Научна конференция „Иновационни ИКТ в бизнеса и обучението: тенденции, приложения и разработване“, Пампорово, 24-25 Ноември 2016

Scientific Conference “Innovative ICT in Business and Education: Future Trends, Applications and Implementation”, Pamporovo, 24-25 November 2016

NOSQL ПОДХОДИ В SQL БАЗИ ОТ ДАННИ

Стоян Черешаров, Христо Крушков

Резюме. С развитието на облачните технологии ролята на така наречените NoSQL бази от данни нараства все повече. Това се дължи на удобствата, които предлагат при работа в реално време в уеб среда с големи обеми от данни (big data). В статията се описват подходи, чрез които част от техните предимства могат да се симулират в релационна SQL база от данни, което позволява оптимизирането на приложения, които са изградени над релационни бази от данни.

Ключови думи: SQL, NoSQL, бази от данни

1. Въведение

В последните години разбиранията ни за архитектурата на едно приложение претърпяха огромно развитие. Все по-популярна е архитектурата базирана на микро услуги, които си комуникират посредством шина за събития (event driven micro services with event bus). Тази шина запазва събитията и е възможно всичко случващо се в приложението да бъде пресъздадено на по-късен етап. Приложенията работят с обекти, които не сменят състоянието си (immutable objects). Микро-услугите представляват мини приложения със собствени понякога силно денормализирани бази от данни. Тези услуги могат да са написани на различни програмни езици и да използват различни сървърни технологии. Но независимо от технологиите и архитектурните решения в повечето случаи се стига до запазване на данните в слой за запазване. Обикновено това са бази от данни. Поради сложността и противоречивите изисквания към приложенията, възниква необходимост от

използването на много бази от данни (New Data Base или Polyglot Persistence) в едно приложение. Експериментира се с използването на различни по модел и вид бази данни. Всяка от базите данни има своите предимства и недостатъци. Комбинирането им спомага да се използват предимствата на всяка една от тях и да се избегнат недостатъците.

Потребителите очакват да могат да достъпят желаното от тях приложение навсякъде и по всяко време. В същото време, за да запазят или повишат своята конкурентоспособност, фирмите се борят да предоставят неповторимо потребителско преживяване на своите клиенти във всеки един момент. Всичко това предполага създаването на потребителски интерфейси, които могат лесно да се адаптират към индивидуалните нужди на потребителите, промените в околната среда и платформата [1]. Бизнес изискванията стават все по-високи, силно непредсказуеми и променливи. Като пример можем да посочим ситуацията в която на потребителя се дава възможност сам да определя типа и броя на полетата в една форма. Трудно може да се отговори на тези изисквания с бази данни в които схемата е предварително заложена.

Разработчиците от своя страна се опитват да изградят приложенията така, че промяната в един елемент да не предизвиква верижна реакция от промени в други части на софтуера (strong cohesion, loos coupling, non rigid, non fragile). Използването на ORM (Object Relational Mapping) техниките способства също вълните на промените да бъдат ограничени до определени слоеве. Промяната в схемата на една релационна база от данни води до промяна на всички слоеве над нея. А обикновено базата с данни е в ядрото на приложението, което значи, че то цялото ще претърпи промени при промяна в схемата на базата данни. Започвайки отдолу нагоре вълната от промени преминава през целия софтуер. Липсата на схема в една NoSQL (точка 3) база от данни позволява промяната да се ограничи [7, 8]. Промяната може да се направи отгоре надолу, като не предизвиква вълна от промени. Но NoSQL базите от данни не поддържат транзакции и не следват ACID принципите (Atomicity, Consistency, Isolation, Durability), което ограничава техните възможности [2]. От друга страна една релационна база от данни поддържа транзакции и следва ACID принципите, но се нуждае от ясни изисквания и предварително заложена схема, което не позволява гъвкавост при промени [5, 6]. В тази публикация предлагаме някои подходи, които съчетават предимствата на двата вида бази данни в една обединена SQL релационна база.

В изложението релационните или SQL бази от данни ще бъдат наричани просто **SQL бази от данни**. Това са бази от данни, които използват

релационния модел, в тях може да се използва стандартен SQL и притежават ACID свойствата.

За нерелационните или NoSQL бази от данни ще използваме просто термина **NoSQL бази от данни**. Това са бази от данни, които не използват релационен модел, не използват стандартен SQL и не притежават всички ACID свойства.

1.1. Цел и задачи

Целта на това изследване е да се намерят начини за използване, някои от предимствата на NoSQL в SQL базите от данни.

От тази цел произтичат и някои конкретни **задачи**:

- Да се отговори на въпроса, кои са предимствата на подход с една SQL база данни и кое налага симулация на някои от предимствата на NoSQL базите от данни в SQL база данни. (точка 1.2.)
- Да се определи кои предимства на NoSQL базите от данни са от значение за изследването и желаем да симулираме. (точка 1.2.)
- Да се намерят начини за симулиране на важните за нас предимства в SQL база данни. (точка 4.)

1.2. Предимства на подхода

- Необходимост от само един сървър за бази данни, което намалява цената на проекта и неговата поддръжка.
- Нужда от познаване само на една база от данни.
- Използване на добре познати стандартни софтуерни инструменти.
- Използване на стандартен SQL.
- Разполагаме с транзакции, които гарантират правилното запазване на чувствителни данни
- Притежаваме гъвкавост, да запазваме разнообразни по тип данни без да се налага промяна в схемата на базата данни и промени във всички слоеве на приложението.
- Можем да отговорим на променящи се динамично изисквания, като взимането на крайните решения може да се отложи, колкото се може по-далеч във времето.
- Описаните подходи дават възможност в приложения вече използващи SQL бази от данни да се внесе гъвкавост.

За това изследване е важно да отговорим и на въпроса кои от предимствата на NoSQL базите данни желаем да симулираме. Основно предимство на една нерелационна NoSQL база от данни, от гледна точка на разработчиците е липсата на схема. Промените в бизнес логиката не предизвикват вълни от промени, които преминават през цялото приложение.

Затова и в това изследване ще се концентрираме главно върху намиране начини за симулиране това поведение.

2. SQL бази от данни (ACID properties)

Това което отличава SQL базите данни от останалите е фактът, че те напълно поддържат ACID свойствата. Често се налага извършването на голям брой действия с базата данни, които възприемаме като транзакция. Именно способността на SQL базите данни да поддържат тези транзакции ги прави уникални в днешно време и важни за това изследване.

ACID свойствата са:

- A – Atomicity – Атомарност. Това е способността на СУБД да реализира множество команди като една. Тази група команди се изпълняват ако всяка една от тях е успешна. Всички команди пропадат ако дори само една от тях пропадне. Принципът „Всичко или нищо“.
- C – Consistency – Последователност. Преди и след изпълнение на транзакцията базата данни трябва да запазва своя интегритет.
- I – Isolation – Изолация. Ако транзакциите се извършват по едно и също време, никоя от тях не трябва да засяга другата. Транзакциите трябва да се изпълняват в пълна изолация, сякаш никаква друга транзакция не се изпълнява.
- D – Durability – Здравина. Веднъж потвърдена, като успешно приключила транзакцията трябва да е запазена в базата с данни дори при отпадане на захранването.

Основното предимство на SQL базите данни е поддръжката на всички ACID свойства, което прави тези СУБД незаменими в ситуациите в които това е задължително условие.

3. NoSQL бази от данни (Polyglot Persistence)

В последните години, особено след 2009 година се заговори за нерелационните NoSQL бази данни [2, 4, 7 – 10]. В литературата няма точно определение на NoSQL базите данни, затова трябва да използваме характеристиките им:

- NoSQL базата от данни не следва релационния модел от което следва, че не може да се използва стандартен SQL.
- Обикновено тези СУБД са с отворен код.
- Създадени са за работа в облака.
- Нямат схема.

В практиката има ситуации в които пълното спазване на ACID свойствата не е задължително условие. Точно в такива ситуации нерелационните NoSQL

бази от данни могат да заменят релационните или SQL базите от данни. Примери за използване на NoSQL и SQL бази от данни могат да се намерят в различни ситуации.

При необходимост от бърз достъп за четене и писане при запазване на потребителски сесии може да се използва Redis. Компромис се прави със свойството здравина. Тоест възможно е някои от потребителските сесии да се загубят.

При необходимост от работа с финансови данни са ни нужни всички свойства на релационна база данни. Компромиси не могат да се правя в този случай използването на релационна СУБД е задължително. Същото се отнася за ситуациите при които се нуждаем от справки от подобни финансови операции.

В случай че има нужда от много четене от базата данни с относително малко писане може да се използва друга NoSQL база от данни MongoDB. За големи обеми от данни, които се налага да се записват при аналитични бази от данни от рода на потребителски логове или аналитични данни може да се използва Cassandra.

Посочените по-горе примери са с цел да се демонстрират възможни приложения на SQL и NoSQL бази от данни и нямат задължителен характер.

При използването на множество бази данни в едно приложение говорим за многоезично запазване на данните (Polyglot Persistence).

4. NoSQL подходи в SQL бази данни

Често при изграждането на система не знаем предварително всички изисквания. Изправени сме пред проблема да изберем типа на базата или базите данни без да имаме достатъчно информация. Релационните бази са доказали качествата си. Затова и те са естествен избор в подобни непредвидими ситуации. Ако в последствие възникне нужда от използване на данни без предварително известна схема, предлагаме няколко подхода, които биха могли да решат проблема.

4.1. Таблица само с полета ключ – стойност

Създаването на таблица само с полета ключ и стойност (meta_key, meta_value) дава възможност под формата на стрингове в тях да бъдат записвани огромни обеми разнородна информация. Полето ключ не е задължително да е от текстов тип. То просто трябва да съдържа уникален идентификатор, който ни позволява да различим един запис от друг. Обикновено все пак е добре то да е от текстов тип, за да може в него да се

запише UUID – Universally Unique ID. Това би позволило идентификаторите да се генерират не само и единствено от конкретната СУБД, но и от други системи. Така не се налага обръщение към СУБД при необходимост от предварително подготвяне на множество взаимосвързани записи. Например при поставяне на външен ключ за осъществяване на връзка от тип едно към много, външният ключ, може да се генерира без обръщение към СУБД. Това придобива все по-голямо значение в съвременните front-end системи, в които се стремим да ограничим обръщенията към сървъра.

В полето meta_value се записват както конкретните данни така и мета данните, служещи ни за правилната интерпретация на данните. Името на полето при този подход не може да служи, като мета данни за правилната интерпретация.

Като недостатък на всички варианти които описваме по-долу и използват подхода ключ – стойност можем да посочим трудностите при търсене, индексиране, филтриране и сортиране на данните [11, 12]. За всичко това трябва да се грижи приложението. Именно затова и по-долу ще разгледаме различни формати за записване на данните в полето meta_value.

4.2. Използване на XML за съхраняване на данните в полето за стойност

В полето за стойност при подхода с използване на ключ – стойност данните могат да са записани под формата на XML [3, 8, 13]. Това е стандартен формат който позволява преобразуване в DOM дървовидна структура. Всяка технология предлага библиотека за работа с подобни структури. Може да се осъществява търсене и извличане на данните със стандартни средства.

Друго голямо предимство на подобен подход е независимостта от технологията. XML може да се чете и преобразува в DOM по един и същи начин в различните технологии.

Недостатък на подобен подход е обемът на XML форматът. XML предлага големи възможности, но изисква доста символи за описание на мета данните.

4.3. Сериализирани обекти в полето за стойност

В полето за стойност могат да се записват сериализирани обекти. Тези символни последователности могат в отделните технологии да се преобразуват в обратно обекти. Като предимство може да се посочи фактът, че преобразуването от символна последователност в обект за съответната технология става по стандартен начин, бързо и лесно.

Недостатък е зависимостта на записаните данни от технологията. Обект сериализиран с една технология не може директно да се използва в друга. Друг недостатък е ограничението, че не всичко в един обект може да се сериализира.

4.4. JSON в полето за стойност

Частен случай на сериализирани обекти е използването на JSON – Java Script Object Notation. Това са сериализирани обекти на JavaScript. Предимство е простотата на този запис както и това, че лесно може да се чете и асимилира от хора. Сериализираните JavaScript обекти под формата на JSON са се превърнали в де-факто стандарт при обмена на информация. Това го прави изключително удобен начин за съхраняване на данни в ключ – стойност таблици. Подобни данни могат да бъдат четени в различни технологии.

4.5. Потребителски формат в полето за стойност

За съхраняване на данните в полето стойност можем да използваме и специален потребителски формат. Предимство е изключителната лаконичност на записите, която можем да постигнем. Недостатък е липсата на стандарт не само между отделните технологии, но и дори между отделни приложения. Данни записани в подобен формат няма как да бъдат прочетени със стандартни средства от друго приложение дори да е създадено със същата технология.

4.6. Добавяне на полета с общо предназначение

Добавянето на полета с общо и неясно предназначение, за да запазват непредвидени стойности крие доста опасности и неудобства. Избягва се проблемът с трудната филтрация и сортиране на данните, но изисква съпътстваща документация понеже имената на полетата не могат да служат като мета данни. По този начин схемата на базата данни не се променя при промяна на изискванията. Но работата с данните става много трудна поради огромния обем информация която разработчиците трябва да познават.

4.7. Добавяне на таблици подмножества

При този подход се използва базова таблица съдържаща полетата за които имаме яснота, включително уникалния идентификатор на всеки запис. При необходимост от допълнителни полета се добавя таблица подмножество с отношение едно към едно с базовата таблица. Недостатък е промяната на схемата. Предимството е в лесната филтрация и сортировка, които могат да се реализират на ниво SQL без необходимост от писане на код в приложението. Данните могат да се използват в сложни SQL заявки на ниво база данни.

4.8. Таблица с полета идентификатор, ключ и стойност

При този подход имаме отново полета за ключ и стойност, но освен тях се добавя допълнително поле за идентификатор. Това ни позволява да имаме повтарящи се ключове с еднакви стойности. Друго характерно е че в полето за стойност се записва единична стойност. Полето ключ играе ролята на мета данни, като определя смисъла на стойността записана в полето стойност. То може да се нарече още „име“ (name). Няма нужда в полето за стойност да се записват стойности и метаданни, записва се само стойност. Друг начин по който можем да гледаме на този подход е сякаш имаме таблица с променлив брой колони. Името на колоната се определя от стойността записана в полето ключ. Полетата за ключ и стойност отново трябва да са от символен тип което ни позволява при нужда да записваме сериализирани обекти, XML, JSON или потребителски низове. Но за предпочитане е запис на единични стойности. Основно предимство на този подход е възможността да се осъществяват търсене, филтрация и сортировка със стандартните средства на езика SQL. Не се налага предварително преобразуване на символните низове в обекти, за да бъдат извлечени стойностите от тях.

4.9. Таблица с полета идентификатор, външен ключ, ключ и стойност

Подобна таблица би могла да се използва, за да се симулира увеличаване до безкрайност на броя на колоните в съществуваща таблица. Полето външен ключ служи да се определи кой ред от родителската таблица се разширява. Полето за ключ играе ролята на име на колона, а в полето стойност се пази обикновено единична стойност. По този начин всеки ред от родителската таблица може да има различен по брой и вид допълнителни колони. Обикновено таблицата съдържаща ключ-стойност се нарича, като родителската таблица, която разширява плюс думата “meta” в името си. Като предимство може да се посочи възможност от използване на стандартни SQL заявки за манипулация на записаните данни.

5. Основни резултати

Два от описаните подходи бяха използвани при създаването на модул за управление на работни процеси. Беше използван подходът с XML в полето за стойност в таблица с полета ключ – стойност. Другият използван подход е с таблици подмножества.

Конкретната реализация на така описания модул е с помощта на програмния език PHP и база данни MySQL. Беше използвана интегрираната

среда за разработка (IDE) Zend Studio. Модулът е базиран на работната рамка Zend. Тази работна рамка предлага висококачествени компоненти за изграждане на уебприложения, включително MVC. Два от описаните подходи са използвани в приложение в областта на финансите. Модулът управлява процеси чиито параметри не са предварително известни и непрекъснато се променят.

Заключение

Описаните подходи за решаване проблема с неизвестността в софтуерните системи използващи SQL бази данни дават възможност да се изградят гъвкави и надеждни системи.

Това изследване би могло да намери приложение в ситуации в които има нужда да се използват съвременни NoSQL подходи при оптимизиране на съществуващи системи използващи SQL бази данни.

Друго възможно приложение може да се намери в системи в които работим в силно непредсказуема среда, но не желаем да увеличаваме сложността на системата чрез увеличаване броя на използваните бази данни.

Благодарности

Настоящият доклад е частично финансиран по проект ИТ15-ФМИИТ-004 към НПД на ПУ „Паисий Хилендарски“.

Литература

[1] Atanasova, M., A. Malinova, *Adaptive User Interfaces in Software Systems*, Doctoral Conference in Mathematics and Informatics MIDOC 2015 October 15-18, 2015 Sofia, Bulgaria, Proceedings, Pages 1-9.

[2] Ayman, A., A. Saleh, H. El-Ghareeb, H. Ali, *A middle layer solution to support ACID properties for NoSQL databases*, Original Research Article Journal of King Saud University - Computer and Information Sciences, Volume 28, Issue 1, January 2016, Pages 133-145.

[3] Böttcher, S., R. Hartel, D. Wolters, *S2CX: From relational data via SQL/XML to (Un-)Compressed XML*, Original Research Article Information Systems, Volume 56, March 2016, Pages 198-213.

[4] Corbellini, A., C. Mateos, A. Zunino, D. Godoy, S. Schiaffino *A Persisting big-data: The NoSQL landscape,* Original Research Article Information Systems, Volume 63, January 2017, Pages 1-23.

[5] Harrington, J., *Relational Database Design and Implementation (Fourth edition),* Morgan Kaufmann, Elsevier, 2016, ISBN: 9780128043998.

[6] Hernandez,, M., *Database design for mere mortals : a hands-on guide to relational database design*, Addison-Wesley, 2003, ISBN: 0201752840.

[7] http://martinfowler.com/articles/nosql-intro-original.pdf последно посетен 24.10.2016.
[8] Lee, K., W. Tang, K. Choi, *Alternatives to relational database: Comparison of NoSQL and XML approaches for clinical data storage*, Original Research Article Computer Methods and Programs in Biomedicine, Volume 110, Issue 1, April 2013, Pages 99-109.
[9] Liao, Y., J. Zhou, C.Lu, S. Chen, C. Hsu, W. Chen, M. Jiang, Y. Chung, *Data adapter for querying and transformation between SQL and NoSQL database*, Original Research Article Future Generation Computer Systems, Volume 65, December 2016, Pages 111-121.
[10] Makris, A., K. Tserpes, V. Andronikou, D. Anagnostopoulos, *A Classification of NoSQL Data Stores Based on Key Design Characteristics*, Original Research Article Procedia Computer Science, Volume 97, 2016, Pages 94-103.
[11] Peng, Z., *A Study on Database Fuzzy Query Method in SQL*, Original Research Article Procedia Engineering, Volume 24, 2011, Pages 340-344.
[12] Rocha, L., F. Vale, E. Cirilo, D. Barbosa, F. Mourão *A Framework for Migrating Relational Datasets to NoSQL1*, Original Research Article Procedia Computer Science, Volume 51, 2015, Pages 2593-2602.
[13] Schweinsberg, K., L. Wegner, *Advantages of complex SQL types in storing XML documents*, Original Research Article Future Generation Computer Systems, In Press, Corrected Proof, Available online 3 March 2016.

Факултет по математика и информатика
Пловдивски университет „Паисий Хилендарски“
Бул. „България“ 236,
4003 Пловдив, България
e-mail: stoyancheresharov@gmail.com, hdk@uni-plovdiv.bg

NOSQL APPROACHES IN SQL DATABASES

Stoyan Cheresharov, Hristo Krushkov

Abstract. With the cloud computing development, the NoSQL databases become increasingly popular. This is due to the amenities offered in building real-time web applications and big data. The article describes the ways in which some of their advantages can be simulated in a relational SQL database, allowing optimization of applications that are built on relational databases.