AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE

WYDZIAŁ INFORMATYKI,
ELEKTRONIKI I TELEKOMUNIKACJI

KATEDRA INFORMATYKI

Praca dyplomowa magisterska

Wspieranie zespołowej analizy danych o charakterze grafowym
Supporting team analysis of graph data

Autor: *Marcin Krupa*
Kierunek studiów: *Informatyka*
Opiekun pracy: *dr inż. Jacek Dajda*

Kraków, 2013

Oświadczam, świadomy(-a) odpowiedzialności karnej za poświadczenie nieprawdy, że niniejszą pracę dyplomową wykonałem(-am) osobiście i samodzielnie i nie korzystałem(-am) ze źródeł innych niż wymienione w pracy.

Serdecznie dziękuję . . . tu ciąg dalszych podziękowań np. dla promotora, żony, sąsiada itp.

Spis treści

1. Wstęp

1.1. Cele pracy

Celem poniższej pracy jest zaprojektowanie i stworzenie systemu kontroli wersji grafowych baz danych, będącego narzędziem wspomagającym zespołową analizę danych o charakterze grafowym.

1.2. Zawartość pracy

W rodziale 2 przedstawiono podstawowe informacje dotyczące baz NoSQL, grafowego modelu danych i systemów kontroli wersji, a także porównano dostępne na rynku grafowe bazy danych.

2. Wprowadzenie

W rozdziale tym przedstawiono podstawowe informacje dotyczące baz *NoSQL* z naciskiem na bazy grafowe oraz wersjonowania danych. Porównano także dostępne na rynku rozwiązania.

2.1. Relacyjne bazy danych

W czerwcu 1970 Edgar Frank Codd, pracując dla IBM, opublikował „A Relational Model of Data for Large Shared Data Bank" – pracę na temat relacyjnego modelu organizacji danych. Model relacyjny i pierwowzór języka SQL po raz pierwszy zostały zaimplementowany przez IBM w projekcie System R (System Rational), będącym protoplastą DB2, prowadzonym w latach 1974–1978. Pierwsze komercyjne rozwiązania to Oracle (1979) i DB2 (1983). Relacyjne bazy danych wkrótce zawładnęły rynkiem i mimo upływu czasu, nadal są dominującą technologią, zwłaszcza wśród aplikacji biznesowych. W ciągu ponad 30 lat dominacji, co jest ewenementem w świecie informatyki, kilkukrotnie pojawiały się technologie bazodanowe z potencjałem do odebrania części rynku systemom relacyjnym, na przykład obiektowe bazy danych w latach 90'. Alternatywy te nigdy jednak nie stały sie popularne. Może więc dziwić panujące od niedawna poruszenie wokół technologii *NoSQL*.

2.1.1. Zalety relacyjnych baz danych

Najbardziej oczywistym zastosowaniem bazy danych jest przechowywanie dużych ilości danych. Pamięć współczesnych systemów komputerowych dzieli się na szybką, ulotną pamięć operacyjną oraz wolniejszą, lecz pojemniejszą pamięć dyskową. Rozmiar pamięci operacyjnej jest ograniczony (z powodu kosztów), a dane są tracone po odłączeniu zasilania – trwałe przechowywanie danych wymaga więc wykorzystania przestrzeni dyskowej. Dla wielu aplikacji wystarczające jest przechowywanie danych w plikach na dysku, jednak większość aplikacji biznesowych wykorzystuje bazy danych.

Model ten bazuje namatematycznej teorii zaproponował relacyjny model danych. Model ten bazuje na Pierwsze komercyjne rozwiązania, Oracle i DB2 nie były dostępne aż do roku około 1980.

Bazy relacyjne - tabele, impedance mismatch pomiedzy strukturami danych w pamieci a tym co jest w bazie

Bazy obiektowe - brak impedance mismatch ale niszowe, nie przyjely sie gdyz nie moga one pelnic

roli integracyjnej tak jak bazy relacyjne - popularne jest uzycie bazy relacyjnej jako elementu integrujacego wiele roznych aplikacji - wiele aplikacji korzysta z jednej bazy danych

Nowe wyzwania - duzy ruch online - na poczatku np. Google, eBay, Amazon Rozwiazanie - zbudowac wiekszy komputer - ale to kosztowne, istneje granica rozbudowy LUB rozproszenie na wiele komputerow ALE relacyjne bazy danych zaprojektowane dla pojedynczego komputera, nie dla rozproszonych baz. Oczywiscie rozproszenie jest stosowane (np. w ebay, betfare - ogromne farmy) ale wymaga to nienaturalnego podejscia i traci sie wiekszosc zalet relacyjnych baz.

Google - Bigtable Amazon - Dynamo

Co czyni NoSQL innym od relacyjnych baz danych -> duzy ruch na wielu wezlach (rozproszony)

Johan Oskarsson #nosql NOSQL meetup San Francisco, nazwa powstala jako twitter hashtag mongoDB, Project Voldemort, CouchDB, Hypertable, Dynomite, Cassandra, Apache Hbase

Proba definicji nosql - cechy wspolne: - non-relational - open-source (wiekszosc ale nie wszystkie) - cluster-friendly (oprocz grafowych baz) - 21st century web - schema-less

Modele danych

1. dokumentowy mongoDB, Raven DB, CouchDB

structured data but no fixed schema ale to ze nie ma schematu w bazie nie znaczy ze nie ma schematu w aplikacji - implicit schema (not schema-less but implicit schema) - w kodzie aplikacji odwolania do nazw pól w dokumencie - zeby sie dowiedziec jakie sa nazwy pol trzeba zajrzec do bazy lub znalezc w kodzie miejsce gdzie te dane sa tworzone problemy np. z migracja danych sa dalej podobne do tych z baz posiadajacych schemat

dokumentowy i klucz-wartosc - granica się rozmywa np. do baz klucz wartosc mozna dolaczac metadane i wtedy zaczynaja wygladac jak dokumenty w bazach dokumentowych mozemy miec pole na identyfikator - klucz i po nim wyciagac dokumenty wielu uzytkowniow traktuje bazy dokumentowe jak bazy KV, wyciagajac dokumenty po kluczu zamiast po zawartosci

M. Fowler uwaza ze rozroznianie pomiedzy document a KV nie jest wielce uzyteczne, moze byc najwyzej podpowiedzia czego sie mozna spodziewac interesujace jest to co te 2 modele maja wspolnego: Aggregate-oriented - operacje na poziomie agregatu a nie pojedynczego wiersza w tabeli value == aggregate | document == aggregate zamiast zapisywac dane do kilku tabel zapisujemy caly agregat - w SQL zeby pobrac jakies dane czesto potrzeba wielu zapytan, tutaj natomiast pobieramy jednym zapytaniem caly agregat, latwiej dystrybuowac agregaty w klastrach

ALE dostep do danych na innym poziomie trudniejszy - koniecznosc uzycia algorytmow Map/Reduce

2. column-family Cassandra, Apache HBase

poziom agregatow, wyciaganie danych po id wiersza i nazwie rodziny kolumn

3. klucz-wartosc Voldemort, riak, redis

lookup data by key

4. grafowe Neo4j

NIE SA AGGREGATE-ORIENTED sa schema-less struktura grafu skupienie na relacjach

RDBMS - take a logical lump of data - an aggregate and split it across lots of rows Aggregate-oriented - store the whole aggregate on its own RDBMS == ACID NoSQL == BASE BUT Graph == ACID - bo nie ma agregatow, dane sa rozdrobnione

Aggregate == Transaction boundary atomic updates only within single aggregate

Transakcje nie rozwiązują wszystkich problemow wspolbieznosci ACID tak naprawde niewiele nam nie daje - bo w srodowisku produkcyjnym nie uzywa sie dlugich transakcji, ale tworzy sie transakcje w momencie zapisue, wiec np. jesli dwoch uzytkownikow modyfikuje ten sam obiekt to dalej bedzie problem bo drugi nadpisze zmiany pierwszego. Aby to kontrolowac uzywany jest offline lock - oznaczanie obiektow numerem wersji, obsluga konfliktow przez aplikacje lub warstwe orm a nie przez baze danych - bo nie uzywamy dlugich transakcji

w NoSQL nie potrzebujemy transakcyjnosci na poziomie pojedynczego agregatu bo operacje na nim sa atomowe. W przypadku operacji na wielu agregatach naraz mozemy natomiast uzyc offline lockow takich jak w RDBMS

Consistency - Logical (on single node) | Replication (across nodes)

replication - disconnected from acid, it is a problem regardless of technology

consistency vs availability - you can't have both more important - it is not for you as a programmer to choose, it is a business decision

Eventual consistency Relaxing durability Quorums Read-your-writes consistency

NOSQL - WHEN AND WHY

- easier development

- large scale data

wzrost popularnosci web serwisow, REST - odejscie od podejscia integracji aplikacji przez RDBMS

nosql is not the future Future - polyglot persistence - uzycie wielu roznych baz w obrebie jednej aplikacji zgodnie z ich przydatnoscia w konretnych zastosowaniach np sesje - Redis, koszyk - Riak, katalog produktow - mongo, dane finansowe i reporting - RDBMS, analiza i logi aktywnosci - Cassandra, rekomendacje - Neo4j

polyglot persistence - problems / opportunities

- decyzje - nie mozemy juz powiedziec - uzyjmy Oracle bo jest standardem korporacyjnym; musimy odpowiedziec na ptanie - jaka jest odpowiednia baza dla tego problemu

- organizational change

- immaturity

- eventual consistency

What project would nosql be useful for? - easier development - rapid time to market - large scale data - data intensive - strategic projects - projekty strategiczne powinny dzialac lepiej i szybciej niz u konkurencji, warto zaryzykowac projekty do uzytku wewnetrznego - nieistotne czy beda szybciej dzialac, lepiej uzyc sprawdzonych technologii niz uczyc sie czegos nowego, niedojrzalego

2.2. Czym jest *NoSQL*

W ciągu ostatnich kliku lat coraz większą popularność zaczynają zdobywać nierelacyjne, skalowalne systemy bazodanowe wysokiej dostępności, określane mianem *NoSQL*. Skrót ten można rozwinąć jako *„Not Only SQL"* (nie tylko SQL) lub, bardziej antagonizująco, *„No to SQL"* (nie dla SQL), jednak nie istnieje jednoznaczna i powszechnie akceptowana definicja. W obecnym znaczeniu termin ten po raz pierwszy użyty został w roku 2009[1] w nazwie spotkania, zorganizowanego przez Johana a w San Francisco, poświęconego zdobywającym coraz większą popularność nierelacyjnym bazom danych. Oskarsson potrzebował dla swojego spotkania chwytliwej nazwy, której mógłby użyć jako *hashtag*[2] w serwisie społecznościowym *Twitter*. *NoSQL* niewątpliwie spełnia to kryterium, jednak nieprecyzyjnie opisuje nowe systemy bazodanowe. Organizator spotkania zamierzał użyć tego terminu jednorazowo i nie spodziewał się, że na stałe przyjmie się on jako określenie całego trendu technologicznego.

Założenia *NoSQL*

- Rezygnacja z wielu elementów baz relacyjnych. Zauważono, ze duża liczba złączen tabel powoduje zdecydowany spadek wydajności, a ścisły schemat bazy danych nie zawsze bywa zaletą, gdyż wiele danych nie ma określonej struktury. Wątpliwości dotyczyły również zbyt restrykcyjnych postulatów ACID.

- Zmniejszenie znaczenia schematów danych.

- Zmiana podejścia w kwestii awarii. Stwierdzono, że awarie to nie wyjątki, a normalna sytuacja i w przypadku, gdy jeden z elementów systemów zostanie uszkodzony – reszta musi działać.

- Łatwe do wprowadzenia i transparentne dla aplikacji skalowanie poziome.

Podział baz *NoSQL*

W tabeli 2.1 przedstawiono podział baz NoSQL ze względu na wykorzystywany model danych. Należy jednak podkreślić, że granice pomiędzy tymi kategoriami często się rozmywają i wielu baz *NoSQL* nie da się w prosty sposób skategoryzować (np. *OrientDB* jest jednocześnie bazą dokumentową i grafową).

2.3. Grafowe bazy danych

Grafowa baza danych używa do reprezentowania danych struktur grafu - wierzchołków i krawędzi. Charakteryzuje się tym, że dostęp do sąsiednich elementów w grafie nie jest realizowany przy pomocy

[1] w roku 1998 Carlo Strozzi użył terminu *NoSQL* do nazwania swojej relacyjnej bazy danych *Strozzi NoSQL*, by podkreślić, iż nie używa ona *SQL* jako języka zapytań. Nie ma ona jednak nic wspólnego z obecnym trendem technologicznym.

[2] oznaczenie wpisów w serwisie *Twitter* ułatwiające wyszukiwanie *tweetów* (wpisów). Jest to słowo poprzedzone znakiem # (ang. *hash*) umieszczone we wpisie

Tablica 2.1: Modele danych NoSQL

Model danych	Opis	Przykłady
Klucz-wartość	Dane przechowywane w postaci par klucz-wartość (słownik), mogą być sortowane po kluczach	Redis Riak Voldemort
Kolumnowy	Dane przechowywanie są w kolumnach zamiast w wierszach	Cassandra HBase Hypertable
Dokumentowy	Wykorzystuje pojęcie dokumentów o wspólnym formacie (np. XML, JSON), lecz bez narzuconego schematu. Umożliwia wyszukiwanie dokumentów nie tylko po kluczu, ale także po zawartości.	MongoDB CouchDB OrientDB
Grafowy	Dane reprezentowane są przez wierzchołki i krawędzie grafu wraz z ich właściwościami.	Neo4J InfiniteGraph Titan

indeksu, a dzięki bezpośrednim wskaźnikom. Wierzchołki grafu reprezentują obiekty biznesowe, natomiast krawędzie - relacje między nimi. Większość istotnych informacji przechowywana jest w krawędziach. Można powiedzieć, że grafowa baza danych jest „bardziej relacyjna" od bazy relacyjnej. Baza grafowa skupia się bowiem na powiązaniach pomiędzy danymi i jest o wiele bardziej skalowalna, gdyż nie wymaga użycia kosztownych złączeń (ang. *Join*) pomiędzy tabelami.

Model grafu z właściwościami

Graf z właściwościami to najbardziej popularny wariant modelu grafowego. Posiada on następujące cechy:

– składa się z węzłów (wierzchołKów) i relacji (krawędzi),

– węzły posiadają właściwości (pary klucz-wartość),

– relacje są nazwane, skierowane i zawsze posiadają węzeł początkowy i końcowy,

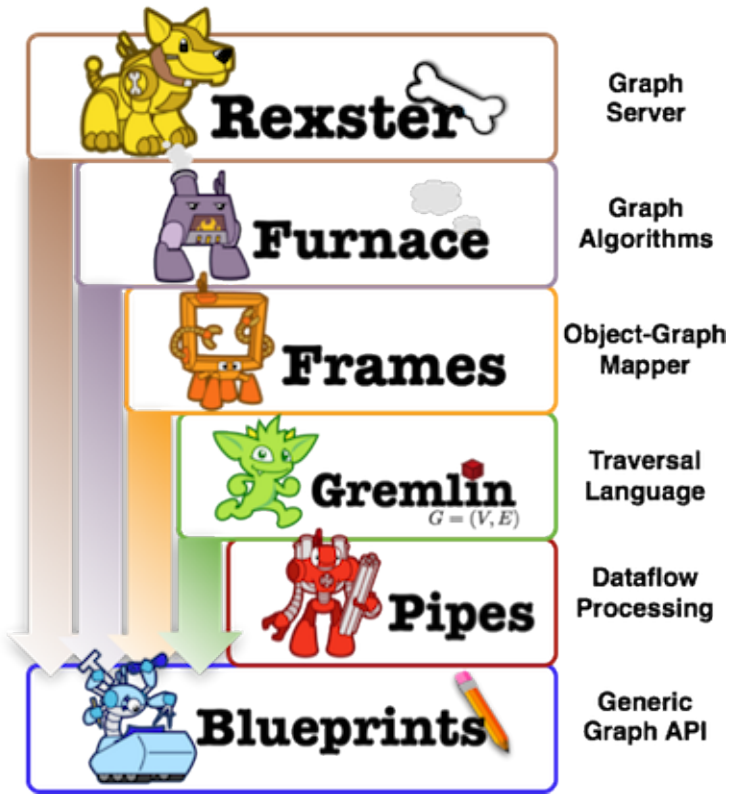
– relacje także mogą posiadać właściwości.

Model grafu z właściwościami, pomimo swojej prostoty, jest wystarczający do opisania znacznej większości przypadków użycia grafów.

2.4. TinkerPop Blueprints

TinkerPop jest grupą programistów tworzących stos technologiczny dla coraz bardziej popularnego grafowego modelu danych. Wszystkie projekty *TinkerPop* udostępnione są na licencji open-source (*BSD*).

TinkerPop Blueprints to zbiór inerfejsów, implementacji i testów dla modelu danych grafu z właściwościami. *Blueprints* jest dla grafowych baz danych tym, czym *JDBC*[3] dla baz relacyjnych. Aplikacje zbudowane z wykorzystaniem *Blueprints* mogą działać z dowolną bazą danych wspieraną przez *Blueprints*. W chwili obecnej są to: *TinkerGraph*, *Neo4J*, *Sparksee*, *Accumulo*, *ArangoDB*, *Bitsy*, *Bigdata*, *Datomic*, *FoundationDB*, *InfiniteGraph*, *MongoDB*, *Oracle NoSQL*, *OrientDB*, *Titan*, *Desired* oraz bazy korzystające z interfejsów *Sail*, *JUNG*, *JPA* i *JDBC*.



Rysunek 2.1: Stos technologiczny *Tinkerpop*

Biblioteka *Blueprints* jest podstawą dla innych projektów, wspólnie tworzących stos technologiczny *Tinkerpop* (rys. 2.1):

– *Pipes*: framework przepływu danych, umożliwiający dzielenie, łączenie, filtrowanie i transformację danych z wejścia do wyjścia,

[3] Java Database Connectivity - interfejs programowania opracowany w 1996 r. przez Sun Microsystems, umożliwiający niezależnym od platformy aplikacjom napisanym w języku Java porozumiewać się z bazami danych za pomocą języka SQL.

– *Gremlin*: język trawersowania grafów bazujący na języku Groovy, pozwalający na tworzenie zapytań grafowych, analizę i manipulację danymi,

– *Frames*: biblioteka realizująca mapowanie obiektowo-grafowe, umożliwiająca tworzenie aplikacji w kontekście obiektów domenowych wraz z ich powiązaniami, a nie krawędzi i wierzchołków,

– *Furnace*: zestaw implementacji standardowych algorytmów analizy grafów,

– *Rexster*: serwer grafowy, udpstępniający grafy *Blueprints* poprzez *REST* oraz protokół *RexPro*.

2.5. Wersjonowanie danych grafowych

Git

Kontrolę wersji danych grafowych można oprzeć o istniejący, sprawdzony system kontroli wersji - np. Git. Zapisane w postaci tekstowej (np. GraphML, GraphSON) snapshoty grafu można łatwo porównywać. Problem pojawia się natomiast, gdy chcemy przeszukać dane historyczne - konieczne jest wtedy odtworzenie grafu z pliku.

Antiquity

Antiquity - wersjonowany graf dla *Tinkerpop Blueprints*, umożliwia pobranie z bazy historycznych wersji wierzchołków oraz krawędzi. Przynależność obiektu do danej wersji jest określana przez właściwości - minimalny i minimalny numer wersji grafu dla której dany obiekt jest aktualny. Wersja grafu jest inkrementowana przy każdej zmianie grafu lub przy każdej transakcji.

FluxGraph

FluxGraph[4] to implementacja *Tinkerpop Blueprints* dla bazy danych *Datomic*. Baza ta przechowuje dane jako niezmienne (ang. *immutable*) obiekty, opisane datą. *FluxGraph* daje możliwość porównania grafu pomiędzy dwiema datami oraz porównania ze sobą dwóch obiektów. Dla każdego wierzchołka i krawędzi można także pobrać jego poprzednią i następną wersję. Datomic pod spodem może korzystać z baz relacyjnych, *DynamoDB*, *Riak*, *Cassandra*, *Couchbase*.

Zapisywanie różnic pomiędzy zmianami

Rozwiązanie zaproponowane w [KSP13]

Graf historyczny o odmiennej strukturze

W artykule [CL13] zaproponowano obiecującą metodę kontroli wersji danych grafowych, która zakłada nieinwazyjność - graf historyczny jest niezależny od danych źródłowych i może posiadać odmienną strukturę - np. wszystkie obiekty (także krawędzie) są zapisywane w grafie historycznym jako wierzchołki.

[4] `https://github.com/datablend/fluxgraph`

Bibliografia

[AG08] Renzo Angles and Claudio Gutiérrez. Survey of graph database models. *ACM Comput. Surv.*, 40(1), 2008.

[Cha09] Scott Chacon. *Pro Git: professional version control*. Apress, July 2009.

[CL13] Arnaud Castelltort and Anne Laurent. Representing history in graph-oriented nosql databases: A versioning system. In Imran Sarwar Bajwa, M. Asif Naeem, and Pit Pichappan, editors, *ICDIM*, pages 228–234. IEEE, 2013.

[KSP13] Georgia Koloniari, Dimitris Souravlias, and Evaggelia Pitoura. On graph deltas for historical queries. *CoRR*, abs/1302.5549, 2013.

[RWE13] Ian Robinson, Jim Webber, and Emil Eifrem. *Graph Databases*. O'Reilly, Beijing, 2013.

[SF13] Pramod J. Sadalage and Martin Fowler. *NoSQL distilled : a brief guide to the emerging world of polyglot persistence*. Addison-Wesley, Upper Saddle River, NJ, 2013.