МЕЃУНАРОДЕН СЛАВЈАНСКИ УНИВЕРЗИТЕТ
„ГАВРИЛО РОМАНОВИЧ ДЕРЖАВИН“
СВЕТИ НИКОЛЕ – БИТОЛА
РЕПУБЛИКА СЕВЕРНА МАКЕДОНИЈА



ФАКУЛТЕТ ЗА ИНФОРМАТИКА
СТУДИСКА ПРОГРАМА: СОФТВЕРСКИ ТЕХНОЛОГИИ

МАГИСТЕРСКА РАБОТА

NOSQL

СТУДЕНТ: (ХАРИ МИРЧЕСКИ)
БРОЈ НА ИНДЕКС: M199001
МЕНТОР: (ПРОФ.ДР.АЛЕКСАНДАР ПЕТРОВСКИ)

Битола, 2022

АПСТРАКТ

Во овој магистерски труд ќе извршиме преглед на NOSQL ,на неговите предности и недостатоци, основите, типовите на складирање на NOSQL ,како и видовите на складирањето на податоци кои се опфатени. NoSQL базите на податоци складираат податоци во документи наместо во релациони табели. Соодветно на тоа, класифицирани се како „ Not only SQL“ и се делат со различни флексибилни модели на податоци. Видовите на NoSQL бази на податоци вклучуваат бази на податоци за чисти документи, складишта за клучни вредности, бази на податоци со широки колони и бази на податоци со графикони.

Клучни зборови: NOSQL, Column oriented, Document store, Key value store, Graph store

СОДРЖИНА

АПСТРАКТ 2

Клучни зборови: NOSQL, Column oriented, Document store, Key value store, Graph store 2

ВОВЕД 6

I. nosql 10

1 преглед и дефинирање на nosql 10

1.1 Што претставува NoSQL, а што не 12

1.2 Предности и недостатоци на NoSQL базите на податоци 17

II. Зошто NOsql ? 20

2 Nosql и релациони бази на податоци 20

2.1 ACID properties 23

2.2 Приспособливост, Типови и Видови на складирање NoSQL 23

III. видови на складирање 26

3 column-oriented 26

3.1 Column-oriented (Wide column) store 33

4 Document store database 35

4.1 Document store 56

5 Key-value store 60

5.1 Key-Value 66

6 Graph store 69

6.1 Graph Store 73

7 Multi-storage type databases 75

7.1 Евентуална конзистентност 79

7.2 MapReduce 82

7.3 MongoDB 83

7.4 Структура на податоци 84

7.5 Оператори ... 85

7.5.1 Reading data (Читање на податоци) ... 85

7.5.2 Manipulating data (Манипулирање со податоци) ... 87

7.6 Правила за интегритет ... 88

7.7 Индексирање ... 88

7.8 Приспособливост ... 90

7.9 Конзистентност ... 91

7.10 Предности и недостатоци на користењето на NoSQL ... 91

7.11 Компјутерска апликација ... 94

7.11.1 Барања за шема на ентитети ... 95

7.11.2 Барања за пристап до податоци ... 95

7.11.3 Што може да направи NoSQL за компјутерска апликација ... 96

7.11.4 Што NoSQL не може да го направи за компјутерска апликација ... 96

7.12 Апликација на веб-скала ... 98

7.12.1 Барања за шема на ентитети ... 99

7.12.2 Барања за пристап до податоци ... 99

7.12.3 Што може да направи NoSQL за веб-скала ... 99

7.12.4 Што NoSQL не може да го направи за веб-скала ... 100

Резиме / Заклучок ... 101

Користена литература ... 102

ЛИСТА НА СЛИКИ

Слика 1 Пример за Key-value store. 8
Слика 2 Scale NoSQL 10
Слика 3 NoSQL database 12
Слика 4 ACID Transactions 13
Слика 5 BASE дефиниција 14
Слика 6 CAP теорема 14
Слика 7 Relational database management system(RDBMS) 15
Слика 8 Iot (Internet of Things) 16
Слика 9 Карактеристики помеѓу NoSQL и релационите бази на податоци 19
Слика 10 Модел на релациски податоци 21
Слика 11 Пример за релациона база на податици што се користи за складирање на информации за луѓе како за работи што им се допаѓаат 22
Слика 12 ACID properties 23
Слика 13 Вертикално и хоризонтално скалирање на податоци 24
Слика 14 Видови на складирање 25
Слика 15 Taxir application 26
Слика 16 Пример за складирање податоци во RDBMS 27
Слика 17 OLAP коцка 28
Слика 18 OLAP vs OLTP пример 28
Слика 19 Apache Cassandra 29
Слика 20 HBase 30
Слика 21 Google Cloud Datastore 30
Слика 22 Пример за складирање податоци во систем со широки колона користејќи дводимениционален клуч 33
Слика 23 Пример податоци претставени во широк склад на податоци во колона 34
Слика 24 Column oriented database 34
Слика 25 Rest Api model 35
Слика 26 Apache Thrift protocol 35
Слика 27 Blob store 40
Слика 28 File store(Document store) 40

Слика 29 Споредба на перформанси помеѓу XSLT и XQuery (лево: recipes, десно: mathml) 41
Слика 30 Пример за MongoDB. 44
Слика 31 CouchDB пример 45
Слика 32 Пример за JavaScript типови 49
Слика 33 Пример за ShoppingDB 53
Слика 34 NoSQL Document store 56
Слика 35 Колекција на Document store 57
Слика 36 Рамката MapReduce со системот за складирање CouchDB 57
Слика 37 Пример шема на GeoJSON 58
Слика 38 Пример податоци претставени во склад за документи. 58
Слика 39 Пример за настан зачуван во базата на податоци MongoDB. 59
Слика 40 Дијаграм кој покажува пример на податоци зачувани како парови со клучни вредности(Key-value) во DynamoDB 60
Слика 41 Пример за Memcached 62
Слика 42 Пример за Redis (Remote Dictionary Server) 63
Слика 43 Системски дизајн за Key-value база на податоци 66
Слика 44 Пример за склад на податоци со клучна вредност 67
Слика 45 Пример податоци претставени во склад со клучна вредност (Key-value store). 68
Слика 46 Графичкото прикажување на графикон 69
Слика 47 Пример за структура на податоци Neo4j 70
Слика 48 Пример за FlockDB graph database 70
Слика 49 Page Rank algorithm 71
Слика 50 HITS algorithm. 71
Слика 51 Пример на податоци претставени како графикон. 74
Слика 52 OrientDB 75
Слика 53 Пример за колекција на рабови во ArangoDB 75
Слика 54 Споредба на моделите за складирање на податоци 76
Слика 55 Пример за основни CRUD операции 77
Слика 56 Пример за Google app engine 78
Слика 57 Пример за CAP theorem 80
Слика 58 Пример за BASE system. 81
Слика 59 Модел на рамката MapReduce. 83

Слика 60 Пример со користење на MapReduce за да се анализира кој работи им се допаѓаат на колку луѓе.....84
Слика 61 Пример за прашања користејќи FIND.....86
Слика 62 MongoDB имплементација на MapReduce (поедноставен излез).....87
Слика 63 Примери за манипулирање со документи во MongoDB користејќи INSERT и UPDATE.....89

КОРИСТЕНИ КРАТЕНКИ

ACID- Atomicity Consistency, Isolation and Durability
CAP- Consistency, Availability and Partition Tolerance
DBMS-Database Management System
RDBMS- Relational Database Management System
IOT- Internet of Things
OLAP - Online analytical processing
OLTP- Online Transaction Processing
HBASE- Hadoop Distributed File System
REDIS- Remote Dictionary Server
HITS- Hyperlink Induced Topic Search
CRUD- CREATE, READ, UPDATE and DELETE
BASE- Basically availability, Soft state, Eventual consistency

ВОВЕД

Во денешната индустрија, се користат многу бази на податоци. Меѓу базите на податоци има SQL бази на податоци и NoSQL бази на податоци. Традиционалната база на податоци е систем на база на податоци SQL кој ги претставува податоците и врските користејќи табеларна релациона парадигма. Базата на податоци NoSQL е понова база на податоци која користи различна техника за складирање и преземање податоци од моделот на табеларни односи на релационите бази на податоци.

NoSQL база на податоци е не-SQL или не-релациона база на податоци која не користи SQL.

Тој обезбедува алтернатива на пристапот на табеларни односи на релационите бази на податоци за складирање и пребарување на податоци. Табелите не се користат во NoSQL базите на податоци за складирање податоци. Најчесто се користи за складирање на големи количини на податоци, како и онлајн апликации во реално време.

Flat file systems беа популарни во раните 1970-ти. Податоците беа зачувани во рамни датотеки, кои имаат неколку недостатоци, вклучувајќи го и фактот дека секоја организација ги користи своите рамни датотеки и нема стандарди. Бидејќи не постои стандарден начин за складирање податоци, исклучително е тешко да се складираат податоци во датотеки и да се преземат податоци од датотеки.

Од страна на EF Codd била осмислена релацискатa база на податоци, која има одговори на прашањето, “ Зошто не постои стандарден начин за складирање на податоци ”. Сепак, релационите бази на податоци подоцна имаа проблем да не можат да ракуваат со големи количини на податоци; како резултат на тоа, се појави побарувачка за база на податоци што може да се справи со сите видови тешкотии и беше создадена базата на податоци NoSQL.

Придобивките од NoSQL

- Има јазик за пребарување.
- Има брзи перформанси.
- Овозможува хоризонтално скалирање.

ИСТОРИЈА

Терминот NoSQL го користел Карло Строци во 1998 година за да ја именува неговата релациона база на податоци со отворен код Strozzi NoSQL која не го изложувала стандардниот интерфејс на структурен јазик за пребарување (SQL), но сепак бил релациски.

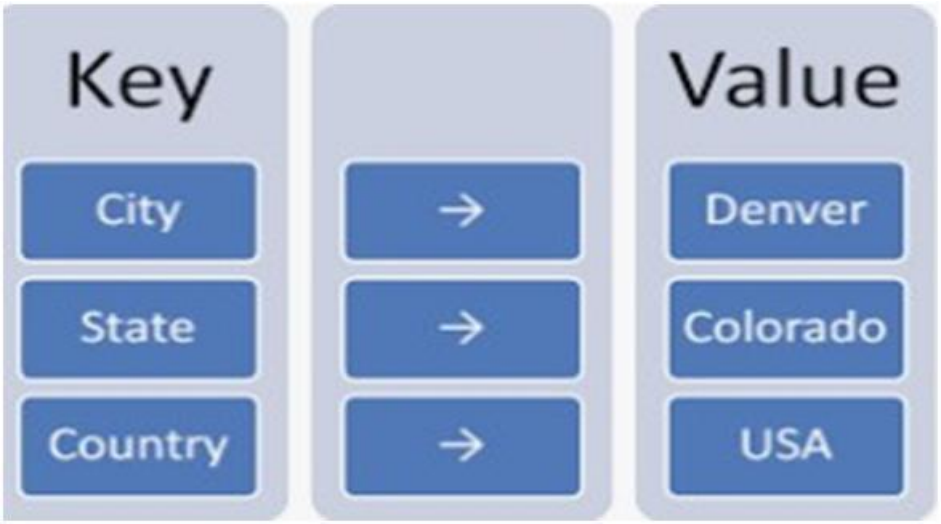
Неговиот NoSQL RDBMS (relational database management system) се разликува од општиот концепт на NoSQL бази на податоци во 2009 година. Strozzi имал сугерирано дека сегашното движење NoSQL целосно отстапува од релацискиот модел,кој требало посоодветно да се нарече NoREL, мислејќи на „не релациско“.

Јохан Оскарсон, тогаш програмер во Last.fm, повторно го воведе терминот NoSQL на почетокот на 2009 година кога организираше настан на кој се дискутираше за „дистрибуирани со отворен код, не-релациони бази на податоци“. Името NoSQL се обиде да ја означи појавата на зголемен број нерелациски, дистрибуирани складишта на податоци, вклучувајќи клонови со отворен код на Bigtable/MapReduce на Google и DynamoDB на Amazon.

Key-value store (Склад за клучни вредности-KV) ја користат асоцијативната низа (исто така наречена мапа или речник) како нивен фундаментален модел на податоци. Во овој модел, податоците се претставени како збирка од парови на клучна вредност, така што секоја можна клучна вредност се појавува најмногу еднаш во колекцијата.

Моделот клучна вредност е еден од наједноставните нетривијални модели на податоци, а побогатите модели на.податоци често се имплементираат како негово продолжение. Моделот клучна вредност може да се прошири на дискретно подреден модел кој ги одржува.клучните вредности по лексикографски редослед. Оваа екстензија е пресметковно моќна, со тоа што може ефикасно повторно да врати селективен опсег на клучни вредности.

Key-value store може да користат модели на конзистентност кои се движат од евентуална конзистентност до серијализација. Некои бази на податоци поддржуваат подредување на клучни вредности. Постојат различни хардверски имплементации, а некои корисници складираат податоци во меморија (RAM), додека други на дискови со цврста состојба (SSD) или ротирачки дискови (познато како хард диск (HDD)).



Слика 1 Пример за Key-value store.

Извор:Google

Истражување на некои знаменитости со клучна вредност во историјата што доведоа до создавањето на NoSQL базите на податоци.

Од Inktomi, кој бил првиот вистински пребарувач,па се до Google, кој во сегашноста е светски лидер, компјутерските научници добро ги препознале ограничувањата на традиционален и широко користен RDBMS конкретно поврзан со прашањата за приспособливост, паралелизација и трошок, исто така забележувајќи дека множеството податоци е минимално вкрстено во споредба со уништените, трансакциски податоци, кои најчесто се внесуваат во RDBMS.

Поточно, ако го земеме само случајот на Google кој добива милијарди барања месечно преку апликации кои можеби се тотално неповрзани во она што го прават, но поврзани со тоа како тие испорачуваат, проблемот на приспособливост треба да се реши на секој слој - директно од податоците до пристапот до конечна испорака.

Затоа, Google морал да работи иновативно и да создаде нов компјутерски екосистем кој се состои од:

- GFS: Дистрибуиран датотечен систем
- Chubby: Дистрибуиран систем за координација
- MapReduce: Систем за паралелно извршување
- Big Data: База на податоци ориентирана кон колони

Овие системи првично беа опишани во наведените трудови објавени од 2003 до 2006 година како што следи:

- Google File System, 2003: http://research.google.com/archive/gfs.html
- Chubby, 2006: http://research.google.com/archive/chubby.html
- MapReduce, 2004: http://research.google.com/archive/mapreduce.html
- Big Data, 2006: http://research.google.com/archive/bigtable.html

Овие и други трудови доведоа до скок на зголемени активности, особено во отворен код, околу дистрибуирани компјутери од големи размери и некои од најневеројатните производи беа создадени. Некои од првичните производи што се појавија вклучуваат:

- Lucene: Java-based indexing and search engine (http://lucene.apache.org)
- Hadoop: For reliable, scalable, distributed computing (http://hadoop.apache.org)
- Cassandra: Scalable, multi-master database with no single point of failure (http://cassandra.apache.org)
- ZooKeeper: High performance coordination service for distributed applications (http://zookeeper.apache.org)
- Pig: High level dataflow language and execution framework for parallel computation (http://pig.apache.org)

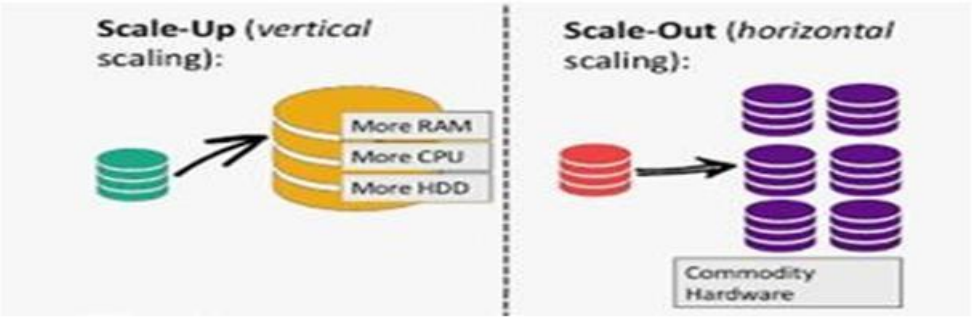
I. NOSQL

1 ПРЕГЛЕД И ДЕФИНИРАЊЕ НА NOSQL

- Што претставува NoSQL
- Што не е NoSQL
- Зошто да се користи NoSQL
- Список на NoSQL бази на податоци

Релационите бази на податоци се користат за складирање на она што е познато како структурирани податоци. Податоците се поделени во групи, наречени табели. Во табелите се складираат добро дефинирани единици на податоци во однос на видот, големината и другите ограничувања. Секоја единица на податоци е позната како колона додека секоја единица од групата е позната како ред. Колоните може да имаат врски дефинирани меѓу себе,, па оттука е и името релациони бази на податоци. И бидејќи конзистентноста е еден од критичните фактори, хоризонталното скалирање е предизвикувачка задача, ако не и невозможна. Во минатото со порастот на големите веб-апликации, се појавиле истражувања во ракување со податоци во голем обем.[1]

Еден од резултатите на овие истражувања е нерелацискатa база на податоци, генерално наречена NoSQL база на податоци. Еден од главните проблеми кои NoSQL базата на податоци ги решава е скалесто.



Слика 2 Scale NoSQL

Извор:Google

[1] *R. Angles and C. Gutierrez. Survey of graph databasemodels. ACM Comput. Surv., 40(1):1:1–1:39, Feb. 2008.*

Според Википедија - Во компјутерите, NoSQL (најчесто се толкува како „not only SQL“) е широка класа на системи за управување со бази на податоци идентификувани со неговото непридржување кон широко користен модел на систем за управување со релациона база на податоци; односно NoSQL базите на податоци не се првенствено изградени на табели, и како резултат на тоа, генерално не користете SQL за манипулација со податоци.[2]

Движењето NoSQL започна во раните години на 21 век кога светот го започна својот длабок фокус на создавање на база на податоци во веб-скала. Под web-scale, се подразбира на скала која се грижи за стотици милиони корисници и сега расте на милијарди поврзани уреди кои се вклучени, но не ограничувајќи се на мобилни телефони, паметни телефони, интернет телевизија, уреди во автомобил и многу други.

Иако Википедија ја третира како „not only SQL“, NoSQL првично започна како едноставна комбинација од два збора - No и SQL - јасно и целосно видлива во новиот термин. Таа нема акроним. Она што буквално значи е: „Не сакам да користам SQL“. „Сакам да пристапам до базата на податоци без да користам никаква SQL синтакса“.

Без оглед на основната фраза, NoSQL денес е терминот што се користи за обраќање до класата на бази на податоци кои не го следат системот за управување со релациона база на податоци (RDBMS) принципи, особено се оние од ACID природа, и се специјално дизајнирани да се справи со брзината и обемот на оние како Google, Facebook, Yahoo, Twitter и многу повеќе.

[2] *Википедија*

1.1 Што претставува NoSQL, а што не

NoSQL е генерички термин што се користи за да се однесува на која било склад за податоци што не го следи традиционалниот RDBMS модел - поточно, податоците се нерелациски и не користат SQL како јазик за пребарување.[3] Се користи за да се однесува на базите на податоци што се обидуваат да ги решат проблемите на приспособливост и достапност наспроти онаа на атомноста или конзистентноста.



Слика 3 NoSQL database

Извор:Google

NoSQL не е база на податоци. Тоа не е ни тип на база на податоци. Всушност, тоа е термин кој се користи за филтриран збир на бази на податоци надвор од екосистемот. Постојат неколку различни достапни семејни стебла.

Традиционалните RDBMS апликации се фокусираа на ACID трансакции:

- Atomicity: Се што е во трансакцијата успева да не се врати назад.
- Consistency: Трансакцијата не може да ја остави базата на податоци во неконзистентна состојба.
- Isolation: Една трансакција не може да и пречи на друга.
- Durability: Завршената трансакција опстојува, дури и по рестартирањена апликациите.

[3] *Apache. Couchdb - a database for the web. http://couchdb.apache.org/, July 2012.*



Слика 4 ACID Transactions

Извор:Google

Колку и да изгледаат неопходни овие квалитети, тие се сосема некомпатибилни со достапност и перформанси на апликации од web-scale.

На пример, ако компанија како Амазон требала да користи ваков систем, тогаш ваквиот систем би бил бавен. Ако во текот на некоја продажба трансакцијата е вклучена, тогаш некои делови од базата на податоци ќе бидат заклучени, во конкретен случај би бил заклучен инвентарот, а секој друг корисник во светот ќе мора да почека да заврши започнатата трансакција.[4]

Амазон може да користи кеширани податоци или отклучени записи што ќе резултира со недоследност. Во екстремен случај, доколку двајца би сакале да го купат последниот примерок од една книга во продавница еден од нив би добил писмо со извинување преку пошта.

[4] *NoSQL For Dummies Adam Fowler Professional NoSQL Shashank Tiwari*

BASE, измислено од Ерик Бруер:

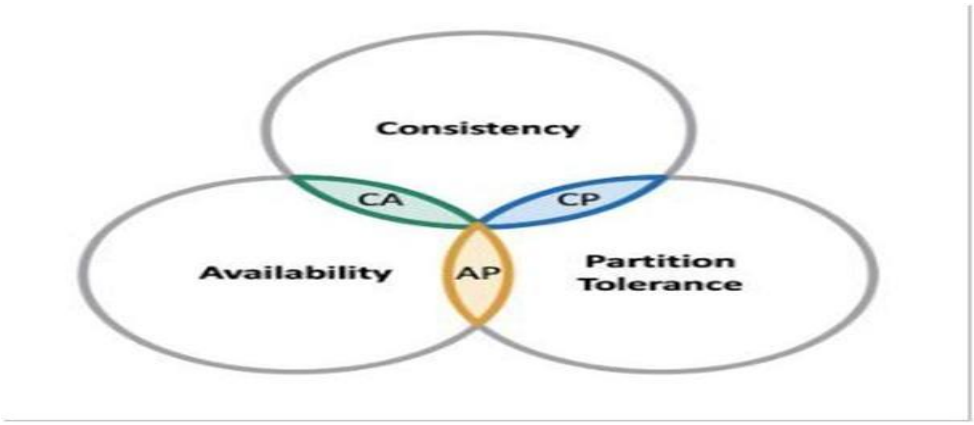
- Basically availability: На секое барање му се гарантира одговор - успешено или неуспешно извршување.
- Soft state: Состојбата на системот може да се менува со текот на времето, понекогаш без кој било влез (за евентуална конзистентност).
- Eventual consistency: Базата на податоци може да биде моментално неконзистентна, но ќе бидат конзистентни на крајот.



Слика 5 BASE дефиниција

Извор:Google

Ерик Бруер, исто така, забележал дека е невозможно дистрибуиран компјутерски систем да обезбеди конзистентност, достапност и толеранција на партиции истовремено. Ова е почесто се нарекува теорема CAP.



Слика 6 CAP теорема

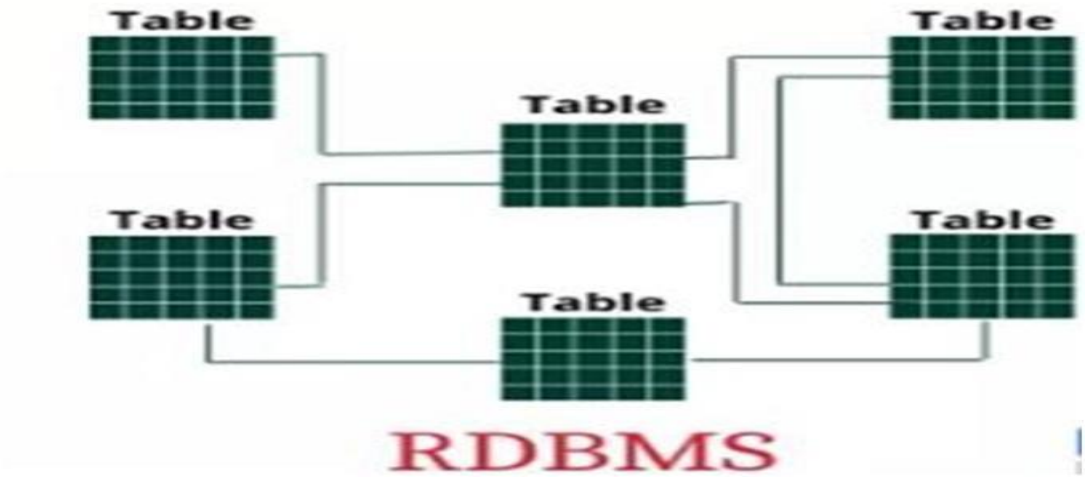
Извор:Google

Сепак, во одредени случаеви како берзи или банкарство каде што трансакциите се критични, кешираните или државните податоци едноставно нема да работат. Значи NoSQL дефинитивно не е решение за сите проблеми поврзани со базата на податоци

Ограничената приспособливост на SQL ја поттикнала индустријата да развие и распореди голем број NoSQL системи за управување со бази на податоци, со фокус на перформансите, доверливоста и конзистентноста. Трендот бил поттикнат од комерцијалните NoSQL бази на податоци развиени од Google и Amazon. На крајот, системите со отворен код како MongoDB, Cassandra и Hypertable го приспособија NoSQL на дофат на секого.

Несомнено е дека начинот на кој веб-апликациите се справуваат со податоците значително се променети во изминатата деценија. Се собираат повеќе податоци и повеќе корисници кои пристапуваат до овие податоци истовремено од кога било досега. Ова значи дека приспособливоста и перформансите се поголем предизвик од кога било за релационите бази на податоци кои се базирани на шема и затоа може да биде потешко да се скалираат.

NoSQL е пристап кон базите на податоци што претставува поместување од традиционалните системи за управување со релациони бази на податоци (RDBMS).

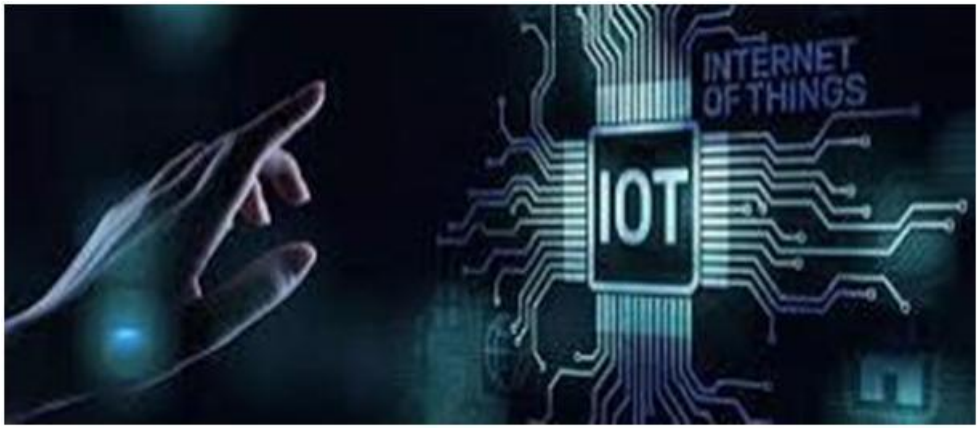


Слика 7 Relational database management system(RDBMS)

Извор:Google

За да се дефинира NoSQL, корисно е да се започне со опишување на SQL, кој е јазик за пребарување што го користи RDBMS. Релационите бази на податоци се потпираат на табели, колони, редови или шеми за организирање и преземање податоци. Спротивно на тоа, NoSQL базите на податоци не се потпираат на овие структури и користат пофлексибилни модели на податоци. NoSQL може да значи „no SQL“ или „not only SQL“.

Бидејќи RDBMS сè повеќе не успева да ги исполни потребите за перформанси, приспособливост и флексибилност што ги бараат апликациите од следната генерација и интензивни податоци, NoSQL базите на податоци беа усвоени од главните претпријатија. NoSQL е особено корисен за складирање на неструктурирани податоци, кои растат многу побрзо од структурираните податоци и не одговараат на релационите шеми на RDBMS. Вообичаени типови на неструктурирани податоци вклучуваат: податоци за корисникот и сесијата; податоци за разговор, пораки и дневници; податоци за временски серии како што се IoT и податоци за уредот; и големи објекти како видео и слики.



Слика 8 Iot (Internet of Things)

Извор:Google

1.2 Предности и недостатоци на NoSQL базите на податоци

Предности - NoSQL базите на податоци имаат многу предности во споредба со традиционалните, релациони бази на податоци. Една голема, основна разлика е тоа што NoSQL базите на податоци имаат едноставна и флексибилна структура. Тие се без шема.

За разлика од релационите бази на податоци, NoSQL базите на податоци се засноваат на парови на клучна вредност. Некои типови складишта на бази на податоци NoSQL вклучуваат складирање на колони, складирање документи, складирање на клучни вредности, складирање на графикони, складирање на објекти, складирање XML формат и други режими за складирање податоци.

Обично, секоја вредност во базата на податоци има клучна вредност. Некои складишта на базата на податоци NoSQL, исто така, им дозволуваат на програмерите да складираат сериски објекти во базата на податоци, а не само едноставни вредности на низата.

Базите на податоци NoSQL со отворен код не бараат скапи такси за лиценцирање и можат да работат на ефтин хардвер, што го прави нивното распоредување економично. Исто така, кога се работи со NoSQL бази на податоци, без разлика дали се со отворен код или сопственички, проширувањето е полесно и поевтино отколку кога се работи со релациони бази на податоци. Тоа е затоа што се прави со хоризонтално скалирање и дистрибуирање на оптоварувањето на сите јазли, наместо со типот на вертикално скалирање што обично се прави со системите на релациона база на податоци, што го заменува главниот домаќин со помоќен. [5]

[5] . *The mongodb company. http://www.10gen.com/, July 2012.*

Недостатоци - NoSQL базите на податоци не се совршени и не се секогаш вистинскиот избор. Повеќето NoSQL бази на податоци не поддржуваат карактеристики за доверливост што се природно поддржани од системите за релациона база на податоци. Овие карактеристики на сигурност може да се сумираат како атомност, конзистентност, изолација и издржливост. Ова исто така значи дека базите на податоци NoSQL, кои не ги поддржуваат тие карактеристики, тргуваат со конзистентност со перформанси и приспособливост.

Со цел да се поддржат карактеристиките на доверливост и конзистентност, програмерите мора да имплементираат својот сопствен код, што додава поголема сложеност на системот.[6] Ова може да го ограничи бројот на апликации што можат да се потпрат на NoSQL базите на податоци за безбедни и сигурни трансакции, како што се банкарските системи.

Други форми на сложеност пронајдени во повеќето NoSQL бази на податоци вклучуваат некомпатибилност со SQL барања. Ова значи дека е потребен код или јазик кој е недостапен за пребарување, што додава уште повеќе време и сложеност.

[6] *MongoDB. The mongodb manual. http://docs.mongodb.org/manual/, July 2012.*

Оваа табела дава кратка споредба на карактеристики помеѓу NoSQL и релационите бази на податоци:

Feature	NoSQL Databases	Relational Databases
Performance	High	Low
Reliability	Poor	Good
Availability	Good	Good
Consistency	Poor	Good
Data Storage	Optimized for huge data	Medium sized to large
Scalability	High	High (but more expensive)

Слика 9 Карактеристики помеѓу NoSQL и релационите бази на податоци

Извор:Google

Треба да се забележи дека табелата покажува споредба на ниво на бази на податоци, а не различни системи за управување со бази на податоци кои ги имплементираат двата модели. Овие системи обезбедуваат свои сопствени техники за да се надминат некои од проблемите и недостатоците во двата системи, а во некои случаи, значително да се подобрат перформансите и доверливоста.

II. ЗОШТО NOSQL ?

2 NOSQL И РЕЛАЦИОНИ БАЗИ НА ПОДАТОЦИ

NoSQL базите на податоци имаат многу повеќе да понудат отколку само решавање на проблемите на обемот кои се споменати како што следува:

- Schemaless data representation(презентација на податоци без шема) Ги нудат речиси сите имплементации на NoSQL без шематски приказ на податоци. Ова значи дека не треба да се размислува за да се дефинира структура и може да се продолжи да се развива со текот на времето - вклучувајќи додавање нови полиња или како и вгнездување на податоците, на пример, во случај на JSON застапеност.[7]

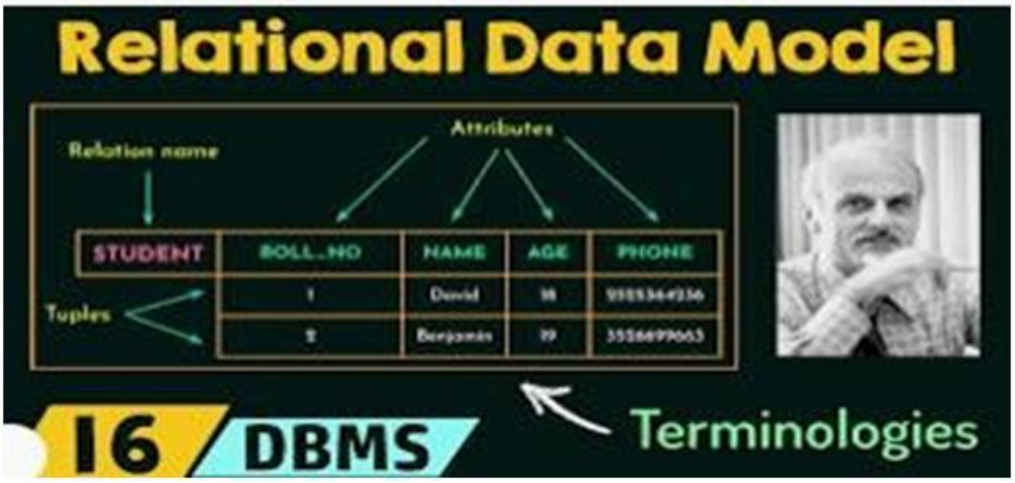
- Speed (брзина) Со мала количина на податоци што се понудени , може да се испорачаат во милисекунди, а не во стотици милисекунди - особено над мобилни и други наизменично поврзани уреди-имате многу повисока веројатност за освојување на корисниците.

- Plan ahead for scalability: (предвремено планирање на приспособливоста) Одредена апликација може да биде прилично еластична - може да се справи со ненадејни скокови на оптоварување.

[7] *NoSQL the Ruby Way Durran Jordan*

Релациони бази на податоци - Едгар Ф. Код, пронаоѓачот на релацискиот модел, идентификувани се три основни компоненти кои го дефинираат моделот на податоци.

1. Структура на податоци –типови на податоци од кои е изграден моделот
2. Оператори – достапните операции за преземање или манипулирање со податоци од оваа структура
3. Правила за интегритет-општите правила што гидефинираат конзистентните состојби на податочниот модел.



Слика 10 Модел на релациски податоци

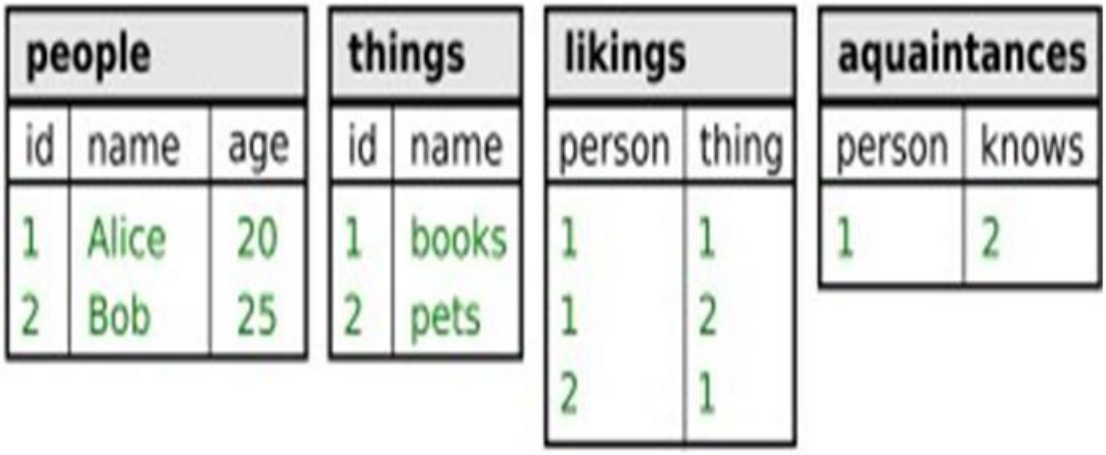
Извор:Google

Структурата на моделот на релациски податоци е главно дадена по релации, атрибути, топки и (примарни) клучеви. Односите обично се визуелизираат како табели, со атрибути како колони и топки како редови.[8] Редоследот на атрибутите и топките не се дефинирани од структурата, па затоа може да бидат произволни.

[8] *D. Bermbach and S. Tai. Eventual consistency: Howsoon is eventual? an evaluation of amazon s3's consistency behavior. In Proceedings of the 6th Workshop on Middleware for Service Oriented Computing, MW4SOC '11, pages 1:1–1:6, New York, NY, USA, 2011. ACM.*

Основните операции дефинирани со релацискиот модел се операции SELECT (вклучувајќи проекции и спојување) за враќање на податоци, како и манипулативни операции како INSERT, UPDATE, DELETE.

Може да се разликуваат две различни групи на правила на интегритет Дополнително, постојат правила за референцијален интегритет помеѓу различни односи.



people

id	name	age
1	Alice	20
2	Bob	25

things

id	name
1	books
2	pets

likings

person	thing
1	1
1	2
2	1

aquaintances

person	knows
1	2

Слика 11 Пример за релациона база на податици што се користи за складирање на информации за луѓе како за работи што им се допаѓаат .

Извор:Google

2.1 ACID properties

Важен концепт во системите на релациони бази на податоци се трансакции. Дефинирани се три својства на трансакција : атомност, конзистентност и издржливост. Подоцна Хардер и Ројтер ги скратија тие својства заедно со четврта - изолација по кратенката ACID[9]. Иако сите четири ACID својства се гледаат како клучни својства на трансакциите на релационите бази на податоци, конзистентноста е особено важна кога се истражува приспособливоста на еден систем.



Слика 12 ACID properties

Извор:Google

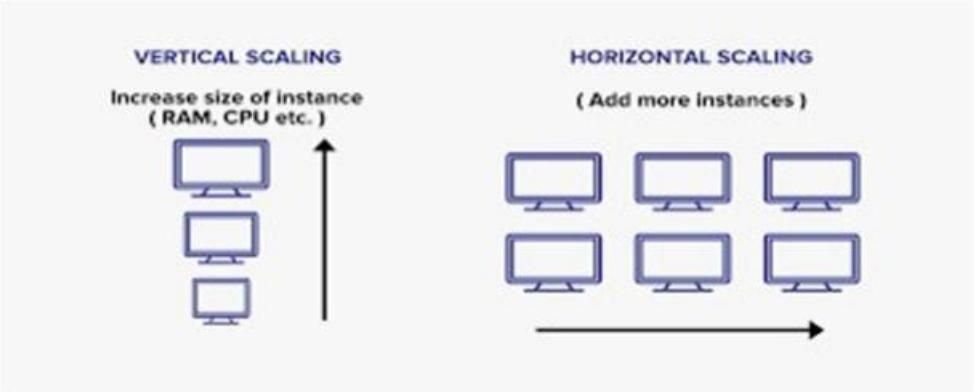
2.2 Приспособливост, Типови и Видови на складирање NoSQL

Приспособливост - Приспособливоста на системот е неговата способност да се справи со растечки обем на работа. Во основа, системот може да се зголеми во две различни насоки : вертикално и хоризонтално .

Вертикално складирање (скала до) значи зголемување на капацитетот на јазли на системот. Ова може да се постигне со користење на помоќен хардвер. Напротив, системот се скалира хоризонтално (намалена) ако се додадат повеќе јазли.

[9] *D. Bermbach and S. Tai. Eventual consistency: Howsoon is eventual? an evaluation of amazon s3's consistency behavior. In Proceedings of the 6th Workshop on Middleware for Service Oriented Computing, MW4SOC '11, pages 1:1–1:6, New York, NY, USA, 2011. ACM.*

Вертикалното скалирање на системите обично е ограничено од достапноста на подобар хардвер и има тенденција да биде ефикасно. Напротив, хоризонталното скалирање на системите овозможува подобри перформанси, додека се користи поевтин хадвер. Иако , хоризонталното скалирање често е нетривијална задача. Ова е главно затоа што гаранциите за конзистентност како што бараат својствата на ACID, е тешко да се постигнат дистрибуирани системи.



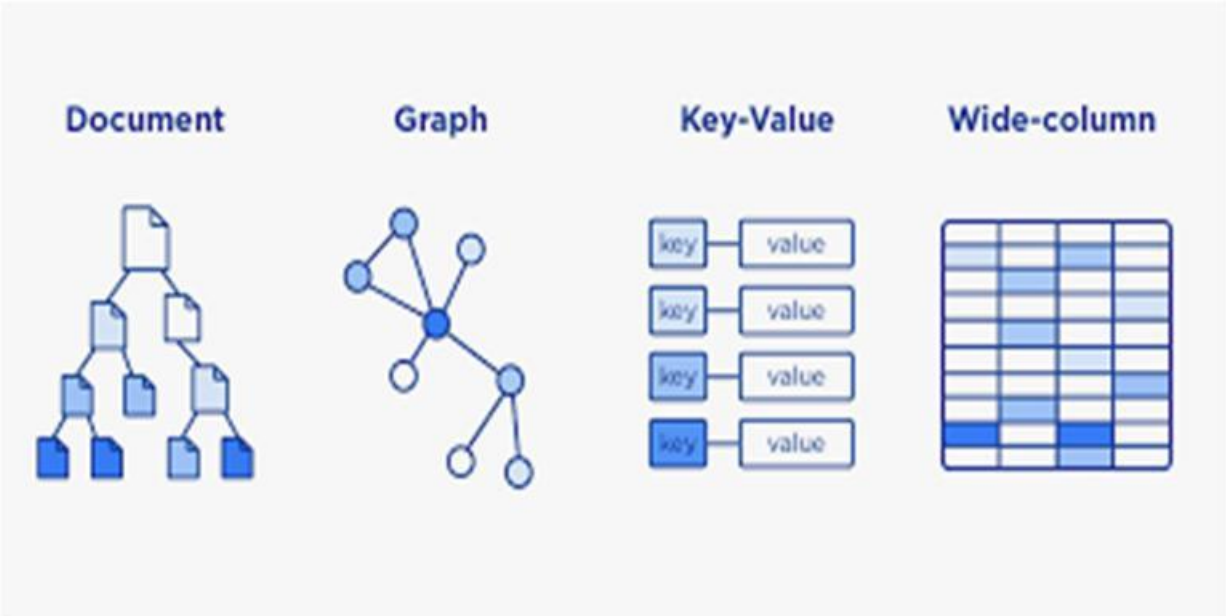
Слика 13 Вертикално и хоризонтално скалирање на податоци

Извор: Google

Типови на складирање NoSQL - Базите на податоци на NoSQL се категоризираат врз основа на тоа како се чуваат податоците. Бидејќи за потребата да се обезбедат одредени информации од големи количини, генерално во близина во реално време, NoSQL главно следи хоризонтална структура. Тие се оптимизирани за вметнување и преземање операции во голем обем со вградени способности за репликација и кластерирање. Некои од функционалностите на SQL базите на податоци како функции, складирани процедури, а PL може да не е присутна во повеќето бази на податоци.

Видови на складирање - Достапни се различни типови на складирање за кои може да се моделира содржината NoSQL бази на податоци. Во следните делови, ќе го истражиме следното складирање типови:

- Document Store - Склад за документи
- Graph Store - Складирање на податоци со помош на графикони
- Key Value Store- Складирање на податоци со клучна вредност
- Column-oriented(Wide-column)- Складирање на податоци ориентирани кон колони



Слика 14 Видови на складирање

Извор: Google

III. ВИДОВИ НА СКЛАДИРАЊЕ

3 COLUMN-ORIENTED

Базите на податоци ориентирани кон колони складираат податоци како колони наспроти редови кои се истакнати во RDBMS.

Базите на податоци ориентирани кон колони се во развој уште од деновите на DBMS. TAXIR, биологија фокусирана на пронаоѓање информации , таа беше првата апликација која користи колона-ориентирана од 1969 година.[10]



Слика 15 Taxir application

Извор: Google

Релациска база на податоци ги прикажува податоците како дводимензионални табели што се состојат од редови и колони, но ги складира, презема и обработува еден ред во исто време, а Колона-ориентирана база на податоци ги складира податоците како колони.

[10] *M. Michael, J. Moreira, D. Shiloach, and R. Wisniewski. Scale-up x scale- out: A case study using nutch/lucene. In Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International, pages 1 –8, march 2007.*

Доколку следните податоци треба да се складираат тоа ќе изгледа вака :

EmployeeID	FirstName	LastName	Age	Salary
SM1	Anuj	Sharma	45	10000000
MM2	Anand		34	5000000
T3	Vikas	Gupta	39	7500000
E4	Dinesh	Verma	32	2000000

Слика 16 Пример за складирање податоци во RDBMS

Извор: Google

Во RDBMS, податоците може да се серијализираат и да се складираат внатрешно на следниов начин:

SM1,Anuj,Sharma,45,10000000 MM2,Anand,,34,5000000

T3,Vikas,Gupta,39,7500000 E4,Dinesh,Verma,32,2000000

Меѓутоа, во базите на податоци ориентирани кон колони, податоците ќе се складираат внатрешно на следниов начин:

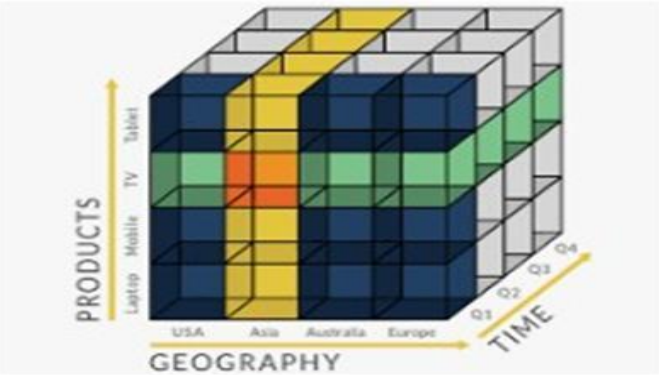
SM1,MM2,T3,E4

Anuj,Anand,Vikas,Dinesh Sharma,,Gupta,Verma, 45,34,39,32

10000000,5000000,7500000,2000000

Претходниот начин е премногу поедноставен. Базите на податоци ќе имаат пософистицирани и оптимизирани механизми за складирање податоци. Задачи како партиционирање, кеширање, индексирање и способност за креирање

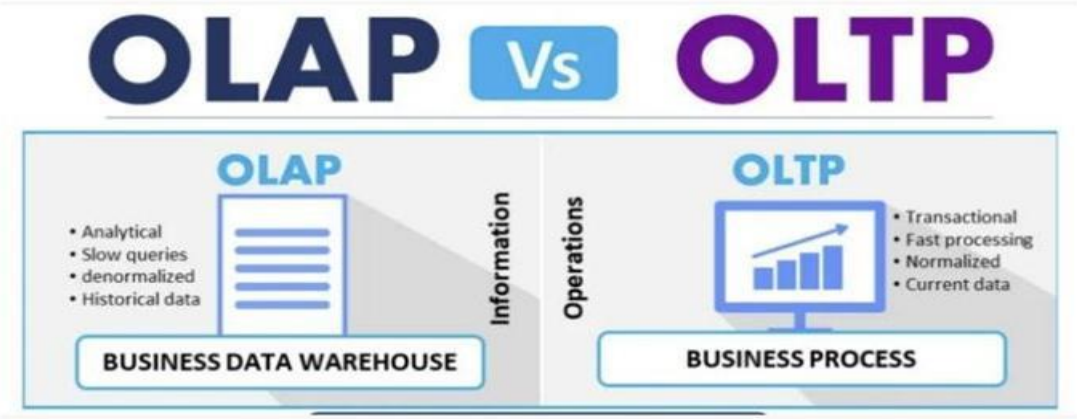
Коцките OLAP и други влијаат на основната физичка организација на податоците во системот.



Слика 17 OLAP коцка

Извор: Google

Релационите бази на податоци се фокусирани на обработка на трансакции ориентирани преку Интернет (OLTP). Системи за онлајн аналитичка обработка (OLAP) кои бараат обработка на податоци треба да има пристап ориентиран кон колона. Операциите на OLTP исто така може да бараат пристап ориентиран кон колона кога работите на подмножество колони и кога работите на нив.



Слика 18 OLAP vs OLTP пример

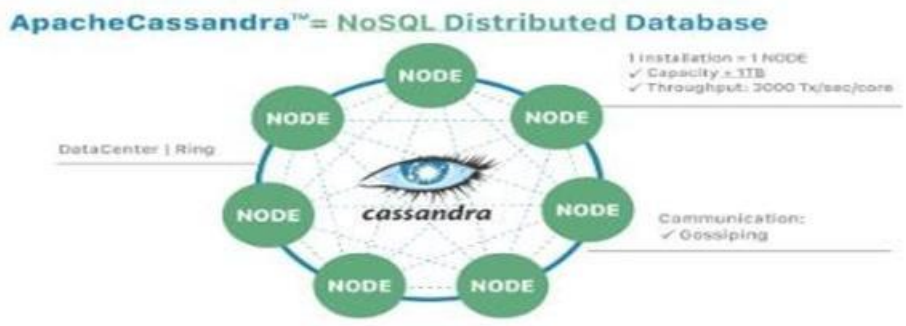
Извор: Google

Пристапот до податоците до овие бази на податоци вообичаено се врши со користење на комерцијален протокол во случај на комерцијални решенија или со отворен бинарен стандард (на пример, [11]Повикување на методот од далечина). Транспортниот протокол е генерално бинарен.

Некои од базите на податоци кои спаѓаат во оваа категорија вклучуваат:

- Oracle RDBMS Columnar Expression
- Microsoft SQL Server 2012 Enterprise Edition
- Apache Cassandra
- HBase
- Google BigTable (available as part of Google App Engine branded Datastore)

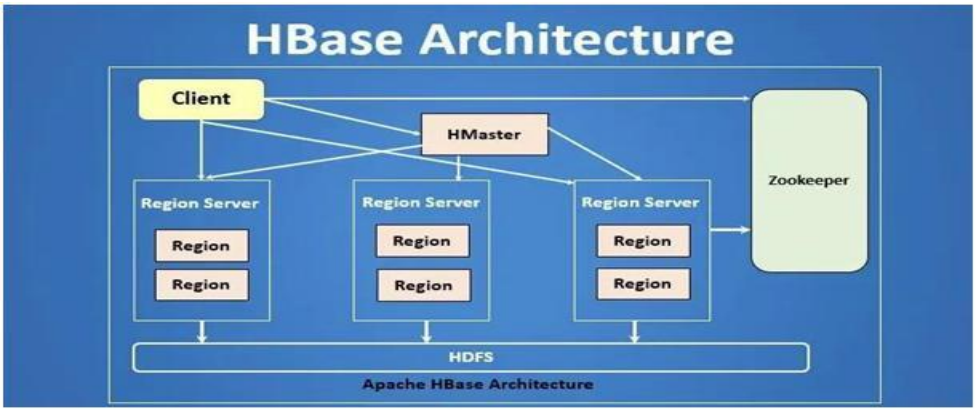
Предности - Повеќето решенија, како што се Apache Cassandra, HBase и Google Cloud Datastore, дозволуваат додавање колони со текот на времето за пополнување на стандардните вредности за постоечките редови за новите колони. Ова дава флексибилност во моделот и дизајн на ентитет што овозможува да се земат предвид новите колони во иднина за непредвидени сценарија и нови барања.[12]



Слика 19 Apache Cassandra

Извор: Google

[11] *S. Edlich, A. Friedland, J. Hampe, and B. Brauer. NoSQL: Einstieg in die Welt nichtrelationaler Web 2.0 Datenbanken. Hanser Fachbuchverlag, 10 2010.*
[12] *NoSQL the Ruby Way Durran Jordan*



Слика 20 HBase

Извор: Google



Слика 21 Google Cloud Datastore

Извор: Google

Има предности при работа со подмножество од достапните колони. За на пример, пресметување на максимум, минимум, проседи и збирови, конкретно на големи збирки на податоци, е местото каде што овие складишта на податоци ориентирани кон колони се подобри во перформансите.[13]Слично на тоа, кога се применуваат нови вредности или за сите редови одеднаш или со исти- филтри за колони, овие бази на податоци ќе овозможат делумен пристап до податоците без допирање неповрзани колони и да бидат многу побрзи во извршувањето.

[13] *D. Borthakur, J. Gray, J. S. Sarma, K. Muthukkaruppan, N. Spiegelberg, H. Kuang, K. Ranganathan,*

Бидејќи колоните ќе бидат од униформен тип и најчесто (освен во случаите со променлива должина на мрежата) со иста должина, постојат можности за ефикасно складирање во однос на големината. Како колона со исти вредности низ редовите (на пример, одделот за кориснички профил или дали профилот на корисникот е јавен или приватен или дури и на возраста на корисникот), на исти или слични соседни вредности може ефикасно да се компресираат.

Пример-Во следниот пример, ќе се забележи пример од код за работа со Google datastore (може да се најде на https://developers.google.com/appengine/docs/ java/datastore/) на Google App Engine користејќи го Objectify (http://code. google.com/p/objectify- appengine/) библиотека:

```
public class UserProfile

{

@Id String employeeID;

String firstName;

String lastName;

String age;

long salary;

}

ObjectifyService.register(UserProfile.class);

Objectify ofy = ObjectifyService.begin();

//Adding records
```

```
UserProfile up1 = new UserProfile(...);

UserProfile up2 = new UserProfile(...);

ofy.put(up1, up2);

//Retrieve by Id

UserProfile upg = ofy.get(UserProfile.class, "SM1");

//Filter all profiles by first name

 Iterator<UserProfile> profiles

= ofy.query(UserProfile.class)

.filter("firstName", "Alice");

//Query all profiles by age greater than 30, ordered by salary

 Iterator<UserProfile> agedProfiles

= ofy.query(UserProfile.class)

.filter("age >", 30)

.order("salary");
```

3.1 Column-oriented (Wide column) store

Системите за складирање од оваа класа се нарекуваат и Extensible Record store. Широк склад за колони може да се гледа како складиште за клучна вредност, со дводимензионален клуч, зачувана вредност се референцира со клуч за колона и клуч за ред. Клучот може да биде понатаму продолжен со временски печат, како што е случајот во BigTable на Google. Во зависност од имплементацијата, има повеќе можни екстензии на клучот, најчесто наречени keyspaces или domains. Така клучевите во складот со широки колони може да имаат многу димензии. Што резултира со структура слична на повеќедимензионална, асоцијативна низа.

Складот на колони поддржува податоци како колони наместо редови, аранжман што обично е оптимизиран за прашања преку големи збирки на податоци. Барањето низ редови е интензивно на меморија и бара огромен пристап на дискот, особено кога секој ред содржи многу колони. [14]Со складирање на колони, колоните се групирани во фамилии на колони и секоја фамилија на колони може да има неограничен број на колони. На овој начин е многу полесно да се побара целата збирка колони за сите редови. BigTable, HBase и Cassandra на Google се најпопуларните бази на податоци засновани на колони.

Во примерот подолу, прикажани се различните структури на складиштето засновано на редови (лево) и складиштето засновано на колони (десно) .

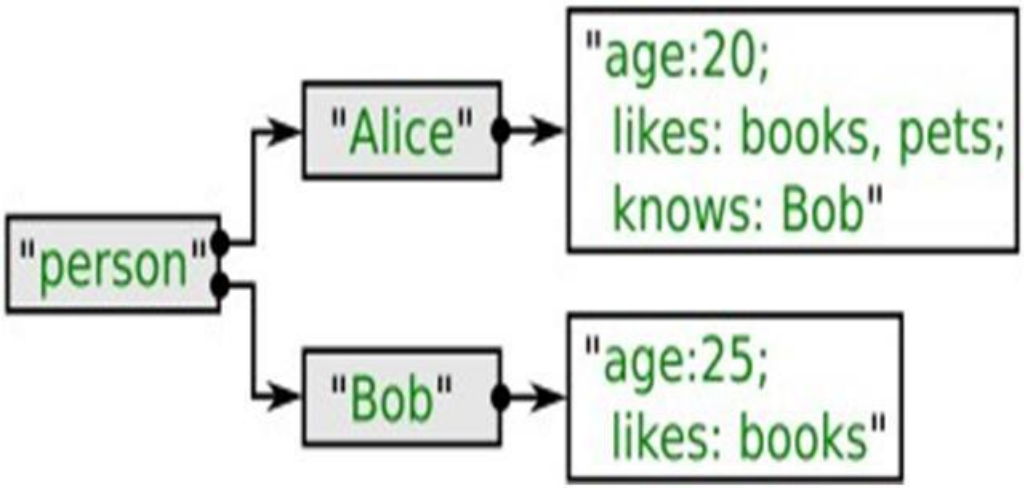
Name	GPA	Year	Program
Tom	3.4	2	GEOG
Mary	4.0	1	CS
David	3.7	3	ENV

Name	GPA	Year	Program
Tom	3.4	2	GEOG
Mary	4.0	1	CS
David	3.7	3	ENV

Слика 22 Пример за складирање податоци во систем со широки колона користејќи дводимениционален клуч

Извор: Google

[14] *A. B. Bondi. Characteristics of scalability and their impact on performance. In Proceedings of the 2nd international workshop on Software and performance, WOSP '00,pages 195–203, New York, NY, USA, 2000. ACM.*



Слика 23 Пример податоци претставени во широк склад на податоци во колона .

Извор: Google

Овде “лицето ” се користи како клуч за колона и името на секое лице како клуч за ред.

Како во Key-Value store , складираните вредности дополнително не се толкува од системот.

Name	ID
John	001
Karen	002
Bill	003

Grade	ID
Senior	001
Freshman	002
Junior	003

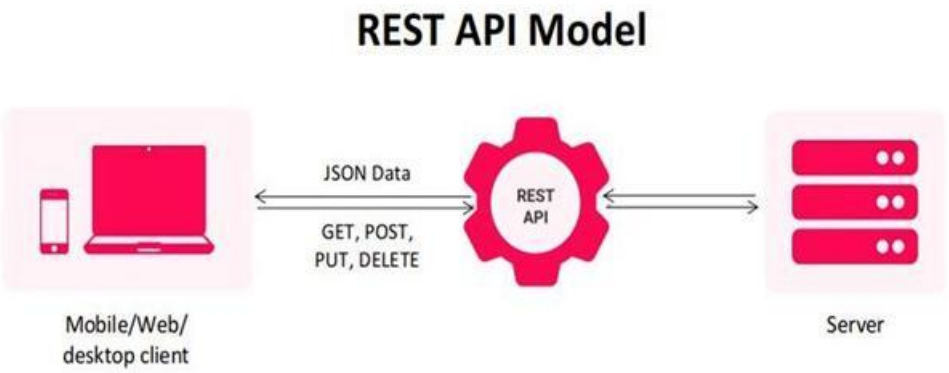
GPA	ID
4.00	001
3.67	002
3.33	003

Слика 24 Column oriented database

Извор: Google

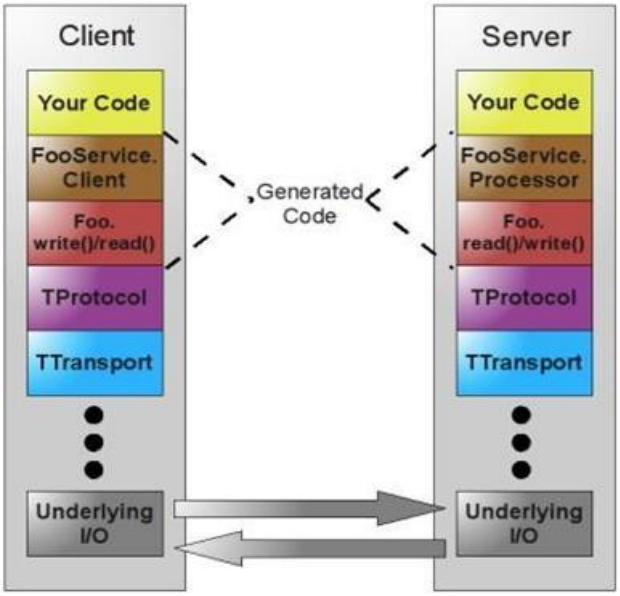
4 DOCUMENT STORE DATABASE

Исто така наречена база на податоци ориентирана кон документи, складот за документи дозволува вметнување, преземање и манипулирање со полуструктурирани податоци. Повеќето од базите на податоци достапни под оваа категорија користат XML, JSON, BSON или YAML, со пристап до податоци обично преку HTTP протокол користејќи RESTful API или преку Apache Thrift протокол за меѓујазична интероперабилност.



Слика 25 Rest Api model

Извор: Google



Слика 26 Apache Thrift protocol

Извор: Google

Во споредба со RDBMS, самите документи дејствуваат како записи (или редови), меѓутоа, тој е полуструктуриран во споредба со крутиот RDBMS.

На пример, два записи може да имаат сосема различни полиња или колони.

Записите може или не може да се придржуваат до одредена шема (како дефинициите на табелата во RDBMS). За таа работа, базата на податоци може да не поддржува шема или потврдување документ против шемата.

Иако документите не следат строга шема, може да се креираат индекси и прашања. Наведени се неколку примери на содржина на документи користејќи JSON формат:

Еден документ може да обезбеди вработен чии што детали не се потполно познати:

```
{
"EmployeeID": "SM1",
"FirstName" : "Anuj",
 "LastName" : "Sharma",
 "Age" : 45,
"Salary" : 10000000
```

Вториот документ може да содржи целосни детали за друг вработен:

```
{
"EmployeeID": "MM2",
 "FirstName" : "Anand",
 "Age" : 34,
"Salary" : 5000000
```

"Address" : {

"Line1" : "123, 4th Street",

"City" : "Bangalore",

"State" : "Karnataka"

},

"Projects" : [

"nosql-migration",

"top-secret-007"

]

}

Трет документ може да има информации за една од локациите на канцелариите:

{

"LocationID" : "Bangalore-SDC-BTP-CVRN",

"RegisteredName" : "ACME Software Development Ltd"

"RegisteredAddress" : {

"Line1" : "123, 4th Street",

"City" : "Bangalore",

"State" : "Karnataka"

},

}

Ако се забележат претходните примери, првите два документи се нешто слични при што вториот документ има повеќе детали во споредба со првиот. Сепак, ако го погледнете третиот документ, содржината нема корелација со првите два какви било документи - се работи за локација на канцеларија, а не за вработен. ID на вработениот или ID на локација може да не се ID на документот.

Базите на податоци обезбедуваат пристап со користење на RESTful API каде што ID на документот е дел од самиот URL или се предвидени во телото на барањето, содржината на документот не треба да го содржи неговиот ID. [15]Всушност, една од најдобрите практики наведува дека ID документот мора да биде вграден во документот некаде и по можност на стандардна локација. На пример, изменетата содржина може да биде:

```
{
"docId": "SM1",
...
}
{
"docId": "MM2",
...
}
{
"docId": "Bangalore-SDC-BTP-CVRN",
...
}
```

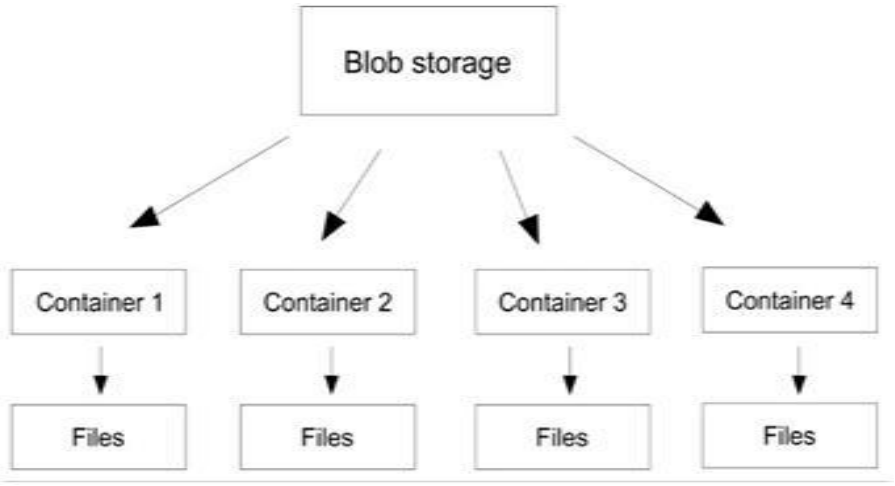
[15] *S. Edlich. Nosql databases. http:// nosql-database.org/, July 2012.*

Базите на податоци ориентирани кон документи ја обезбедуваат оваа флексибилно - динамична или променлива шема или документи без шема. Поради неограничената обезбедена флексибилност во овој модел, ова е еден од најпопуларните модели кои се имплементирани и користени. [16]Некои од популарните бази на податоци кои обезбедуваат складирање ориентирано кон документи вклучуваат:

- MongoDB
- CouchDB
- Jackrabbit
- Lotus Notes
- Apache Cassandra
- Terrastore
- Redis
- BaseX

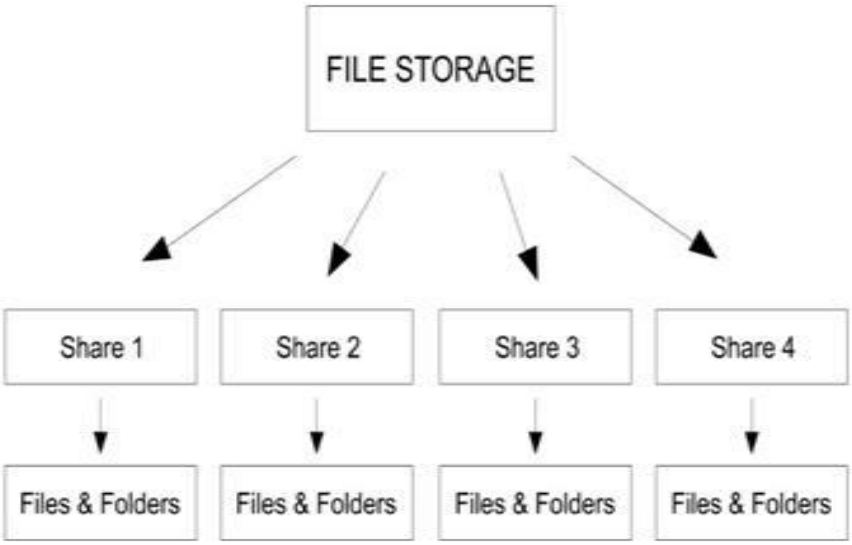
Предности - Најистакнатата предност, како што е видливо во претходните примери, е тоа содржината да е без шема, или во најдобар случај лабаво дефинирана. Ова е многу корисно во веб-базирани апликации каде што има потреба од складирање на различни типови на содржина што може еволуираат со текот на времето. [17]На пример, за самопослуга, информации за корисниците, инвентарот и нарачките може да се складираат како едноставни JSON или XML документи. Забележително е тоа што „Document store“ не е исто што и „blob store“ каде што податоците не можат да се индексираат.

[16] *G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels. Dynamo: amazon's highly available key-value store. SIGOPS Oper. Syst. Rev., 41(6):205–220, Oct. 2007.*
[17] *NoSQL and SQL Data Modeling Ted Hills*



Слика 27 Blob store

Извор :Google

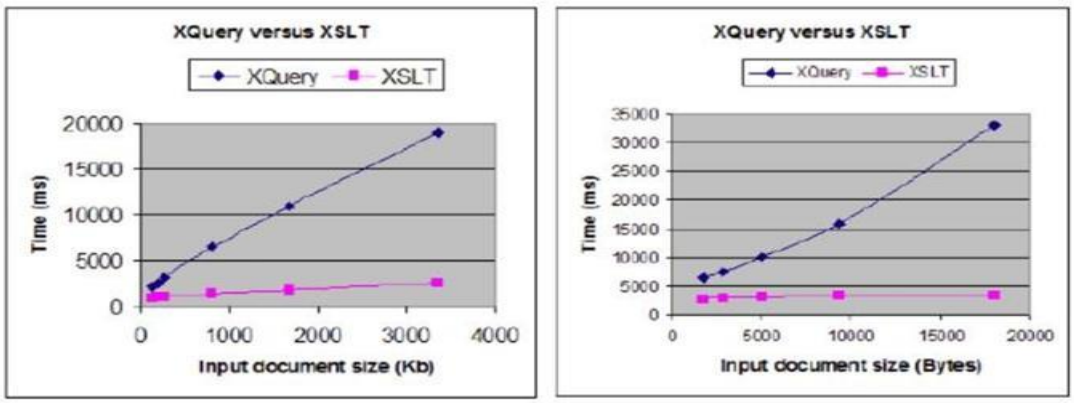


Слика 28 File store(Document store)

Извор :Google

Врз основа на имплементацијата, може или не е можно да се преземе или ажурира и делумно да се надоградува . Ако е можно да се направи тоа, има голема предност.[18] Складовите базирани на XML, BSON, JSON и YAML обично го поддржуваат ова. Складовите базирани на XML BaseX може да бидат навистина моќени, додека интегрираат повеќе системи кои работат со XML имајќи предвид дека поддржуваат XQuery 3.0 и XSLT 2.0.

[18] *SQL & NoSQL Databases Andreas Meier, Michael Kaufmann*



Слика 29 Споредба на перформанси помеѓу XSLT и XQuery (лево: recipes, десно: mathml)

Извор :Google

Пребарувањето низ повеќе типови ентитети е многу потривијално во споредба со тоа во традиционалните RDBMS или дури и во бази на податоци ориентирани кон колони. Затоа што, сега, нема концепт на табели - кој во суштина не е ништо повеќе од дефиниција на шема – еден може да пребарува низ записите, без оглед на основната содржина или шема или со други зборови, барањето е директно против целата база на податоци. [19]Базите на податоци овозможуваат создавање на индекси (со користење на заеднички параметри или на друг начин и се развиваат прекувремено).

Складовите базирани на JSON лесно се дефинираат како проекции. Секој клуч од највисоко ниво за објектот JSON може да биде проекција на ентитетот низ други делови од системот со што се овозможува шемата да се развива со текот на времето со повратна компатибилност.

Примери

Документ базиран на JSON кој ги демонстрира предностите:

{

"me": {

[19] *R. Cattell. Scalable sql and nosql data stores. SIGMOD Rec., 39(4):12–27, May 2011.*

```
"id" : "document-uuid",
"version" : "1.0.0.0",
"create_time" : "2011-11-11T11:11:11Z",
"last_update" : "2012-12-12T12:12:12Z"
},
"type": "UserProfile",
"personal": {
"firstName" : "Alice",
"lastName" : "Matthews",
"date_of_birth": "1901-01-01T01:01:01Z"
},
"financial": {
 "bank" : { ... },
"trading" : { ... },
"credit-history" : { ... }
},
"criminal": {
}
}
```

Атрибутот me е основна информација за записот. Тоа го опфаќа единствен ID на документот кој никогаш не се менува, верзија која мора да се ажурира секој пат кога записот се менува, означување на creation_time кога на записот е создаден, а last_update покажува кога записот бил последен ажуриран. Ова може да биде задолжително за разумност.

Атрибутот тип го одредува типот на ентитет претставен во даден документ. Ова, повторно, може да биде задолжително.

Други атрибути како што се лични, финансиски, криминални и уште неколку можат да се додадат со текот на времето.

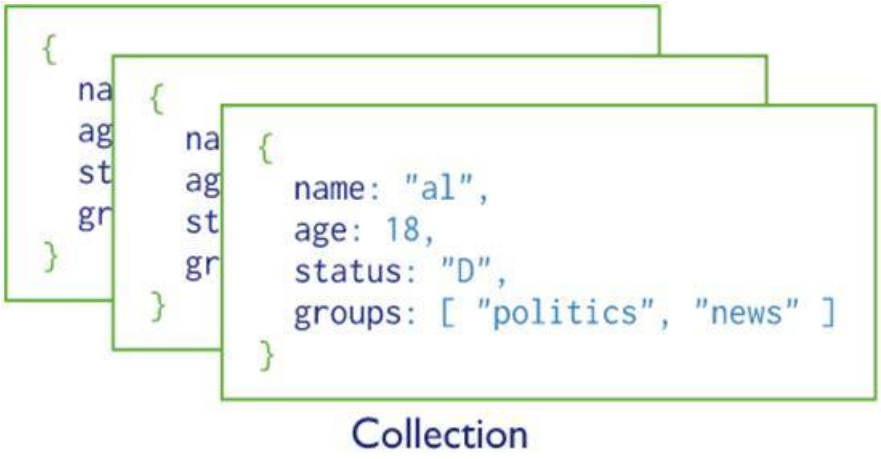
Токму овие атрибути се нарекуваат проекции кои обезбедуваат специфичен контекст податоци.

Овие контексти не треба првично да се дефинираат и генерално да се развиваат прекувремено. [20]Предноста, како што се гледа, е што сите податоци поврзани со ентитетот се наоѓаат во еден запис(документ) и вишокот може да помогне со забрзување на прашањата.

Базите на податоци како MongoDB овозможуваат создавање на ентитети без шема, така што може да се ослободи корисниците од атрибутот "тип" и да ги поддржува погледите на кои може да се користи пребарување низ различни типови на ентитети слично на она што го прави JOIN во SQL.

MongoDB складира документи во колекции. Колекциите се аналогни на табелите во релационите бази на податоци.

[20] *F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber. Bigtable: a distributed storage system for structured data. In Proceedings of the 7th symposium on Operating systems design and implementation, OSDI '06, pages 205–218, Berkeley, CA, USA, 2006. USENIX Association.*



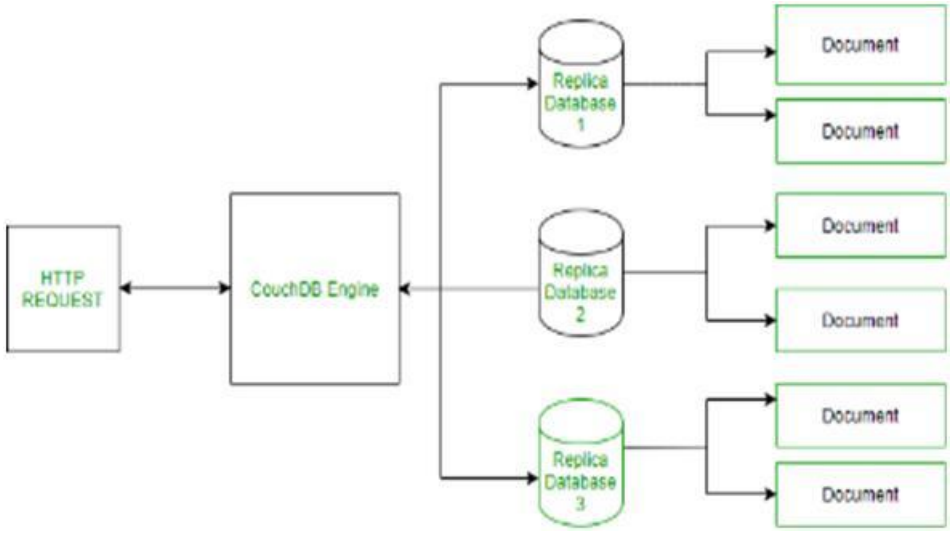
Слика 30 Пример за MongoDB

Извор: Google

Следниот пример ја покажува употребата на JSON со CouchDB и како тие концепти може да се применат. Бидејќи CouchDB нема концепт за табели и сè што ќе се додаде е едноставен, неструктуриран, но легален JSON документ, кој го чува документот како операции кои може да се извршат на податоците.[21]

CouchDB обезбедува RESTful HTTP интерфејс со стандардните HTTP методи мапирање на операциите со податоци - GET (превземи), POST (создавање или ажурирање, имплицитно/имплициран ID), PUT (создади или ажурира, експлицитно/одреден ID), DELETE (бришење). Да се претпостави дека серверот CouchDB HTTP е вклучен и работи на локален хост стандардно порта 5984. Ќе ги истражуваме операциите на CRUD заедно со основните операции на базата на податоци. За олеснување на операции, ја користиме командната линија програма curl (http://curl.haxx.se/docs/ manpage.html) за извршување на барањата HTTP.

[21] *NoSQL with MongoDB in 24 Hours Brad Dayley*



Слика 31 CouchDB пример

Извор: Google

Како CouchDB го користи ID, ревизија и лабавост во шемата:

1) Создавање база на податоци со име ShoppingDB: curl –X PUT http://localhost:5984/ShoppingDB

Ќе се добие следниот одговор:

{ "ok": true }

2) Се добива резиме на базата на податоци:

curl http://localhost:5984/ShoppingDB

{

"db_name" : "ShoppingDB"

"doc_count": 0

// Отстранети се другите атрибути за краткотрајност.

}

3) Според претпоставка дека содржината е зачувана во датотеката data.json, со додавање на документ до складот користејќи документ-uuid како лична карта како што е наведено во документот. Технички гледано, документот-uuid може да биде било кој единствен идентификатор - едноставно 123.
   Важно е да се напомене дека, ако не е обезбедено, MongoDB автоматски генерира ID за секој вметнат документ. Името на полето е _id.

curl–XPUT–H"Content-Type:application/json"\
http://localhost:5984/ShoppingDB/document-uuid \

-d @data.json

Ќе се добие следниот одговор:

{

"ok": true,

"id": "document-uuid",

"rev": "1-Revision-UUID"

}

4) Враќање на документот:

curl http://localhost:5984/ShoppingDB/document-uuid

Ќе се добие следниот одговор:

{

"_id": "document-uuid",

"_rev": "1-Revision-UUID",

"me": {

"id": "document-uuid",

}

}

5) Ажурирање на документот. Ако има известување дека ревизијата треба да се ажурира потребно е истата да биде ажурирана:

curl –X PUT –H "Content-Type: text/json" \

http://localhost:5984/ShoppingDB/document-uuid

-d '{ "name": "Alice Taylor", "_rev": "1-Revision-UUID" }'

Ќе се добие следниот одговор:

{

"ok": true,

"id": "document-uuid",

"rev": "2-Revision-UUID"

}

6) Бришењето на документот е едноставно како и извршувањето на методот DELETE на ID на документ:

curl –X DELETE http://localhost:5984/ShoppingDB/document-uuid

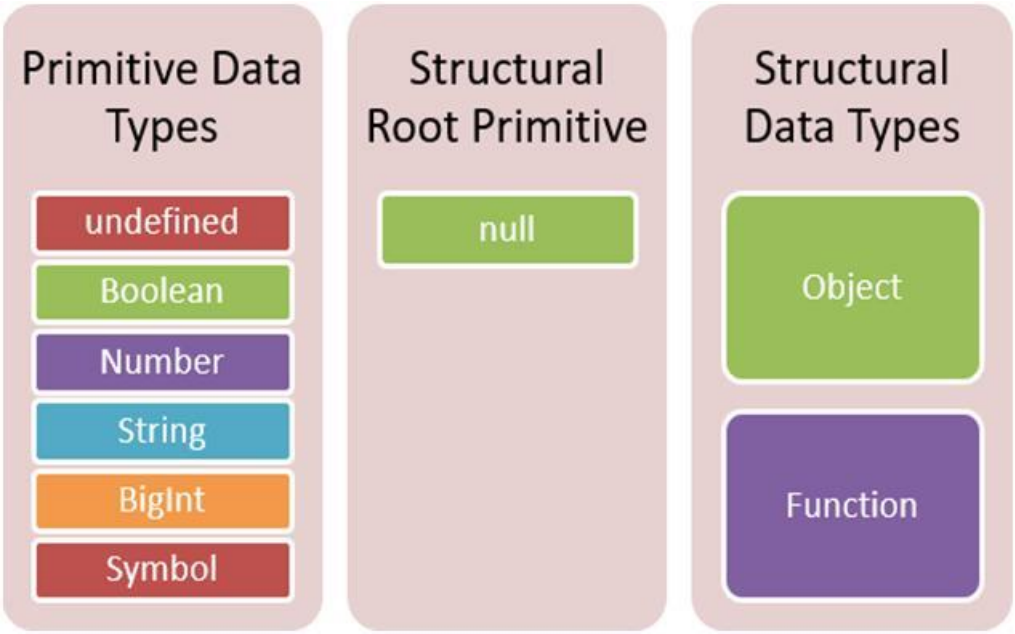
Ќе се добие следниот одговор:

{ "ok": true }

7) Тоа се едноставни операции и изгледаат главно тривијално, на пример, треба да се изврши хипотетичко барање за преземање на сите документи низ базата на податоци што имаат наслов или експертиза на атрибут што содржи NoSQL без грижа за капитализацијата. Насловот треба да биде едноставна низа, а експертизата е низа во документот JSON.

Профилите може да ја имаат нивната техничка експертиза наведена, или таму може да биде компанија чие име ја содржи низата.

Ова е прилично застрашувачка задача во секоја база на податоци што има поим за табели затоа што при пребарување низ табели и при презентирање во унифициран начин е речиси невозможно. Сепак теоретски или хипотетички ова барање може да биде, Тоа ја покажува моќта зад таквата имплементација на базата на податоци NoSQL. CouchDB поддржува прегледи и излезот од приказот е исто така JSON документ со јазик за спроведување на логиката за дефинирање на погледот е JavaScript и функциите се нарекуваат функции за мапирање и намалување (http://en.wikipedia.org/wiki/Mapreduce).



Слика 32 Пример за JavaScript типови

Извор : Google

Функцијата зема документ (JSON објект) како параметар и емитува JSON објект што го претставува излезот од приказот и единствена идентификација за идентификација на записот (може да биде кој било валиден JavaScript тип).

Функционирањето на мапата ја претставува вредноста што ќе ја намали потрошената функција (се користи ID на документот како ID на вратениот запис):

```
function(doc) {
var pattern = /nosql/i;
if(pattern.test(doc.title)) {
emit(doc['_id'], doc);
} else if(doc.expertise && doc.expertise.length) {
pattern = /,nosql,/i;
if(pattern.test(',' + doc.expertise.join(',') + ',')) {
emit(doc['_id'], doc);
```

}

}

}

Функцијата за намалување ги дава конечните податоци што можат да се потрошат во бизнис- слој на апликација. Следниве функции се за намалување:

- Го враќа целосниот документ:

```
function(key, value, rereduce) {
return value;
}
```

- Го враќа атрибутот id и адреса од документот:

```
function(key, value, rereduce) {
return {
"_id": value["_id"],
"address": value["address"]
};
}
```

- Ги враќа сите атрибути освен атрибутот адреса:

```
function(key, value, rereduce) {
delete value.address;
return value;
}
```

Следниот пример покажува користење на JSON со MongoDB што овозможува сегрегација на записи користејќи го поимот збирки (слично на табелите во SQL). Во MongoDB, не треба да се креира база на податоци пред да се изврши вметнувањето на податоците.

MongoDB не бара да се креира база на податоци.Кога нешто се вметнува , MongoDB ја создава основната колекција (слично на базите на податоци во светот на RDBMS) и базата на податоци. Колекција која не постои, MongoDB ја третира како празна колекција (корисникот никогаш не добива грешки).

Бидејќи MongoDB не обезбедува RESTful интерфејс преку HTTP надвор од кутијата, препорачливо е да се изврши кодот на конзолата MongoDB.

-Иако не е задолжително, постојат повеќе начини како да се креира колекција:

```
> db.createCollection("userprofile");
```

-Вметнување запис во колекцијата на кориснички профили: Се забележува дека документот не го содржи типот атрибут бидејќи MongoDB го поддржува поимот на колекција.

```
> db.userprofile.insert({
"me": {
"id": "document-uuid",
...
},
"personal": {
"firstName": "Alice",
...
}
});
```

Додека се работи со MongoDB, секогаш е добра идеја да се има идентификациски број во самиот документ.

-За да се ажурира запис, се користи методот за ажурирање што очекува два аргументи. Првиот аргумент е барање за филтрирање на записот што треба да се ажурира. Вториот аргумент дава детали за ажурираните вредности. Овие чекори се слични на онаа во SQL за изјава за UPDATE каде што првиот аргумент е сличен на клаузулата WHERE и вториот аргумент е сличен на клаузулата SET:

```
> db.userprofile.update( {
"me.id": "uuid-to-search-for"
}, {
"$set": {
"personal.lastName": "Taylor"
}
}
);
```

MongoDB ја користи точката (.) како напречен раздвојувач во рамките на објектот,затоа препорачливо е да не се користи точка во имињата на атрибутите. “ firstName ” треба да се смета како одобрен атрибут додека “ Name.first ” како име на атрибут што треба да се избегнува.[22]

-За да се вратат записите, може да користи кој било од атрибутите:

```
> db.userprofile.find( {
"personal.firstName": "Bob"
}
);
```

[22] *NoSQL For Dummies Adam Fowler*

-За да се избриши еден или повеќе од еден запис, се што е потребно е да се повика еден соодветен атрибут или повеќе атрибути. [23]На пример, за отстранување на профили каде возраста е поголема од 30 години, а градот на нивната лична адреса е Мадрид:

```
> db.userprofile.remove({
"personal.age":{ $gt: 30 },
"personal.address.city": "Madrid"
}
);
```

-Откако ќе се разгледаат некои од дадените основни операции треба да се разгледаат и некои реални сценарија за пребарување на податоците, конфигурирање на специфични индекси и враќање на делумен документ.



Customers

CustomerID	FirstName	LastName	Address	City	State	PostalCode	PhoneNumber
1111	Michael	Jordan	123 Anywhere Rd	Wilmington	NC	28401	(910) 555-5555
2222	Larry	Bird	456 Somewhere St	French Lick	IN	47432	(812) 555-5555
3333	Irvin	Johnson	789 Nowhere Ave	Los Angeles	CA	90001	(213) 555-5555

Products

ProductID	ProductName	Category	Cost
00001	Basketball	Sporting Goods	$25.00
00002	Shorts	Apparel	$20.00
00003	Shoes	Apparel	$75.00

Orders

OrderNo	CustomerID	ProductID	OrderDate
1	1111	00002	9 /18/2012
2	2222	00001	9 /18/2012
3	3333	00003	9 /18/2012
4	1111	00001	9 /20/2012

Слика 33 Пример за ShoppingDB

Извор : Google

[23] *Beginning Database Programming Using ASP.NET Core 3 Bipin Joshi*

Колекцијата - кориснички профил – е само збирка документи која може безбедно да се преименува во ShoppingDB, инкорпорирање на поголемиот опсег кој е прикажан во следните примери:

```
>       db.shoppingDB.find( {
$or: [
{ "title": /NoSql/i },
{ "expertise": /NoSql/i }
]
}
);
```

Профили кои имаат продажба на низа од објекти кои одговараат на месечна продажба од времето кога одреден корисник се приклучил на одредена компанија со даден објект има информации за месецот и фигуративната продажба. Она што е прикажано е проверката на сите профили кои ја надминуваат продажбата од 500.000 во нивниот прв месец од приклучувањето:

```
{
"me": { ... },
"sales": [ { "month": 201201", "value": 100000 }, ...]
}
>       db.shoppingDB.find({
 "sales.0.value >= ": 500000
}
);
```

Во овај случај, се избира само personal.firstName and sales:

```
>       db.shoppingDB.find( { ... }, {
"personal.firstName": 1,
"sales": 1
}
);
```

Во следниот случај, се избираат сите полиња освен оние со криминално досие:

```
>       db.shoppingDB.find( { ... }, {
 "criminal": 0
}
);
```

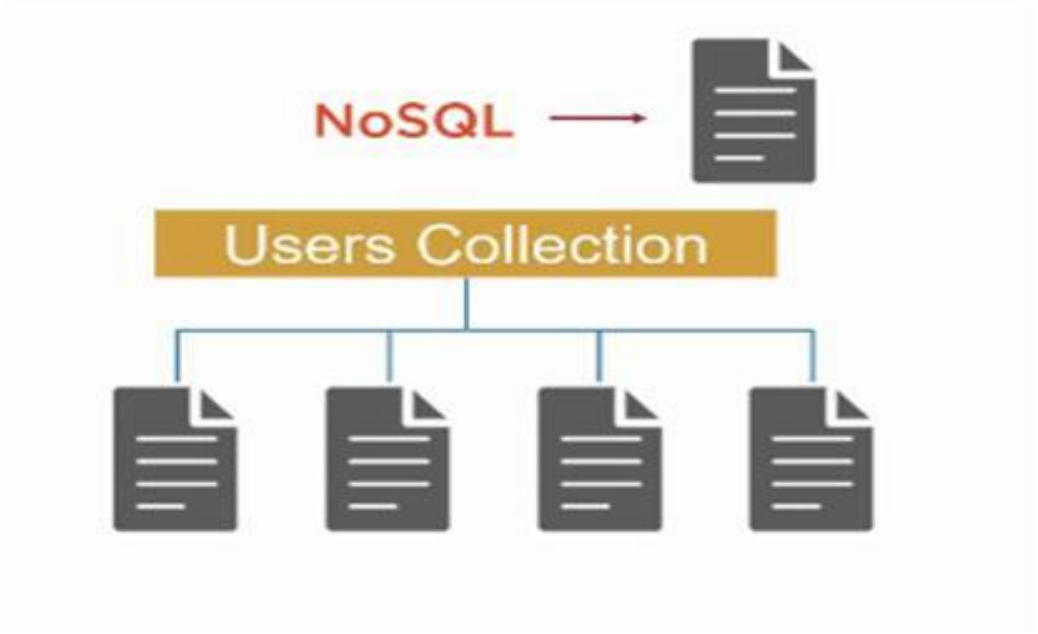
Во последниот случај, се избираат само последните пет продажни елементи низ сите документи.

```
>       db.shoppingDB.find( { }, {
"sales": { $slice: -5 }
}
);
```

Забележително е дека иако MongoDB поддржува колекции, тоа не спроведува ограничувања на шемата. Ова во суштина значи дека MongoDB е слично на тоа да има повеќе примероци на CouchDB кои работат под еден шешир.

4.1 Document store

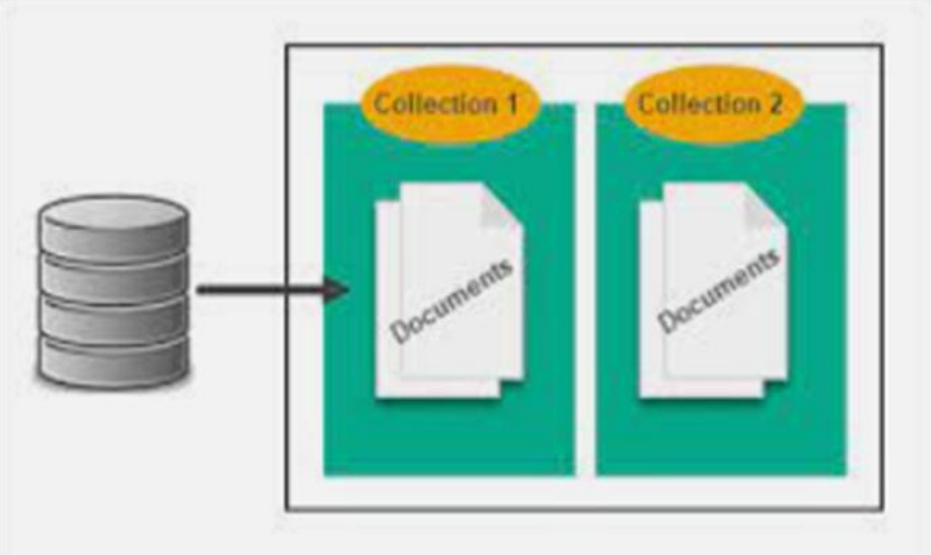
Во складот за документи, податоците се чуваат во т.н документи. Терминот документи се однесува на произволни податоци во некој формат на структурирани податоци. Примери за користени форми се JSON, BSON или XML. Додека типот на формат на податоци обично го определува складот за документи, структурата е флексибилна. Во складот за документи базирана на JSON, на пример, документи со сосема различна група на атрибутите може да се складираат заедно, се додека се валидни JSON документите. [24]Избраниот формат на податоци може да се толкува преку системот за складирање. Ова му овозможува на системот да понуди ситно читање и пишување операции за својствата на зачуваниот документ, спротивно на системот Key-Value. Таму каде што се чува вредноста типично не е дополнително разбирлива од системот воведена од MongoDB , претставник на класа на document store.



Слика 34 NoSQL Document store

Извор : Google

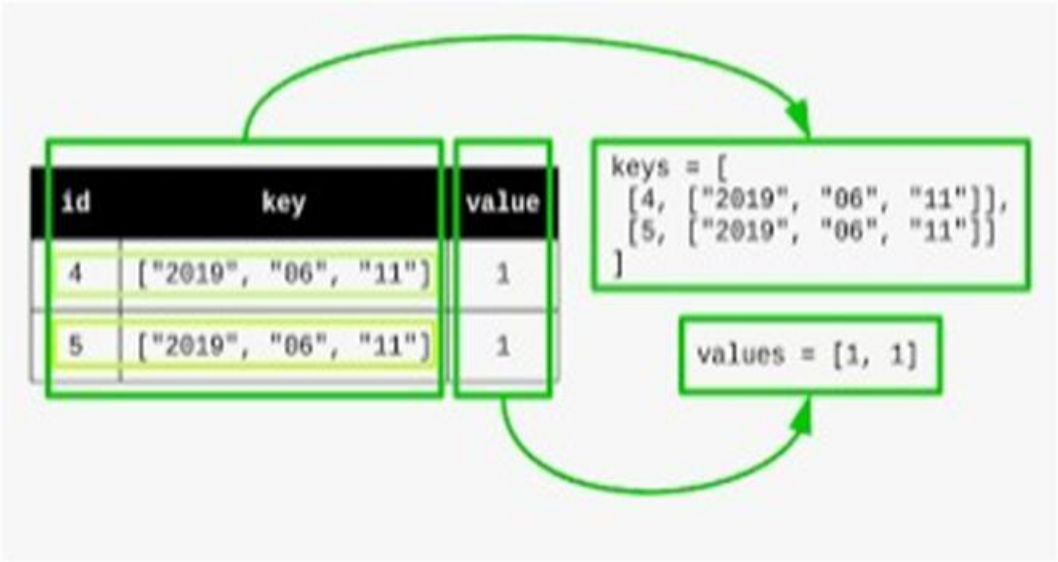
[24] *NoSQL A Complete Guide, The Art of Service - NoSQL Publishing*



Слика 35 Колекција на Document store

Извор : Google

Друг широко користен систем за складирање на оваа категорија е CouchDB, кој многу се потпира на рамката MapReduce за барање податоци.



Слика 36 Рамката MapReduce со системот за складирање CouchDB

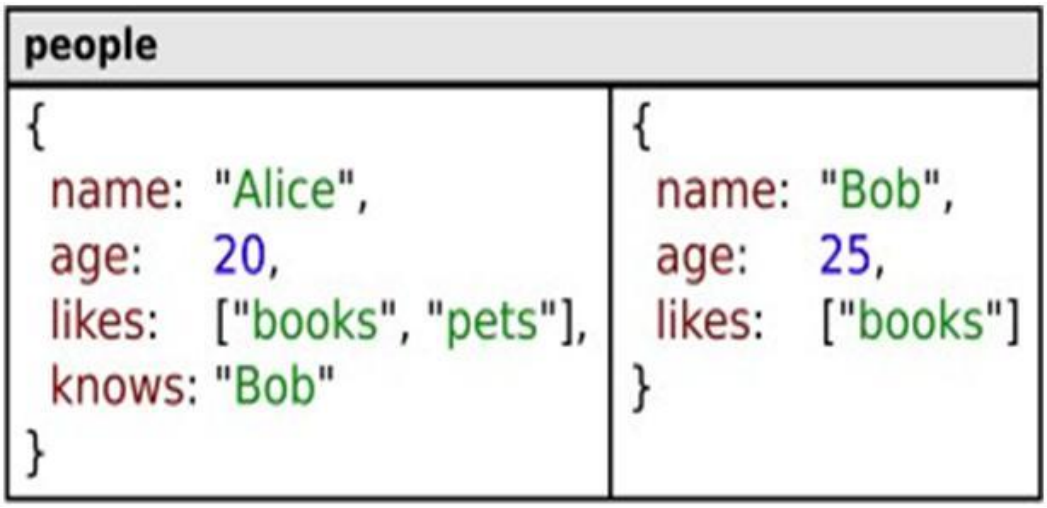
Извор : Google



Слика 37 Пример шема на GeoJSON

Извор : Google

Складирањето на документи е слично на складирањето со клучна вредност со една разлика: бара неговата складирана вредност (наречена документ) да биде структурирана и кодирана со метаподатоци. Вообичаените шифрирања вклучуваат XML, JSON, BSON, а за просторни податоци, GeoJSON е добро прифатен од многу NoSQL бази на податоци за складирање документи. MongoDB и Apache CouchDB се примери за оваа категорија.

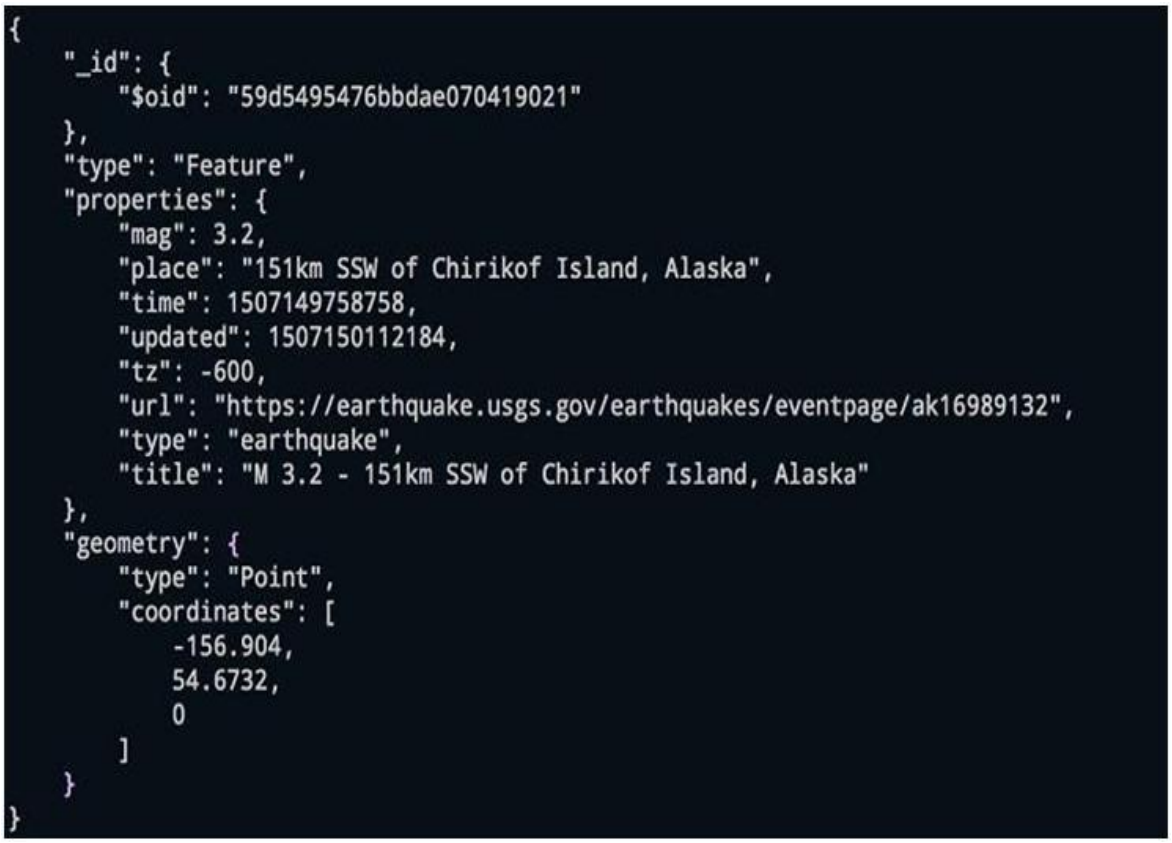


Слика 38 Пример податоци претставени во склад за документи.

Извор : Google

На слика 38 JSON се користи како формат на податоци. Бидејќи системот е разбирлив за форматот, директни прашања за атрибути (на пр. “име” или ”воздраст”) се множат.[25]

Сликата подолу прикажува настан од земјотрес зачуван во MongoDB како документ. Документот е во формат GeoJSON со својства на земјотресот и неговата геометрија.



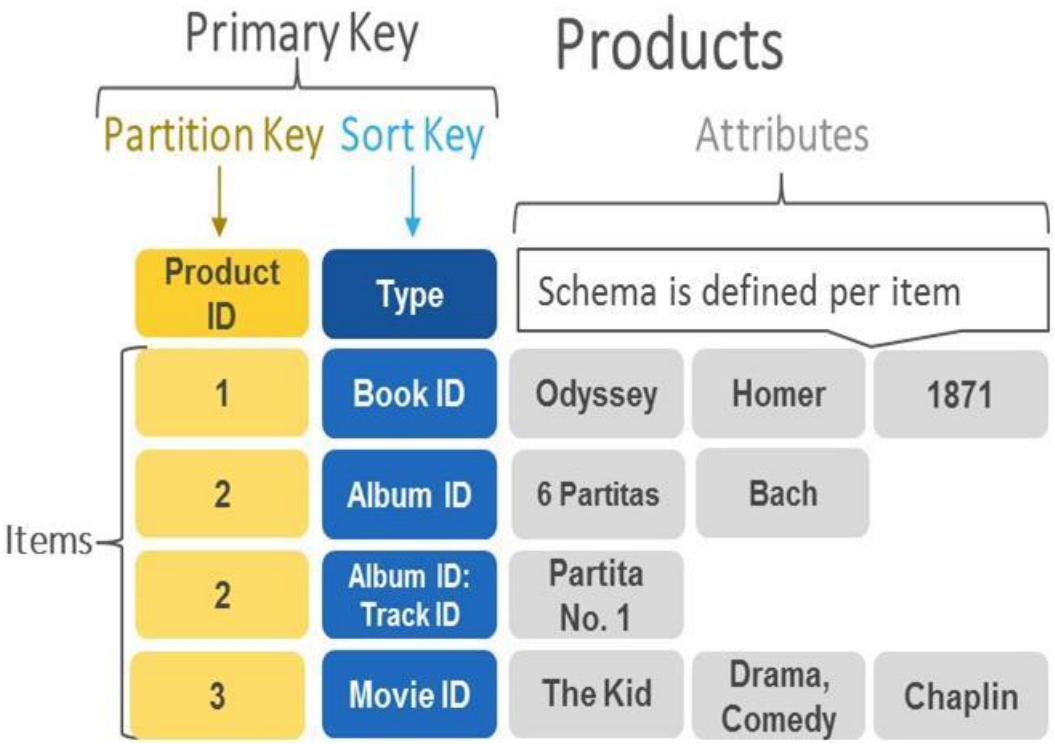
Слика 39 Пример за настан зачуван во базата на податоци MongoDB.

Извор : Google

[25] *Making Sense of NoSQL Dan McCreary , Ann Kelly*

5 KEY-VALUE STORE

Key-value store е многу тесно поврзан со Document store кој овозможува складирање на клучна вредност кој нема потреба од шема за да да се наметне на вредноста. Сепак, постојат неколку ограничувања што се наметнуваат со складот за клучна вредност (Key-value).



Слика 40 Дијаграм кој покажува пример на податоци зачувани како парови со клучни вредности(Key-value) во DynamoDB.

Извор : Google

-За разлика од складот за документи (Document store) кој може да креира клуч кога е вметнат нов документ, складиштето со клучна вредност (Key-value store) бара клучот да биде наведен.

-За разлика од Document store каде што вредноста може да се индексира и да се бара, за да се зачува клучна вредност, вредноста е матна и како таква, клучот мора да му биде познат.[26]

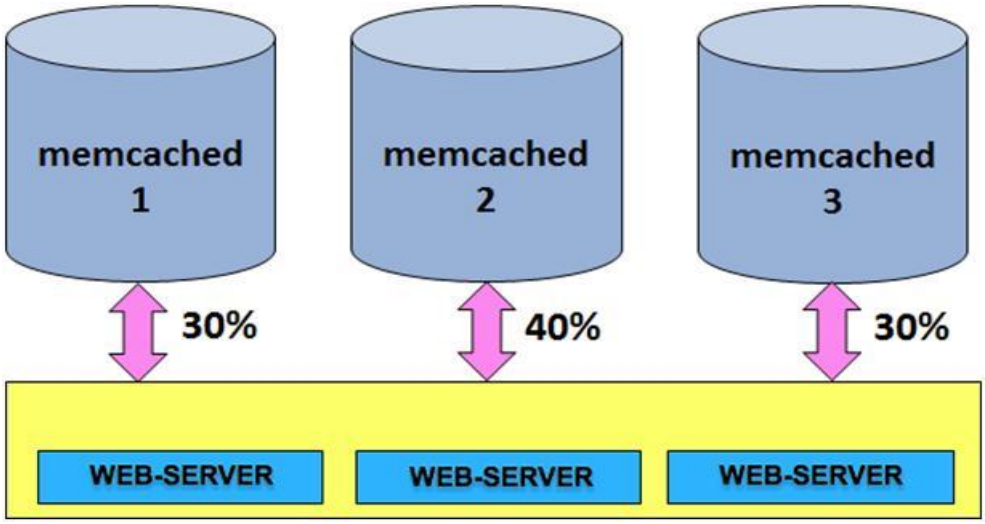
Најистакнатата употреба на работа со складот на документи со клучна вредност (Key-value store) е во меморијата дистрибуирана или на друг начин во cache. Сепак, имплементациите постојат за да обезбедат постојано складирање.

Неколку од популарните складови за клучни вредности се:

- Redis (In memory-во меморијата, со упорност на dump или command-log)
- Memcached (in-memory-во меморијата)
- MemcacheDB (направена според Memcached)
- Berkley DB
- Voldemort (Имплементација со отворен код на Amazon Dynamo)

[26] *J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. Commun. ACM, 51(1):107–113, Jan. 2008.*

Предности- Складовите со клучна вредност (Key-value store) се оптимизирани за барање во однос на клучевите. Како такви, тие одлично служат за cache(кеш) во меморијата. Memcached и Redis (Remote Dictionary Server) поддржуваат истекување на клучот (опаѓачки или апсолутно истекување) по што влезот е исфрлен од складот.[27]

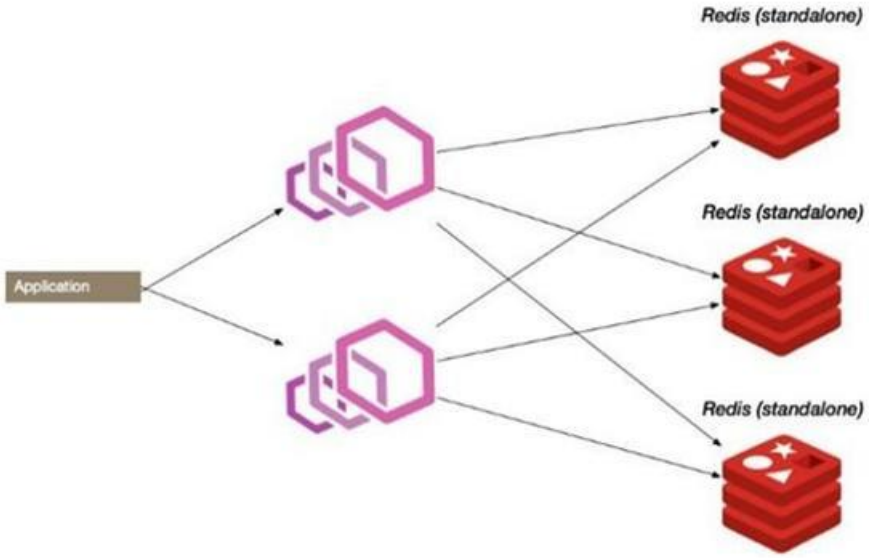


Слика 41 Пример за Memcached

Извор : Google

Понекогаш, потребно е да се генерираат клучевите - UUID во корпа - и може да се побара наспроти опсегот на клучеви. На пример, Redis (Remote Dictionary Server) овозможува враќање на списокот со сите клучеви со појавување на (glob-style schema) шема во стил на глоб.

[27] *D. Molkov, A. Menon, S. Rash, R. Schmidt, and Aiyer. Apache hadoop goes realtime at facebook.In Proceedings of the 2011 ACM SIGMOD International Conference on Management of data, SIGMOD '11, pages 1071–1080, New York, NY, USA, 2011. ACM.*



Слика 42 Пример за Redis (Remote Dictionary Server)

Извор : Google

Redis(Remote Dictionary Server) работи на влезно ниво, а лаптопот може да скенира база на податоци од 1 милион клучеви за 40 милисекунди. [28]

Иако зачувувањата на клучните вредности не можат да ги бараат вредностите, тие сепак можат да го разберат типот на вредноста. Складови како Redis поддржуваат различни типови на вредности - низи, хашови, списоци, множества и подредени множества. Врз основа на типовите на вредност, напредните функционалности можат да бидат обезбедени. Некои од нив вклучуваат атомско зголемување, поставување/ажурирање на повеќе полиња на хаш (еквивалент на делумно ажурирање на документот), и пресек, унија, и разлика при работа со множества.

[28] http://redis.io/commands/keys

Примери- Основни операции со податоци користејќи го Redis (Remote Dictionary Server) :

Поставување или ажурирање на вредноста во однос на клучот:

SET company "My Company" //String

HSET alice firstName "Alice" //Hash – set field value

HSET alice lastName "Matthews" //Hash – set field value

LPUSH "alice:sales" "10" "20" //List create/append

LSET "alice:sales" "0" "4" //List update

SADD "alice:friends" "f1" "f2" //Set – create/update

SADD "bob:friends" "f2" "f1" //Set – create/update

Истражување на неколку операции на множества и списоци:

Поставени се следните операции:

//Intersection – Прикажи ги заедничките пријатели на Алис и Боб

SINTER "alice:friends" "bob:friends"

//Difference – Пријателите од Алис ги нема кај пријателите на Боб

SDIFF "alice:friends" "bob:friends"

//Union – Сите пријатели на кои им е потребна покана

SUNION "alice:friends" "bob:friends"

Листа на операции:

// Поставете ја првата ставка или вратете ја нула

POP "key:name"

//Blocking pop – Пропуштете ја првата ставка, почекајте ја следната или почекајте до истекот на времето

is available (check across lists – l1, l2, l3)

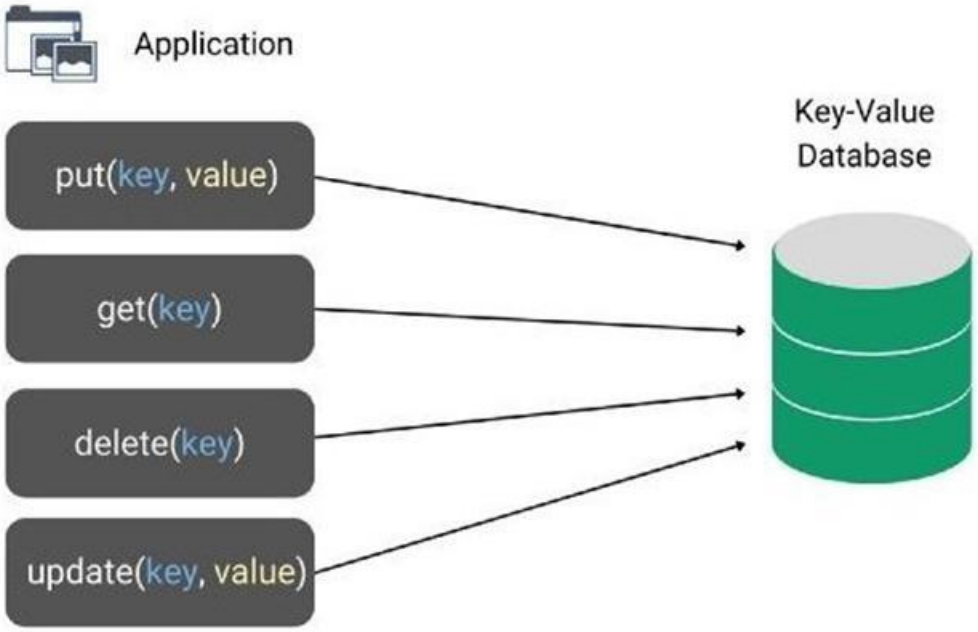
BLPOP l1 l2 l3

// Избришете ја ставката од списокот1, додајте ја на листата2 и вратете ја вредноста RPOPLPUSH list1 list2

Складовите за клучна вредност не се дизајнирани за апликации на кои им треба индекси на вредностите. Поради оптимизација на клучните прашања, имплементациите како Memcached или Redis се одлични кандидати за дистрибуирана, скалабилна, кеш во меморијата.

5.1 Key-Value

Системите за складирање базирани на клучна вредност во основа се асоцијативни низи, составени од клучеви и вредности. Секој клуч мора да биде единствен, а да обезбеди недвосмислена идентификација на вредностите. Додека клучевите се претежно едноставни објекти , вредностите можат да бидат листи, поставува или пак кеш, овозможувајќи посложени податочни структури.[29]



Слика 43 Системски дизајн за Key-value база на податоци

Извор : Google

Key-value е најинтуитивниот склад за податоци на NoSQL. Секоја податочна ставка во базата на податоци е зачувана како пар клучна вредност, сличен на конвенционален речник. Клучот е типично единствен ID што укажува на податоците со кои е поврзан. Можните операции за користење со клучна вредност вклучуваат добивање на вредноста за клучот (get), ставање или доделување вредност за клуч (put) и бришење клуч (delete).

[29] *R. Cattell. Scalable sql and nosql data stores. SIGMOD Rec., 39(4):12–27, May 2011.*

Вредноста може да биде кој било објект како стринг, број, датум, низа, JSON итн., така што системот е без шема. Апликацијата е одговорна да го разбере типот на објектот и соодветно да го анализира.[30]

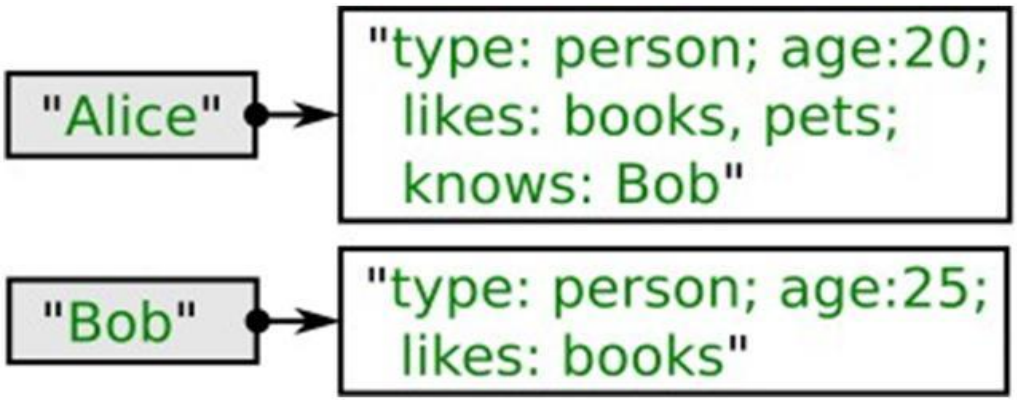
Пример за складирање на клучни вредности е прикажан во Табела подолу. Се забележува дека клучот за адреса е поврзан со вредност JSON која содржи три атрибути: улица, град и држава. Во релациона база на податоци, таквиот склад би бил неважечки.

Key	Value
ID	"a78d1238dfedhz"
Building	"COOR Hall"
Address	{street: "976 S Forest Mall", city: "Tempe", state: "Arizona" }

Слика 44 Пример за склад на податоци со клучна вредност

Извор : Google

[30] *S. Gilbert and N. Lynch. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. SIGACT News, 33(2):51–59, June 2002.*



Слика 45 Пример податоци претставени во склад со клучна вредност (Key-value store).

Извор : Google

Зачуваната вредност (од типот String) обично не може да биде интерпретирана од системот за складирање.

Типични операции што ги нудат складовите на Key-Value се тие кои им се познати на програмерите од иплементациите на Hash-Table:

- ENTER нови парови со клучна вредност
- Вредности LOOKUP за одреден клуч
- Клучот DELETE и вредноста поврзана со него

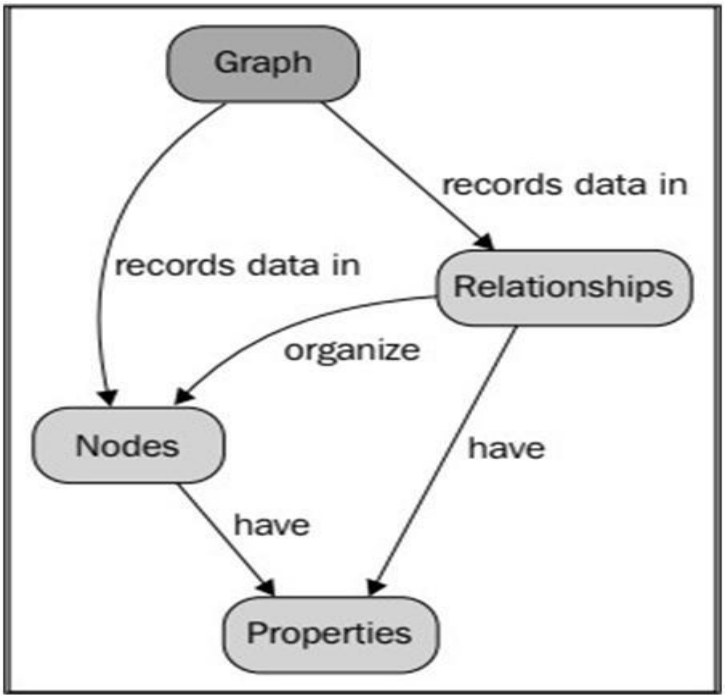
Едноставниот модел обезбеден од складот на податоци Key-Value го дозволува тоа да работи многу брзо и ефикасно.[31] Цената на ова е со намалена комплексност со која е намалена и флексибилноста на можностите за барање.

Познат претставник на Key Value store е Amazon Dynamo, Dynamo нуди ефикасна и високо скалабилно решение за складирање, по цена на многу ограничен интерфејс за барање.

[31] *J. Gray. The transaction concept: virtues and limitations (invited paper). In Proceedings of the seventh international conference on Very Large Data Bases - Volume 7, VLDB '81, pages 144–154. VLDB Endowment, 1981.*

6 GRAPH STORE

Графиконските бази на податоци претставуваат посебна категорија на NoSQL бази на податоци каде односите се претставени како графикони. Може да има повеќе врски помеѓу две јазли во графиконот - претставувајќи ги повеќекратните врски што ги споделуваат двата јазли. Претставените односи може да вклучуваат општествени односи меѓу луѓето, транспортни врски помеѓу места или мрежни топологии помеѓу поврзаните системи.

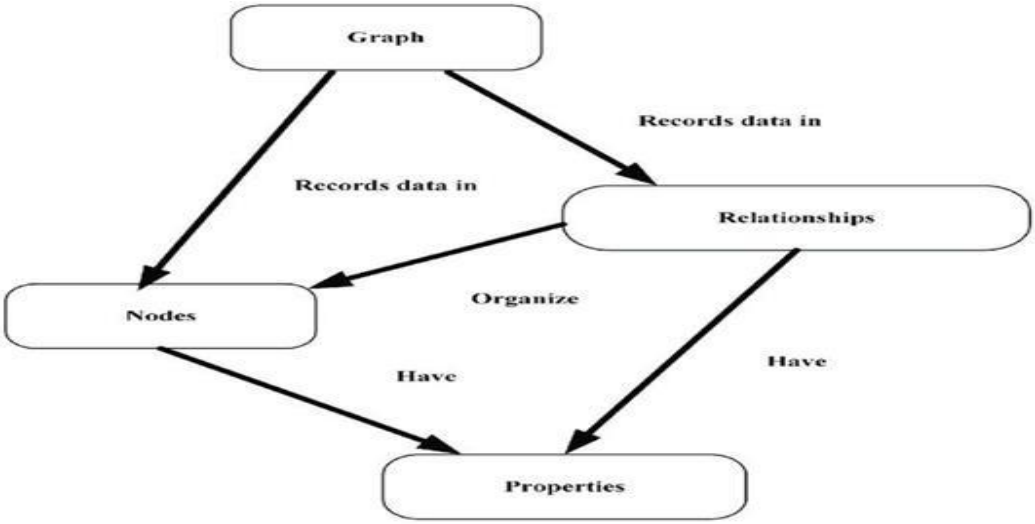


Слика 46 Графичкото прикажување на графикон

Извор : Google

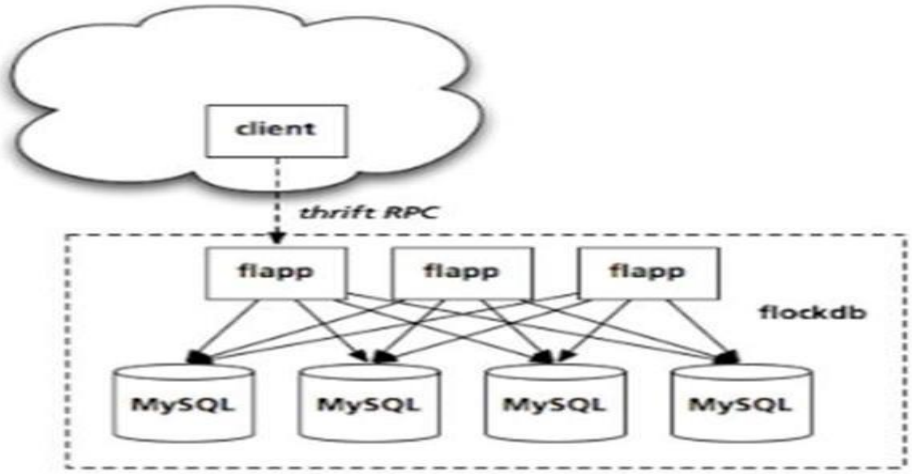
Базите на податоци за графикони се прилично нови на пазарот со само неколку докажани решенија:

- Neo4j
- FlockDB (from Twitter)



Слика 47 Пример за структура на податоци Neo4j

Извор : Google



Слика 48 Пример за FlockDB graph database

Извор : Google

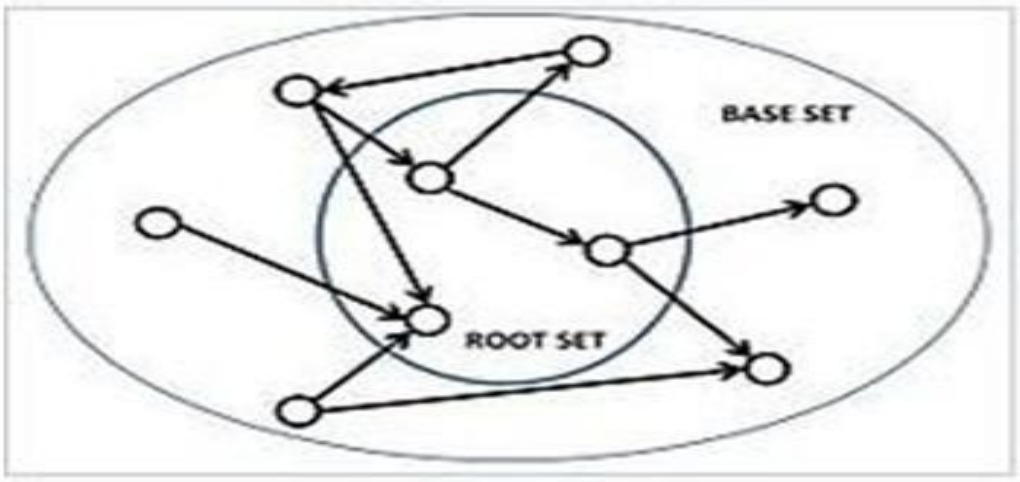
Предности- Теоријата на графикони има голема корист и релевантност во многу проблеми низ целиот свет со различни домени. [32]Најприменетите теоретски алгоритми за графикони вклучуваат различни типови пресметки на најкратки патеки, геодетски патеки, мерки на централност како PageRank алгоритам, централност на сопствени вектори, блискост, меѓусебност, HITS алгоритам и многу други.

[32] *JSONiq The SQL of NoSQL Ghislain Fourny, Matthias Brantner, Federico Cavalieri*



Слика 49 Page Rank algorithm

Извор : Google



Слика 50 HITS algorithm

Извор : Google

Базите на податоци со графикони може да се сметаат за оптимизирани бази на податоци за NoSQL за посебна намена за податоци тешки за релација. Ако нема врска меѓу субјектите, тогаш нема потреба ни од употреба за бази на податоци со графикони. Една од предностите што ја имаат базите на податоци со графикони е лесното претставување, пронаоѓање и манипулација со односите меѓу субјектите во системот. Не е невообичаено да се складираат податоци во Document store и Graph database links.

Примери- Создавање едноставна врска помеѓу два јазли со Neo4J:

```
// Да претпоставиме дека некако ја добиваме основната услуга за база на податоци
GraphDatabaseService db = ...
Node node1 = db.createNode();
node1.setProperty("documentId", "alice");
Node node2 = db.createNode();
node2.setProperty("documentId", "bob");
RelationType friendRel = new RelationType() {
public String name() { return "friend"; }
};
Relationship reln = node1.createRelationshipTo(node2,
friendRel);
reln.setProperty("initatedBy", "alice");
reln.setProperty("createdOn", "1980-01-01T07:30+0530");
```

Следниот пример покажува како да се повлечат сите соседни јазли против врска и стожерен јазол:

```
//Дефинирање на индекс
Index<Node> nodeIndex = db.index().forNodes("nodes");
// Додека креирате јазол, додадете го во индексот
nodeIndex.add(node1, "documentId",
(String) node1.getProperty("documentId"));
```

```
//Пребарајте одреден јазол

Node aliceNode = nodeIndex.get("documentId, "alice").single();

//Добијте ги сите пријателски врски

Iterable<Relationship> rels =

aliceNode.getRelationships(friendRel);

// Добијте пријатели од другата страна на врската

for(Relationship r: rels)

{

Node friendNode = r.getEndNode();

assert "bob".equals(friendNode.getProperty("documentId"))
```

Користење на бази на податоци за графикони и за складирање на врските. Не се користат за целосно складирање на податоци;бидејќи перформансите може да се намалат. Neo4j е целосна база на податоци во согласност со ACID .

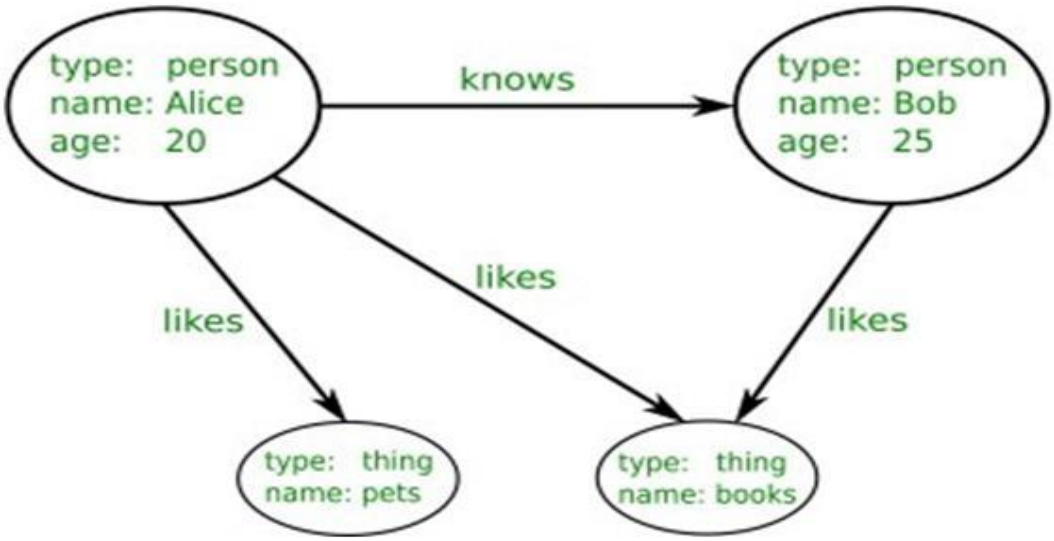
6.1 Graph Store

Како што покажува името, во системите од оваа категорија податоците се претставени со графикони. Базите на податоци за графикони се најпогодни за претставување на податоци со висок, но сепак флексибилен број на меѓусебни врски кои се претставени како важни податоци. Ваквите информации можат да бидат општествени односи или географски податоци.

Базите на податоци за графикони овозможуваат прашања за структурата на графиконот на пр. На односите помеѓу јазлите или најкратките патеки. Имплементациите на базите на податоци со графикони можат ефикасно да ги поддржат таквите барања со користење на добро проучени графички алгоритми.

Складот за графикони (Graph store) се користи за складирање на мрежните податоци што се состојат од јазли (ентитети) и рабови (врски) и својства на јазли и рабови. [33]Пример за мрежни модели вклучува социјални мрежи, мрежи за наука за животот, патни мрежи и ИТ/мрежи, каде што ентитетите се многу поврзани преку врски.

Во релационите бази на податоци, ентитетите се чуваат како табели. Барањето односи меѓу ентитетите се врши со извршување на JOIN на тие табели, операција која е интензивна за пресметување и меморија. Спротивно на тоа, во базата на податоци NoSQL која овозможува складирање на графикони, приоритет се односите. Секој ентитет содржи листа на записи за врски поврзани со други јазли. При извршување на барање за наоѓање на тие врски, базата на податоци ќе ја користи оваа листа за да ги пронајде поврзувачките јазли, со што ќе го намали времето од пребарување и совпаѓање. Достапните бази на податоци за графикони вклучуваат Neo4J, InfiniteGraph и OrientDB.



Слика 51 Пример на податоци претставени како графикон.

Извор : Google

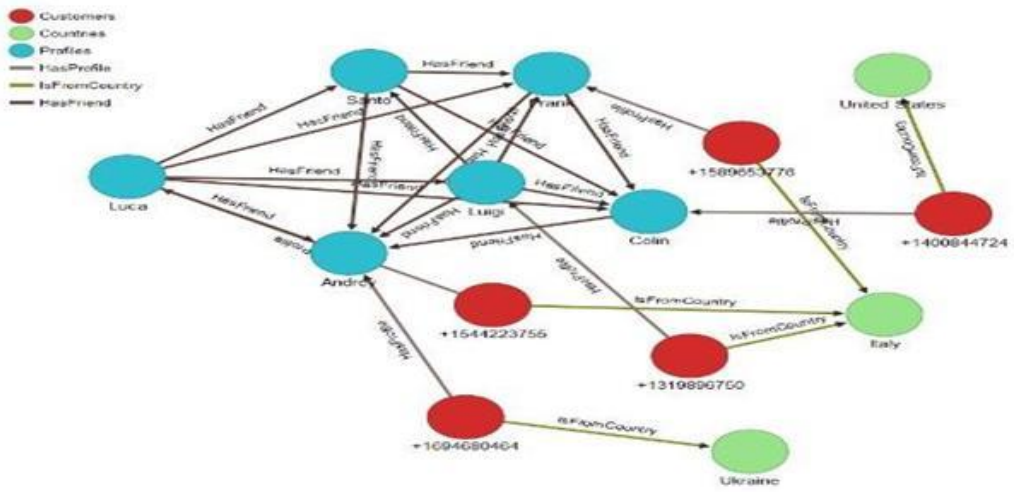
Рабовите можат да се користат за складирање на информации за односите, додека другите атрибути на објектите треба да се складираат во темињата.

[33] *J. Cheng, Y. Ke, and W. Ng. Efficient query processing on graph databases. ACM Trans. Database Syst., 34(1):2:1–2:48, Apr. 2009.*

7 MULTI-STORAGE TYPE DATABASES

Освен базите на податоци, следна е листата на бази на податоци кои поддржуваат повеќе типови складирање, што дава флексибилност:

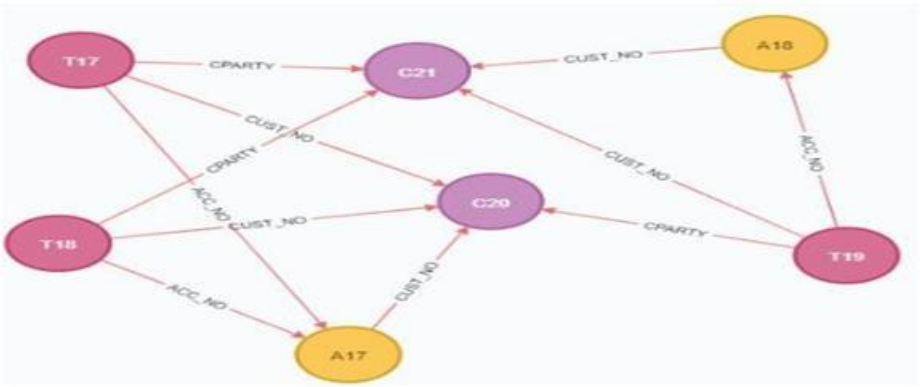
- OrientDB: Поддржува складирање документи со клучна вредност, како и графикони.



Слика 52 OrientDB

Извор : Google

- ArangoDB: Универзална база на податоци со поддршка за складирање документи со клучна вредност и графички модели.



Слика 53 Пример за колекција на рабови во ArangoDB

Извор : Google

- Aerospike: База на податоци која е хибрид помеѓу RDBMS и NoSQL. Поддржува складирање на документи со клучна вредност, графикони, како и RDBMS.

Споредување на моделите- Модели за складирање на податоци, со кои се споредуваат клучните параметри на карактеристиките:

Feature	Column Oriented	Document Store	Key Value Store	Graph
Table-like schema support (columns)	Yes	No	No	Yes
Complete update/ fetch	Yes	Yes	Yes	Yes
Partial update/ fetch	Yes	Yes	Yes	No
Query/Filter on value	Yes	Yes	No	Yes
Aggregates across rows	Yes	No	No	No
Relationships between entities	No	No	No	Yes
Cross-entity view support	No	Yes	No	No
Batch fetch	Yes	Yes	Yes	Yes
Batch update	Yes	Yes	Yes	No

Слика 54 Споредба на моделите за складирање на податоци

Извор : Google

Иако се прикажани со “Yes “ и “No” низ колоните, разликата е помеѓу различните модели кои стануваат заматени со текот на времето. На пример, неколку години наназад со Key-value store би ги разгледале вредностите како матни дамки каде што и денес поддржуваат грануларни барања и ги ажурираат.

Слично на тоа, претходните складишта на документи дозволуваат CRUD операции базирани на ID, но бази на податоци како што е MongoDB имаат поддршка за масовни ажурирања.

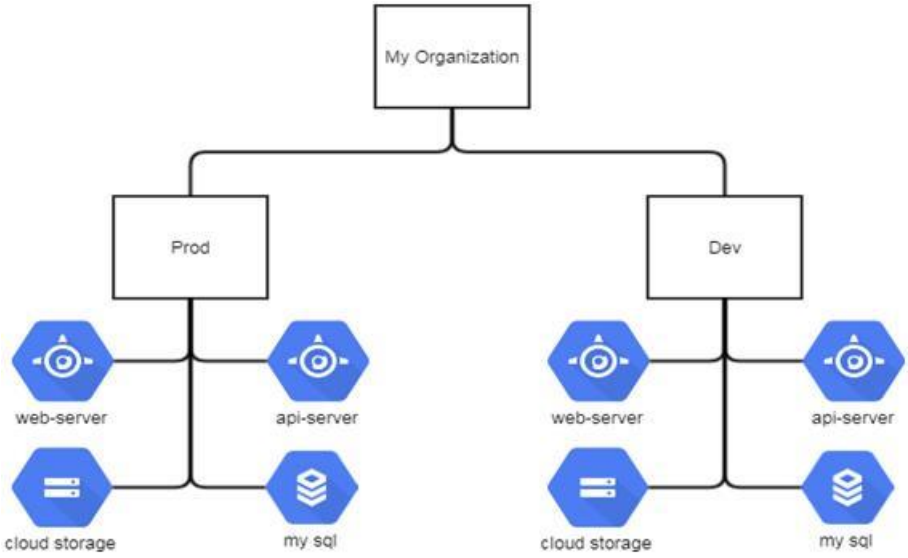
CRUD	HTTP	REST
Create	POST	/api/movie
Read	GET	/api/movie/{id}
Update	PUT	/api/movie
Delete	DELETE	/api/movie/{id}

Слика 55 Пример за основни CRUD операции

Извор : Google

Клучното прашање е - каков модел да се користи?

Во светот на RDBMS, моделирање со базите на податоци ориентирани кон колони може да дадат брзо практично искуство со NoSQL моделирање. Уште еден случај од реалниот свет е кога се има повеќе потреба да се работат збирни вредности во споредба со индивидуалните вредности - на пример, социјално-тешки апликации. Ако се има намера да се работи со Google App Engine , нема друг избор освен да се користи овој модел. Фејсбук користи HBase за да ги напојува нивните пораки и инфраструктура додека Yahoo! го користи истото за складирање на отпечаток од прст од документ за откривање дупликати.



Слика 56 Пример за Google app engine

Извор : Google

Системите за Document store се еден вид што може да се користат низ целиот свет - типови на апликации, вклучително и онаму каде што се има голема потреба од агрегати преку субјектите.

Имплементациите базирани на MapReduce даваат неверојатна контрола над пребарување на податоците како работа со JSON користејќи вгнездени својства и со XML користејќи XQuery; и создавање на структури за одговор како што се создавање агрегати, и приспособени прикази низ типовите на ентитети. Компании како Facebook (HBase), Netflix, Твитер и eBay (Cassandra) дадоа огромен поттик на еволуцијата на документите, тие се складираат и продолжуваат да го водат пакетот со огромна разлика.

Key-value store имаат одлични опции слични на Document store но единствената карактеристика што недостасува е барањето во однос на вредноста. Меѓутоа, со текот на времето, ова се чини дека диференцијацијата е насочена кон заматување. Еден случај каде што се складира клучната вредност Win hands-down се дистрибуира во кешот во меморијата. Широка употреба на Memcached и Redik го докажуваат истото.

Базите на податоци со графикони се особено корисни во дефинирањето на односите меѓу ентитетите на ниво на база на податоци за разлика од другите модели каде што врските се само видливи на ниво на апликација. Твитер има отворен код FlockDB што го користи исклучиво за складирање социјални графикони .Меѓутоа, ако се дизајнира базата на податоци за лесната апликација за купување, пристапот RDBMS е совршено прилагоден за истиот.

7.1 Евентуална конзистентност

Важна разлика помеѓу релационите бази на податоци на NoSQL системите се дадени модели на конзистентност. NoSQL системите треба да ги ублажат дадените гаранции за ACID со релациски трансакции, со цел да се овозможи хоризонтална приспособливост. За да се разбере причината за ова, прво се објаснети три пожелни својства на хоризонтално распределените системи.

Конзистентност- Конзистентноста во дистрибуираниот систем бара вкупен редослед на сите операции низ сите јазли на системот. [34]Ова на пример би било случај ако после успешна операција за пишување, сите последователни операции за читање ја вратат напишаната вредност, без разлика на кој јазол се извршуваат операциите.

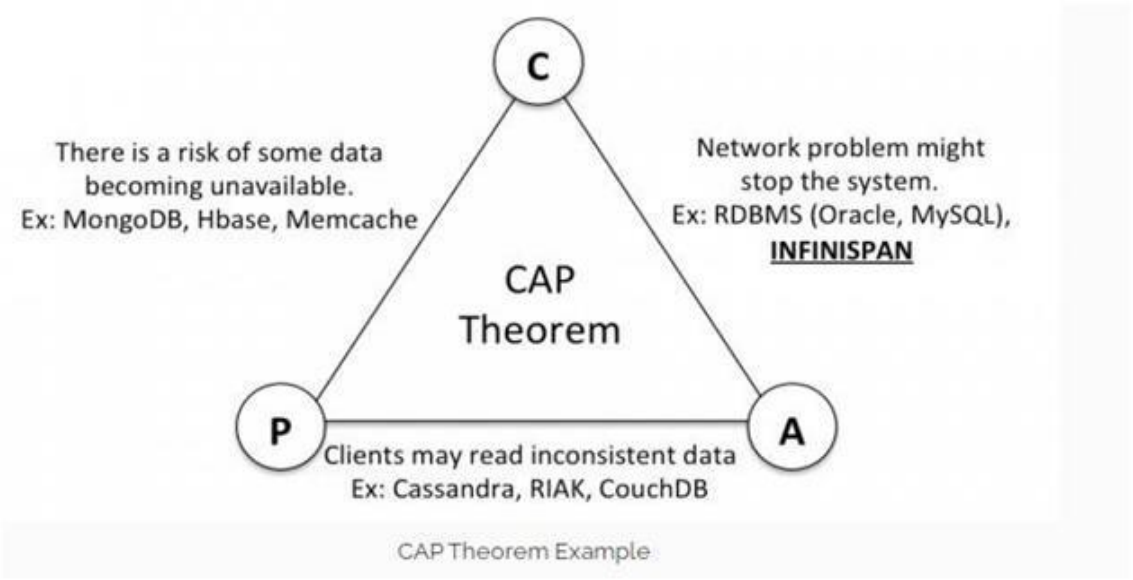
Достапност- Системот ја задоволува достапноста, ако сите операции , извршени на јазол на системот завршуваат со одговор.

[34] *E. F. Codd. Data models in database management.SIGMOD Rec., 11(2):112–114, June 1980.*

Толеранција на партиции- Системот се нарекува поделен, ако постојат најмалку две групи на јазли, така што сите јазли од истото множество можат да комуницираат додека сите пораки испратени помеѓу јазлите од различни множества се губат. Пример за поделен ситем би бил систем каде што добива еден јазол кој е недостапен поради мрежна грешка.

Системот е толерантен ако е достапен и конзистентен иако многу внатрешни пораки се губат.

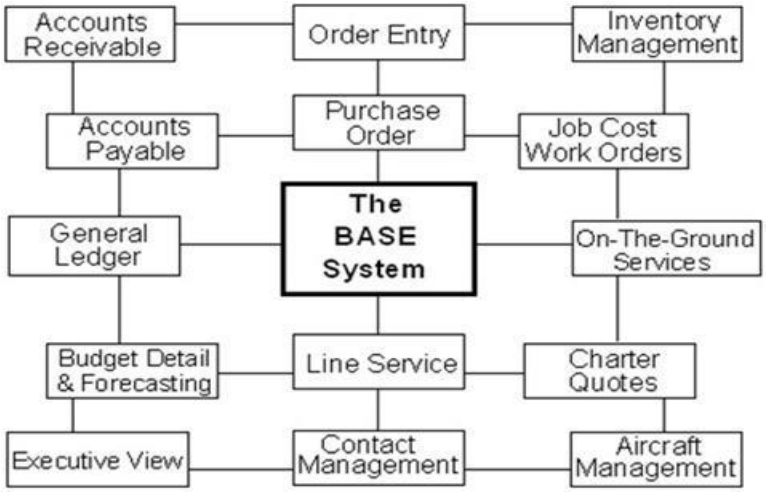
Во 2000 год. Бруер ја изјавил дека претпоставува дека нема веб сервис кој може да ги гарантира сите тие својства (конзистентност, достапност и толеранција на партиции) во исто време. Две години подоцна оваа претпоставка, наведена како CAP theorem или Brewer theorem е формализирана и докажана од Гилберт и Линч.



Слика 57 Пример за CAP theorem

Извор : Google

Во пракса тоа значи дистрибуираните бази на податоци мора да загубат едно од тие својства. За да се избегнат партиции, мора да се има сигурност дека секој поединечен јазол на системот е секогаш достапен од другите јазли, што не може да се гарантира во големите дистрибуирани системи. Така да за скалабилните системите одлуката останува помеѓу достапноста и конзистентноста.



Слика 58 Пример за BASE system

Извор : Google

BASE во основа претставува достапна, мека состојба , на крајот доследна која се фокусира главно на достапноста на системот и на трошоците за олабавување на гаранциите за конзистентност. Својството за евентуална конзистентност на BASE системот прифаќа периоди каде што клиентите може да читаат неконзистентни (т.е застарени) податоци. Сепак, тоа гарантира дека тие периоди на крајот ќе бидат крај. Системот со особини BASE веќе не е ограничен со CAP theorem, на тој начин нуди висока хоризонтална приспособливост. [35]Пример за должината на неконзистентни периоди за BASE системите може да се најдат каде што се анализира системот за складирање S3 на Amazon. Таму авторите набљудуваат флуктирачки времиња на недоследност, од неколку милисекунди до неколку секунди.

[35] *NoSQL Databases A Complete Guide - 2020 Edition Gerardus Blokdyk*

7.2 MapReduce

MapReduce потекнува од Google, тоа е рамка како и програмски модел, дизајниран да обработува огромни количини на податоци со користење на кориснички дефинирана логика. Примарната цел од оригиналната рамка на MapReduce треба да обезбеди апстракција за обработка на податоци, без да се занимава со барањата кои доаѓаат со приспособливост, како паралелизација или балансирање на оптоварување[36]. Влез во извршување на MapReduce е речник, т.е. база на податоци што се состои од парови клучеви/вредности. Корисникот треба да имплементира две функции, MAP и REDUCE. Функцијата MAP добива клуч и вредност и враќа листа на парови клучеви/вредности како среден резултат. MapReduce рамката ја повикува дадената MAP функција на сите записи на влезниот речник. Бидејќи сите тие повици за функции се независни една од друга, рамката може да ја паралелизира обработката по потреба. За ова, влезните податоци се поделени на неколку делови, од кои секоја се ракува со посебен процес на MAP.

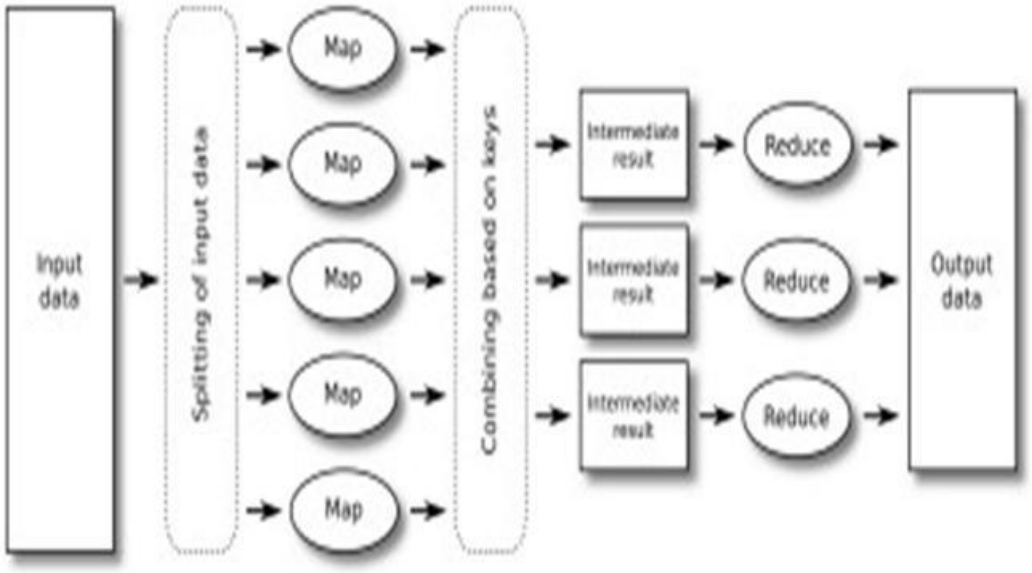
Во рамките на средните резултати, сите парови клучеви/вредности се комбинирани со помош на клучеви, така што секој клуч се пресликува на листа на вредностите. За секој клуч се повикува функцијата REDUCE. Како влез се добива клуч, како и список на сите вредности кои припаѓаат на тој клуч. Функцијата REDUCE ги обработува вредностите и ги комбинира до вредност на конечниот резултат, што се поврзува со клучот. Повторно сите повикувања на REDUCE може да се извршат паралелно.

[36] *J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. Commun. ACM, 51(1):107–113, Jan. 2008*

7.3 MongoDB

MongoDB е отворена база на податоци за документи, иницирана од софтверската компанија 10gen [37], која нуди и комерцијална поддршка. Како што името MongoDB - потекнува од зборот „хумонгозно“ , кое е негова главна цел да се справи со огромни количини на податоци. Додека MongoDB е имплементиран во C++ кој користи JavaScript како јазик за барање.

Во следните делови, моделот на податоци обезбеден од MongoDB е анализиран, земајќи ги во предвид трите основни компоненти

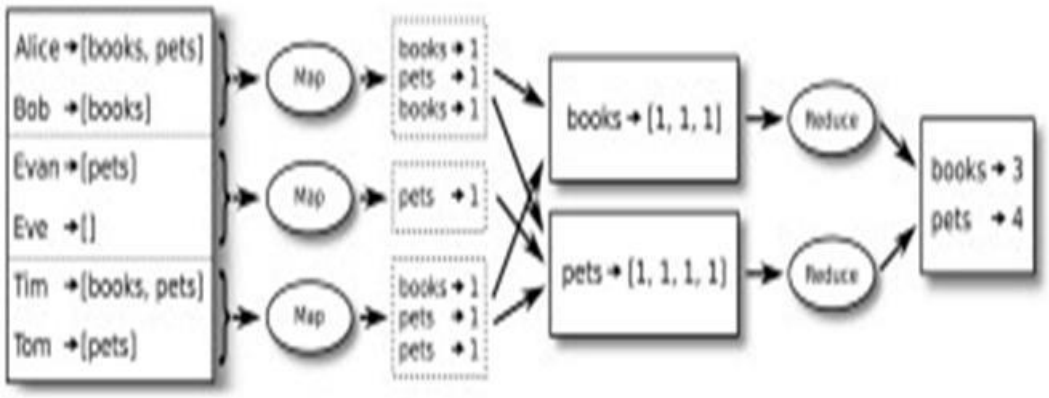


Слика 59 Модел на рамката MapReduce.

Извор : Google

Разделувањето на влезните податоци ги предаваат на неколку процеси кои ја извршуваат MAP функцијата. Повратните вредности на функциите на картата се комбинираат до средни резултати. За секој среден резултат по еден процес REDUCE е извршен.

[37] *10gen. The mongodb company. http://www.10gen.com/, July 2012*



Слика 60 Пример со користење на MapReduce за да се анализира кој работи им се допаѓаат на колку луѓе.

Извор : Google

Списокот на луѓе и допаѓања служи како влезни податоци. Влезните податоци се поделени на три сегменти, кои сите се обработуваат паралелно. на модели на податоци како што е дефинирано претходно. Повеќето од тие делови се засноваат на информации од тековната верзија на прирачникот на MongoDB[38].

7.4 Структура на податоци

Основните градежни блокови што ги користи MongoDB се колекциите. Релацискиот еквивалент на збирка е релација (или маса). Спротивно на односите, збирката не спроведува фиксна шема, но може да има сосема поинакви документи. Сепак, давајќи ги сите документи од збирката со малку слична структура може да овозможи полесно и поефикасно пребарување на податоци.

[38] *MongoDB. The mongodb manual. http://docs.mongodb.org/manual/, July 2012.*

Документите се споредливи со релациони множества и може да се гледа како асоцијативни низи или кешови. MongoDB store ги пренесува документите во формат BSON (Binary JSON).

Документот BSON е главно бинарна репрезентација на JSON документ, проширен со информации за полесно анализирање на податоците (на пр. префикси за должина). Како JSON објекти, документите BSON се состојат од парови атрибути-вредности, каде секоја вредност може да биде едноставен тип (на пр. низа или цел број), повторно комплексен BSON објект, или збирка која се состои од едноставни типови или предмети. Дополнително BSON се протега според спецификацијата JSON според некои едноставни типови, на пр. типови за датуми и времиња. [39]

7.5 Оператори

7.5.1 Reading data (Читање на податоци)

Една важна операција за читање на податоци од колекција на MongoDB е FIND. Кога се користи без дополнителни параметри, ги враќа сите документи од збирка. Операцијата FIND прифаќа објект со критериуми форматиран BSON како параметар. MongoDB ги филтрира документите според дадените критериуми, враќајќи ги само резултатите од барањето што одговараат на сите наведени атрибути и вредности. Дополнително, објектот за критериуми може да содржи посебни условни оператори дефинирани од MongoDB. Ваквите оператори се движат од едноставни споредби и регуларни изрази кои се сложено дефинирани од корисникот за функции на JavaScript, кои се оценуваат од базата на податоци.

[39] *E. A. Brewer. Towards robust distributed systems. In Symposium on Principles of Distributed Computing (PODC), 2000. BSON. Bson - binary json. http://bsonspec.org,July 2012.*

Друг начин за барање податоци е користење на MapReduce оператор на MongoDB. Овој оператор очекува MAP и функција REDUCE, и двете дадени како JavaScript функции на одреден формат., прво Функцијата MAP се извршува на сите документи. [40]Потоа на средните резултати се обработуваат со функцијата REDUCE. Повратните вредности на чекорот REDUCE го претставуваат барањето

```
1  db.people.find({name: 'Alice'})
2  // {_id: 1234, name: 'Alice', age: 20, likes: ['
       books', 'pets'], knows: ['Bob']}
3
4  db.people.find({age: 25})
5  // {_id: 5678, name: 'Bob', age: 25, likes: ['books
       ']}
6
7  db.people.find({age: {$lt: 30}})
8  // [{_id: 1234, name: 'Alice', ... }, {_id: 5678,
       name: 'Bob', ... }]
9
10 db.people.find({knows: {$exists: true}})
11 // {_id: 1234, name: 'Alice', ... }
12
13 db.people.find({likes: {$size: 1}})
14 // {_id: 5678, name: 'Bob', ... }
```

Слика 61 Пример за прашања користејќи FIND.

Извор : Google

Колекцијата „people“ се претпоставува дека има два записи: „Alice“ и „Bob“.како резултат.

[40] *E. A. Brewer. Towards robust distributed systems. In Symposium on Principles of Distributed Computing (PODC), 2000. BSON. Bson - binary json. http://bsonspec.org,July 2012.*

7.5.2 Manipulating data (Манипулирање со податоци)

Операцијата INSERT може да се користи за додавање и собирање на документи. Документот што треба да се вметне се предава на операција во формат BSON. Пред да го зачува дадениот документ, MongoDB додава дополнителен атрибут

„id“, со една единствена вредност во целата колекција. Ова може да се види како примарен клуч, уникатно идентификување на документот. [41]За да се сменат постоечките објекти на колекцијата, може да се користи операцијата UPDATE. Повторно очекува и критериумски објект. Објект што ги претставува промените што треба да се применат. Ако вториот објект претставува нормален документ, првиот документ претставува одговарањето на дадените критериуми целосно да се замени.

За да се модифицират само поединечни атрибути на соодветните документи, MongoDB обезбедува дополнителна синтакса на модификатор.

Бришењето на документи е можно со помош на операцијата REMOVE. Документите што треба да се избришат повторно може да се адресираат.

```
1  map = function() {
2    for (i in this.likes) {
3      emit(this.likes[i], {count: 1});
4    }
5  };
6
7  reduce = function(key, values) {
8    var total = 0;
9    for (i in values) {
10     total += values[i].count
11   }
12   return {count: total}
13 };
14
15 db.people.mapReduce(map, reduce)
16 // [{ books: {count: 3} },
17 //  { pets: {count: 4} }]
```

Слика 62 MongoDB имплементација на MapReduce (поедноставен излез).

Извор : Google

[41] *R. H. Guting. Graphdb: Modeling and querying ¨ graphs in databases. In Proceedings of the 20th International Conference on Very Large Data Bases, VLDB '94, pages 297–308, San Francisco, CA, USA, 1994. Morgan Kaufmann Publishers Inc.*

7.6 Правила за интегритет

Спротивно на релационите системи, нудејќи широк опсег на правила за интегритет, MongoDB нуди само можности за спроведување на уникатноста на документите. [42]Секој документ има единствен атрибут „id“. Дополнителни атрибути, како и комбинации на атрибути, може да се дефинираат и како единствени индекси . Индексите обезбедуваат дека вредностите за наведените атрибути се единствени низ колекцијата.

Бидејќи MongoDB нема вградена поддршка за приклучување на различни документи, исто така, воопшто не нуди правила за референцијален интегритет.

7.7 Индексирање

Индексите на бази на податоци се важни структури на податоци, кога тоа доаѓа до оптимизирање на прашањата за читање. Во релационите бази на податоци, колоната со примарен клуч е типично стандардно индексирана.

Истото важи и за атрибутот „id“ на документот MongoDB. Со цел да се овозможи ефикасен пристап до податоците, MongoDB нуди можност за создавање произволни многу различни индекси за колекција. Во основа, секој атрибут на документот може да се индексира, како и секоја комбинација на атрибути.

Ова се протега на атрибути во под-документи (т.е. атрибути на постоечките атрибути).

[42] *MongoDB. Sharding-mongodb.http://www.mongodb.org/display/DOCS/Sharding, July 2012.*

Структурата на документот не е фиксирана од колекцијата. Бидејќи индексите се креираат по колекција, може да има документи во колекцијата, кои го прават да немаат специфичен атрибут, иако индекс за атрибутот постои. MongoDB ги третира таквите документи како да имаат атрибут со вредност null. За збирки кои содржат различни документи, ова предизвикува значителни трошоци.[43]

Затоа MongoDB нуди друга опција за индексирање, наречено ретко индексирање. Ако индексот е редок, тој ги игнорира сите документи од збирка, кои немаат индексиран атрибут.

Ова овозможува подобри перформанси и го намалува прекумерното складирање, бидејќи не треба да се креираат и одржуваат „празни“ индекси.

Како што е познато од релационите системи, иако индексите ги прават операциите за читање поефикасни, кое има цена на поскапи операции за запишување. Секогаш кога некој документ е креиран, ажуриран или избришан, индексите треба да се ажурираат. Особено за апликации со интензивно пишување, создавањето премногу индекси може значително да ги намали перформансите на системите.

```
1  db.people.insert({name: 'Alice', age: 20})
2  db.people.update({name: 'Alice'}, {name: 'Eve'})
3  db.people.find({name: 'Eve'})
4  // {_id: 123, name: 'Eve'}
5
6  db.people.insert({name: 'Alice', age: 20})
7  db.people.update({name: 'Alice'}, {$set: {name: 'Eve
       '}})
8  db.people.find({name: 'Eve'})
9  // {_id: 456, name: 'Eve', age: 20}
10
11 db.people.insert({name: 'Alice', age: 20})
12 db.people.update({name: 'Alice'}, {$inc: {age: 5}})
13 db.people.find({name: 'Alice'})
14 // {_id: 789, name: 'Alice', age: 25}
```

Слика 63 Примери за манипулирање со документи во MongoDB користејќи INSERT и UPDATE

Извор : Google

[43] *W. Vogels. Eventually consistent. Queue, 6(6):14–19,Oct. 2008.*

7.8 Приспособливост

Бидејќи MongoDB не се справува со никакви референци помеѓу збирки, секое барање за читање или пишување од клиент секогаш вклучува точно една колекција. Скалирањето на апликации со многу помали колекции вертикално може да се постигнат со ставање различни колекции на различни машини.

Сепак, вертикалното скалирање станува покомплицирано кога се управува со огромни количини документи складирани во една збирка. MongoDB нуди механизам наречен „sharding“ на дистрибуираните колекции преку повеќе јазли, транспарентно за апликацијата, бројот на парчиња што треба да се користат.

Fragment key претставува атрибут што мора да го имаат сите документи од збирката, кои треба да бидат фрагментирани. [44]Клучот за фрагменти се користи за да се подели колекцијата на повеќе делови. Еден дел е подмножество на збирката, во која се наоѓаат сите документи, кои имаат вредности за тастерот во одреден опсег 7.

Ако атрибутот “ a ” е дефиниран како клучен фрагмент, а збирката има n делови (ci)i=1..n, тогаш секој дел “ci” ги содржи сите документи со вредности “ v ” за атрибутот “a”, така што min ≤ v < max . Вредностите за min и мах , т.е опсезите за секој дел, се одредуваат од системот, така што сите документи може да се распределат подеднакво низ достапните делови. Секогаш кога делот станува поголем од претходно дефинираниот праг, системот повторно го дели и деловите се прераспределуваат, за да се обезбеди еднакво оптоварување на сите повторувачки фрагменти. Ефикасноста на пристапот за читање и запишување до уништената збирка значително зависи од изборот на клучот за фрагменти.

Користејќи атрибут како деловен клуч, кој ја има истата вредност за повеќето документи, спречува делот во кој се наоѓаат тие документи да се подели. Така колекцијата не може да се дистрибуира подеднакво низ сите делови 8. Атрибутот „home country“, би бил лош фрагментен клуч, ако најверојатно доаѓаат сите луѓе во колекцијата од истата земја. Во тој случај MongoDB не може да создаде рамномерно и распоредено раздвојување.

[44] *NoSQL for Mere Mortals Dan Sullivan*

7.9 Конзистентност

Сè додека за секој фрагмент има само еден сервер за ракување со сите барања за читање и пишување, MongoDB нуди силна конзистентност на поединечни ентитети. Тоа значи дека сите ги читаат и пишуваат единечните документи кои се атомски, и FIND операциите што вклучуваат само поединечни документи се гаранција дека ја одразуваат најновата состојба на базата на податоци.[45] Дополнително, операциите FIND во MongoDB може да се наведат како опција за да се дозволи секундарно читање. Во тој случај барањето може да се одговори со реплициран (т.е. робот) јазол, доколку се користи репликација. На овој начин достапноста на базата на податоци може да се зголеми. Иако, ова доаѓа по цена на конзистентност, како што бара "Теоремата на Кап" (Cap's Theorem). Бидејќи на реплицираниот јазол можеби сè уште има застарени информации за бараниот субјект, овде може да биде само евентуалната конзистентност загарантирана.

7.10 Предности и недостатоци на користењето на NoSQL

Разбирање на опциите за моделирање достапни во NoSQL заедно со RDBMS знаење сега можат да се разбираат добрите и лошите страни на користење на NoSQL. [46]Има неколку различни репрезентативни барања за апликација за предностите и недостатоците на користењето NoSQL

- Entity schema requirement (Барање за шема на ентитет)
- Data access requirements (Барања за пристап до податоци)
- What NoSQL can do (Што може да направи NoSQL)
- What NoSQL cannot do (Што NoSQL не може да го направи)
- Whether or not to use NoSQL (Дали да се користи или да не се користи NoSQL)

[45] *D. Pritchett. Base: An acid alternative. Queue, 6(3):48–55, May 2008.*
[46] *NoSQL for Mere Mortals Dan Sullivan Sullivan*

На широко ниво, класифицирани се апликациите во следните категории:

- Апликации кои се занимаваат со релациски податоци кои се тешки за трансакции
- Апликации засновани на податоци кои се тешки за пресметки
- Апликации во веб-скала, управувани од податоци каде што се прифатливи мали задоцнувања.

Трансакциска апликација- Овој тип на апликација има податоци кои се многу релациски по природа. Карактеристиките на апликацијата се:

- Апликацијата се потпира на конзистентноста и интегритетот на податоците
- Конкурентноста (користењето) е релативно помала
- Lower е релативна придавка – потребно е апликацијата да може да превзема податоци од една база на податоци без какви било барања за репликација или рамнотежа на оптоварување
- Може да избере да има одраз на репликација за автоматско откажување, но апликацијата покажува за најмногу еден пример – “can failover” (може да не успее)

Пример за ова е апликација која се користи во продажно место во продавница за храна или апликација за управување со ресурси на одредено претпријатие. [47]Главната цел овде е конзистентноста на податоците.

[47] *T. Haerder and A. Reuter. Principles of transactionoriented database recovery. ACM Comput. Surv., 15(4):287–317, Dec. 1983.*

Барања за шема на ентитети- За трансакциска апликација, општите барања за шемата на ентитетот вклучуваат:

- Високо структурирана definition-properties , типови и ограничувања, доколку се применува
- Способност да се дефинираат врски
- Шемата не се развива или варира премногу со текот на времето.

Барања за пристап до податоци- Од гледна точка на пристап до податоци, има неколку барања:

- Конзистентност — секое читање треба да ги врати најновите ажурирани податоци
- Најчесто, целиот запис (ред) ќе биде преземен
- Пристапот до податоци меѓу ентитети (data) може да биде постојан

Што може да направи NoSQL- За трансакциска апликација, NoSQL може да помогне на следниов начин:

- Базата на податоци ориентирана кон колона помага да се дефинира структурата. Ако е потребно, може да се променува со тек на време.
- Базата на податоци ориентирана кон документи може да помогне во имплементацијата на JOIN или во создавање прикази.

Она што NoSQL не може да го направи- Користењето на NoSQL ќе резултира со неколку ограничувања за трансакциска апликација:

- Неможност да се дефинираат односите
- Отсуство на трансакции (Neo4j е исклучок)
- Недостапност на ACID својства во операциите (CouchDB и Neo4j се исклучоци)
- Користењето на Cassandra или CouchDB може да биде претерано ако се споредат со MySQL или PostgreSQL - тие не ја ослободуваат својата вистинска моќ во инсталација со една машина
- Отсуство на поддршка за JOIN и барање помеѓу ентитети, иако документ-ориентираните (document-oriented) складови го поддржуваат преку MapReduce, но напорите може да бидат значителни бидејќи прашањата стануваат сложени.

7.11 Компјутерска апликација

Овој тип на апликација прави многу пресметки во апликацијата. Клучните карактеристики на овој тип на апликација се:

- Повеќето од операциите се по должина на даден сет на својства низ записите
- Податоците што се складирани може да бидат сè уште релациони, но податоците центрирани на пресметките дефинитивно имаат ретки односи.

Пример за овој тип на апликација е апликација за анкета или апликација за управување со евиденцијата за продажба. Главната цел овде е дека пресметката ги има надминато записите, но на подмножеството од својства или колони кои се достапни (подмножеството во краен случај може да биде 100 процентен од достапните колони). Исто така, апликацијата за анкета би побарала способност да се дефинираат сопствени полиња и операции.

7.11.1 Барања за шема на ентитети

За компјутерска апликација, општите барања за шемата на ентитетите вклучува:

- Дефиниција на структурна schema-properties, типови и ограничувања, во случај да бидат применливи.[48]
- Шемата не се развива или варира премногу со текот на времето.

7.11.2 Барања за пристап до податоци

Од гледна точка на пристап до податоци, ги има следниве барања:

- Делумен пристап до евиденција.
- Вертикални барања, односно обработка според колона.

На пример, за субјект кој ги чува податоците за дневните приходи и расходи на дадена локација на работа, најчесто ќе се пресмета низ колоната за приходи и/или колона за трошоци отколку да се работи на целиот ред.

- Пристапот до податоци меѓу ентитетите е редок.

На пример, за да се креира конечен биланс на состојба за локација, може да се користи locationID еднаш за да се добиат деталите за локацијата.

- Конзистентност -податоците треба да бидат конзистентни. Меѓутоа, во некои случаи, дозволени се мали задоцнувања.

На пример, бидејќи извештаите може да се генерираат секојдневно наместо во реално време, корисникот е задоволен да работи со еднодневни податоци.

---

[48] *C. Strozzi. Nosql relational database management system. http://www.strozzi.it/cgi-bin/CSA/tw7/I/en_US/NoSQL/HomePage, July 2012.*

7.11.3 Што може да направи NoSQL за компјутерска апликација

За компјутерска апликација, NoSQL може да помогне на следниов начин:

- Базите на податоци ориентирани кон колони би помогнале да се дефинира ригорозна шема. Document или Key-value бази на податоци сè уште можат да се користат.На пример, форматите JSON може да се користат за дефинирање на формална шема. Само овие складови не можат да ја применат шемата.
- (Column-oriented, Key-value, и Document stores) можат да обезбедат брзина и размер при преземање делумни податоци.
- Складирањето на документи заедно со обработката на MapReduce може да помогне во извршувањето на пресметувањето веднаш до нивото на податоци, со што значително се зголемува брзината на извршување. Не е пожелно непотребните податоци да лебдат низ мрежата.[49]
- Складот на документи (Data store) може да помогне во спроведувањето JOIN или создавање на прегледи.

7.11.4 Што NoSQL не може да го направи за компјутерска апликација

Користењето на NoSQL ќе резултира со неколку ограничувања за компјутерска апликација:

- Дефинирањето на односите може да биде незгодно. Освен за бази на податоци со графикони, овие односи мора да се одржуваат во примената.
- Бидејќи врските не постојат во базата на податоци, податоците може да бидат неконзистентни - макар и моментални.

[49] *H. Lu, J. X. Yu, G. Wang, S. Zheng, H. Jiang, G. Yu, and A. Zhou. What makes the differences: benchmarking xml database implementations. ACM Trans. Internet Technol., 5(1):154–194, Feb. 2005.*

Одлука- Списокот на задачи што се можни во NoSQL се можни и со RDBMS. Единствена точка која NoSQL ја добива овде, што може да испадне дека е голем диференцијатор, е брзината и скалата со која податоците може да се поделат хоризонтално и да се преземат.

Системите RDBMS овозможуваат филтрирање на прашања на она што е дадено: Choose приход, трошок from location_finance where location_id=1234

Сепак, внатрешно RDBMS системите се подесени да го преземат целиот ред во тоа време од основните физички складови и потоа да го нанесат во филтерот-особено колоните избрани користејќи ја изјавата SELECT. Од друга страна, NoSQL бази на податоци - особено ориентирани кон колони бази на податоци - се високо оптимизирани за враќање на делумни записи и може да резултираат со драматична разлика во перформансите наспроти RDBMS додека се справува со стотици милиони или милијарди записи.

Ова е тежок избор помеѓу базите на податоци RDBMS и NoSQL во овој случај. Ако апликацијата е апликација за претпријатие каде што бројот на записи ќе биде ограничена на околу сто милиони или помалку, RDBMS може само да опслужи правилно, иако може да се користи и NoSQL. За која било големина помала од милион записи, NoSQL може да има режиски трошоци во однос на поставување и подесување; додека за над неколку стотици милиони записи, RDBMS ќе се намали во перформансите.

7.12 Апликација на веб-скала

Овој последен тип на апликација е веројатно порелевантен денес во апликациите за потрошувачи, дали се целосно веб-базирани или мобилни апликации или комбинација од двете.

Некои од клучните карактеристики на овој тип на апликација се:

- Апликацијата треба да може да се скалира поради огромниот волумен на содржината на која работи, огромниот број на корисници и огромната географија каде пристапот на корисниците го прави центарот за податоци неостварлив.
- На корисниците на оваа апликација можеби им е добро да работат со нереално време, релативно застарени податоци. Застоеноста може да се движи од неколку десетици милисекунди до неколку дена, но најновите податоци може дефинитивно да не се достапни во рамките на дел од милисекунда.
- Шемата може да се развива со текот на времето бидејќи апликацијата дозволува интеграција со други апликации.
- Бидејќи податоците никогаш не можат целосно да се нормализираат или денормализираат, односите ќе постојат.

Пример за оваа апликација е апликација за веб-аналитика или социјална платформа за микроблогирање. Клучот овде е обемот на работа и застојот на податоците. Друг сет на примери вклучува апликации за претпријатие засновани на SaaS како што се CRM или ERP. Еден брилијантен пример за ова е SalesForce - тоа е SaaS апликација која овозможува да се интегрираат податоците од корисничката шема.

7.12.1 Барања за шема на ентитети

За апликација на веб-скала, општите барања за шемата на ентитетот вклучува:

- Дефиниција на структурна шема
- Способност за промена на шемата со текот на времето без да се влијае на постоечките записи во каков било начин - во екстремен случај, латентна шема
- Врските може да бидат опционални во слојот на базата на податоци и може да се туркаат до слој на апликација главно поради малата густина

7.12.2 Барања за пристап до податоци

Од гледна точка на пристап до податоци, постојат следниве барања:

- Делумен пристап до евиденција
- Брзина на извршување на операција-CRUD
- Неконзистентни податоци - за краток момент - се поднесливи

7.12.3 Што може да направи NoSQL за веб-скала

За апликација во веб-скала, NoSQL може да помогне на следниов начин:

- Сѐ што е веќе опфатено во претходното сценарио. Document store би биле фантастичен избор за латентна шема
- Може да обезбеди обем на операции бидејќи не имплементира ACID операции, но најчесто обезбедуваат особини на BASE.

7.12.4 Што NoSQL не може да го направи за веб-скала

Нема никакво барање што NoSQL не може да го исполни во овој случај. Ако се нема ACID ограничувања - тоа е една од главните причини зошто е измислен NoSQL.

Одлука- Препорачливо е да се користи NoSQL. Изборот на типот на сладишта (модел на податоци) може да варира во зависност од вистинскиот основен услов:

- Во случај на SaaS апликации каде што шемата треба да биде флексибилна инкорпорираат атрибути специфични за корисникот, складирањето документи се оптимален избор.

  Примери за овој подтип вклучуваат апликации како што се CRM, ERP – главно апликации насочени кон претпријатието каде секој краен потрошувач (организација) може да имаат своја специфична шема.
- Во случај на апликации како е-учење или други социјални апликации, чија основната шема се менува и се развива со брзо темпо и би требало способност да се промени шемата со текот на времето, но сепак да се биде под строга контрола на развивач на апликации, база на податоци ориентирана кон колона е одличен избор.
- Во случај на социјални апликации на кои им е потребна способност да се интегрираат со други апликации, можеби ќе треба да користи мешавина од Column oriented и Document oriented store за да се ублажи ризикот од целосна шема на ремонт на непредвидени формати на складови на нова апликација која останува без оваа апликација мора да се интегрира со новата апликација на пазарот.
- За зачувување на врските, базите на податоци со графикони може да бидат додаток на вистинскиот склад за податоци. На пример, Твитер користи база на податоци за графикони, FlockDB за складирање на врски додека ја користи Cassandra за аналитика во реално време и HBase за постојан склад на податоци.

РЕЗИМЕ / ЗАКЛУЧОК

Може да се заклучи дека NoSQL базите на податоци ги обработуваат податоците побрзо од релационите бази поради нивниот поедноставен модел. Има различни студии и евалуации кои може да се направат врз основа на различни фактори за квалитет за секоја од NoSQL базите на податоци. Тоа ќе им помогне на корисниците да се одлучат за вистинската база на податоци според барања. Има голем број на достапни бази на податоци NoSQL, така што, изборот на добро прилагоден NoSQL базите на податоци станува се потежок. Изборот може да се заснова на одредени параметри како приспособливост што ќе помогне корисникот и организациите да изберат вистинска база на податоци од достапните. Исто така, може да се заклучи дека неопходните функционалности и механизми влијаат на изборот на базата на податоци. NoSQL базата на податоци ни обезбедува многу придобивки, конзистентност, достапност и толеранција на партиции. Исто така, обезбедува можности за лесно складирање на податоците од графиконот, кои не се достапни со SQL базите на податоци. Наместо тоа, има некои проблеми со NoSQL базите на податоци.

NoSQL базите на податоци се способни да складираат различни типови на податоци и не треба да бидат структурирани како SQL базите на податоци. Оттука, нерелационите бази на податоци овозможуваат голема приспособливост и флексибилност, што го прави посоодветен избор при ракување со големи групи на неструктурирани и неповрзани податоци.

Вообичаено, колку е пообемна базата на податоци, толку е поголема веројатноста NoSQL базата на податоци да е подобра опција. Не-релационите бази на податоци имаат тенденција да се истакнуваат во барањата за приспособливост и достапност, што се идеални за социјални мрежи и апликации во реално време (на пример, онлајн игри, инстант пораки).

NoSQL базите на податоци бараат програмско знаење. За разлика од SQL, кој исто така може да го научи персоналот од други области како што се менаџментот и маркетингот, на базите на податоци на NoSQL обично им треба некој со искуство во кодирање и способност да стекне други јазици според системите на бази на податоци што се користат.

КОРИСТЕНА ЛИТЕРАТУРА

1. 10gen. The mongodb company. http://www.10gen.com/, July 2012.
2. R. Angles and C. Gutierrez. Survey of graph databasemodels. ACM Comput. Surv., 40(1):1:1–1:39, Feb. 2008.
3. Apache. Couchdb - a database for the web. http://couchdb.apache.org/, July 2012.
4. D. Bermbach and S. Tai. Eventual consistency: Howsoon is eventual? an evaluation of amazon s3's consistency behavior. In Proceedings of the 6th Workshop on Middleware for Service Oriented Computing, MW4SOC '11, pages 1:1–1:6, New York, NY, USA, 2011. ACM.
5. A. B. Bondi. Characteristics of scalability and their impact on performance. In Proceedings of the 2nd international workshop on Software and performance, WOSP '00,pages 195–203, New York, NY, USA, 2000. ACM.
6. D. Borthakur, J. Gray, J. S. Sarma, K. Muthukkaruppan, N. Spiegelberg, H. Kuang, K. Ranganathan,
7. D. Molkov, A. Menon, S. Rash, R. Schmidt, and Aiyer. Apache hadoop goes realtime at facebook.In Proceedings of the 2011 ACM SIGMOD International Conference on Management of data, SIGMOD '11, pages 1071–1080, New York, NY, USA, 2011. ACM.
8. E. A. Brewer. Towards robust distributed systems. In Symposium on Principles of Distributed Computing (PODC), 2000. BSON. Bson - binary json. http://bsonspec.org,July 2012.
9. R. Cattell. Scalable sql and nosql data stores. SIGMOD Rec., 39(4):12–27, May 2011.
10. F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber. Bigtable: a distributed storage system for structured data. In Proceedings of the 7th symposium on Operating systems design and implementation, OSDI '06, pages 205–218, Berkeley, CA, USA, 2006. USENIX Association.
11. J. Cheng, Y. Ke, and W. Ng. Efficient query processing on graph databases. ACM Trans. Database Syst., 34(1):2:1–2:48, Apr. 2009.

12. E. F. Codd. Data models in database management.SIGMOD Rec., 11(2):112–114, June 1980.

13. J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. Commun. ACM, 51(1):107–113, Jan. 2008.

14. G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels. Dynamo: amazon's highly available key-value store. SIGOPS Oper. Syst. Rev., 41(6):205–220, Oct. 2007.

15. S. Edlich. Nosql databases. http:// nosql-database.org/, July 2012.

16. S. Edlich, A. Friedland, J. Hampe, and B. Brauer. NoSQL: Einstieg in die Welt nichtrelationaler Web 2.0 Datenbanken. Hanser Fachbuchverlag, 10 2010.

17. S. Gilbert and N. Lynch. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. SIGACT News, 33(2):51–59, June 2002.

18. J. Gray. The transaction concept: virtues and limitations (invited paper). In Proceedings of the seventh international conference on Very Large Data Bases - Volume 7, VLDB '81, pages 144–154. VLDB Endowment, 1981.

19. R. H. Guting. Graphdb: Modeling and querying ¨ graphs in databases. In Proceedings of the 20th International Conference on Very Large Data Bases, VLDB '94, pages 297–308, San Francisco, CA, USA, 1994. Morgan Kaufmann Publishers Inc.

20. T. Haerder and A. Reuter. Principles of transactionoriented database recovery. ACM Comput. Surv., 15(4):287–317, Dec. 1983.

21. H. Lu, J. X. Yu, G. Wang, S. Zheng, H. Jiang, G. Yu, and A. Zhou. What makes the differences: benchmarking xml database implementations. ACM Trans. Internet Technol., 5(1):154–194, Feb. 2005.

22. M. Michael, J. Moreira, D. Shiloach, and R. Wisniewski. Scale-up x scale- out: A case study using nutch/lucene. In Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International, pages 1 –8, march 2007.

23. MongoDB. The mongodb manual. http://docs.mongodb.org/manual/, July 2012.

24. MongoDB. Sharding - mongodb. http:

//www.mongodb.org/display/DOCS/Sharding, July 2012.

25. D. Pritchett. Base: An acid alternative. Queue, 6(3):48–55, May 2008.

26. C. Strozzi. Nosql relational database management system. http://www.strozzi.it/cgi-bin/CSA/tw7/I/en_US/NoSQL/HomePage, July 2012.

27. W. Vogels. Eventually consistent. Queue, 6(6):14–19,Oct. 2008.

28. NoSQL for Mere Mortals Dan Sullivan

29. NoSQL For Dummies Adam Fowler

30. NoSQL with MongoDB in 24 Hours Brad Dayley

31. NoSQL and SQL Data Modeling Ted Hills

32. NoSQL the Ruby Way Durran Jordan

33. NoSQL Databases A Complete Guide - 2020 Edition Gerardus Blokdyk

34. NoSQL for Mere Mortals Dan Sullivan Sullivan

35. NoSQL A Complete Guide, The Art of Service - NoSQL Publishing

36. NoSQL For Dummies Adam Fowler Professional NoSQL Shashank Tiwari

37. SQL & NoSQL Databases Andreas Meier, Michael Kaufmann

38. JSONiq The SQL of NoSQL Ghislain Fourny, Matthias Brantner, Federico Cavalieri

39. Making Sense of NoSQL Dan McCreary , Ann Kelly

40. Beginning Database Programming Using ASP.NET Core 3 Bipin Jos

A