уровня для алгебры нечетких запросов. Язык должен быть понятен пользователю, легко расширяем, а также должен полностью охватывать возможности алгебры.
4. Реализовать транслятор для языка высокого уровня в модель алгебры нечетких запросов.
5. Провести апробацию реализованной функциональности.

3. Обзор существующих решений

В данном разделе представлены существующие решения и интерфейсы для взаимодействия с разнородными источниками данных и способы параметризации запросов. Также в этом разделе приведены примеры языков с поддержкой нечетких запросов, которые задают определенное направление дизайна языка высокого уровня.

3.1. Интерфейсы и решения для взаимодействия с разнородными источниками

Задача обработки данных с различных источников стоит давно. Существует множество решений и у каждого есть свои преимущества и недостатки. До появления ныне широко-известного направления NoSQL (Not only SQL) [3], символизирующего собой ответвление от традиционных концепций реляционный СУБД (Систем Управления Базами Данных), основным средством доступа к различным системам хранения данных являлись технологии OLE DB (Object Linking and Embedding, Database) и ODBC (Open Database Connectivity). Они предоставляют универсальный интерфейс для передачи данных между приложениями и базами данных. Универсальность интерфейса является и недостатком: не все системы хранения данных могут его реализовать полностью, что часто и вовсе приводит к отказу от его имплементации.

В связи с этим в последнее время начала расти популярность веб-интерфейсов REST (Representational State Transfer), где пользователь взаимодействует с системой хранения данных посредством обычных HTTP-запросов (HyperText Transfer Protocol). В качестве результата, удобном как для обработки, так и для чтения, отладки используются средства

сериализации XML (eXtensible Markup Language) и JSON (JavaScript Object Notation).

Многие промышленные базы данных поддерживают обработку этих форматов [4], [5], но они не могут быть легко расширены для нужного пользователю типа информации и, тем более, эту цель не преследуют.

В то же время существует несколько проектов, чья внутренняя архитектура позволяет решать задачу разнородности данных. Система Forward [6] позволяет обрабатывать данные с различных информационных источников, для которых реализованы специальные компоненты-адаптеры, инкапсулирующие в себе взаимодействие с источником.

Проект Pentaho [7] включает в себя множество бизнес-решений для работы с данными, в том числе и составление отчетов, где источниками данных могут служить любые СУБД, поддерживающие интерфейс JDBC (Java DataBase Connectivity).

В итоге, обработка разнородных данных требует от систем универсальной модели данных и интерфейса, с помощью которого информация может быть извлечена из источников.

3.2. Параметризация запросов

Как упоминалось раннее, большинство языков запросов являются декларативными и избавляют пользователя от деталей исполнения, за которые отвечает оптимизатор запросов. Этот модуль отвечает за поиск оптимального плана запроса по различным критериям. Разумеется, найденный план может быть далек от идеального и у специалистов есть возможность подсказать оптимизатору некоторые детали.

Этот механизм есть во многих СУБД и позволяет конфигурировать алгоритмы операций, использование индекса, специальные ограничения и т.д. Ниже представлен пример использования подсказок на языке PL/SQL – языке запросов базы данных Oracle:

```
SELECT
      /*+ LEADING(e2 e1)
      USE_NL(e1) INDEX(e1 emp_emp_id_pk)
      USE_MERGE(j) FULL(j) */
   e1.first_name, e1.last_name, j.job_id, sum(e2.salary) total_sal
FROM employees e1, employees e2, job_history j
WHERE e1.employee_id = e2.manager_id
   AND e1.employee_id = j.employee_id
   AND e1.hire_date = j.start_date
GROUP BY e1.first_name, e1.last_name, j.job_id
ORDER BY total_sal
```

В начале запроса после ключевого слова SELECT происходит конфигурация оптимизатора для составления плана с конкретными деталями. Обработка неоднородных данных вносит новый смысл для конфигураций оптимизатора. Это и определение специального доступа к источнику данных, и переопределение поведения удаленного исполнителя запросов, и указание на использование особой функции. Разработчики проекта Forward определили особый язык для запросов к гетерогенным источникам – SQL++ [8]. Этот язык расширяет SQL (Structured Query Language) синтаксисом для переопределения семантики запроса и конфигурирования оптимизатора. Ниже представлен пример запроса с параметрами на этом языке:

```
@SELECT  {path:…}
@FROM    {bag_order:…}
FROM readings AS r
SELECT attribute r.gas : r.val
```

Из вышесказанного следует, что параметризация является неотъемлемой частью языков запросов, с помощью которых пользователь может специфицировать детали исполнения, источники данных и семантику запроса.

3.3. Поддержка нечетких запросов

Что касается поддержки нечетких запросов в языках высокого уровня, то существует несколько проектов по внедрению этой возможности в синтаксис SQL. Язык FSQL [9] расширяет традиционную грамматику новыми операторами для работы с вероятностными функциями и нечеткими множествами. Ниже описан пример запроса на данном языке:

```
SELECT * FROM person
WHERE hair feq $fait thold 0.5
AND height fgt $tall thold 0.8
```

В книге [10] описывается надстройка над реляционными базами данных, позволяющая решать задачи классификации с помощью специального синтаксиса.

Поддержку нечетких запросов для изображений вводит язык FOQL [11], который также надстраивается над базами данных изображений и дает возможность пользователю, используя вводимые языком конструкции, проводить приближенный анализ данных.

Все вышеперечисленные языки являются расширениями SQL и тем или иным способом транслируются в базовый язык СУБД, применяя нечеткие операторы как заранее определенные функции. Из этого можно сделать вывод о том, что поддержка приближенных запросов может быть реализована также на уровне функций и представлена специальным синтаксисом.

4. Модель алгебры нечетких запросов

Целью данной работы является описание грамматики языка высокого уровня для алгебры нечетких запросов [2], для которой в этом разделе будет описана модель. Главной особенностью этой алгебры является концепция q-set – это тройка (q, B, S), где:

- q – это представление запроса;
- B – это источник данных для запроса, который также может содержать набор q-set;
- S – это функция оценки релевантности объекта запросу.

С помощью описанных выше троек определяются все операторы данной алгебры. Это такие операторы, как:

- фильтр – это оператор, исполняющийся для каждого объекта из источника;
- соединение – этот оператор поддерживает как обычное соединение, так и приближенное с возможностью конфигурации;
- агрегирование – это оператор группировки объектов из источника на основе определенных параметров;
- слияние – это оператор, производящий слияние наборов атрибутов источников данных с заполнением пропущенных значений.

В тоже время представление каждого оператора в модели трансляции состоит из следующих составляющих.

1. Строковый идентификатор оператора – эта часть введена для предоставления возможности ссылки на атрибуты результата исполнения данного оператора.
2. Схема – эта часть содержит в себе описание атрибутов объектов, которые являются результатом исполнения оператора. Для упрощения модели было принято решение позволять только фильтру иметь эту часть, так как все остальные операторы можно

обернуть специальным фильтром, создающим нужную проекцию атрибутов.

3. Аргументы – эта часть содержит в себе описание источников данных для оператора. Здесь также могут содержаться другие операторы, формирую таким образом подзапросы.
4. Параметры – эта часть служит контейнером для параметров оператора: условий фильтрации или соединения, функций оценки, подсказок оптимизатору и т.д.

Данная модель алгебры определяет универсальное представления операторов алгебры, которое будет использоваться в процессе трансляции.

5. Язык высокого уровня для представления алгебры нечетких запросов

В данном разделе описываются требования к языку высокого уровня для алгебры нечетких запросов в неоднородной среде, а также перечисляются основные правила грамматики для него.

5.1. Требования к языку высокого уровня

В начале работы над созданием языка высокого уровня был проведен анализ существующих решений на предмет поддержки разнородных источников, параметризации запросов и приближенной обработки, результатом чего стал набор требований к языку и к самой системе обработки информации в целом.

1. Работа с различными форматами данных и гетерогенными источниками должна быть скрыта от пользователя универсальной моделью памяти, тем самым будет достигаться унификация подхода к анализу неоднородных данных.
2. Каждый оператор алгебры может иметь несколько параметров, согласно определенной раннее модели, в связи с чем такую же гибкость должен предоставлять и высокоуровневый язык.
3. Язык должен являться расширением существующего решения, так как это избавляет пользователя от изучения нового языка запросов и сокращает время разработки.
4. Ошибка несоответствия типов во время исполнения запроса может приводить к потере огромного количества времени и ресурсов систем распределенной обработки неоднородных данных, поэтому становится важно предотвращать такие ошибки в момент трансляции запроса. Из этого следует, что язык должен предоставлять возможности для статического анализа запросов.

5.2. Грамматика языка высокого уровня

С учетом требований к языку было принято решение о выборе SQL в качестве базового языка. Была проведена работа по изменению грамматики стандарта SQL-1992 с целью внедрения поддержки параметризации, но при этом все возможности языка, кроме выборки и обработки данных были убраны, так как это не входило в цели данной работы. Грамматика SQL, в основном, не претерпела изменений, за исключением нескольких новых возможностей.

Во всех правилах грамматики, представленных в данном разделе, автор придерживается следующих обозначений.

- Текст, обрамленный одинарными кавычками, определяет неделимый текстовый блок, который должен присутствовать в выражении.
- Символ «?» обозначает опциональный блок.
- Символ «*» обозначает ноль или более повторений блока.
- Символ «+» обозначает одно и более повторений блока.
- Остальные текстовые обозначения являются названиями правил грамматики.

Также стоит отметить, что для описания грамматики используется расширенная форма Бэкуса-Наура, где приоритет правил задается очередностью их описания – самый высокий приоритет будет иметь первое правило.

Некоторые из основных правил грамматики языка высокого уровня представлены ниже:

1. query: 'fusion' paranthesizedParamList? query ('with' query)+ 'end'
   | '(' query ')'
   | 'select' paranthesizedParamList? selectList tableExpression?;

2. tableExpression: fromClause whereClause? groupByClause? havingClause? orderByClause? limitClause?.

Первое правило описывает вид самого запроса. Центральное место в нем занимает операция выборки. Также поддерживается синтаксис для слияния результирующих структур данных нескольких подзапросов и возможность задания приоритета выполнения запросов с помощью скобочной структуры.

Второе правило отвечает за выборку данных из источников и их обработку. Как можно заметить, большинство блоков опциональны, как и в стандартной грамматике SQL, что дает определенную гибкость в составлении запросов. Пример правила грамматики для операции группировки выглядит следующим образом:

groupByClause : 'group' paranthesizedParamList? 'by' exprList.

Данное правило определяет стандартную конструкцию группировки данных по нескольким выражениям, которые определяются правилом «exprList»:

exprList : expr (',' expr)*.

Каждое подправило «expr» задает определенное арифметическое выражение языка, ссылку на атрибут или подзапрос.

В перечисленных выше правилах грамматики фигурировал опциональный блок параметров ключевого слова– «paranthesizedParamList». Данный блок предназначен для конфигурирования работы алгоритмов обработки данных. Правила грамматики для данного блока представлено ниже:

```
paranthesizedParamList : '(' paramList ')';
paramList : param (',' param)*;
param : name=(String | StringLiteral) '=' expr.
```

Как можно заметить, этот блок является перечислением именованных значений, которые в свою очередь также могут быть подвыражениями языка и даже скалярным подзапросом.

Основные моменты грамматики языка высокого уровня перечислены в краткой форме. Полная грамматика в расширенной форме Бэкуса-Наура представлена в Приложении 1.

Для формирования представления о возможностях языка ниже приведены несколько запросов:

```
SELECT * FROM Table1
FUZZY JOIN (algorithm='merge_join', precise_bound=0.8)
Table2 ON Table1.a = Table2.a.
```

В данной примере запроса производится соединение двух таблиц специальным алгоритмом по приближенному равенству полей с определенной точностью.

```
SELECT Airports.name,
NEST(Hotels.name,
DISTANCE(Hotels.location, Airports.location; unit="km")) as Hotel
FROM Hotels JOIN Airports
ON DISTANCE(Hotels.location, Airports.location; unit="km") < 20
GROUP BY Hotels.name
```

Данный запрос для каждого отеля ищет набор аэропортов в пределах 20 км. Для вычисления расстояния применяется функция «distance»,

принимающая в качестве параметра единицы измерения.

Также стоит обратить внимание на поддержку нечетких запросов в данном языке. В обзоре существующих решений было рассмотрено несколько языков нечетких запросов и большинство из них достигали своих целей определением функций для работы с приближенными алгоритмами. В данной работе было также принято решение не нагружать синтаксис конструкциями для поддержки нечетких запросов, так как применение специальных функций и их параметризация будут решать поставленную задачу в полном объеме.

6. Транслятор языка высокого уровня в модель алгебры

В предыдущих разделах было описана модель алгебры нечетких запросов, грамматика языка высокого уровня и требования к нему. В этом разделе будет затронуто описание решения главной задачи данной работы – трансляции из определенного раннее языка в модель алгебры. В общем случае, под процессом трансляции будет подразумеваться преобразование одного синтаксического дерева в другое. Этот процесс следует разделить на трансляцию арифметических выражений, функций, ссылок на атрибуты и трансляцию самих операторов.

6.1. Трансляция арифметических выражения

Под арифметическими выражениями в данной работе подразумевается все, что не относится к операторам алгебры – математические выражения, вызовы функций, ссылки на атрибуты и т.д. Синтаксические деревья таких выражений практически не изменяются, поэтому детали данного этапа трансляции будут опущены. Осталось только заметить, что выражения могут содержать в том числе и подзапросы, о трансляции которых рассказывается в следующем разделе.

6.2. Трансляция операторов алгебры нечетких запросов

Исходя из описания синтаксиса языка высокого уровня, представленного выше, основную сложность представляет трансляция синтаксического дерева для операции выборки «SELECT».

В процессе работы над дизайном языка было принято решение, что за каждое ключевое слово в операции выборки будет отвечать свой оператор алгебры. Поэтому результат трансформации синтаксического дерева данной операции представляет собой набор вложенных операторов, которые

применяются согласно семантическому порядку ключевых слов в языке высокого уровня. Он определяется следующим образом:

1. FROM
2. WHERE
3. GROUP BY
4. HAVING
5. ORDER BY
6. LIMIT
7. SELECT

Согласно данному порядку определяется процесс трансляции операции выборки. Ниже пошагово описан данный процесс трансляции операции выборки в набор операторов алгебры нечетких запросов. Для наглядности входной запрос будет описываться на вводимом в данной работе языке, а результат трансляции – в формате XML, схема которого будет отражать структуру узла синтаксического дерева, отвечающего за конкретный оператор.

6.2.1. Ключевое слово FROM

Данное ключевое слово отвечает за выборку данных из источников и подзапросов, а также за соединение их с различными параметрами. Если в запросе представлен лишь один источник, то применяется специальный фильтр, предназначенный для извлечения данных. Если же источников несколько, то будет применяться оператор соединения, в котором источники данных будут являться аргументами, а условия соединения и другие параметры будут переведены в параметры оператора. Также важно определять для каждого аргумента оператора уникальный идентификатор, который может быть использован в условиях соединения. Ниже представлен пример части запроса с ключевым словом «FROM», в котором осуществляется соединение двух таблиц, и результат его трансляции в

выражение алгебры.

Запрос:

```
FROM first JOIN second ON…
```

Результат трансляции:

```
<query id="from" type="join">
  <parameters>
    ...
    <expression id="joinCondition">...</expression>
  </parameters>
  <arguments>
    <query id="first">...</query>
    <query id="second">...</query>
  </arguments>
</query>
```

6.2.2. Ключевое слово WHERE

Результат трансляции поддерева, за которое отвечает ключевое слово «FROM» переходит в аргумент к оператору фильтрации по условию, которое описывает ключевое слово «WHERE». Само условие транслируется как параметр алгебраического оператора фильтрации. Важно заметить, что у этого оператора часть, отвечающая за атрибуты – схема – преднамеренно оставляется пустой, что говорит о наследовании списка атрибутов от источника данных. Ниже представлен пример части запроса с ключевым словом «WHERE», в котором производится фильтрации данных по условию, и результат ее трансляции в выражение алгебры.

Запрос:

```
FROM first JOIN second ON…
WHERE …
```

Результат трансляции:

```
<query id="where" type="filter">
  <parameters>
    <expression id="whereCondition">...</expression>
  </parameters>
  <arguments>
    <query id="from">...</query>
  </arguments>
</query>
```

6.2.3. Ключевое слово GROUP BY

Если в запросе присутствует ключевое слово «GROUP BY», то эта конструкция будет транслироваться в алгебраический оператор . Единственным аргументом оператора будет являться результат трансляции предыдущих стадий, а в параметры попадут все выражения для группировки и сами параметры ключевого слова. Далее описан пример трансляции части запроса, содержащего операцию группировки.

Запрос:

```
FROM first JOIN second ON…
WHERE …
GROUP (algorithm=”k_means”) BY expr1, expr2
```

Результат трансляции:

```
<query id="group" type="aggregation">
  <parameters>
    <parameter name="algorithm">
      <expression type="const" value="k_means"/>
    </parameter>
    <expression id="groupByArg0">...exp0...</expression>
    <expression id="groupByArg1">...exp1...</expression>
  </parameters>
  <arguments>
    <query id="where">...</query>
  </arguments>
</query>
```

6.2.4. Ключевое слово HAVING

Данное ключевое слово отвечает за отсеивание результатов группировки, поэтому эту операция будет транслироваться в оператор фильтрации. Ниже описан пример трансляции операции фильтрации агрегированных значений.

Запрос:

```
FROM first JOIN second ON…
WHERE …
GROUP (algorithm=”k_means”) BY expr1, expr2
HAVING first.a = second.b
```

Результат трансляции:

```
<query id="having" type="filter">
  <parameters>
    <expression id="havingCondition" type="EQUAL">
      <expression type="ATTRIBUTE" ref="first.a"/>
      <expression type="ATTRIBUTE" ref="second.b"/>
    </expression>
  </parameters>
  <arguments>
    <query id="group" type="aggregation">...</query>
  </arguments>
</query>
```

6.2.5. Ключевое слово ORDER BY

Часть запроса, в котором присутствует ключевое слово «ORDER BY» будет транслироваться в специальный сортирующий фильтр по перечисленным в запросе аргументам с модификаторами порядка. Пример данной трансляции можно опустить ввиду ее схожести с предыдущими.

6.2.6. Ключевое слово LIMIT

Данная операция ограничивает мощность результирующего множества и

транслируется также в специальный фильтр, отвечающий за это действие. Пример данной трансляции будет также опущен из-за его схожести с другими.

6.2.7. Ключевое слово SELECT

Этот оператор единственный, у которого есть непустое определение выходных атрибутов, в которые транслируется список пользовательских аргументов проекции. Параметры самого ключевого слова кладутся в параметры оператора фильтрации. Далее представлен пример заключительного этапа трансляции запроса – проекции на выбранные выражения.

Запрос:

```
SELECT a, COUNT(b)
FROM first JOIN second ON…
WHERE …
GROUP (algorithm=”k_means”) BY expr1, expr2
HAVING first.a = second.b
```

Результат трансляции:

```
<query id="select" type="filter">
  <schema>
    <expression type="ATTRIBUTE" ref="first.a"/>
    <expression type="FUNCTION" name="count">
      <expression type="ATTRIBUTE" ref="second.b"/>
    </expression>
  </schema>
  <arguments>
    <query id="having" type="filter">...</query>
  </arguments>
</query>
```

6.2.8. Ключевое слово FUSION

Трансляция данной операции сводится к использованию оператора

слияния, в аргументы которого транслируются подзапросы. Пример данного не будет представлен ввиду отсутствия новых деталей.

6.2.9. Особые случаи трансляции

Описанная выше схема трансляции обошла стороной два важных оператора алгебры – вложения и развертки. Они оба не были упомянуты в списке операторов алгебры нечетких запросов в описании модели, так как являются частными случаями группировки и соединения соответственно. Оператор вложения создает атрибут-контейнер для сгруппированных значений, указанных в запросе. Основываясь на семантике данного оператора было принято решение представлять его в языке высокого уровня в качестве специальной функции, принимающей в качестве аргументов те атрибуты и выражения, группа из которых должна быть вложена в новый именованный атрибут.

Что касается оператора развертки, который является противоположным к оператору вложения, то есть предназначенный для разворачивания вложенных структур, то было решено использовать для его обозначения специальную конструкцию соединения источника данных и атрибута его объекта:

SELECT * FROM **TABLE JOIN TABLE.NESTED_ATTRIBUTE** …

Такой подход позволяет сократить количество новых деталей в языке, является семантически интуитивно понятным, а также применяется в существующих базах данных, имеющих такие возможности.

6.3. Особенности реализации

В данном разделе рассматриваются особенности реализации поставленных в этой работе задач, а также технологии, которые применяются для их выполнения.

6.3.1. Архитектура

Для более наглядного представления архитектуры реализации на рис. 1 изображена диаграмма компонентов транслятора:

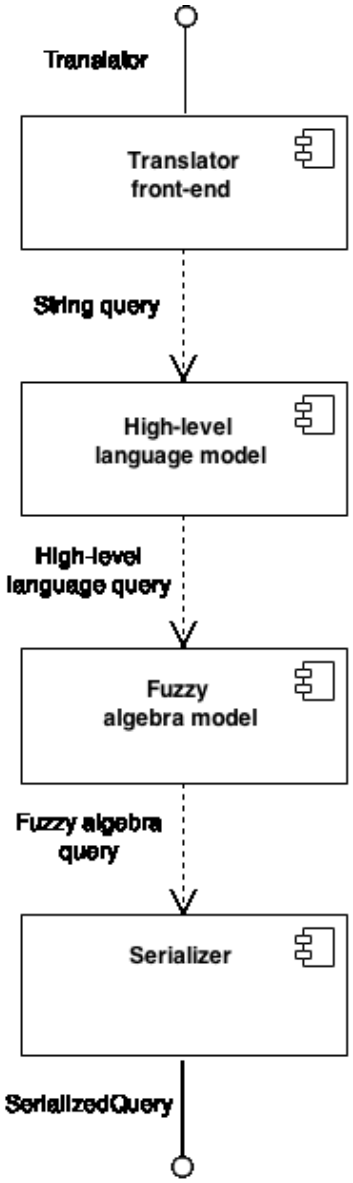

Рисунок 1: Диаграмма компонентов транслятора

Реализуемый в данной работе транслятор имеет несколько независимых компонент: модель языка высокого уровня, модель алгебры нечетких запросов и модель сериализации. Такое разделение позволяет уменьшить зависимость между стадиями трансляции и независимо над ними работать [12], оставляя межмодульное взаимодействие на уровне интерфейсов. Работа данных модулей организована последовательно, но при этом позволяет добавить дополнительные стадии трансляции. В итоге, полученная схема взаимодействия дает возможности для расширения, масштабирования и независимой разработки компонент.

6.3.2. Технологии

Разработка программной функциональности велась на объектно-ориентированном языке программирования Java с применением библиотеки ANTLR [13]. С ее помощью был сгенерирован синтаксический анализатор по заданным правилам в расширенной форме Бэкуса-Наура, полный список которых представлен в Приложении 1. На рис. 2 представлена диаграмма последовательности работы транслятора с пользовательским запросом:

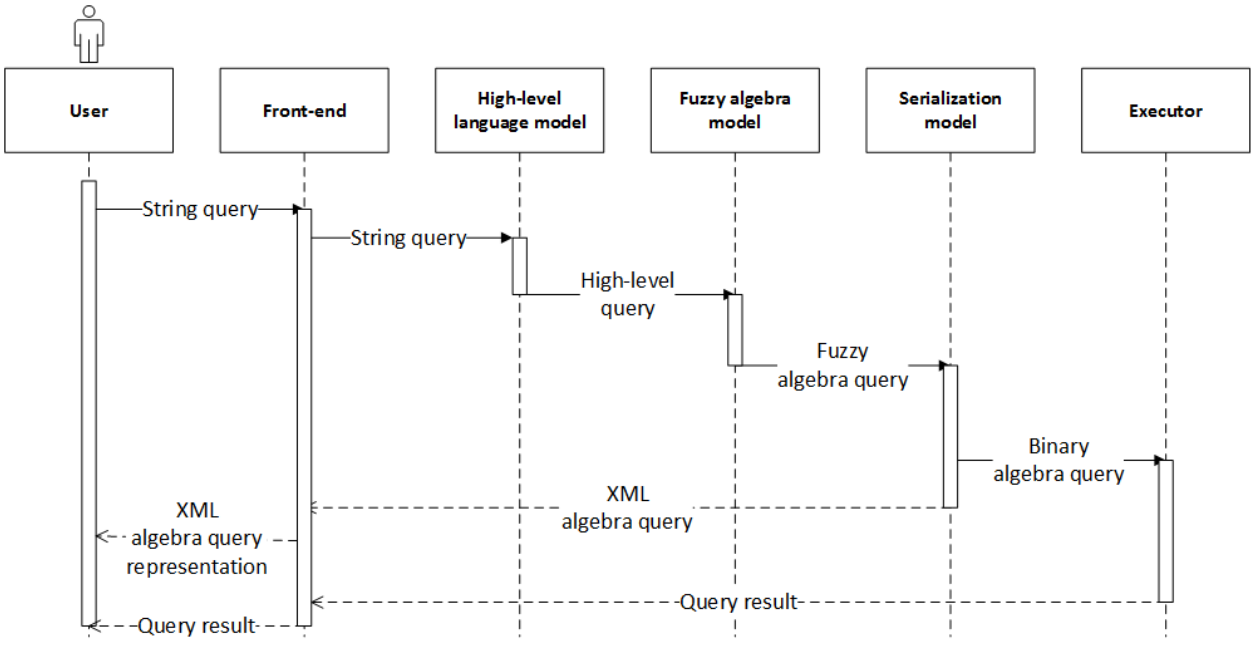

Рисунок 2: Диаграмма последовательности работы транслятора

Данный анализатор принимает на вход строку пользовательского запроса и строит синтаксическое дерево в соответствии с определенными правилами грамматики. Далее производится обход дерева, целью которого является построение синтаксического дерева языка высокого уровня, которое является более удобным для дальнейшей трансляции. После построения итогового дерева выражений алгебры нечетких запросов пользователь имеет возможность сериализовать его в удобный для представления формат XML или отправить дерево данного запроса на исполнение с помощью бинарного протокола Thrift [14], который позволяет сократить объем передаваемой по сети информации.

Для тестирования реализованной функциональности применялась библиотека модульного тестирования Junit [15]. Для каждого правила грамматики были написаны тесты, проверяющие правильность построения синтаксического дерева.

В качестве системы сборки проекта использовалась система Maven [16]. Данная система сборки позволяет собрать проект как в качестве библиотеки, так и в качестве самостоятельного модуля для трансляции.

7. Апробация

Как упоминалось в предыдущем разделе, реализованная функциональность была тщательно протестирована с помощью модульных тестов. Это тестирование включало в себя как независимые тесты моделей алгебры, языка, правил грамматики, так и тесты на совместное взаимодействие.

В данной работе тестирование играет важную роль ввиду множества различных деталей, которые должны взаимодействовать друг с другом правильным образом. Грамматика созданного языка включает в себя 28 синтаксических правил. Так как грамматика описана в расширенной форме Бэкуса-Наура, то каждое правило может иметь несколько ветвлений. Исходя из этого, каждое правило и каждое его ветвление было протестировано с помощью модульных тестов. Тесты проверяли правильность построения синтаксического дерева языка высокого уровня по текстовому запросу. Не менее важную роль играло и тестирование самого процесса трансляции в модель алгебры нечетких запросов, каждое правило которого также проверялось тестами. В итоге правильность работы реализованной функциональности проверяется 47 модульными тестами, что дает полное тестовое покрытие транслятора.

Например, рассмотрим комплексный тест транслятора, принимающий на вход пользовательский запрос и возвращающий в качестве результата синтаксическое дерево алгебраических операторов, выраженное с помощью формата XML:

```
SELECT asterix.name, file.name
FROM file JOIN (algorithm = "merge_join") asterix
        ON asterix.department_id = file.id
WHERE asterix.age > 30
```

В этом запросе совмещено несколько особенностей как стандартного SQL, так и расширенного в данной работе: выборка нескольких значений,

соединение двух источников данных с использованием определенного алгоритма, указанного в параметрах, и фильтрация по значению поля. Результатом трансляции запроса является следующая XML конструкция:

```
<query type="FILTER">
 <schema>
  <expression id="name" type="ATTRIBUTE" value="asterix.name"/>
  <expression id="name" type="ATTRIBUTE" value="file.name"/>
 </schema>
 <arguments>
  <query type="FILTER">
   <parameters>
    <parameter name="filterExpression">
     <expression type="GT">
      <arguments>
       <expression id="age" type="ATTRIBUTE" value="asterix.age"/>
       <expression type="CONSTANT" value="30"/>
      </arguments>
     </expression>
    </parameter>
   </parameters>
   <arguments>
    <query id="asterix" type="JOIN">
     <parameters>
      <parameter name="joinType">
       <expression type="CONSTANT" value="inner"/>
      </parameter>
      <parameter name="onCondition">
       <expression type="EQUAL">
        <arguments>
         <expression id="department_id"
                     type="ATTRIBUTE" value="asterix.department_id"/>
         <expression id="id" type="ATTRIBUTE" value="file.id"/>
        </arguments>
       </expression>
      </parameter>
      <parameter name="algorithm">
       <expression type="CONSTANT" value="merge_join"/>
      </parameter>
     </parameters>
     <arguments>
      <query id="file" type="PRIMARY">
       <parameters>
        <parameter name="sourceName">
         <expression type="CONSTANT" value="file"/>
        </parameter>
       </parameters>
      </query>
      <query id="asterix" type="PRIMARY">
       <parameters>
        <parameter name="sourceName">
         <expression type="CONSTANT" value="asterix"/>
        </parameter>
```

```
            </parameters>
          </query>
        </arguments>
      </query>
    </arguments>
  </query>
 </arguments>
</query>
```

Данное представление синтаксического дерева операторов алгебры для указанного запроса выглядит громоздко, но в то же время полностью описывает его специфику. Структура дерева создается в соответствии с семантическим порядком ключевых слов и порядком их трансляции. Сначала происходит трансляции блока «FROM» - это операция соединения двух источников. После этого происходит фильтрация данных, согласно блоку «WHERE». В конце результирующие данные проецируются на заданный список атрибутов.

Совместно с Алексеем Самариным – студентом кафедры системного программирования СПбГУ - был также проведен интеграционный тест, который включал в себя взаимодействие транслятора с исполнителем запросов, разрабатываемым Алексеем в рамках дипломной работы, и двумя разнородными удаленными источниками данных.

В качестве запроса к системе выступал описанный раннее запрос, представление в формате XML которого отправлялось на удаленную машину исполнителя. В запросе фигурируют два источника данных – «asterix» и «file». Первый источник является адаптером к системе хранения данных AsterixDB [17], который был разработан автором в рамках курсовой работы [18]. Этот адаптер транслирует выражения расширенной алгебры в язык запросов AsterixDB и возвращает результат исполнения. Второй источник является адаптером для файловой системы удаленной машины, способным извлекать данные из файла и отправлять их обработчику.

Исполнитель запросов с помощью оптимизатора изменил структуру дерева запроса, которое он получил от транслятора языка высокого уровня. Изменение заключалось в применении операции фильтрации блока

«WHERE» до того, как будет произведено соединение источников, так как это уменьшает количество данных, передаваемых по сети. В связи с этим адаптер AsterixDB отправил обработчику операции соединения, только отфильтрованные данные и в дальнейшем фильтрация не применялась. Сам обработчик также принял данные с файлового адаптера и успешно соединил их по заданному условию, что и являлось целью данного интеграционного теста.

8. Заключение

В ходе выполнения данной работы были получены следующие результаты.

1. Изучена предметная область – системы для обработки разнородных данных и языки нечетких запросов
2. Реализовано представление модели алгебры нечетких запросов для использования в процессе трансляции.
3. Создан язык высокого уровня для алгебры нечетких запросов. Язык является расширением существующего языка запросов SQL, и полностью охватывает возможности алгебры.
4. Реализован транслятор для языка высокого уровня в модель алгебры нечетких запросов. Транслятор имеет несколько независимых модулей, отвечающих за различные стадии процесса трансляции, что позволяет его легко расширять и вести независимую доработку модулей.
5. Выполнена апробация реализованной функциональности; проведено модульной тестирование независимых компонент, правил грамматики, а также проведен интеграционный тест с использованием исполнителя запросов и двух источников данных – системы хранения данных AsterixDB и файловой системы удаленной машины.
6. Результаты выполнения данной дипломной работы представлены на ISSAT 2015.

Созданный в рамках данной работы язык высокого уровня и транслятор для него может использоваться в дальнейших исследованиях в рамках проекта кафедры информационно-аналитических систем Математико-механического факультету СПбГУ по созданию системы точной и приближенной обработки разнородных данных.

Кроме того, грамматика языка и функционал его транслятора может быть расширен и доработан в следующих направлениях.

- Работа над синтаксисом, его конкретизация в рамках нужд пользователей.
- Введение строгой типизации данных: поддержка данной возможности будет требовать знание метаинформации об источниках данных, сигнатурах функций и возможностях исполнителей запросов.
- Поддержка дополнительных операторов алгебры нечетких запросов: в рамках данной работы были рассмотрены все базовые оператора, но в стороне остались несколько операторов, являющихся частными случаями базовых: например, оператор одновременной группировки и соединения, позволяющий значительно оптимизировать план исполнения запроса.

Список литературы

[1] *Y. Ioannidis*. Query optimization // ACM Computing Surveys, Volume 28 Issue 1, pages 121-123, 1996.

[2] *B. Novikov, A. Vassilieva, A. Yarygina*. Querying Big Data // Proceedings of the 13th International Conference of Computer Systems and Technologies, pages 1-10, 2012.

[3] *C. Strauch*. NoSQL Databases // Stuttgart Media University, Stuttgart, 2011.

[4] Oracle Database.

URL: http://www.oracle.com/ru/database/overview/index.html

[5] PostgreSQL. URL: http://www.postgresql.org/

[6] Project Forward. URL: http://forward.ucsd.edu/

[7] Pentaho Business Intelligence Suite. URL: http://www.pentaho.com/

[8] *Y. P. Kian Win Ong, Yannis Papakonstantinou, Romain Vernoux*. The SQL++ Query Language: Configurable, Unifying and Semi-structured // Technical report. 2015.

[9] FSQL. URL: http://www.lcc.uma.es/~ppgg/FSQL/

[10] *J. Galindo*. Handbook of Research on Fuzzy Information Processing in Databases // IGI Global, New York, 2008.

[11] *S. Nepal, M.V. Ramakrishna, J.A. Thom.* A Fuzzy Object Query Language(FOQL) for Image Databases // Proceedings of the 6th International Conference on Database Systems for Advanced Applications, pages 117-124, Hsinchu, 1999.

[12] *M. Fowler*. Domain-Specific Languages // Addison-Wesley, 2010.

[13] ANTLR. URL: http://www.antlr.org/

[14] Apache Thrift/ URL: https://thrift.apache.org/

[15] JUnit URL: http://junit.org/

[16] Apache Maven. URL: https://maven.apache.org/

[17] *S. Alsubaiee, Y. Altowim, H. Altwaijry, A. Behm, V. Borkar, Y. Bu, M. Carey, K. Faraaz, E. Gabrielova, R. Grover, Z. Heilbron, Y.-S. Kim, C. Li, G. Li, J. M. Ok, N. Onose, P. Pirz.* AsterixDB: A Scalable, Open Source BDMS // Proceedings of the VLDB Endowment, Vol. 8, No. 2, 2014.

[18] *N. Saveliev.* Implementation of Generalized Relational Algebraic Operations with AsterixDB BDMS // 18th East European Conference on Advances in Databases and Information Systems and Associated Satellite Events, Part X, pages 323-332, Ohrid, 2014.

Приложение 1. Грамматика языка высокого уровня

В данном приложении приводится полный список правил грамматики для созданного языка в расширенной форме Бэкус-Наура с использованием формата записи библиотеки ANTLR.

1. Запросы

```
query : ('FUSION'|'fusion') paranthesizedParamList? query ( ('WITH'|'with') query)+ ('END'|'end')
      | LEFT_PAREN query RIGHT_PAREN
      | ('select'|'SELECT') paranthesizedParamList? (selectList | MULTIPLY) tableExpression?
selectList : selectItem (COMMA selectItem)* ;
selectItem : ('NEST' | 'nest') LEFT_PAREN selectList (SEMI_COLON paramList)? RIGHT_PAREN asClause?
           | expr asClause?
tableExpression : fromClause whereClause? groupByClause? havingClause? orderByClause? limitClause? ;
```

2. Ключевое слово FROM

```
fromClause : ('FROM'|'from') paranthesizedParamList? tableReferenceList;
tableReferenceList :tableReference (COMMA tableReference)*;
tableReference : tablePrimary joinedTablePrimary*;
joinedTablePrimary : JoinType? ('JOIN'|'join') paranthesizedParamList? tablePrimary (('ON'|'on') expr)?;
JoinType : ('INNER' | 'inner')
         | ('FUZZY' | 'fuzzy')
         | ('LEFT' | 'left')
         | ('RIGHT' | 'right')
         | ('FULL' | 'full')
tablePrimary: tableOrQueryName asClause? paranthesizedColumnNameList?
            | LEFT_PAREN query RIGHT_PAREN asClause paranthesizedColumnNameList?
tableOrQueryName : StringLiteral | Field;
paranthesizedColumnNameList : LEFT_PAREN columnNameList RIGHT_PAREN;
columnNameList : StringLiteral (COMMA StringLiteral)*;
```

3. Ключевое слово WHERE

whereClause : ('where'|'WHERE') expr;

4. Ключевое слово GROUP BY

groupByClause : ('group'|'GROUP') paranthesizedParamList? ('by'|'BY') exprList;

5. Ключевое слово HAVING

havingClause : ('having'|'HAVING') expr;

6. Ключевое слово ORDER BY

orderByClause : ('order'|'ORDER') paranthesizedParamList? ('by'|'BY') orderByArg (COMMA orderByArg)*;

orderByArg : expr OrderType?;

OrderType : 'ASC' | 'asc' | 'desc' | 'DESC';

7. Ключевое слово LIMIT

limitClause : ('limit'|'LIMIT') expr;

8. Выражения

```
expr : BIT_NEG expr
    | MINUS expr
    | <assoc=right> expr POWER expr
    | expr op=(MULTIPLY|DIVIDE|MODULAR) expr
    | expr op=(PLUS|MINUS|BIT_AND|VERTICAL_BAR|BIT_XOR) expr
    | expr CONCATENATION_OPERATOR expr
    | ('NOT'|'not') expr
    | expr comp=(EQUAL|GTH|LTH|GEQ|LEQ|NOT_EQUAL) expr
    | expr ('IS'|'is') expr
    | expr ('AND'|'and') expr
    | expr ('OR'|'or') expr
    | ('CAST'|'cast') expr ('as'|'AS') StringLiteral
    | ('CASE'|'case') expr? caseWhenClause+ caseElseClause? ('END'|'end')
    | query
    | (NumberLiteral | String)
    | Field
    | StringLiteral (LEFT_PAREN exprsAndParams RIGHT_PAREN)?
    | LEFT_PAREN expr RIGHT_PAREN
```

```
asClause : ('as'|'AS')? StringLiteral;

exprsAndParamS : (exprListOrAll SEMI_COLON)? paramList
                | exprListOrAll?

paranthesizedParamList : LEFT_PAREN paramList RIGHT_PAREN;

paramList : param (COMMA param)*;

param : name=(String | StringLiteral) EQUAL expr;


exprListOrAll : exprList | MULTIPLY ;

exprList : expr (COMMA expr)*;

caseWhenClause : ('WHEN' | 'when') w=expr ('THEN' | 'then') t=expr;

caseElseClause : ('ELSE' | 'else') expr;
```

9. Литералы

```
NumberLiteral : Real
              | Int
              | BoolConstant

BoolConstant : 'FALSE' | 'false' | 'TRUE' | 'true' | 'NULL' | 'null' ;

Real : (RealWithoutExp | Int) ([eE] Sign? Int)* ;

RealWithoutExp : ('0' | (Int+)) '.' Digit*
                | '.' Digit+

Int : (NonZeroDigit+ Digit*) | Digit ;

Field : StringLiteral (DOT StringLiteral)+;

StringLiteral: [a-zA-Z] [a-zA-Z0-9_]*;

String: "" (Esc|.)*? "" ;

LineComment : MinusSign MinusSign .*? '\r'? '\n' -> skip ;

ASSIGN  : ':=';

EQUAL  : '=';

COLON :  ':';

SEMI_COLON :  ';';

COMMA : ',';

CONCATENATION_OPERATOR : VERTICAL_BAR VERTICAL_BAR;

NOT_EQUAL  : '<>' | '!=';

LTH : '<' ;
```

```
LEQ : '<=';

GTH   : '>';

GEQ   : '>=';

LEFT_PAREN :  '(';

RIGHT_PAREN : ')';

PLUS  : '+';

MINUS : '-';

MULTIPLY: '*';

POWER : '**';

DIVIDE  : '/';

MODULAR : '%';

DOT : '.';

UNDERLINE : '_';

BIT_AND : '&';

BIT_XOR : '^';

BIT_NEG : '~';

VERTICAL_BAR : '|';

QUOTE : '\'';

DOUBLE_QUOTE : '"';

Digit : [0-9] ;

fragment Sign : [+-] ;

fragment MinusSign : '-' ;

fragment NonZeroDigit : [1-9] ;

fragment Esc : '\\"' | '\\\\' ;

WS : [\t \r\n] -> skip;
```