УДК 004.02 Дата подачи статьи: 04.06.18
DOI: 10.15827/0236-235X.124.659-666 2018. Т. 31. № 4. С. 659–666

Грамматика запросов для хранилища разнородных данных в проактивных системах

Чан Ван Фу [1], *аспирант, vanphu.vstu.russia@gmail.com*
М.В. Щербаков [1], *д.т.н., старший научный сотрудник, maxim.shcherbakov@gmail.com*
Сай Ван Квонг [1], *аспирант, svcuonghvktqs@gmail.com*

[1] *Волгоградский государственный технический университет, г. Волгоград, 400005, Россия*

Проблема хранения и обработки разнородных данных (как структурированных, так и неструктурированных), получаемых из различных источников, актуальна при реализации систем проактивной поддержки этапа функционирования жизненного цикла сложных многообъектных распределенных систем. Разнородность данных требует хранения их описания в виде метаданных для последующей автоматической обработки. Для решения проблемы эффективного хранения разнородных данных используется архитектура, называемая «озеро данных». Она реализует механизмы пакетной обработки данных и обработки данных в режиме реального времени.

Задача совершенствования методов для эффективного доступа к разнородным данным включает в себя следующие подзадачи: разработка грамматики запросов типа SQL к разнородным данным, построение парсера для распознавания запросов в соответствии с новой грамматикой, разработка модулей обработки разнородных данных в соответствии с запросом, разработка рекомендаций (методики) применения разработанных модулей в проактивных системах поддержки принятия решений.

Предлагаемая грамматика основана на расширениях DML языка SQL, в частности, расширениях оператора SELECT. Для обработки сформированных запросов в соответствии с новой грамматикой сгенерирован парсер с использованием библиотеки ANTLR 3.0. В результате генерации созданы классы на языке Java, объекты которых используются для разбора запросов. Реализовано и испытано ПО на основе сгенерированного парсера и модулей обработки разнородных данных. В результате испытания ПО, реализующего предложенную грамматику в системе проактивной поддержки принятия решений, проанализировано время выполнения унифицированных запросов с различными объемами разнородных данных.

Главным результатом применения грамматики является снижение времени на обработку разнородных данных в рамках одного запроса.

Ключевые слова: *проактивные системы, поддержка принятия решений, сбор данных, распределенная обработка данных, озеро данных, разнородные данные.*

Обеспечение функционирования сложных систем на всех этапах жизненного цикла является актуальной задачей, направленной на минимизацию стоимости владения на заданном временном горизонте [1, 2]. При реализации обеспечивающих систем решается задача поддержки принятия эффективных управленческих решений по влиянию на целевой объект. Одним из направлений развития систем поддержки принятия решений при обеспечении эффективного функционирования целевых систем являются проактивные вычисления. Под проактивной поддержкой обеспечения этапа функционирования сложных систем понимается двухуровневая система, на первом уровне которой осуществляются формирование и поддержка принятия *предсказательных* управленческих решений, направленных на устойчивое функционирование целевой системы (то есть функционирование обеспечивающей системы), на втором – управление процессом поддержки принятия предиктивных управленческих решений, то есть управление обеспечивающей системой при минимизации участия человека. На первом уровне необходимо решить задачу сбора и хранения данных о целевой системе и идентификации.

Традиционно схема данных определяется на этапе проектирования, и любые изменения приводят к пересмотру архитектуры систем. Задача усложняется при наличии большого объема данных и разнотипных данных о целевой системе, поступающих из различных источников. Например, данными могут быть показания датчиков о состоянии целевой системы и видео с камер наблюдения за объектом.

Для решения проблемы больших данных используются распределенные архитектуры с пакетной обработкой данных и обработкой в режиме реального времени [3–6]. Вторая проблема требует большего внимания, так как типовые решения, основанные на организации хранилища сырых данных и метаданных, могут быть неэффективны с позиции выполнения запроса, охватывающего разнородные данные. Например, необходимо извлечь информацию из данных, полученных с датчиков и из видеопотоков. В этой ситуации при выполнении запроса пользователю необходимо проанализировать метаданные, выбрать необходимые данные из разнотипных источников и подключить соответствующие механизмы их обработки. Все это сказывается на времени выполнения запроса, а следовательно, на времени принятия решений.

Поэтому цель данной работы заключается в создании единого интерфейса запросов к хранилищу данных, сформированного по принципу озера дан-

ных. Предлагается грамматика для формирования запросов к разнородным данным с целью сокращения времени обработки запросов. Приведем пример использования грамматики для извлечения информации из разнородных источников для задачи запроса к разнородным данным, представленным лог- и видеофайлами с информацией о перемещении транспортных средств.

Обзор существующих подходов

К настоящему времени накопилось достаточно много теоретических и практических результатов в области эффективного распределенного хранения и пакетной обработки больших объемов данных. Тем не менее, крупные компании (такие как Microsoft, Amazon, Google) направляют усилия на поиск новых инфраструктурных решений для поддержки обработки корпоративных данных. В частности, технология Azure – пример инфраструктурных решений, использующих понятие озера данных (data lake). Идея этой концепции очень проста в формулировке (размещаем разнотипные данные с разнородных источников в едином хранилище и обеспечиваем единый интерфейс доступа), но сложна в реализации. В озере данных Azure предоставляются все возможности (ограниченные моделями монетизации) хранения данных любых объема, формата и скорости передачи, а также выполнения различных видов обработки и анализа на разных платформах и языках. Озеро данных Azure решает задачу получения и хранения всех данных одновременно в пакетном, потоковом и интерактивном режимах [7].

Для решения задач аналитики данных в Azure используется расширенный язык U-SQL. Он объединяет декларативный SQL и императивный C#, позволяя обрабатывать данные любого объема. С помощью U-SQL можно обрабатывать неструктурированные данные, применяя схему к пользовательской логике чтения и вставки, а также к определяемым пользователем функциям. Озеро данных Azure использует современные технологии для хранения и обработки данных в потоковом и пакетном режимах, как Spark [5]. Несмотря на то, что U-SQL также используется для интеграции структурированных, полученных из регуляционных БД и бинарных данных (как изображения), недостатком U-SQL является отсутствие методов для обработки разнородных данных в формате одного запроса [8].

Для решения задачи интеграции больших данных часто используется технология процессов ETL (Extract – Transform – Load), реализуемых в платформах Informatica Big Data Edition, SAS Data Integration Studio. Платформа Informatica Big Data Edition используется как ETL-платформа для пакетной обработки данных на основе Hadoop, SAS Data Integration Studio – как ETL-платформа для обработки данных в корпоративном хранилище данных (Greenplum).

Однако ETL-технологии, предназначенные в основном для реляционных БД, обычно построены на использовании вспомогательного ПО, которое в результате нивелирует преимущества обработки больших данных на стороне сервера. Для решения этой задачи используется модификация ELT (Extract – Load – Transform), адаптированная для интеграции больших данных. В [9] разработчиками из компании Oracle описана архитектура системы интеграции больших данных на основе ELT. Кроме этого, существует ряд платформ компаний, которые поддерживают процесс ELT, такие как Hadoop/Apache Hive & Spark, IBM, Informatica, Talend, Teradata, Microsoft SQL Server/SSIS, Vertica. На рисунке 1 представлены традиционная и современная архитектуры системы интеграции больших данных [9].

В современной архитектуре системы интеграции больших данных Sqoop используются как инструмент, входящий в экосистему Hadoop, для пересылки данных между структурированными хранилищами и HDFS [11]. Этот инструмент предназначен для разовой загрузки данных в HDFS Hadoop из реляционной БД.

Можно выделить два подхода к организации хранения сырых данных. При первом подходе данные сохраняются как есть и обработка происходит

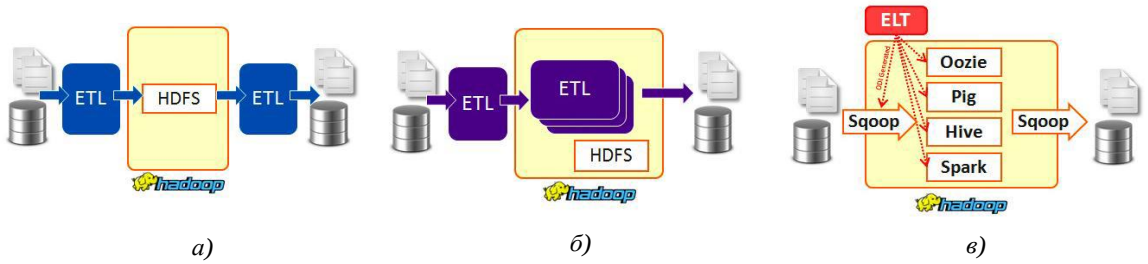

Рис. 1. Примеры архитектуры системы интеграции больших данных:
а) традиционная архитектура ETL, б) традиционная архитектура ETL на основе Hadoop,
в) современная архитектура ELT нативная в Hadoop

Fig. 1. The examples of architecture of a big data integration system: a) traditional ETL architecture,
б) traditional Hadoop ETL architecture, в) modern native Hadoop ELT architecture

в процессе запроса. При втором подходе, кроме исходных данных, в хранилище размещаются предварительно обработанные данные. Второй способ позволяет сократить время на обработку запросов. В частности, в [3, 4] описаны методы для предобработки и интеграции данных, полученных из различных источников при анализе транспортных задач. Однако можно выделить ряд проблем при разработке систем обработки больших данных:

– высокая сложность при сборе разнородных данных, приводящая к снижению производительности всей системы обработки больших данных;

– задержки при реализации запросов к хранилищу данных, содержащих разнородные данные.

Для решения этих актуальных задач требуется разработать эффективный механизм унифицированных запросов к хранилищу с разнородными данными. В работе используются два типа данных, которые должны обрабатываться запросами, – лог-файлы и видеопотоки.

Грамматика запросов к озеру данных

Была спроектирована система обработки разнородных данных с использованием унифицированных запросов для хранилища разнородных данных в концепции проактивных систем поддержки принятия решений. На рисунке 2 представлена схема предлагаемого решения.

Пользователь задает запрос к разнородным данным, пример которых представлен в таблице 1.

Модуль парсинга и анализа SQL-запроса используется для разбора пользовательского запроса, грамматика которого будет описана далее. На основе результата модуля парсинга и анализа SQL-запроса модуль обработки разнородных данных использует различные библиотеки, например, библиотеку для обработки лог-файлов, библиотеку обработки изображений для обращения к хранилищам разнородных данных. Результаты агрегируются и выводятся пользователю.

Таблица 1

Пример запросов к хранилищу формата «озеро данных»

Table 1

The example of queries to the data lake storage

Запрос	Цель запроса
select count from DATALOG, Camera1 where MODIFIED > 1516051435073 and MODIFIED < 1516661699999	Вернуть количество транспортных объектов из источников DATALOG (в виде лог-файла) и CAMERA1 (в виде фреймов видеофайла), которые отправляют данные на сервер в интервалах $t_{modified}$ > 1516051435073 и $t_{modified}$ < 1516661699999
select count from DATALOG, CAMERA2 where longitude > 44 and longitude < 45 and latitude > 45 and latitude < 46	Вернуть количество транспортных объектов в определенной области (longitude > 44 and longitude < 45 and latitude > 45 and latitude < 46) из источников Datalog (в виде лог-файла) и camera2 (в виде фреймов видеофайла)

В результате системного анализа процесса сбора и структуры данных была предложена модель хранения разнородных данных:

$$S = \langle \{DT\}_{i=1}^{n_{ST}}, \{SS\}_{j=1}^{m_{SS}}, \{E\}_{k=1}^{p_E}, IS, DS \rangle, \qquad (1)$$

где $\{DT\}_{i=1}^{n_{ST}}$ – множество шаблонов данных; n_{ST} – количество шаблонов данных; $\{SS\}_{j=1}^{m_{SS}}$ – методы разбиения разнородных данных; m_{SS} – количество типов данных; $\{E\}_{k=1}^{p_E}$ – множество исполнителей задач сбора данных; IS – метод индексирования данных в хранилище озера данных; DS – структура озера разнородных данных.

Лог-файлы размещаются в файловой системе в определенных каталогах и имеют структуру, аналогичную JSON:

```
{
  "VehicleID":"0a955f61-65c3-44f9-8893-f49163225c05",
  "Speed":78.0,
  "Timestamp":"1516661614478",
  "Latitude":"45.07256",
  "Longitude":"43.995274"
}
```

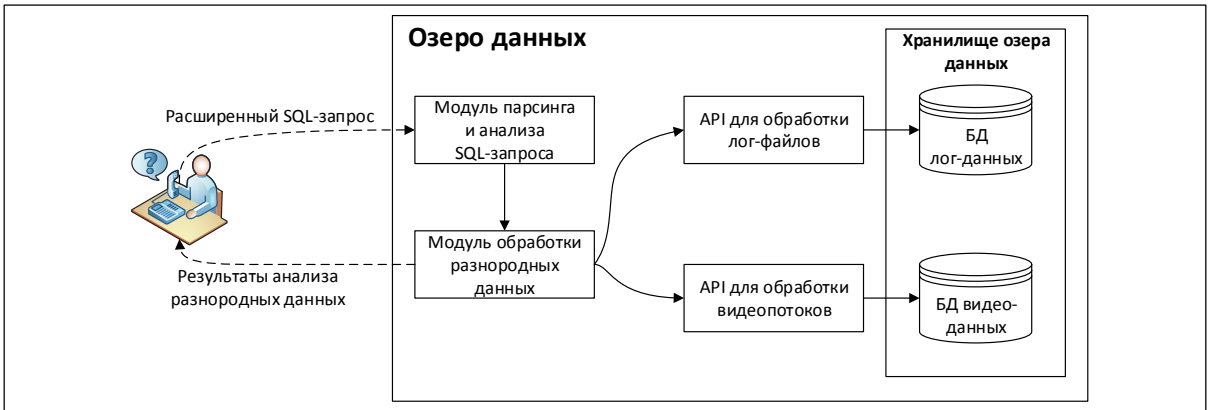

Рис. 2. Архитектура системы обработки разнородных данных с использованием унифицированных запросов

Fig. 2. The architecture of a heterogeneous data processing system using unified queries

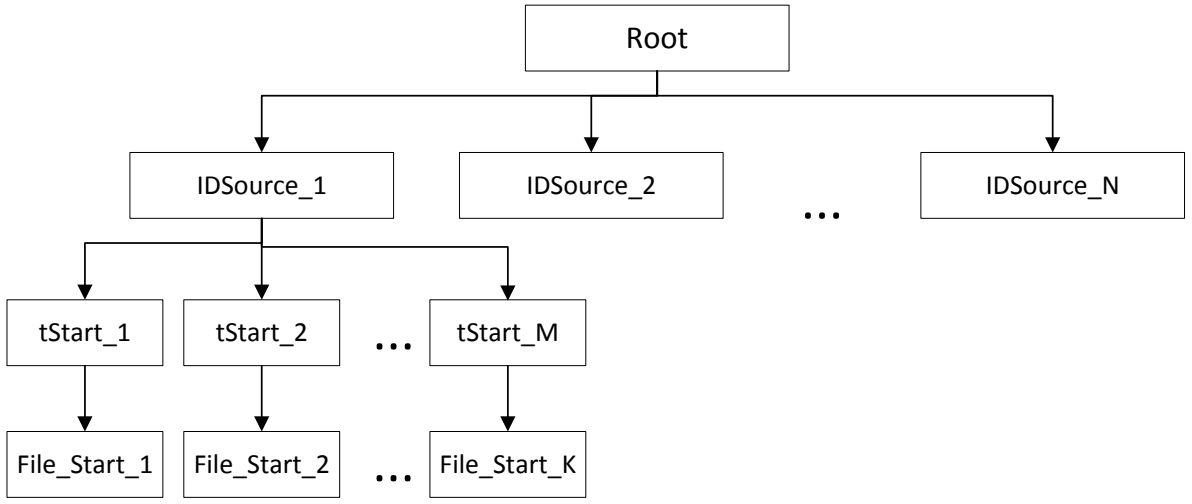

Рис. 3. Общая структура хранения данных

Fig. 3. A general data storage structure

Видеопотоки хранятся как в исходном формате, так и в виде фреймов (изображений в формате PNG) в заданном заранее каталоге. Общая структура хранения данных представлена на рисунке 3.

Для выборки данных в соответствии с запросами (табл. 1) были сформулированы и решены следующие задачи.

Задача 1. Создание расширенного языка SQL. Построена грамматика, соответствующая структурам запросов типа SQL для выборки данных, определены необходимые лексеры и токены для построенной грамматики.

Задача 2. Построение эффективного парсера для распознавания запросов типа SQL. На данном этапе использована библиотека ANTLR 3.0 для генерации кода парсера. Результатом являются классы, которые используются для разбора запросов.

Задача 3. Разработка модулей для извлечения и обработки разнородных данных (видеофайлы, лог-файлы) из хранилища разнородных данных.

Задача 4. Разработка методики применения разработанных модулей для выборки разнородных данных.

Опишем грамматику для формирования запросов к разнородным данным.

Общая грамматика инструкции, реализующей выбор данных (select), имеет следующий вид:

query : select_stmt where_stmt;

где *select_stmt* – оператор выбора;

where_stmt – оператор условий фильтрации результатов;

Визуальная нотация инструкции SELECT представлена на рисунке 4.

Инструкция SELECT выглядит следующим образом:

select_stmt
: 'select' 'count' from_stmt
;

где *from_stmt* – оператор выбора источников данных

from_stmt
: ('from' directoryname (',' directoryname)) ?*

count – оператор подсчета количества объектов.

Схема инструкции SELECT с оператором *COUNT* представлена на рисунке 5.

Рис. 4. Схема инструкции SELECT

Fig. 4. The SELECT statement schema

Рис. 5. Схема инструкции SELECT с оператором COUNT

Fig. 5. A schema of the SELECT statement with COUNT

На рисунке 6 представлена схема инструкции SELECT с оператором *from_stmt* выбора источников данных.

Оператор условия результата имеет вид:

WHERE_STMT
: ('where' clause ('and' clause))?*

Схема оператора *WHERE_STMT* представлена на рисунке 7.

Выражение *CLAUSE* может содержать информацию о названии файла, типе файла или времени модификации/добавления файла, из которых извлечены данные (рис. 8).

Название файлов может быть последовательностью букв и/или цифр:

clause : file_name | pattern | modified ;
file_name : 'file' '=' IDENT ;
IDENT : LETTER(LETTER | DIGIT) ;*

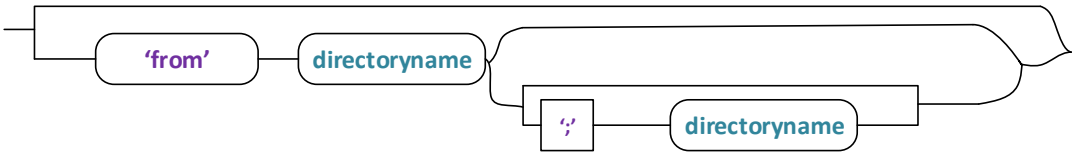

Рис. 6. Схема инструкции SELECT с оператором from_stmt
Fig. 6. A schema of the SELECT statement with from_stmt

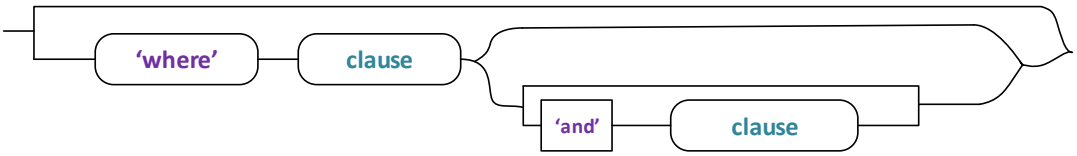

Рис. 7. Схема оператора where_stmt
Fig. 7. The where_stmt operator schema

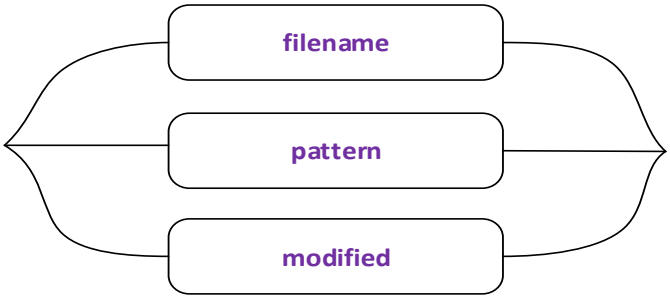

Рис. 8. Схема выражения CLAUSE
Fig. 8. The CLAUSE expression schema

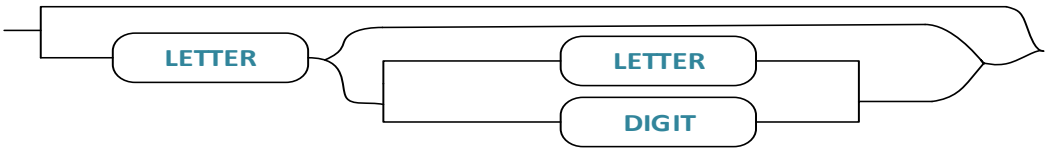

Рис. 9. Схема выражения IDENT
Fig. 9. The IDENT expression schema

Выражение *IDENT* иллюстрируется схемой, отображенной на рисунке 9.

Выражение *LETTER* содержит последовательность букв:

LETTER : 'a'..'z'|'A'..'Z' ;

Выражение *DIGIT* содержит последовательность цифр:

DIGIT : '0'..'9' ;

В представленной грамматике пропускаются символы пробелов, переноса строки и табуляции:

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

Символ * означает, что выражения условий результатов могут быть последовательностью условий с логической связью *AND*.

Выражение *DIRECTORYNAME* содержит наименования источников данных, из которых извлекаются данные. Это выражение содержит последовательность только букв и цифр. Выражение *CLAUSE* содержит условие фильтра результатов.

Для реализации предложенной грамматики была разработана программа на языке Java с использованием фреймворка ANTLR 3.0. Все переменные, которые используются в грамматике, объявлены в теге @members:

```
@members{
  public List directories = new ArrayList();
  public List clauseses = new ArrayList();
  public HashMap<String, String> hmapclause = new HashMap<String, String>();
  public Hashtable table = new Hashtable();
}
```

Рассмотрим последовательность операций генерации механизмов разбора запросов, составленных в соответствии с грамматикой. Все Java-операции описаны в тегах {.}.

Шаг 1. Генерация Java-кода (преобразование грамматики в виде *.g4 в парсер). На данном этапе происходит генерация парсера с помощью фреймворка ANTLR 3.0. Для ручной генерации парсера

можно использовать следующую команду:

java -jar antlr-3.2.jar -o VFSQL.g4

В результате будут получены классы для дальнейшего использования в модуле разбора запросов:

VFSQL.tokens – служебный файл со списком токенов грамматики;

VFSQLBaseListener.java – абстрактный класс для каждого правила парсера;

VFSQLLexer.java – класс лексера;

VFSQLLexer.tokens – служебный файл со списком токенов лексера;

VFSQLListener.java – абстрактный класс;

VFSQLParser.java – класс парсера.

Шаг 2. Реализация модуля для разбора реальных запросов. На данном этапе будет описан метод для разбора SQL-запроса, заданного пользователем с использованием методов класса парсера *VFSQLParser.*

Пример использования

Для испытания были использованы синтетические данные, генерируемые авторской системой EVGEN [12]. Она разработана на основе распределенной системы обмена сообщениями-подписками Kafka и состоит из двух подсистем:

– подсистемы EventDataGenerators, генерирующей события, структура которых задается шаблонами JSON о транспортных средствах и/или транспортных перекрестках;

– подсистемы EventDataGenerationConsumers, состоящей из потребителей (consumers), которые получают сгенерированные события из топиков (topics) и обрабатывают их.

Представим шаблоны, на основе которых генерируются данные в подсистеме EventDataGenerators:

– JSON-шаблон данных о транспортных средствах:

```
{
data :
[
"uid" : {Integer},              // Object ID
"eventStart" : {Long int},  // Starting time of the event
"eventEnd" :  {Long int},  // End time of the event
"long" : {Double},              // Longitude of the object space
"lat": {Double},                // Latitude of the object space
"velocity": {Double},   // The current velocity of the object
"status": {String},         //  Status of the object
"country" :{String}       // Country
]
}
```

– JSON-шаблон данных о транспортных перекрестках:

```
{
data :
[
   uid : {Integer} // ID transportation intersections
   Long: {Double}  // Longitude of the object (Node)
   Lat : {Double}  // Latitude of the object (Node)
   Count: {Integer} // The number of vehicles in the transportation intersections
   Status : {String} // Status of the transportation intersections
   Image: {Frame} // Frame captured from camera
]
}
```

Обработанные подсистемой EventDataGenerationConsumers данные сохраняются в озере данных для дальнейшей аналитики.

Для тестирования предложенной грамматики и метода обработки разнородных данных использованы синтетические данные из EVGEN. В качестве запроса (запрос 1) использовалось выражение «Вычислить количество транспортных объектов, которые отправляют данные на сервер или появляются на участке дороги, охватываемом камерой camera1 во временном интервале [$t1$, $t2$]. Запрос с использованием предложенной грамматики представлен в следующем виде: *select count from Datalog, camera1 where modified > 1516051435073 and modified < < 1516661699999;* где *Datalog* – путь к JSON-файлу, содержащему данные о транспортных объектах; *camera1* – путь к файлу, содержащему изображения с камеры № 1.

Все эксперименты проводились на ноутбуке под управлением операционной системы Ubuntu x86_64 с процессором Intel(R) Core(TM) i5-2430M CPU с частотой 2.40GHz.

Результат тестирования предложенного подхода для различных источников данных представлен в таблице 2. Каждый эксперимент повторялся 10 раз. В таблице 2 показано, что время парсинга экспериментов немного отличается. Обработка лог-данных происходит значительно быстрее по сравнению с обработкой видеоданных, потому что структура хранения лог-данных проста для эффективной обработки. К тому же при обработке видеоданных требуется реализовать алгоритмы для обработки изображений.

Таблица 2

Результаты тестирования выполнения запроса 1 в соответствии с разработанной грамматикой для различных источников данных

Table 2

The results of testing the execution of the query 1 according to the developed grammar for various data sources

Объем лог-данных (Мб)	Объем видеоданных (Мб)	Время разбора запросов (сек.)	Время выполнения запросов (сек.)
2	288.1	0.0135 ± 0.0045	148.3 ± 36.7
2	230.4	0.0132 ± 0.0052	101.7 ± 10.7
1	172.8	0.0139 ± 0.0041	87.1 ± 10.3
1	115.2	0.0134 ± 0.0056	64.6 ± 7.4
1	57.6	0.012 ± 0.005	36.2 ± 6.2

В таблице 3 представлены результаты выполнения запроса (запрос 2), аналогичного предыду-

щему, за исключением того, что в качестве источников данных использовались лог-файлы: *select count from Datalog1, Datalog3 where modified > 1520540895935 and modified < 1546661699999;*

Таблица 3

Результаты тестирования выполнения запроса 2 в соответствии с разработанной грамматикой для различных источников данных

Table 3

The results of testing the execution of the query 2 according to the developed grammar for various data sources

Объем лог-данных (Мб)	Время разбора запросов (сек.)	Время выполнения запросов (сек.)
22.5	0.0119	0.9241 ± 0.303
45.1	0.0124	1.2617 ± 0.639
67.6	0.0137	1.6622 ± 0.651
90	0.0133	2.3284 ± 0.4398
108.8	0.0162	2.7487 ± 0.425

Третий эксперимент включал запрос (запрос 3) только к видеоданным: *select count from camera1, camera2 where modified > 1516051435073 and modified < 1546661699999;*

Результаты выполнения представлены в таблице 4.

Таблица 4

Результаты тестирования выполнения запроса 3 в соответствии с разработанной грамматикой для различных источников данных

Table 4

The results of testing the execution of the query 3 according to the developed grammar for various data sources

Объем видео-данных (Мб)	Время разбора запросов (сек.)	Время выполнения запросов (сек.)
68.8	0.0138	31.529 ± 3,565
137.6	0.009	81.0256 ± 4.0076
184.6	0.013	109.1793 ± 10.9377
285.8	0.0129	133.0363 ± 12.2547
442.7	0.0108	213.6505 ± 20.2645

Эксперименты проводились для различных объемов разнородных данных (лог-файлов и видеофайлов). Среднее время парсинга SQL-запроса и реализации одного запроса для разнородных данных (2 Мб лог-файлов и 288.1 Мб видеофайлов в виде изображений) составило соответственно 0.0135 и 148.3 сек., то есть в сумме 148.3135 сек. Также менялась конфигурация экспериментов:

Объем лог-файлов, Мб	Объем видеофайлов в виде изображений, Мб
2	230.4
1	172.8
1	115.2
1	57.6

Заключение

В статье представлена грамматика SQL-подобного языка запросов к хранилищу, построенному по принципу озера данных. Грамматика предназначена для решения задачи унификации запросов к хранилищам, содержащим разнородные типы данных. В результате испытания ПО, реализующего предложенную грамматику в системе проактивной поддержки принятия решений, проанализировано время выполнения унифицированных запросов с различными объемами разнородных данных.

Работа выполнена при поддержке РФФИ, проект № 16-37-60066_мол_дк.

Литература

1. Golubev A., Chechetkin I., Solnushkin K.S., Sadovnikova N., Parygin D., Shcherbakov M.V. Strategway: web solutions for building public transportation routes using big geodata analysis. Proc. 17th Intern. Conf. on Information Integration and Web-Based Applications and Services, 2015, article 91, pp. 1–4. DOI: 10.1145/2837185.2843851.
2. Golubev A., Chechetkin I., Parygin D., Sokolov A., Shcherbakov M. Geospatial data generation and preprocessing tools for urban computing system development. Procedia Comput. Sci. 2016, vol. 101, pp. 217–226. DOI: 10.1016/j.procs.2016.11.026.
3. Ван Фу Чан, Щербаков М.В., Туан Ань Нгуен, Скоробогатченко Д.А. Метод сбора и слияния разнотипных данных в проактивных системах интеллектуальной поддержки принятия решений // Нейрокомпьютеры: разработка, применение. 2016. № 11. С. 40–44.
4. Ван Фу Чан, Щербаков М.В., Туан Ань Нгуен Yet. Another method for heterogeneous data fusion and preprocessing in proactive decision support systems: distributed architecture approach. Proc. 20th Intern. Conf. DCCN 2017, 2017, pp. 319–330. URL: http://www.springer.com/gp/book/9783319668352?wt_mc=ThirdParty.SpringerLink.3.EPR653.About_eBook (дата обращения: 03.06.2018). DOI: 10.1007/978-3-319-66836-9_27.
5. Ван Фу Чан, Щербаков М.В., Туан Ань Нгуен. Обзор архитектур систем поддержки принятия решений, использующих аналитику данных в режиме реального времени // Изв. ВолгГТУ. Сер. Актуальные проблемы управления, вычислительной техники и информатики в технических системах. 2016. № 3. С. 95–100.
6. Benchmarking Apache Kafka: 2 Million Writes Per Second (On three cheap machines). URL: https://getpocket.com/%40Gawth/share/900049 (дата обращения: 05.05.2017).
7. Платформа и службы облачных вычислений Microsoft Azure. URL: https://azure.microsoft.com (дата обращения: 03.06.2018).
8. Helge Rege Gardsvoll, Diving into your Azure Data Lake with U-SQL. URL: http://www.sqlnexus.com/wp-content/uploads/2017/05/SQL-Nexus-2017-Diving-into-Azure-Data-Lake-with-U-SQL_Helge.pdf (дата обращения: 03.06.2018).
9. The Five Most Common Big Data Integration Mistakes to Avoid. URL: http://www.oracle.com/us/products/middleware/data-integration/big-data-integration-mistakes-wp-2492054.pdf (дата обращения: 03.06.2018).
10. Bhardwaj A., Kumar V.A., Narayan Y., and Kumar P. Big data emerging technologies. A Case Study with analyzing twitter data using Apache Hive. Proc. 2nd Intern. Conf. RAECS, 2015. DOI: 10.1109/RAECS.2015.7453400.
11. Extract, Transform, and Load Big Data with Apache Hadoop. URL: https://pdfs.semanticscholar.org/dcd9/ce3591738b98e2cc9da63ee1fe9932c24500.pdf (дата обращения: 12.01.2018).
12. Ван Фу Чан, Щербаков М.В., Туан Ань Нгуен. EVGEN: A framework for event generator in proactive system design. Proc. 7th Intern. Conf. IISA IEEE, 2016. URL: http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=7774711 (дата обращения: 05.05.2017). DOI: 10.1109/IISA.2016.7785402.

Software & Systems
DOI: 10.15827/0236-235X.124.659-666

Received 04.06.18
2018, vol. 31, no. 4, pp. 659–666

Grammar for queries for heterogeneous data storage in proactive systems

Tran Van Phu [1], *Postgraduate Student, vanphu.vstu.russia@gmail.com*
M.V. Shcherbakov [1], *Dr.Sc. (Engineering), Senior Researcher, maxim.shcherbakov@gmail.com*
Sai Van Cuong [1], *Postgraduate Student, svcuonghvktqs@gmail.com*

[1] *Volgograd State Technical University, Volgograd, 400005, Russian Federation*

Abstract. The problem of storage and processing of heterogeneous data (both structured and non-structured) from various data sources is an important issue when implementing proactive support systems for a life cycle operation stage of complex multi-object distributed systems. The data are heterogenous, so it is required to store data descriptions (metadata) for subsequent automatic processing. In order to solve the problem of heterogeneous data efficient storage, an architecture called *data lake* is used. It implements mechanisms for data batch processing and real-time data processing.

The task of improving methods for effective access to heterogeneous data includes the following subtasks: development of SQL query grammar for heterogeneous data; building a parser for recognizing queries according to new grammar; development of modules for processing heterogeneous data according to a query; development of recommendations (methods) for applying the developed modules in proactive decision support systems.

The proposed grammar is based on the DML extension of the SQL language, in particular the SELECT statement extension. There is the generated parser using the ANTLR 3.0 library for processing the generated queries according to new grammar. Due to generation, there are some created classes in JAVA with their objects used for parsing queries. The generated parser and processing modules for heterogeneous data have become a basis for the new software. After testing the software that implements the proposed grammar in the proactive decision support system, the authors have analyzed the time of execution of unified queries with different volumes of heterogeneous data.

The main result of the grammar application is the reduction in the heterogeneous data processing time within a single query.

Keywords proactive computing, decision support systems, data acquisition, distributed data processing, data lake, heterogeneous data.

Acknowledgements. *The work has been supported by RFBR, project no. 16-37- 60066_мол_дк.*

References

1. Golubev A., Chechetkin I., Solnushkin K.S., Sadovnikova N., Parygin D., Shcherbakov M.V. Strategway: web solutions for building public transportation routes using big geodata analysis. *Proc. 17th Intern. Conf. on Information Integration and Web-Based Applications and Services*. 2015, article 91, pp. 1–4. DOI: 10.1145/2837185.2843851.
2. Golubev A., Chechetkin I., Parygin D., Sokolov A., Shcherbakov M. Geospatial data generation and preprocessing tools for urban computing system development. *Procedia Comput. Sci*. 2016, vol. 101, pp. 217–226. DOI: 10.1016/j.procs.2016.11.026.
3. Van Phu Tran, Shcherbakov M.V., Nguyen Tuan Anh, Skorobogatchenko D.A. A method for data acquisition and data fusion in intelligent proactive decision support systems. *J. Neurocomputers*. 2016, no. 11, pp. 40–44 (in Russ.).
4. Van Phu Tran, Shcherbakov M.V., Nguyen Tuan Anh. Yet. Another method for heterogeneous data fusion and preprocessing in proactive decision support systems: distributed architecture approach. *Proc. 20th Intern. Conf. DCCN 2017*. 2017, pp. 319–330. Available at: http://www.springer.com/gp/book/9783319668352?wt_mc=ThirdParty.SpringerLink.3.EPR653.About_eBook (accessed June 3, 2018). DOI: 10.1007/978-3-319-66836-9_27.
5. Van Phu Tran, Shcherbakov M.V., Nguyen Tuan Anh. Overview of decision support system architectures using real-time data analytics. *Izvestia VSTU. Engineering Sciences*. 2016, no. 3, pp. 95–100 (in Russ.).
6. *Benchmarking Apache Kafka: 2 Million Writes Per Second (On three cheap machines)*. Available at: https://getpocket.com/%40Gawth/share/900049 (accessed May 5, 2017).
7. *Microsoft Azure*. Available at: https://azure.microsoft.com (accessed June 3, 2018).
8. *Helge Rege Gardsvoll, Diving into your Azure Data Lake with U-SQL*. Available at: http://www.sqlnexus.com/wp-content/uploads/2017/05/SQL-Nexus-2017-Diving-into-Azure-Data-Lake-with-U-SQL_Helge.pdf (accessed June 3, 2018).
9. *The Five Most Common Big Data Integration Mistakes to Avoid*. Available at: http://www.oracle.com/us/products/middleware/data-integration/big-data-integration-mistakes-wp-2492054.pdf (accessed June 3, 2018).
10. Bhardwaj A., Kumar V.A., Narayan Y., Kumar P. Big data emerging technologies. A Case Study with analyzing twitter data using Apache Hive. *Proc. 2nd Intern. Conf. RAECS*. 2015. DOI: 10.1109/RAECS.2015.7453400.
11. *Extract, Transform, and Load Big Data with Apache Hadoop*. Available at: https://pdfs.semanticscholar.org/dcd9/ce3591738b98e2cc9da63ee1fe9932c24500.pdf (accessed January 12, 2018).
12. Van Phu Tran, Shcherbakov M.V., Nguyen Tuan Anh. EVGEN: A framework for event generator in proactive system design. *Proc. 7th Intern. Conf. IISA IEEE*. 2016. Available at: http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=7774711 (accessed May 5, 2017). DOI: 10.1109/IISA.2016.7785402.