FRI-ONLINE-1-CCT1-16

USING SEMANTIC NETWORK AS MEANS OF DATA MODELING IN THE NOSQL GRAPH STORES [16]

Assoc. Prof. Milko Marinov, PhD
Department of Computer Systems and Technologies,
"Angel Kanchev" Univesity of Ruse
Phone: 082-888 356
E-mail: milko@uni-ruse.bg

Abstract: *Graph stores are especially important for applications where there is a need to analyze the relationships between objects or to provide access to graph nodes in a certain sequence (graph traversal). Characteristics of this type of data warehouses are that they have a high degree of optimization in storing the nodes of the graph and the connections between them, thus ensuring efficient query management. Graph based repositories are useful for solving problems where there are complicated connections between objects, such as social networks, rule based data retrieval mechanisms, and systems that need to analyze complex network structures to find relevant patterns and templates. Graph stores are suitable when there are many elements that are connected to all the others in a complicated way and the relationships in turn have specific properties. With this type of storage, queries to neighboring nodes are simplified, and in-depth access is easily provided when matching templates are found. The graph data model can be formally represented as a directed graph. The semantic network is defined as a directed graph in which nodes and arcs (semantic links) are named. When converting it to a list of elements, it is necessary to decide when a node will be included as an element in the list and when a semantic link between nodes. In the proposed algorithm, this issue is solved by introducing in the definition of the semantic network the function, reflecting the priority of the semantic links.*

Keywords: *Graph data stores, NoSQL data models, Semantic network, Traversing algorithm.*

ВЪВЕДЕНИЕ

Всички, които работят с релационни БД знаят колко сложни могат да бъдат те. В повечето случаи тяхното изграждане започва като прости системи, така че след изпълнение на съответната заявка се връща избраният кортеж от плоската файлова структура. С течение на времето изискванията към тях се увеличават и се разширяват в системи, които управляват голям брой таблици, изпълняват операции съединение, извършват оптимизация на заявките, управляват транзакциите и процедурите по съхраняване на данните, ускоряват търсенето чрез индексиране, осигуряват сигурност при достъпа до данните. Нерелационните (NoSQL) системите използват различен подход при решаване на тези сложни проблеми чрез създаване на прости приложения, които разпределят необходимите функционалности (характеристики) в компютърната мрежа (Akid, H., & Ayed. M., 2017; Marz, N., & Warren, J., 2015). Чрез поддържането на прости архитектурни компоненти се осигурява многократното им използване между приложенията, подпомагат разработчиците при тестването им и осигурява лесно вграждане на приложението в нови архитектурни решения (McCreary, D., & Kelly, A., 2014).

От гледна точка развитието на NoSQL системите поддържането на прости архитектурни компоненти е добро решение. Когато се създава едно приложение, не е необходимо всички негови функционалности да бъдат вградени в един програмен модул. Функциите на приложението могат да бъдат разпределени към много NoSQL бази от данни, които се състоят от инструментални средства, които имат прости интерфейси и добре дефинирани роли.

Едно от предизвикателствата за потребителите на NoSQL системите е, че съществуват голям брой различни архитектурни модели на данните, от които трябва да се направи съответен избор. В таблица 1 е представена връзката между нерелационните модели на

[16] Докладът е представен на заседание на секция 3.2 на 29 октомври 2021 с оригинално заглавие на български език: ИЗПОЛЗВАНЕ НА СЕМАНТИЧНА МРЕЖА КАТО СРЕДСТВО ЗА МОДЕЛИРАНЕ НА ДАННИТЕ В ГРАФОВИ ХРАНИЛИЩА

данните и техните основни приложения (McCreary, D., & Kelly, A., 2014; Robinson, I., Webber, J., & Eifrem, E., 2015; Khan, W., Ahmed, E., & Shahzad, W., 2017).

Таблица 1. Нерелационни модели на данните

Модел на данните	***Описание***	***Типични приложения***
Хранилища от типа ключ-стойност (key-value store)	Това е опростен начин за свързване на голям обем от данни с един обикновен символен низ	Речници на данни, хранилища на изображения, хранилища на документи, таблици за търсене, кеш за съхраняване на резултатите от заявките
Хранилища с графова (мрежова) структура (Graph store)	Начин да се съхраняват възлите и дъгите на един граф	Заявки в социалните мрежи, Запитвания от типа приятел-към-приятел, извличане на информация от графи, системи базирани на правила, модели за съвпадение
Хранилища от типа фамилия от колони (Column family, BigTable)	Начин разпръснато да се съхраняват матрични данни, използвайки ключа на реда и името на колоната	Обхождане на web страници, съхраняване на големи разпръснато запълвани страници, системи с висока степен на адаптивност, системи с висока степен на променливост на данните
Хранилища на документи	Начин да се съхранява дървовидно структурирана йерархична информация в рамките на един елемент	Всякакви данни, които имат контейнерна структура, включително офис-документи, поръчки за продажби, фактури, описание на продукти, форми, обмяна на документи, търсене на съдържание в документи.

ИЗЛОЖЕНИЕ

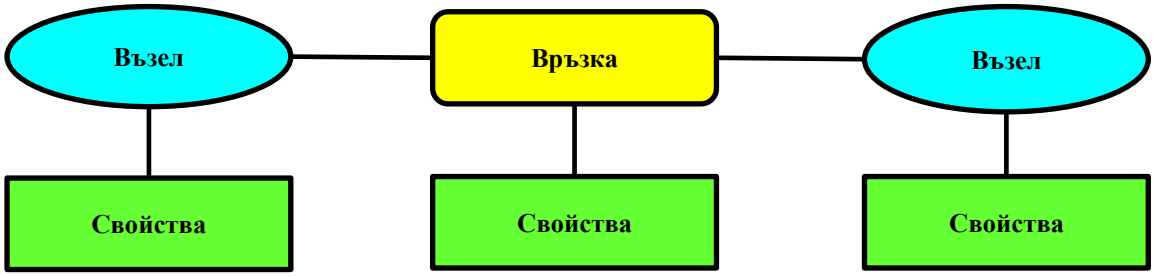
Основни характеристики на нерелационни хранилища, базирани на графов модел на данните

Хранилищата с графова структура са особено важни за приложения, при които има нужда да се анализират връзките между обектите или да се осигури достъп до възлите на графа в определена последователност (graph traversal). Характерното за този тип хранилища на данни е, че имат висока степен на оптимизация при съхраняване възлите на графа и връзките между тях, като по този начин се осигурява ефективно управление на заявките. Хранилищата с графова структура са полезни при решаването на проблеми, при които има сложни връзки между обектите, като социалните мрежи, механизмите за извличане на данни, базирани на правила, и системи, които трябва бързо да анализират сложни мрежови структури, за да открият в тях съответни закономерности и шаблони (Vela, B., Cavero, J., Cáceres, P., Sierra, A., & Cuesta, C.,2018).

Хранилището на данни с графова структура представлява система, която съдържа съвкупност от възли и връзки между тях. При хранилищата от типа ключ/стойност има два типа полета – ключ и стойност. Хранилищата с графова структура имат три типа полета – възли, връзки и свойства (фиг. 1). Хранилището с графова структура се състои от множество от структури от типа възел-връзка-възел. Свойствата се използват за описание на възлите и връзките

Хранилищата с графова структура са подходящи, когато има много елементи, които са свързани с всички останали по сложен начин и връзките от своя страна имат особености

(свойства). При този тип хранилища заявките към съседните възли се опростяват и лесно се осигурява достъп в дълбочина при откриване на съответни шаблони.



Фиг. 1. Структура на графовите хранилища

Графовите модели на данните преставят обектите от реалния свят като съществителни. Възлите могат да бъдат хора, организации, телефонни номера, web страници или дори биологични клетки в един жив организъм. Отношенията могат да бъдат представени като връзки между тези обекти и основно се представят чрез дъги между възлите. Заявките към този тип хранилища представляват алгоритми за обхождане възлите в граф. Например (Akid, H., & Ayed. M., 2017; Agrawal, S., & Patel, A., 2016):

- Кой е най-краткият път между два възела в графа?

- Кои възли имат съседи със специфични свойства?

- За всеки два възела в графа, каква е степента на подобие между техните съседи?

- Каква е средната свързаност на различни точки на графа с всяка останала?

Хранилищата с графова структура могат да бъдат използвани за ефективно решаване на конкретни бизнес проблеми (Parmar, R., & Roy, S., 2018; Khan, W., Ahmed, E., & Shahzad, W., 2017):

- Анализ на връзките (Link Analysis) – се използва когато потребителят иска да осъществи търсене за шаблони и връзки в социалните мрежи, в записите от телефонни разговори или в съобщенията, предавани по електронната поща;

- Правила и механизми на извод (Rules and inference) – се използват когато потребителят трябва да изпълни заявки към сложни структури – библиотеки от класове, класификации и системи, базирани на правила;

- Интегриране на свързани данни (Integrating linked data) – се използва при големи обеми от свързани данни, събирани в реално време, в резултат на което се изграждат компилации без съхраняване на данните.

Представяне графовия модел на данните чрез семантична мрежа

Графовият модел на данните формално може да бъде представен като ориентиран граф от вида:

$$SN = (C, R, Q, s, F, P),$$

където

- $C=\{c_1,c_2,\ldots c_n\}$ е множеството от възли;
- $R=\{r_1,r_2,\ldots r_m\}$ - множеството от типове семантични връзки, определящи отношенията между възлите;
- $Q : C \times C \rightarrow R$ - функция на съответствие между свързана двойка възли и семантична връзка;
- $s \in C$ – начален възел на мрежата;

- $F = \{f_1, f_2, \ldots f_y\}$ – е множеството от терминални възли, нямащи изходни връзки с останалите възли, т.е. $\boldsymbol{F \subseteq C}$;
- $P : R \rightarrow U$ - функция, отразяваща приоритета на семантичните връзки в зависимост от типа на семантичната връзка и начина на обхождане на семантичната мрежа U;

$$\mathbf{U} = \begin{cases} 1, \text{ ако обхождането на семантичната мрежа трябва да продължи с обработването на следващата, изходяща от възела семантична връзка;} \\ 2, \text{ ако обхождането на семантичната мрежа трябва да продължи с обработването на следващия възел, принадлежащ на семантичната мрежа.} \end{cases}$$

В таблица 2 е представен пример, отразяващ съответствието между типовете семантични връзки, изходящи от възела c_i, начина на обхождане на семантичната мрежа U и стойностите на функцията P, отразяваща приоритета им. Чрез израза $P_{max}(c_i)$ ще се определя връзката с най-голям приоритет, изходяща от възела c_i.

Таблица 2. Речник на семантичните връзки

Тип на семантичната връзка	***Начин на обхождане на семантичната мрежа, U***	**Приоритет на семантичните връзки, $P(c_i)$**
Is-a	1	***1 → най-висок приоритет***
A-Kind–Of	1	***2***
Type-Kind	1	***3***
Whole-Part	1	***4***
Part–Of	2	***3***
Instance–Of	2	***4***
Attribute–Of	2	***5***
Has–Parts	2	***6***
Connected–To	2	***7***
Made–Of	2	***8***
Has–Attribute	2	***9***
Object-Property	2	***10***
Object-Action	2	***11***
Action-Result	2	***12***
Object-Example	2	***13 → най-нисък приоритет***

За по-прецизна дефиниция на s и F ще бъдат дефинирани следните допълнителни функции:

<u>*Изходна функция на възела c_i*</u>

$$OUT(c_i) = (C_i \times R_i),$$

където

- $R_i = \{r_{i1}, r_{i2}, \ldots, r_{im}\}$ е множеството от типове на семантични връзки, изходящи от възела c_i ;
- $C_i = \{c_{i1}, c_{i2}, \ldots, c_{im}\}$ е множеството от възли имащи входни връзки с възела c_i .

<u>*Входна функция на възела c_i*</u>

$$IN(c_i) = (C_i \times R_i),$$

тук

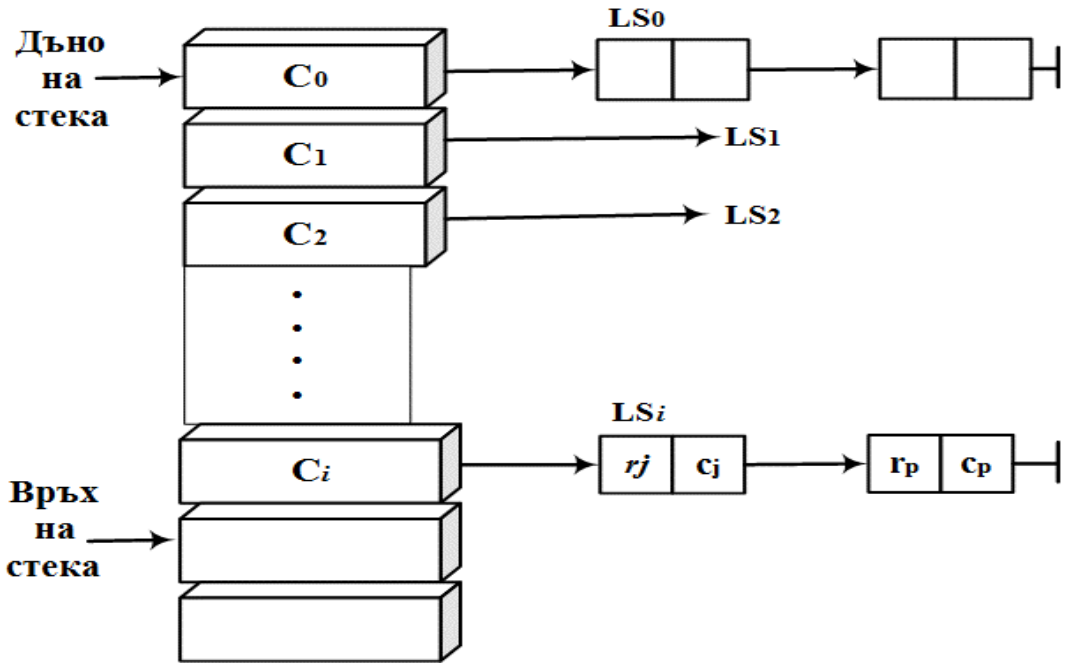
- $R_i=\{r_{i1},r_{i2},\ldots,r_{im}\}$ е множеството от типове семантични връзки, входящи във възела c_i ;
- $Ci=\{ci1,ci2,\ldots,cim\}$ е множеството от възли, имащи изходни връзки с възела c_i

Началният възел $s \in C$ няма входни връзки с останалите възли, т.е. $IN(c_i) = \phi$. Всеки терминален възел $f_i \in F \subseteq C$ няма изходни връзки с останалите понятия, т.е. $OUT(c_i) = \phi$.

Алгоритъм за обхождане на семантичната мрежа

Връзките в семантичната мрежа идентифицират типа отношения, които съществуват между свързаните възли. Тук възниква въпросът как може да се извлече информация, съхранявана в семантична мрежа? Едно добро решение би било да се разработи алгоритъм, чрез който да се обходи семантичната мрежа.

Предлаганият алгоритъм преобразува семантичната мрежа в списък от елементи (възли и семантични връзки) като обхожда мрежата в зависимост от приоритета на изходящите от възлите семантични връзки. За целта се използва стек, на върха на който се намира текущият възел, подлежащ на обработка и указател към списък LS_i от необработените семантични връзки, изходящи от възела c_i (фиг.2).



Фиг. 2. Структура на стека при обхождане на семантичната мрежа

Преди описанието на алгоритъма ще бъдат дефинирани следните функции:

$PUSH(c_i)$ – добавя възела c_i на върха на стека, създава списъка LS_i и включва в него изходящите от възела c_i връзки, определени чрез функцията $OUT(c_i)$, премества указателя на стека към следващия елемент;

$POP(c_i, r_j, c_j)$ – изключва връзката r_j, между понятията c_i и c_j, от списъка LS_i на необработените семантични връзки, изходящи от възела c_i.

Ако списъкът LS_i е празен, то функцията

$POP(c_i)$ – премества указателя на стека към възела c_i и изключва възела c_i от върха на стека.

Входни данни: Семантичната мрежа SN, речник на семантичните връзки, речник на възлите.

Изходни данни: L – списък от елементи (възли и семантични връзки)

Предполага се, че в началния момент списъкът L е празен.

(1) Определя се началният възел *s* на семантичната мрежа *SN*, за който $IN(c_i) = \phi$, включва се *s* в списъка *L* и се изпълнява функцията *PUSH(s)*.
(2) Ако съществуват семантични връзки, изходящи от текущия възел c_i, за които **U=1**, то те се добавят в списъка *L*, като се започва от тези, за които $P(c_i)$ е с най-малка стойност;
(3) Ако съществуват семантични връзки, изходящи от текущия възел c_i, за които **U=2**, определя се $P(c_i)$ е с най-малка стойност и се добавя съответната семантична връзка r_j в списъка *L*. Нека r_j свързва c_i с c_j. Изпълнява се $POP(c_i, r_j, c_j)$, след което възелът, подлежащ на обработка, става c_j. Добавя се c_j в списъка L и се изпълнява функцията $PUSH(c_j)$. Преход към (2).
(4) Ако списъкът LS_i не е празен и съществуват семантични връзки, за които **U=2**, преход към (3).
(5) Ако списъкът LS_i не е празен и не съществуват семантични връзки, за които **U=2,** определя се за **U=1** $P(c_i)$ с най-малка стойност. Нека това е връзката r_p, свързваща възела c_i с c_p. Изпълнява се $POP(c_i, r_p, c_p)$, след което възелът, подлежащ на обработка, става c_p. Добавя се c_p в списъка *L*, изпълнява се $PUSH(c_i)$. Преход към (2).
(6) Ако списъкът LS_i е празен и стекът не е празен, изпълнява се $POP(c_i)$, извлича се от стека следващият възел c_{i-1} за обработка. Преход към (4).
(7) Край.

Ако съществуват повече от един възела, които са свързани с текущия възел c_i с изходящи семантични връзки, за които $P(c_i)$ е с една и съща стойност, избира се възелът, който има най-малък пореден номер в речника на възлите.

Обхождането на семантичната мрежа завършва, когато стекът се изпразни (указателят на стека съвпадне с неговото дъно), т.е. когато всички възли и семантични връзки са обработени.

ЗАКЛЮЧЕНИЕ

Тъй като семантичната мрежа е дефинирана като ориентиран граф, в който възлите и дъгите (семантичните връзки) са именовани, затова при нейното преобразуване в списък от елементи е необходимо да се реши кога като елемент на списъка ще се включи възел и кога семантична връзка между възлите. В предложения алгоритъм този въпрос е решен чрез въвеждане в дефиницията на семантичната мрежа функцията P, отразяваща приоритета на семантичните връзки.

Предложеният алгоритъм генерира решение за семантична мрежа, имаща само един начален възел и поне един терминален. Това изискване е включено в дефиницията на SN. То може да се реализира чрез избор на подходящ тип семантични връзки.

БЛАГОДАРНОСТ

Този доклад се публикува с подкрепата на проект 21-ФЕЕА-01 „Интелигентни компютърни системи: изследване на тяхното развитие, приложение и управление“, финансиран от фонд „Научни изследвания“ на Русенски университет „Ангел Кънчев“.

REFERENCES

Vela, B., Cavero, J., Cáceres, P., Sierra, A., & Cuesta, C. (2018). Using a NoSQL graph oriented database to store accessible transport routes. *In Proc. of the EDBT/ICDT*, 62-66.

McCreary, D., & Kelly, A. (2014). *Making sense of NoSQL*, Manning Publications.

Akid, H., & Ayed. M. (2017). Towards NoSQL graph data warehouse for big social data analysis. *in Proc. of the International Conference on Intelligent Systems Design and Applications*, vol. 557, Springer, Cham.

Robinson, I., Webber, J., & Eifrem, E. (2015). *Graph databases: new opportunities for connected data*. Neo Technology, Inc.

Marz, N., & Warren, J. (2015). *Big Data: principles and best practices of scalable real-time systems*. Manning Publications.

Parmar, R., & Roy, S. (2018). MongoDB as an efficient graph database: an application of document oriented NoSQL database. *Data Intensive Computing Applications for Big Data, M. Mittal et al. (Eds.), IOS Press*, 331-361.

Agrawal, S., & Patel, A. (2016). A study on graph storage database of NoSQL. *International Journal on Soft Computing, Artificial Intelligence and Applications*, 5(1), 33-39.

Khan, W., Ahmed, E., & Shahzad, W. (2017). Predictive performance comparison analysis of relational & NoSQL graph databases. *International Journal of Advanced Computer Science and Applications*, 8(5), 523-530.