УДК 004.434

Разработка языка запросов к графовому хранилищу биллинговой информации

© М.В. Бартенев, И.Э. Вишняков

МГТУ им. Н.Э. Баумана, Москва, 105005, Россия

Выполнен краткий обзор языков запросов к графовым базам данных Cypher и Gremlin. Сформулированы требования к языку запросов к графовому хранилищу биллинговой информации с учетом специфики задач обработки биллинговых данных. Сделан вывод о необходимости создания специализированного языка запросов. Спроектирован и реализован язык запросов, сочетающий удобство и наглядность декларативного подхода с простотой расширения его функциональности. Приведены синтаксис и семантика основных конструкций языка. Представлены результаты тестирования времени выполнения отдельных запросов, отражающие также производительность используемого графового хранилища.

Ключевые слова: *NoSQL, графовые базы данных, анализ биллинговой информации, язык запросов.*

Введение. В настоящее время очень остро стоит проблема анализа больших объемов биллинговой информации [1]. Реляционные базы данных могут справиться с ее хранением, но извлечение и любая обработка таких данных превращаются в очень ресурсоемкий и медленный процесс, зачастую вообще невыполнимый в классических табличных системах управления базами данных (СУБД) [2, 3].

Для решения подобных проблем используются нереляционные хранилища данных, которые пока находятся на раннем этапе развития и потому не имеют хорошо развитой системы взаимодействия с конечным пользователем. Кроме того, большинство таких хранилищ данных создается для внутрикорпоративного использования с целью решения узкоспециализированных задач, стандартизированные интерфейсы доступа и языки запросов встречаются очень редко. Одним из немногих примеров стандартизации может служить язык запросов SPARQL, применяемый для работы с данными спецификации RDF [4, 5].

Языки запросов к нереляционным хранилищам данных должны обеспечивать возможность выполнения операций с данными и хранилищем в целом не только разработчику, но и любому пользователю, не знакомому с особенностями внутренней организации системы. Поскольку при обработке биллинговой информации мы имеем дело с графовым хранилищем данных [1] и система имеет сложную внутреннюю структуру, то возникает необходимость создания такого языка запросов к этому хранилищу, который упростит взаимодействие с ним.

Задачи языка запросов. Рассмотрим основные функции, которые должен предоставлять язык запросов к графовому хранилищу биллинговой информации:

- создание экземпляра графовой базы данных с определенным идентификатором;
- удаление экземпляра базы данных по идентификатору;
- очистка базы данных от всех хранящихся в ней объектов;
- добавление вершины со строковым идентификатором в граф;
- добавление ребра, соединяющего две вершины, в граф (возможно указание даты добавления ребра);
- удаление вершины из графа по идентификатору;
- удаление ребра из графа по идентификаторам вершин, которые оно соединяет;
- импорт данных в хранилище из файла, содержащего список ребер;
- выборка данных из хранилища с помощью последовательно примененных аналитических операций.

При импорте данных каждая строка файла имеет следующий вид:

$$\{source\_node;\ destination\_node;\ [time]\}\ ,$$

где *source_node* — строковое значение, идентифицирующее начальную вершину ребра; *destination_node* — строковое значение, идентифицирующее конечную вершину ребра; *time* — необязательная временна́я метка ребра.

Аналитические операции должны быть представлены в виде одной из функций [1]:

- поиск соседних вершин для заданной вершины;
- пересечение найденных соседей двух вершин.

На вход любой функции подаются список вершин и глубина поиска. Список вершин может быть задан изначально или представлять собой результат работы другой функции. Аналогично к подграфу, являющемуся результатом работы некоторой функции, может быть применена другая аналитическая функция.

Обзор существующих языков запросов. На текущий момент имеется два языка запросов к графовым базам данных, которые используются в тех или иных системах и развитие которых активно продолжается: Cypher [6] и Gremlin [7].

Язык запросов Cypher — самый распространенный язык запросов к графовым базам данных, что обусловлено его использованием в СУБД Neo4J [2]. Cypher является декларативным языком и позволяет создавать, обновлять и удалять вершины, ребра, метки и свойства, а также управлять индексами и ограничениями. Для извлечения дан-

ных из хранилища используется запрос, содержащий шаблон фильтрации, позволяющий получать:

- *(n)-->(m)* — все направленные ребра из вершины *n* в вершину *m*;
- *(n:Person)* — все вершины с меткой *Person*;
- *(n:Person:Russian)* — все вершины, имеющие обе метки *Person* и *Russian*;
- *(n:Person {name:{value}})* — все вершины с меткой *Person* и отфильтрованные по дополнительному свойству;
- *(n:Person)-->(m)* — ребра между вершинами *n* с меткой *Person* и *m*;
- *(n)--(m)* — все ненаправленные ребра между вершинами *n* и *m*;

Одно из главных преимуществ языка Cypher — выразительный SQL-подобный синтаксис, привычный для большинства пользователей баз данных. Кроме того, независимо от потенциального расширения своих возможностей этот языка будет оставаться достаточно простым.

Язык запросов Gremlin поддерживается базой данных Titan [8]. Gremlin позволяет выполнять базовые операции с элементами графа (создание, обновление и удаление вершин, ребер, меток и свойств). Однако, в отличие от Cypher, данный язык является императивным и основан на идее последовательного применения функций к исходному графу и промежуточным результатам, благодаря чему можно выполнять обходы и проводить фильтрацию. Основные функции языка Gremlin позволяют получить:

- *V* — все вершины (с возможностью фильтрации по заданному ключу и значению);
- *E* — все ребра (с возможностью фильтрации по заданному ключу и значению);
- *id* — идентификатор элемента;
- *label* — метку элемента;
- *out(labels,…)* / *in(labels,…)* / *both(labels,…)* — смежные вершины (по исходящим / входящим / всем ребрам) для заданного набора вершин;
- *outE(labels,…)* / *inE(labels,…)* / *bothE(labels,…)* — инцидентные исходящие / входящие / все ребра по заданному набору вершин.

Благодаря императивному подходу язык Gremlin дает наиболее полное представление о выполняемых над графом операциях и их последовательности. Кроме того, поддержка интерфейса Blueprints [9] на основе реализации языка Gremlin обеспечивает возможность применения библиотек стандартных алгоритмов на графах к данным графового хранилища.

Разработка собственного языка запросов. Подводя итог рассмотрения существующих языков запросов, необходимо отметить,

что используемое нами графовое хранилище биллинговой информации берет на себя основную роль в обработке данных, в связи с чем на язык запросов должна ложиться минимальная ответственность за формирование логики анализа. Если какая-либо аналитическая операция реализована в хранилище, то ее выполнение должно быть наиболее простым с точки зрения пользователя. Применить рассмотренные языки в чистом виде невозможно в силу их универсальности: в языке Cypher потребуется большое количество шаблонов фильтрации, а в языке Gremlin для реализации необходимой операции придется выполнять длинную последовательность из существующих функций. Таким образом, основная идея разработки собственного языка заключается в объединении простоты и наглядности декларативного синтаксиса с простотой добавления новых методов анализа данных путем реализации аналитической функции в графовом хранилище с одновременным ее добавлением к языку запросов.

Рассмотрим лексические, синтаксические и грамматические особенности разрабатываемого языка. Будем различать пять типов лексем: идентификаторы, ключевые слова, константы, знаки операций и прочие разделители. Компилятор языка игнорирует пробелы, знаки табуляции и перевода строки, расположенные между лексемами. При этом требуется, чтобы хотя бы один из этих символов разделял смежные идентификаторы, ключевые слова и константы. При разбиении входного потока на лексемы должно учитываться правило са́мой длинной лексемы, а именно: если входной поток был разбит на лексемы вплоть до некоторой позиции, то, начиная с этой позиции, из входного потока выбирается самая длинная последовательность символов, которая может составлять лексему.

Идентификатор представляет собой последовательность букв и цифр, начинающуюся с буквы. Константы разделяются на целочисленные и строковые. *Целочисленные константы* в языке запросов представляют собой неотрицательные целые числа в диапазоне от 0 до $(2^{31} - 1)$. Они записываются в виде последовательности цифр, перед которыми в фигурных скобках может указываться основание системы счисления. Цифрами считаются арабские цифры от 0 до 9 и латинские буквы: A означает 10, B означает 11 и т. д. Основание системы счисления записывается в виде десятичного числа в диапазоне от 2 до 36. Если основание не указано, то константа считается десятичной. *Строковая константа* представляет собой последовательность символов, заключенных в кавычки. Комментарием является любой фрагмент языка запросов, который заключен между парами символов /* и */.

Следующие идентификаторы зарезервированы для использования в качестве *ключевых слов* и не зависят от регистра:

CREATE	DROP	DATABASE	IN	TRUNCATE
INSERT	NODE	INTO	VALUE	EDGE
TIME	DELETE	FROM	SELECT	WHERE
IMPORT	FILE	AND	OR	NEIGHBORS
INTERSECTION				

Множество знаков операций и прочих разделителей содержит следующие символы:

( ) + – * / % ^ <> ; , < > = <= >=

Описание операций, уровень приоритета и порядок выполнения приведены в таблице.

Описание операций

Уровень приоритета	Операция	Описание	Порядок выполнения
1	*x* ^ *y*	Возведение *x* в степень *y*	←
2	*x* * *y*	Умножение *x* на *y*	→
	x / *y*	Частное от деления *x* на *y*	
	x % *y*	Остаток от деления *x* на *y*	
3	*x* + *y*	Сумма *x* и *y*	→
	x – *y*	Разность *x* и *y*	
4	*x* = *y*	Сравнение *x* и *y*	→
	x < > y		
	x < *y*		
	x > *y*		
	x <= *y*		
	x >= *y*		
5	*x* and *y*	Логическое И	→
6	*x* or *y*	Логическое ИЛИ	→

Сценарий запросов к графовому хранилищу представляет собой последовательность запросов определенного вида, применяемых к графовой базе данных. Каждый запрос отвечает за одну из описанных функций. Запросы могут разделяться точкой с запятой. Рассмотрим синтаксис каждого запроса:

- *CREATE DATABASE database_id [IN directory_path]* – создание экземпляра графовой базы данных с идентификатором *database_id* в директории *directory_path*, где *directory_path* представляет собой

строковую константу (если путь в запросе не указан, директорией по умолчанию считается корневой каталог программы, выполняющей запрос);

- *DROP DATABASE database_id* — удаление экземпляра графовой базы данных с идентификатором *database_id*;
- *TRUNCATE database_id* — удаление всех объектов из экземпляра графовой базы данных с идентификатором *database_id*;
- *INSERT NODE INTO database_id VALUE node_value* — вставка вершины *node_value* в базу данных с идентификатором *database_id*, где *node_value* — строковая константа;
- *INSERT EDGE INTO database_id VALUE source_node, dest_node [TIME date_time]* — вставка ребра из вершины *source_node* в вершину *dest_node* в базу данных с идентификатором *database_id*, где *source_node* и *dest_node* — строковые константы (возможно указание дополнительного параметра *date_time* — временно́й метки ребра);
- *DELETE NODE FROM database_id VALUE node_value* — удаление вершины *node_value* из базы данных с идентификатором *database_id*;
- *DELETE EDGE FROM database_id VALUE source_node, dest_node* — удаление ребра из вершины *source_node* в вершину *dest_node* из базы данных с идентификатором *database_id*;
- *IMPORT FILE file_path INTO database_id* — импорт файла с данными *file_path* в базу данных с идентификатором *database_id*, где *file_path* представляет собой строковую константу.

Более подробно остановимся на операторе выборки данных из хранилища — SELECT. Этот оператор имеет следующую грамматику:

<select_operator> ::= FROM <database_id>

*SELECT { * | <selection_cond> } [WHERE <where_cond>]*

<selection_cond> ::= <nodes_list> . <functions>

<nodes_list> ::= (node_id [,...n])

<functions> = <analysis_function>[.<functions>]

<analysis_function> = { NEIGHBORS | INTERSECTION }

Таким образом, оператор SELECT позволяет выбрать все данные из хранилища или применить к определенному подграфу последовательность аналитических операций. Для вызова конкретных аналитических функций используются следующие ключевые слова:

- *NEIGHBORS* — поиск соседних вершин;
- *INTERSECTION* — пересечение найденных соседей двух вершин.

Тестирование языка запросов. Корректность и производительность выполнения запросов были протестированы путем запуска сценария, в который входили операции создания экземпляра графовой базы данных, добавления в нее графа и применения к этому графу различных последовательностей аналитических операций. Сценарий включал следующие запросы в приведенном порядке:

1. *CREATE DATABASE database1 IN "c:\database1\"*
2. *IMPORT FILE "c:\test.txt" INTO database1*
3. *FROM database1 SELECT **
4. *FROM database1 SELECT ("4", "5").intersection(1)*
5. *INSERT EDGE INTO database1 VALUE "4", "6"*
6. *INSERT EDGE INTO database1 VALUE "5", "7"*
7. *FROM database1 SELECT **
8. *FROM database1 SELECT ("4", "5").intersection(1).neighbors(2)*
9. *DELETE NODE FROM database1 VALUE "2"*
10. *DELETE EDGE FROM database1 VALUE "4", "6"*
11. *TRUNCATE database1*
12. *DROP DATABASE database1*

Время выполнения запросов было измерено для двух исходных графов — малого и большого. Первый (7 вершин, 10 ребер) использовался прежде всего для проверки корректного выполнения операций, а второй (10 млн вершин, 200 млн ребер) — для тестирования времени работы со значительными объемами данных.

Стенд, на котором проводилось тестирование, имеет следующую конфигурацию: Intel Core i5 3,1 ГГц, 8 Гбайт DDR3, жесткий диск 1 Тбайт, 7200 об/мин, Windows Server 2008 × 64. Результаты тестирования времени выполнения запросов представлены на рис. 1 и 2. Ранее было отмечено, что используемое графовое хранилище биллинговой информации (на основе B-дерева) берет на себя основную роль в обработке данных, поэтому результаты второго теста также отражают производительность самого хранилища при выполнении соответствующих операций. Создание, очистка и удаление хранилища данных являются крайне редкими операциями, поэтому время их выполнения не очень важно. Продолжительность операции импорта относительно велика, но время ее выполнения остается приемлемым, поскольку импорт — одноразовая операция, позволяющая загрузить в хранилище достаточно много данных. Остальные операции создания, удаления и анализа занимают приблизительно такое же время, кроме операции выборки всех данных, которая является крайне быстрой благодаря внутренней оптимизации хранилища.

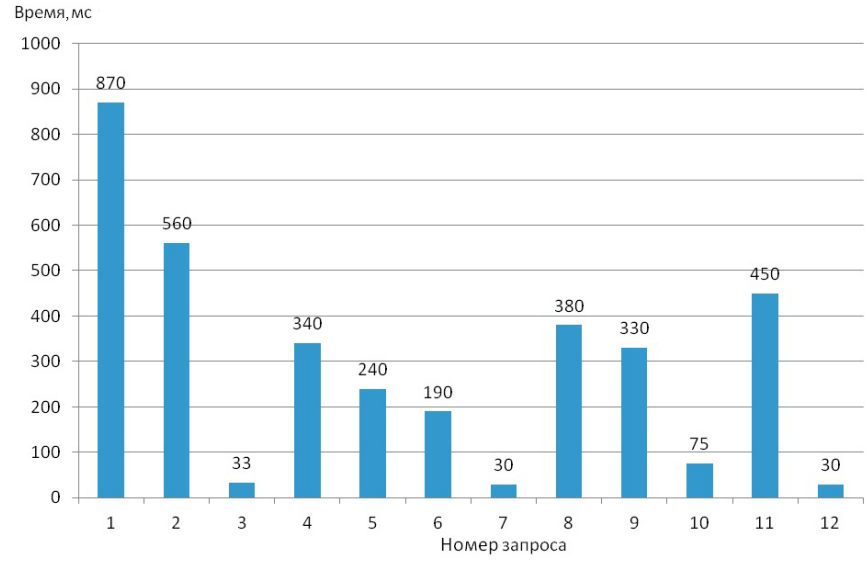

Рис. 1. Результаты тестирования времени работы на малом графе

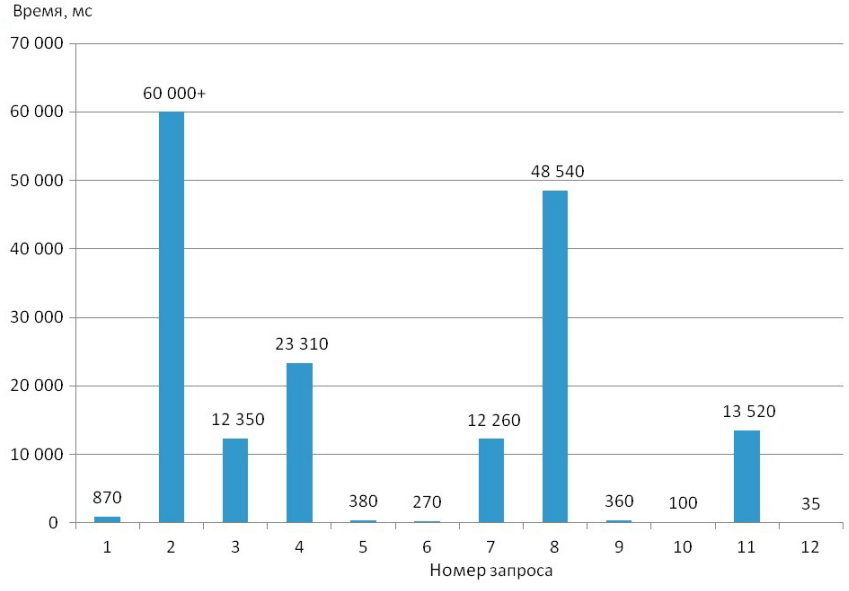

Рис. 2. Результаты тестирования времени работы на большом графе

Заключение. В работе рассмотрены языки запросов к графовым хранилищам — Cypher и Gremlin. На их основе был разработан собственный язык запросов к графовому хранилищу биллинговой информации. В языке реализованы все необходимые операции по работе с графовыми данными. Благодаря SQL-подобному синтаксису язык прост для понимания и обладает широкими возможностями для расширения.

Основное применение языка заключается в анализе биллинговых данных с помощью последовательного применения аналитических функций. В конечном итоге тестирование показало, что реализованный язык запросов выполняет все поставленные задачи корректно и за приемлемое время. Можно отметить также более высокую производительность используемого графового хранилища по сравнению с аналогичными разработками [1].

ЛИТЕРАТУРА

[1] Бартенев М.В., Вишняков И.Э. Использование графовых баз данных в целях оптимизации анализа биллинговой информации. *Инженерный журнал: наука и инновации*, 2013, вып. 11. URL: http://engjournal.ru/catalog/it/hidden/1058.html (дата обращения 23.05.2014).

[2] Robinson I., Webber J., Eifrem E. *Graph Databases*. O'Reilly Media, 2013, 224 p.

[3] Dominguez-Sal D., Urbon-Bayes P., Gimenez-Vano A., Gomez-Villamor S., Martinez-Bazan N., Larriba-Pey J.L. Survey of graph database performance on the HPC scalable graph analysis benchmark. *Proceedings of the 2010 int. conf. on Web-age information management* (WAIM'10). Berlin, Heidelberg, Springer-Verlag, 2010, pp. 37–48.

[4] Алексиянц А., Коршунов А., Кузнецов С. СУБД для социальных сетей. *Открытые системы: СУБД*, 2014, № 2. URL: http://www.osp.ru/os/2014/02/13040051 (дата обращения 23.05.2014).

[5] Головков В., Портнов В., Чернов В. RDF — инструмент для неструктурированных данных. *Открытые системы: СУБД*, 2012, № 9. URL: http://www.osp.ru/os/2012/09/13032513 (дата обращения 23.05.2014).

[6] *Cypher Query Language*. URL: http://docs.neo4j.org/chunked/stable/cypher-query-lang.html (дата обращения 23.05.2014).

[7] *Gremlin Query Language*. URL: https://github.com/tinkerpop/gremlin/wiki (дата обращения 23.05.2014).

[8] *Titan*. URL: https://github.com/thinkaurelius/titan/wiki (дата обращения 23.05.2014).

[9] *Blueprints interfaces*. URL: http://blueprints.tinkerpop.com (дата обращения 23.05.2014).

Статья поступила в редакцию 03.10.2014

Ссылку на эту статью просим оформлять следующим образом:

Бартенев М.В., Вишняков И.Э. Разработка языка запросов к графовому хранилищу биллинговой информации. *Инженерный журнал: наука и инновации*, 2014, вып. 11. URL: http://engjournal.ru/catalog/it/hidden/1319.html

Бартенев Максим Владимирович родился в 1993 г., студент 5-го курса кафедры «Теоретическая информатика и компьютерные технологии» МГТУ им. Н.Э. Баумана. Автор одной публикации. Специализируется в области графовых баз данных. e-mail: max.bartenev@yandex.ru

Вишняков Игорь Эдуардович родился в 1980 г., окончил МГТУ им. Н.Э. Баумана в 2003 г. Старший преподаватель кафедры «Теоретическая информатика и компьютерные технологии» МГТУ им. Н.Э. Баумана. Автор 8 публикаций. Специализируется в областях хранения, обработки и визуализации больших массивов данных e-mail: vishnyakov@bmstu.ru

The query language for graph database containing billing information

© M.V. Bartenev, I.E. Vishnyakov

Bauman Moscow State Technical University, Moscow, 105005, Russia

The article presents a brief overview of Cypher and Gremlin query languages for graph databases. The requirements to the query language for billing information storages take into account special features of the billing data processing tasks. The necessity of creating a specialized query language is justified. A query language, which combines both convenience and clarity of declarative paradigm with the simplicity of functionality extension, is designed and implemented. Basic syntax and semantics of the language constructions are provided. We show the results of the test execution time of individual queries, which also reflect the performance of graph storage used.

Keywords: *NoSQL, graph databases, billing data analysis, query language.*

REFERENCES

[1] Bartenev M.V., Vishnyakov I.E. *Inzhenernyi zhurnal: nauka i innovatsii — Engineering Journal: Science and Innovations*, 2013, iss. 11. Available at: http://engjournal.ru/catalog/it/hidden/1058.html

[2] Robinson I., Webber J., Eifrem E. *Graph Databases*. O'Reilly Media, 2013, 224 p.

[3] Dominguez-Sal D., Urbon-Bayes P., Gimenez-Vano A., Gomez-Villamor S., Martınez-Bazan N., Larriba-Pey J.L. Survey of graph database performance on the HPC scalable graph analysis benchmark. *Proceedings of the 2010 Int. Conf. on Web-age Information Management* (WAIM'10). Berlin, Heidelberg, Springer-Verlag, 2010, pp. 37–48.

[4] Alexiyants A., Korshunov A., Kuznetsov S. *Otkrytye sistemy: SUBD — Open Systems Journal: Databases,* 2014, no. 2. Available at: http://www.osp.ru/os/2014/02/13040051 (accessed on 23.05.2014).

[5] Golovkov V., Portnov V., Chernov V. *Otkrytye sistemy: SUBD — Open Systems Journal: Databases*, 2012, no. 9. Available at: http://www.osp.ru/os/2012/09/13032513 (accessed on 23.05.2014).

[6] *Cypher Query Language*. Available at: http://docs.neo4j.org/chunked/stable/cypher-query-lang.html (accessed on 23.05.2014).

[7] *Gremlin Query Language*. Available at: https://github.com/tinkerpop/gremlin/wiki (accessed on 23.05.2014).

[8] *Titan*. Available at: https://github.com/thinkaurelius/titan/wiki (accessed on 23.05.2014).

[9] *Blueprints interfaces*. Available at: http://blueprints.tinkerpop.com (accessed on 23.05.2014).

Bartenev M.V. (b. 1993), a 5th year student of the Computer Science and Technologies Department at the Bauman Moscow State Technical University. Author of one publication. Specializes in the field of graph databases. e-mail: max.bartenev@yandex.ru

Vishnyakov I.E. (b. 1980) Graduated from the Bauman Moscow State Technical University in 2003, assistant professor of the Computer Science and Technologies Department at the Bauman Moscow State Technical University. Specializes in the fields of storage, processing and visualization of large data sets, author of 8 publications.
e-mail: vishnyakov@bmstu.ru