För- och nackdelar med noSQL i motsats till MySQL

Johan Ollas

Examensarbete

Informationsteknik

2018

EXAMENSARBETE	
Arcada	
Utbildningsprogram:	Informationsteknik
Identifikationsnummer:	5688
Författare:	Johan Ollas
Arbetets namn:	För- och nackdelar med NoSQL i motsats till MySQL
Handledare:	Jonny Karlsson
Uppdragsgivare:	

Sammandrag:
Arbetets mål är att ta reda på vad NoSQL-databaser är och hur de strukturerar data och att jämföra deras prestanda mot relationsbaserade varianter. För att utföra arbetet har re olika NoSQL-databaser valts ut: kolumnbaserade Cassandra, dokumentbaserade MongoDB och grafbaserade Neo4j. Relationsdatabasen som behandlas i arbetet är MySQL. Arbetet börjar med att ta upp allmän information om alla de olika databaserna som behandlas för att skapa en bild av hur de fungerar, hur de är utvecklade och för att ge exempel på möjliga användningsområden. Sedan tas varje databas upp i detalj. Först berättar arbetet hur de olika databaserna strukturerar data för att lyfta upp eventuella skillnader. Sedan tas det upp hur det i praktiken går till att installera och konfigurera varje databas samt hur data matas in och sedan behandlas. Arbetet tar också upp alternativa verktyg för administrering av databaserna. Prestandajämförelserna baserar sig på analytisk genomgång av existerande forskningslitteratur. Arbetet tar upp ett flertal olika belastningstest där varje NoSQL-databas testas mot en relationsbaserad variant i olika miljöer. Testerna bevisar att NoSQL-databaserna fungerar bra inom sina specifika användningsområden samt när det handlar om att hantera stora mängder data. Testerna visar också att relationsdatabaser i de flesta fall presterar bättre på traditionell serverhårdvara. Arbetet avslutas med en spekulation om att NoSQL kommer att ha en mycket stor plats i framtiden, speciellt eftersom mängden data som måste sparas och behandlas hela tiden ökar vilket sätter en stor belastning på relationsdatabaser. En annan orsak som pekar på en ljus framtid för NoSQL är att de är mer kostnadseffektiva för att det går att sprida ut dem med hjälp av t.ex. molnteknologi.

Nyckelord:	NoSQL,SQL,icke-relationsdatabas,relationsdatabas
Sidantal:	30
Språk:	Svenska
Datum för godkännande:	18.12.2018

Degree thesis	
Arcada	
Education:	Information technology
Identification number:	5688
Author:	Johan Ollas
Name:	För- och nackdelar med NoSQL i motsats till MySQL
Supervisor:	Jonny Karlsson
Commissioned by:	

Abstrakt:
The goal of this thesis is to find out what NoSQL is and how they structure data and to compare their effectiveness against relational variants. Three different NoSQL-databases have been chosen: column based Cassandra, document based MongoDB and graph based Neo4j. The relational database used is MySQL. The thesis first goes through general information about the databases to paint a picture about how they work, how they're developed and to give examples of possible areas of use. After this we go more into detail. First, we go through data structure in order to find differences, then it shows how to install and configure each of the databases and how to save and process data. The thesis also shows alternative tools for database administration. Performance comparisons are based on existing literature. The thesis goes through several different stress tests where each NoSQL-database is tested against a relational variant. The tests show that NoSQL-databases work well in their intended areas of use and when the amount of data that needs to be processed becomes large. The tests also show that relational databases perform well on more traditional server hardware. The thesis concludes with a speculation that NoSQL will have a large place in the future especially because the amount of data that needs to be saved and processed is growing each day. Another reason would be that NoSQLis cheaper to use when using cloud technology.

Keywords:	NoSQL,SQL,non-relational,relational
Number of pages:	30
Language:	Swedish
Date of acceptance:	18.12.2018

INNEHÅLL

1 INLEDNING

De senaste åren har kraven på databaser ändrats på grund av växande datamängder och antalet användare som måste ha tillgång till dem. Den traditionella relationsbaserade modellen fungerar bra med data som är tydligt strukturerat. Problem uppstår eftersom en stor mängd tabeller med komplicerad struktur kräver ökade resurser och förfrågningar i SQL blir långa och svårlästa.

För att lösa det här problemet har olika NoSQL-databaser utvecklats. Huvudsakligen siktar dessa databaser på att vara snabba och horisontellt skalbara med molnteknologi efter behov. NoSQL-databaser är icke-relationsdatabaser vilket betyder att strukturen av data som lagras kan vara en hel del friare än i en relationsbaserad variant. (Yishan, L & Sathiamoorthy, M. 2013 s.1)

Målsättningen med arbetet är att ta reda på hur ett antal olika NoSQL-varianter fungerar i praktiken, och vilka deras användningsområden är, genom att skapa en av vardera med samma innehåll. Efter detta kan man jämföra hur data struktureras till skillnad från relationsbaserade databaser och hur användare sedan kommer åt den via de olika språken som används för förfrågningar. Ett annat syfte är att presentera en jämförelse mellan de olika databaserna baserat på prestanda.

Metoderna för arbetet är att jämföra strukturskillnader mellan flera olika databastyper genom att strukturera och lagra data i dem. Prestandajämförelserna baserar sig på genomgång av forskningslitteratur.

Valet av databaser i arbetet baserar sig på popularitet samt hur de strukturerar data.

Kapitel 2 beskriver de olika databastyperna. Kapitel 3 tar upp själva utvecklingen samt struktureringen av data och kapitel 4 handlar om prestandajämförelserna mellan NoSQL och relationsbaserade varianter.

2 DATABASTYPER

I det här kapitlet tas det fram information om de olika databastyperna som behandlas i arbetet.

2.1 Relationsdatabaser

MySQL är en populär relationsdatabas som är skriven i C och C++ och använder sig av SQL som står för Structured Query Language. Det finns andra populära relationsdatabaser som MS SQL Server och PostgreSQL men jag har i arbetet valt att använda MySQL på grund av hur den strukturerar data. MySQL stöder flera olika datatyper som används för att bygga upp själva databasen i tabeller som består av rader och kolumner. Tabellerna kan vara relaterade med en eller flera andra tabeller för att skapa en fungerande struktur som sedan kan användas i praktiken.

MySQL använder sig av ACID som står för atomicity, consistency, isolation och durability. Atomicity ser till att varje del av en transaktion fungerar förrän transaktionen godkänns. Consistency betyder att all ny data som skrivs in i databasen måste följa de regler som konfigurerats i databasen för att kunna sparas. Om två transaktioner behandlar samma data samtidigt ser Isolation till att de två transaktionerna inte kan påverka varandra. Durability ser till att alla ändringar som sker i databasen säkert blir sparade även om det skulle ske något slags hård- eller mjukvaruproblem. ACID modellen används för att försäkra att alla transaktioner är pålitliga vilket i vissa fall kan påverka prestandan negativt, speciellt i större databaser, eftersom den inte tillåter data att bli korrupterad. För att allt detta skall fungera ordentligt kräver det att data är strukturerad vilket kan vara ett problem med större datamängder som i dagens läge börjar vara vanliga. (MySQL 5.7 Reference Manual 2017)

2.2 Icke-relationsdatabaser

När datamängden som måste sparas växer och när den måste vara tillgänglig för miljoner användare samtidigt kan ACID modellen skapa problem med prestanda för att den sätter höga krav på hur data kan behandlas. Icke-relationsdatabaser som kallas NoSQL skapades för att lösa just det problemet.

Eftersom datastrukturen kan vara mer odefinierad blir sättet man sköter olika transaktioner mindre komplexa. Istället för att skapa en SQL-databas, t.ex. för användare på social media, där all relevant information sparas som rader i tabeller, kan man använda sig av

en dokumentorienterad NoSQL version så att varje individuell användares information sparas i ett enda dokument vilket gör att all information är mer tillgänglig. I teorin skapar det här en databas som är enklare och snabbare att installera och konfigurera medan alla transaktioner i framtiden inte behöver gå igenom lika många lager av tabeller för att hitta den data som man vill komma åt vilket i sin tur ökar prestandan och skapar mindre krav på hårdvara.

Det här kräver dock att NoSQL offrar pålitlighet genom att lämna bort stöd för ACID modellen. NoSQL lämpar sig därför bäst för okritiska data som inte behöver vara specifikt strukturerad. När stödet för ACID modellen lämnas går det också enklare att sprida ut databasen över flera olika servrar genom att utnyttja molnteknologi vilket betyder att en NoSQL-databas i teorin kan skalas oändligt.

För att transaktioner som körs skall lyckas kräver det att data inte kan vara helt ostrukturerad, NoSQL-databaser kategoriseras enligt hur de sköter detta. (Moniruzzaman, A B M & Hossain, Syed Akhter 2013 s.1-4)

2.2.1 Cassandra

Cassandra är en kolumnbaserad NoSQL-databas vilket betyder att strukturen på enskilda tabeller påminner om relationsbaserade alternativ. Skillnaden mellan de båda är att Cassandra sätter fokus på stora tabeller som inte har relationer med varandra för att öka på prestandan och för att göra förfrågningar simplare.

Cassandras arkitektur baserar sig på noder som samlas i kluster. En server är alltså en nod och flera noder kan kopplas ihop till ett kluster så att användare som kommunicerar med en viss nod har tillgång till all data, som ligger på andra servrar, i samma kluster automatiskt. (About Apache Cassandra 2016)

Ett exempel på användningsområde är Netflix som använder sig av Cassandra för att spara 36 miljoner användares tittarhistoria.

2.2.2 MongoDB

MongoDB är en dokumentorienterad databas, vilket innebär att om man t.ex. har ett kundregister så skulle all information för varje individuell kund sparas i ett skillt dokument. Alla dokument som hör till samma kundregister kategoriseras sedan i sin egen kollektion.

MongoDB strukturerar all data i dokument som JSON-objekt. Eftersom de flesta programmeringsspråken stöder JSON betyder det att applikationer kan kommunicera direkt med databasen vilket ökar på prestandan.

För att bygga upp en databas använder sig MongoDB av sharding som betyder att databasen delas upp i fragment som kan existera på en eller flera olika servrar samtidigt. En användare kan sedan kommunicera med alla fragment, som ligger i samma kluster, samtidigt. (The MongoDB 3.4 Manual 2016)

MongoDB används t.ex. av EA FIFA Online 3 för att spara all information om 15000 olika fotbollsspelare i 30 olika ligor.

2.2.3 Neo4j

Grafbaserade databaser fokuserar på visualisering av förfrågningar för att underlätta analys av data. Hela databasen, eller delar av den, kan visualiseras som en graf i alla olika skeden vilket underlättar utvecklingen

Neo4j använder sig av noder istället för tabeller. Fast noderna har relationer med varandra skiljer de sig från den traditionella relationsbaserade modellen genom att ge de olika relationerna attribut. På grund av detta kommer en förfrågan, som i SQL använder sig av JOIN funktioner, alltid att vara en lineär operation i Neo4j vilket sparar på resurser när användare kommunicerar med databasen.

Databaser i Neo4j kan delas upp i kluster som sedan sprids över flera servrar som gör det möjligt att skala databasen enligt behov. (The Neo4j Operations Manual v3.2 2017)

Neo4j används t.ex. av eBay för att ta reda på de snabbaste fraktningsmetoderna för sina kunder.

3 STRUKTUR, UPPBYGGNAD OCH FUNKTIONALITET

Det här kapitlet handlar om hur de olika databaserna byggs upp från början samt hur de sköter strukturering och behandling av data.

3.1 Verktyg

Detta kapitel tar upp de olika verktygen som användes för att utföra arbetet, som i sin korthet går ut på att skapa och konfigurera en databas av varje typ för att se skillnader mellan hur de strukturerar och behandlar data.

3.1.1 Mjukvara

För installation och administrering av databaserna användes följande mjukvara:
MySQL:

- MySQL Community Server 5.7.18
- MySQL WorkBench 6.3.9

Cassandra:

- Apache Cassandra 3.0.9

MongoDB:

- MongoDB 3.4

Neo4j:

- Neo4j 3.2.1

3.1.2 Data

Data som användes i arbetet är en samling postnummer från USA. Från början hämtades data som JSON och varje objekt innehöll följande information om de olika städerna:

- Stadens namn
- Stadens koordinater
- Invånarantal
- Stat
- Postnummer

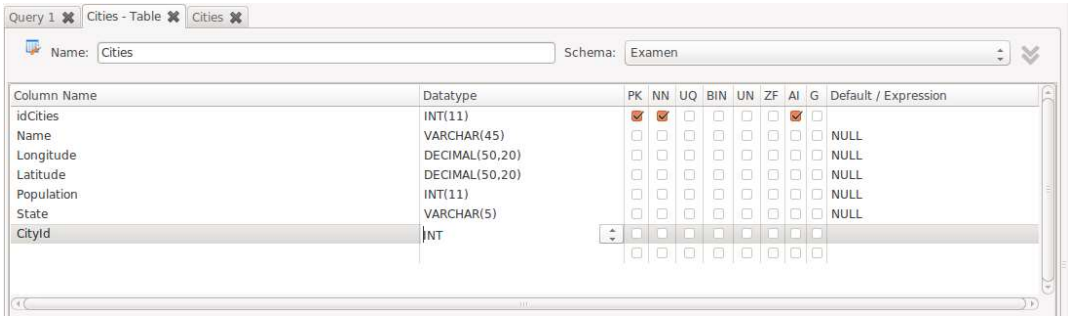
Eftersom arbetet handlar om hur de olika databastyperna struktureras valdes ett antal städer ut från listan som sedan inkluderades i varje databas.

3.2 MySQL

För att köra en MySQL server krävs mysql-server som installeras via kommandot: sudo apt-get install mysql-server. Efter lyckad installation krävs det ett verktyg för att skapa och administrera själva databasen, i det här arbetet användes MySQL Workbench som installeras via kommandot: sudo apt-get install mysql-workbench (MySQL Workbench 2018).

3.2.1 Struktur

I MySQL Workbench skapas först en tabell med namnet Cities som sedan fylls med de kolumner som krävs för att spara all information om städerna. Ett exempel på hur strukturen av en tabell ser ut i Workbench framgår ur figur 1.

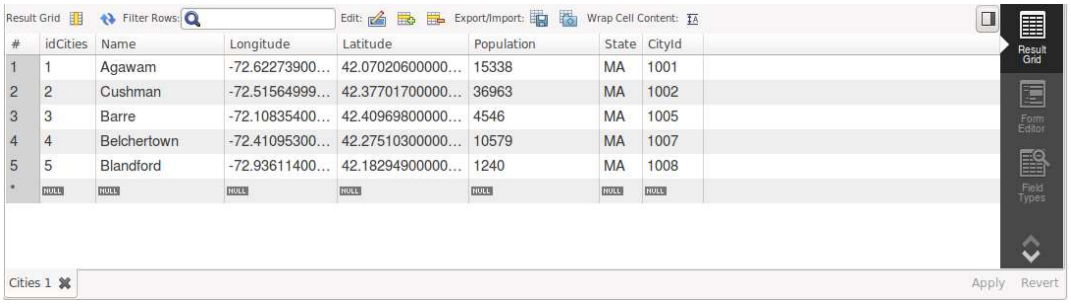


Figur 1. En översikt av en tabell i MySQL.

3.2.2 Uppbyggnad

Inmatningen av data sköts via MySQL Workbench med INSERT kommandot.

När ett antal städer finns i databasen ser tabellen ut som i figur 2.



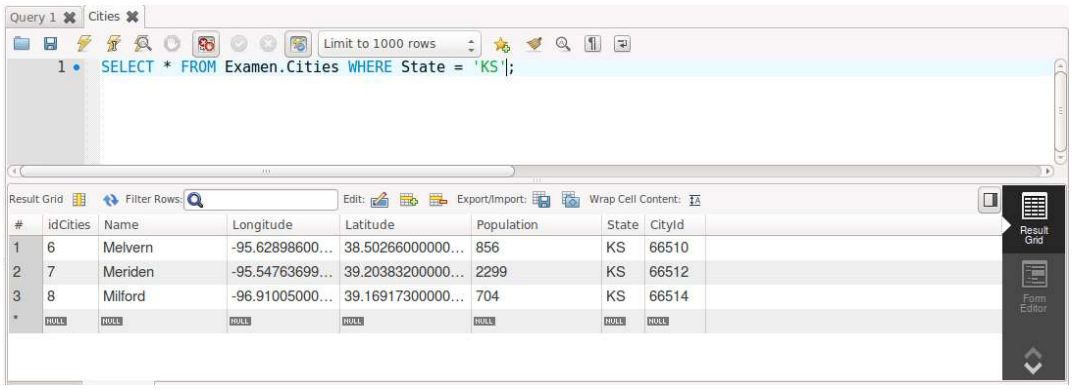
#	idCities	Name	Longitude	Latitude	Population	State	CityId
1	1	Agawam	-72.62273900…	42.07020600000…	15338	MA	1001
2	2	Cushman	-72.51564999…	42.37701700000…	36963	MA	1002
3	3	Barre	-72.10835400…	42.40969800000…	4546	MA	1005
4	4	Belchertown	-72.41095300…	42.27510300000…	10579	MA	1007
5	5	Blandford	-72.93611400…	42.18294900000…	1240	MA	1008
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Figur 2. En översikt av innehållet i en tabell i MySQL.

3.2.3 Funktionalitet

För att köra förfrågningar manuellt i MySQL används MySQL Workbench verktyget.

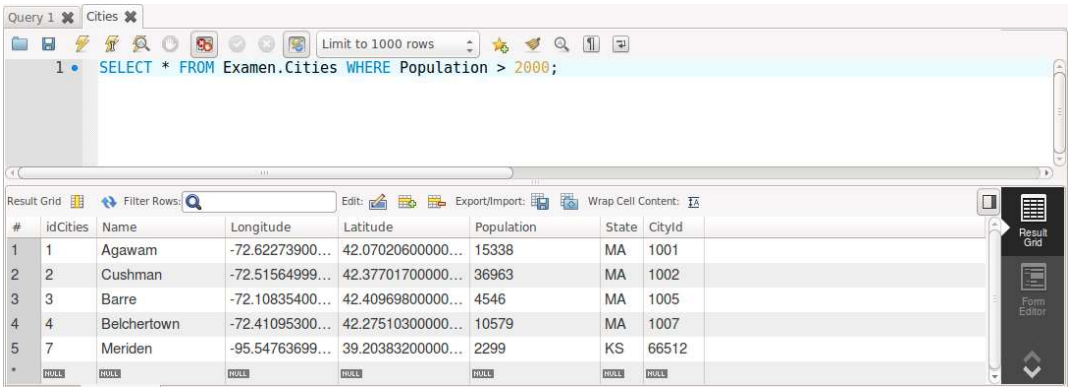
För att plocka ut alla städer i Kansas används SELECT * FROM och WHERE State = ’KS’. Vilket visas i figur 3.



#	idCities	Name	Longitude	Latitude	Population	State	CityId
1	6	Melvern	-95.62898600…	38.50266000000…	856	KS	66510
2	7	Meriden	-95.54763699…	39.20383200000…	2299	KS	66512
3	8	Milford	-96.91005000…	39.16917300000…	704	KS	66514
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL

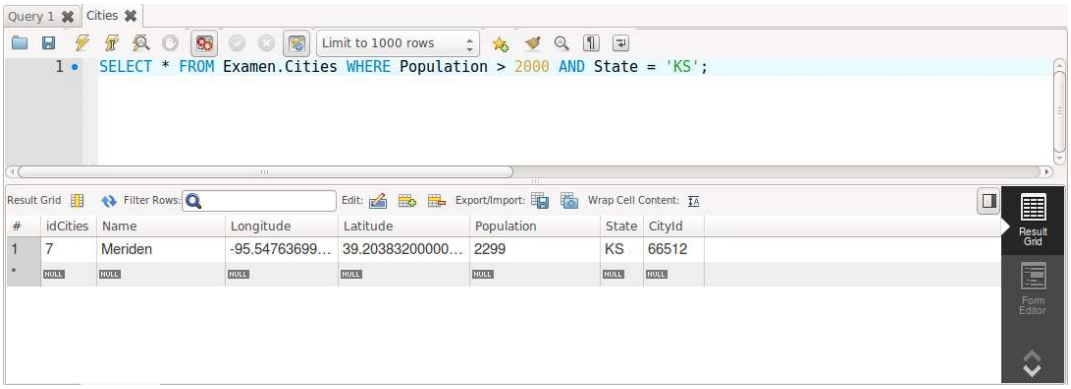
Figur 3. Alla städer i Kansas i MySQL Workbench.

Syntaxen för att få fram alla städer med fler än 2000 medborgare använder sig av WHERE Population > 2000, vilket framgår ur figur 4.



Figur 4. Alla städer med ett befolkningsantal över 2000 i MySQL Workbench.

Den sista kombinerar de två tidigare förfrågningarna med AND, som visas i figur 5.



Figur 5. Alla städer i Kansas med ett befolkningsantal över 2000 i MySQL Workbench.

3.2.4 Objektorienterade verktyg

MySQL stöds av flera olika ramverk för objektorienterad utveckling för de flesta programmeringsspråken som t.ex. Hibernate för Java, Sequelize för JavaScript, SQLAlchemy och Django för Python m.m.

3.3 Cassandra

Själva installationen av Cassandra sköts via apt-get. För administrering av själva databasen används CQLSH som är ett inbyggt verktyg som kommunicerar med Cassandra via Cassandra Query Language. (About Apache Cassandra 2016)

3.3.1 Struktur

Cassandra strukturerar data i kolumner vilket betyder att varje stad är en rad i en tabell med flera kolumner som innehåller all information.
Varje rad måste ha en PRIMARY KEY som i det här arbetet representeras numeriskt som ett heltal.

3.3.2 Uppbyggnad

För att börja utveckla en databas med Cassandra måste det först skapas en nod via CQLSH. Tabellerna som bygger upp själva databasen skapas sedan inom deras egen nod genom CREATE TABLE. Ett exempel i figur 6.

```
cqlsh:examen> CREATE TABLE Cities(
          ... id int PRIMARY KEY,
          ... name text,
          ... location text,
          ... population int,
          ... state text,
          ... CityId int
          ... );
```

Figur 6. Hur man skapar en tabell i Cassandra.

Efter detta är det möjligt att tillägga städer till tabellen via INSERT, som syns i figur 7.

```
cqlsh:examen> INSERT INTO cities (id,cityid,location,name,populatio
n,state)
          ... VALUES(1,01001,'-72.622739,42.070206','Agawam',15338,
'MA');
```

Figur 7. Inmatningen av en stad i Cassandra.

När flera städer har blivit inkluderade ser databasen ut som i figur 8.

```
 id | cityid | location                      | name        | population | state
----+--------+-------------------------------+-------------+------------+-------
  5 |   1008 |         -72.936114,42.182949 |   Blandford |       1240 |    MA
  1 |   1001 |         -72.622739,42.070206 |      Agawam |      15338 |    MA
  2 |   1002 | -72.51564999999999,42.377017 |     Cushman |      36963 |    MA
  4 |   1007 | -72.41095300000001,42.275103 | Belchertown |      10579 |    MA
  3 |   1005 | -72.10835400000001,42.409698 |       Barre |       4546 |    MA

(5 rows)
```

Figur 8. Översikt av innehållet i en tabell efter imatning av städer i Cassandra.

3.3.3 Funktionalitet

Syntaxen I CQLSH påminner om SQL. Skillnaderna mellan de två kommer fram i praktiken beroende på hur kolumnerna är konfigurerade i Cassandra. CQLSH använder sig av SELECT för att göra vissa förfrågningar men som standard stöds kommandot endast för kolumner som är av typen PRIMARY KEY.s

För att göra förfrågningar på andra kolumner måste de först få ett sekundärt index i den tabell de ligger i genom kommandot CREATE INDEX. När kolumnen som förfrågan kommer att köras på har ett index fungerar SELECT kommandot som vanligt, som syns i figur 9.

```
cqlsh:examen> CREATE INDEX ON cities(state);
cqlsh:examen> SELECT * FROM cities WHERE state='KS';

 id | cityid | location                      | name    | population | state
----+--------+-------------------------------+---------+------------+-------
  8 |  66514 |          -96.91005,39.169173 | Milford |        704 |    KS
  7 |  66512 | -95.54763699999999,39.203832 | Meriden |       2299 |    KS
  6 |  66510 |          -95.628986,38.50266 | Melvern |        856 |    KS

(3 rows)
cqlsh:examen>
```

Figur 9. Alla städer i Kansas i Cassandra.

Eftersom Cassandra är fokuserad på snabbhet blockerar den förfrågningar som potentiellt kan ha en negativ påverkan på databasens prestanda. T.ex. förfrågningen för vilka städer som har fler medborgare en 2000 går över gränsen. För dessa tillfällen måste det specificeras att det är okej om prestandan sjunker med ALLOW FILTERING, som visas i figur 10.

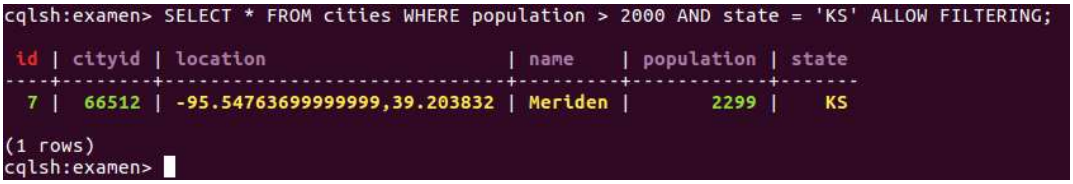
```
cqlsh:examen> SELECT * FROM cities WHERE population > 2000 ALLOW FILTERING;

 id | cityid | location                      | name        | population | state
----+--------+-------------------------------+-------------+------------+-------
  1 |   1001 |         -72.622739,42.070206 |      Agawam |      15338 |    MA
  2 |   1002 | -72.51564999999999,42.377017 |     Cushman |      36963 |    MA
  4 |   1007 | -72.41095300000001,42.275103 | Belchertown |      10579 |    MA
  7 |  66512 | -95.54763699999999,39.203832 |     Meriden |       2299 |    KS
  3 |   1005 | -72.10835400000001,42.409698 |       Barre |       4546 |    MA

(5 rows)
```

Figur 10. Alla städer med ett befolkningsantal över 2000 i Cassandra.

Samma problem med prestandan kommer fram när de två tidigare förfrågningarna kombineras med AND, och ALLOW FILTERING krävs för att förfrågningen skall godkännas. Figur 11 visar detta.



Figur 11. Alla städer i Kansas med ett befolkningsantal över 2000 i Cassandra.

3.3.4 Objektorienterade verktyg

Cassandra stöds av flera olika verktyg för objektorienterad utveckling. För Java och Python erbjuder Cassandra ett eget alternativ genom sin Object Mapper som är en lättviktig lösning för simpla transaktioner. (Java Driver for Apache Cassandra 3.2 2016)

Mer avancerade alternativ som t.ex. Achilles och Kundera för Java och Express-Cassandra för NodeJS finns tillgängliga.

3.4 MongoDB

För att kunna skapa databaser med MongoDB krävs endast installation av deras community edition från deras hemsida, alla verktyg som behövs för administrering är inbyggda i samma paket.

3.4.1 Struktur

Eftersom MongoDB använder sig av JSON behövs ingen omstrukturering av data.
T.ex. ser all information om staden Agawam ut så här:

```
{
 "city": "AGAWAM",
 "loc": [
  -72.622739,
  42.070206
 ],
 "pop": 15338,
 "state": "MA",
```

```
  "_id": "01001"
}
```

Informationen går alltså att importera rakt in till en databas i MongoDB, men om man inte har tillgång till redan existerande data och istället kommer att mata in all information manuellt måste man strukturera om all data till JSON.

3.4.2 Uppbyggnad

För manuell inmatning av data används Mongoshell som är ett inbyggt verktyg som används för administrering av databaser i MongoDB. Eftersom MongoDB är en dokumentorienterad databas sparas varje stad som ett eget dokument i databasen. Dokumenten sparas i collections som skapas automatiskt om de inte finns från tidigare så i praktiken sker inmatningen av en stad i databasen med följande kommando som visas i figur 12.

```
> db.cities.insert(
... { "city" : "AGAWAM", "loc" : [ -72.622739, 42.070206 ], "pop" : 153
38, "state" : "MA", "_id" : "01001" })
WriteResult({ "nInserted" : 1 })
> show dbs
admin  0.000GB
local  0.000GB
test   0.000GB
> show collections
cities
>
```

Figur 12. Skapandet av ett dokument i MongoDB.

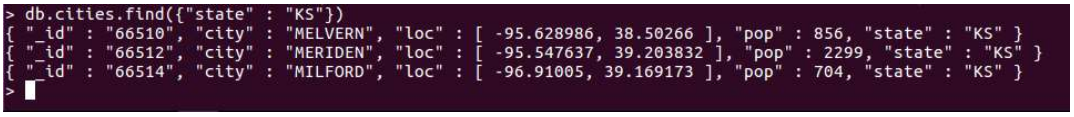
Eftersom cities inte fanns från tidigare skapades den automatiskt under testdatabasen. När flera städer har blivit tillagda i databasen kommer den i sin helhet att innehålla den information som syns i figur 13.

```
> db.cities.find()
{ "_id" : "01001", "city" : "AGAWAM", "loc" : [ -72.622739, 42.070206 ]
, "pop" : 15338, "state" : "MA" }
{ "_id" : "01002", "city" : "CUSHMAN", "loc" : [ -72.51565, 42.377017 ]
, "pop" : 36963, "state" : "MA" }
{ "_id" : "01005", "city" : "BARRE", "loc" : [ -72.108354, 42.409698 ],
 "pop" : 4546, "state" : "MA" }
{ "_id" : "01007", "city" : "BELCHERTOWN", "loc" : [ -72.410953, 42.275
103 ], "pop" : 10579, "state" : "MA" }
{ "_id" : "01008", "city" : "BLANDFORD", "loc" : [ -72.936114, 42.18294
9 ], "pop" : 1240, "state" : "MA" }
>
```

Figur 13. En översikt av dokument i MongoDB.

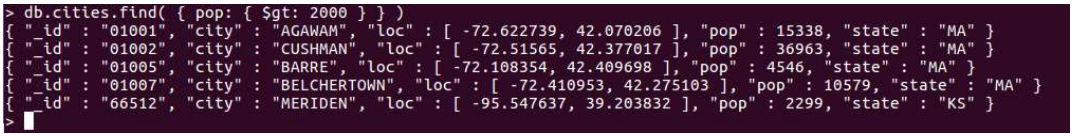
3.4.3 Funktionalitet

I MongoDB sköts förfrågningar via MongoShell. När man gör en förfrågan måste själva databasen först definieras varefter find({”state” : ”KS”}) används för att söka fram alla städer som finns i Kansas. Det här syns i figur 14.



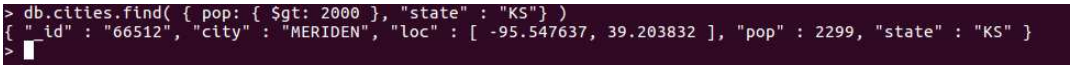
Figur 14. Alla städer i Kansas i MongoDB.

När en förfrågan kräver en jämförelse måste rätt operand väljas, i det här fallet används &gt som står för greater than för att få fram de städer med högre medborgarskap än 2000, som visas i figur 15.



Figur 15. Alla städer med ett befolkningsantal över 2000 i MongoDB.

När flera förfrågningar kombineras används ett kommatecken. Nedan körs de två tidigare kommandona samtidigt i figur 16.



Figur 16. Alla städer i Kansas med ett befolkningsantal över 2000 i MongoDB.

3.4.4 Objektorienterade verktyg

Eftersom MongoDB är ett av de mer populära NoSQL-databaserna finns det stöd för de flesta programmeringsspråken när det kommer till objektorienterade lösningar genom drivrutiner. Mongoid är deras officiella ramverk som använder Ruby. (Mongoid Manual 2016)

Det finns fler andra ramverk som t.ex. Mongoengine för Python och Mongoose för NodeJS.

3.5 Neo4j

För att köra och administrera Neo4j krävs det endast att man installerar mjukvaran från ett .tar paket från deras hemsida. Själva administreringen görs via ett inbyggt verktyg som är tillgängligt via en browser.

3.5.1 Struktur

Neo4j använder sig av noder för att spara data, en stad är alltså en nod. Noderna samlas under en etikett med olika variabelnamn för att skilja på dem. De olika städerna är anslutna genom relationer som definieras skilt t.ex. om det finns i samma stat eller om deras medborgarantal överskrider ett visst antal.

3.5.2 Uppbyggnad

För att skapa noderna använder sig Neo4j av Cypher som är deras eget språk för förfrågningar. Cypher utvecklades för att göra förfrågningar simplare än motsvarande SQL varianter så att Neo4j i praktiken kunde användas mer effektivt. Strukturen är baserad på SQL men Cypher påstås vara mer användarvänligt. (The Neo4j Developer Manual v3.2. 2017)

Inmatningen av Agawam sker enligt följande och visas i det grafiska verktyget i figur 17.

```
$ CREATE (ag:City { name: "Agawam", location: "-72.622739,42.070206",
  population: 15338, state: "MA", cityID: 01001 })
```

Figur 17. Inmatning av en stad i Neo4j.

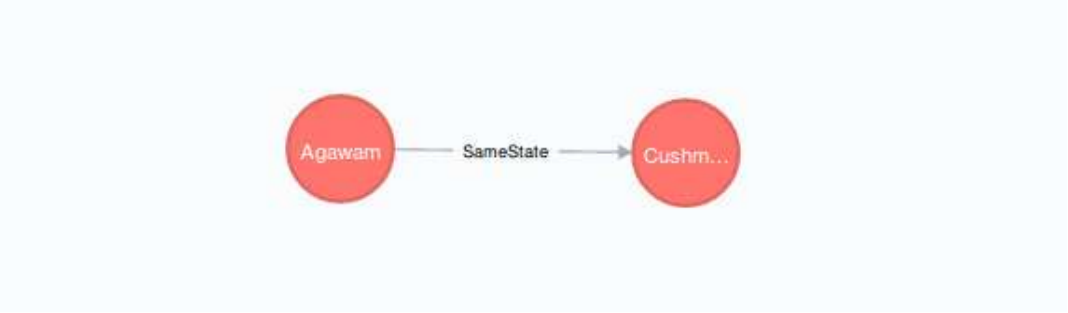
Etiketten i det här fallet heter City med variabelnamnet ag.

För att skapa relationer måste noderna i fråga först paras ihop med MATCH.
Eftersom större databaser kan ha flera noder under samma etikett med samma variabelnamn t.ex. två personers initialer, måste noderna som skall matchas identifieras vidare med WHERE kommandot. När noderna är identifierade används CREATE för att skapa själva relationen, i det här fallet skapas en relation mellan Agawam och Cushman som berättar att de befinner sig i samma stat, vilket syns i figur 18.

```
MATCH (ag:City),(cu:City)
WHERE ag.name = "Agawam" AND cu.name = "Cushman"
CREATE (ag)-[:SameState]->(cu)
```

Figur18. Skapandet av en relation i Neo4j.

Efter att relationen är skapad visualiseras databasen som en graf i figur 19.

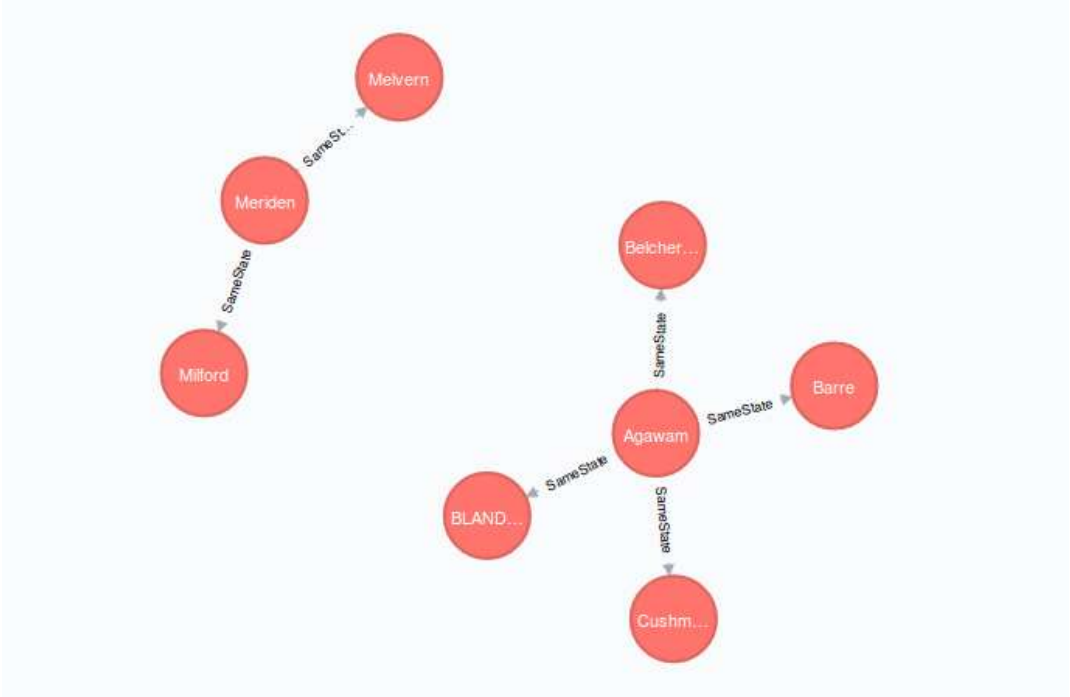


Figur 19. Visualisering av en relation i Neo4j.

Relationen i det här fallet är enkelriktad fast den egentligen är sann åt båda riktningarna. Orsaken till detta är att det anses onödigt för relationen att gå åt båda hållen eftersom Cushman inte kan existera i en annan stat på grund av att vi redan vet att den finns i samma som Agawam, på detta sätt ökas prestandan på databasen.

3.5.3 Funktionalitet

En av de egenskaper som definierar Neo4j är att den alltid visualiserar sina förfrågningar som en graf. Nedan i figur 20 syns hela databasen.



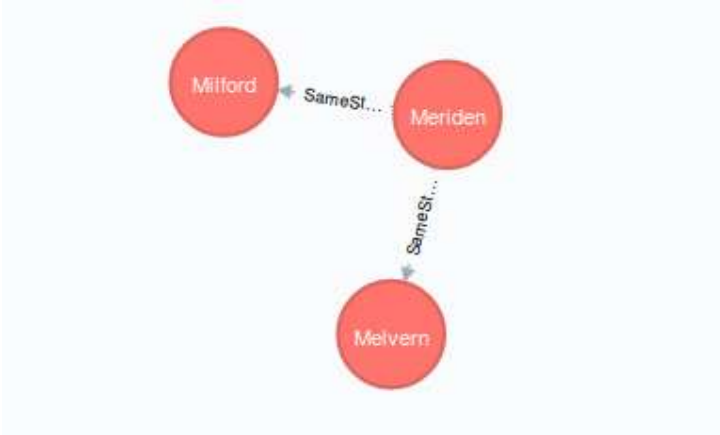
Figur 20. En översikt av databasen i Neo4j.

De olika städerna har relationer som berättar i vilken stat de befinner sig i.

För att göra en förfrågning som visar de städer som finns i Kansas använder Cypher sig av MATCH, som syns i figur 21. Följande förfrågan matchar alltså alla städer med ”state: ’KS’” under variabeln n som sedan returneras, vilket visas i figur 22.

```
1 MATCH (n:City { state: 'KS' })
2 RETURN n
```

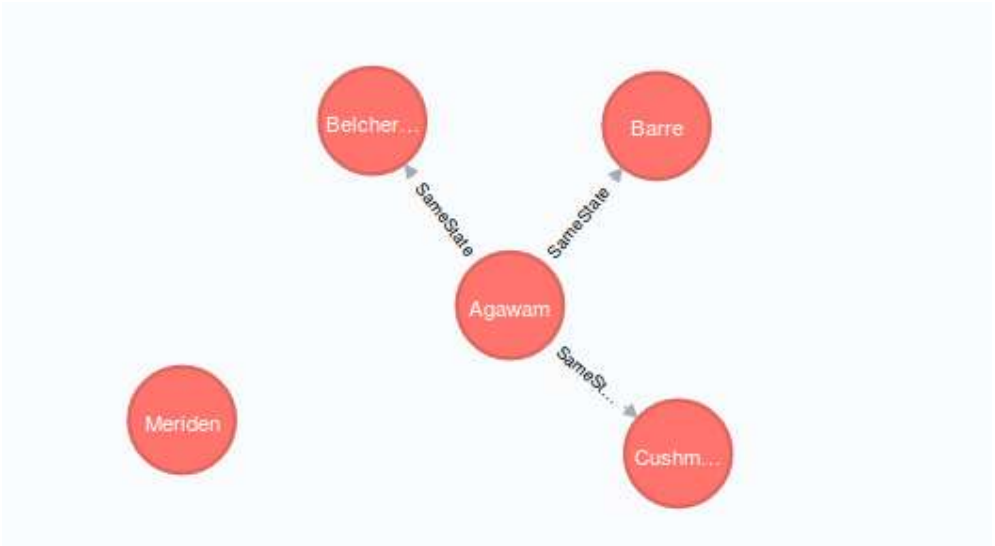
Figur 21, En match förfrågan i Cypher.



Figur 22. Alla städer i Kansas i Neo4j.

När en förfrågan måste jämföra någonting istället för att matcha ett exakt värde måste WHERE användas. Syntaxen kräver att samma variabel anges för att förfrågningar skall lyckas. Nedan i figur 23 visualiseras resultatet för de städer med över 2000 invånare.

```
1 MATCH (n:City)
2 WHERE n.population > 2000
3 RETURN n
```



Figur 23. Alla städer med ett befolkningsantal över 2000 i Neo4j.

Cypher tillåter de två tidigare kommandona att kombineras utan ändringar, vilket visas i figur 24. Resultatet av det här syns i figur 25.

```
MATCH (n:City {state: 'KS'})
WHERE n.population > 2000
RETURN n
```

Figur 24, förfrågningen för alla städer i Kansas i Cypher.

Figur 25. Alla städer i Kansas med ett befolkningsantal över 2000 i Neo4j.

3.5.4 Objektorienterade verktyg

Neo4j OGM är ett officiellt ramverk som erbjuds för Java. Det erbjuds också officiella drivrutiner för Python, JavaScript och .NET (Neo4j OGM Manual v3.1. 2017).

Förutom dessa finns det ett flertal andra verktyg som t.ex. Neode för NodeJS och py2neo för Python.

4 PRESTANDAJÄMFÖRELSE

Det här kapitlet jämför prestandan av de olika NoSQL-databaserna mot relationsbaserade varianter. Informationen är tagen från tidigare studier samt forskningar var man har kört olika belastningstest på databaser.

4.1 Cassandra

I en studie år 2016 testades Cassandra 3.4 mot SQL Server 2012 på en server med endast en processor, data som testades bestod av en miljon rader med 26 kolumner av olika datatyper. I det första testet importerades data in i båda databaserna tre gånger. I medeltal tog det 321 sekunder för Cassandra medan SQL Server klarade av samma på bara 34,42 sekunder.

I nästa test exporterades data från databaserna vilket för Cassandra i medeltal tog 322,6 sekunder men bara 20,3 för SQL Express. Tredje testet gick ut på att plocka ut ett antal rader från databaserna. Testet kördes med 5000, 10000 och 15000 rader och i alla tre fall var SQL Server snabbare, speciellt när antalet rader ökade. Studien visar att relationsbaserade SQL Server klarar sig bra på en traditionell server medan Cassandras prestanda stryps. (Mahmood, K. 2016 s.23-24)

I ett annat test år 2013 testades prestandan på olika databaser, bland dessa Cassandra 1.1.2 och SQL Server Express 10.50.1600.1. Testet använde sig av key-value par som lagrades i databaserna. I ett test där läsförmågan testades visade det sig att SQL Express presterade bättre, 10 par hämtades av Cassandra på 115ms och av SQL Express på 13ms. När antalet par ökades till 100000 klarade Cassandra av det på 228096ms medan det bara tog 17214ms för SQL Express. Skrivförmågan testades likadant, 10 par tog 117ms för Cassandra och 30ms för SQL Express, men när antalet ökades till 100000 tog det bara 88197ms för Cassandra medan SQL Express klarade av det på 216479ms. När det kom till skrivförmåga presterade alltså Cassandra bättre en SQL Express när datamängden ökade. (Yishan, L & Sathiamoorthy, M. 2013 s.17-18)

År 2012 gjorde University of Toronto en studie var databasers prestanda testades genom att se hur många operationer per sekund de klarar av i olika situationer med olika antal noder, varje nod bestod av 10 miljoner uppgifter. I den första situationen testades både läs- och skrivförmågan med 95% läsningar och 5% skrivningar under 10 minuter. Med en nod klarade både Cassandra och MySQL av ungefär 25000 operationer per sekund. Med 12 noder presterade Cassandra en aning bättre med ungefär 180000 operationer per sekund i motsats till 160000 för MySQL.

I nästa test ökades antalet skrivningar till 50%. Med en nod ökade Cassandras prestanda med 10% medan den sjönk med 33% för MySQL. Cassandras prestanda ökade jämnt enda upp till 12 noder medan effektiviteten av MySQL hölls på samma nivå efter 8 noder. När antalet skrivningar ökades till 99% kom det fram att MySQL presterar bäst med en nod med ungefär 19000 operationer per sekund men att effektiviteten inte ökar med fler noder. Cassandra klarade av ungefär 5000 operationer per sekund och prestandan ökade

jämnt desto fler noder som användes, och med 12 noder hade antalet operationer per sekund ökat till över 50000. (Rabl, T. et al. 2012 s.1729-1733)

Cassandras effektivitet över relationsbaserade varianter beror till stor del på hårdvaran. En relationsbaserad databas fungerar bra på en traditionell server men när datamängden ökar och databasen därför måste skalas uppåt med t.ex. molnteknologi presterar Cassandra betydligt bättre.

4.2 MongoDB

År 2015 gjordes en studie var MongoDB 2.6.3 testades mot SQL Server 2005 Express. Data bestod av information om vattenkvalitet tagen från Taiwan Water Corporation. Både databaserna installerades på en traditionell server med en processor. När skrivförmågan testades klarade MongoDB av 10000 skrivningar på 452ms och en miljon skrivningar på 53516ms, för SQL Server tog det däremot 3603ms för 10000 och 329787ms för en miljon skrivningar. När samma test gjordes med läsningar klarade MongoDB av 10000 på 1294ms och en miljon på 155865ms medan det för SQL Server tog 11919ms för 10000 och 1184857 för en miljon. MongoDB presterade alltså bättre i båda fallen speciellt när mängden operationer ökade. (Wu, C. et al. 2015 s.39-40)

Ett annat prestandatest samma år kom fram till liknande resultat. Testet kördes på en processor mellan MongoDB och MS SQL Server. Först testades förmågan att lägga in data. Båda databaserna presterade likadant enda upp till 10000 operationer, efter detta sjönk SQL Servers effektivitet. Vid en miljon operationer tog det 76ms för MongoDB medan det tog 241,88ms för SQL Server. Förmågan att ta bort data testades likadant, MongoDB klarade av en miljon operationer på 75ms motsatt 1258ms för SQL Server. Liknande resultat kom fram när man testade att uppdatera existerande värden. Enda fallet då SQL Server lyckades prestera bättre en MongoDB var när testen kördes med aggregerade förfrågningar. (Aboutorabia, S. et al. 2015 s.5-6)

Den dokumentbaserade modellens förmåga att lagra och ta bort data, samt läs- och skrivförmåga fungerar snabbare än relationsbaserade variationer även på traditionell serverhårdvara. En svaghet hittas dock i förmågan att hantera mer komplicerade förfrågningar där relationsbaserade modeller presterar bättre.

4.3 Neo4j

I ett prestandatest år 2010 testades Neo4j mot MySQL. I testet användes olika förfrågningar som kördes på ett antal databaser som innehöll olika information. I förfrågningar som gick ut på att komma åt själva noderna i databaserna presterade Neo4j i genomsnitt bättre en MySQL. Att t.ex. plocka ut 128 noder och sedan räkna hur många andra noder var tillgängliga, i databaser som innehöll 10000 noder bestående av klartext, tog 4,3ms för Neo4j och 37,4ms för MySQL. I medeltal var Neo4j 10 gånger effektivare en MySQL. Ett undantag var förfrågningar som gick ut på att leta upp alla ensamstående noder, i dessa fall var resultatet relativt jämnt.

Eftersom Neo4j behandlar all data som klartext istället för en specifik datatyp presterade MySQL bättre när det kom till matematiska förfrågningar i databaser som innehöll heltal. T.ex. att räkna alla noder vars värde var mindre en ett annat i en databas med 10000 noder tog 34,8ms för Neo4j men bara 0,6ms för MySQL. (Vicknair, C. et al. 2010 s.4)

En nyare studie från 2017 fann också att Neo4j presterar bättre när det gäller att komma åt själva noderna men att relationsbaserade varianter är effektivare när det gäller analytiska förfrågningar. T.ex. en förfrågan som returnerar alla par av noder som är direkt sammanbundna tog ungefär 10 gånger längre för Neo4j men att relationsbaserade varianter var en hel del långsammare när det gällde att söka fram specifika mönster mellan noderna. (Hölsch, J. et al. 2017 s.4-5)

Grafiska databaser presterar bra när det gäller att hitta specifika mönster mellan noder vilket ger dem ett användningsområde i t.ex. logistik eller social media. Relationsbaserade varianter är fortfarande effektivare när det gäller analytisk behandling av själva informationen som finns i databasen.

5 SLUTSATS

Syftet med arbetet var att ta reda på vad olika NoSQL varianter är samt hur de i praktiken strukturerar data och behandlar den jämfört med MySQL. Ett annat syfte var att ta reda på hur de olika NoSQL varianternas prestanda skiljer sig från relationsbaserade varianter.

I arbetet har jag gått igenom själva skapningen av de olika databastyperna samt tagit upp prestandajämförelser.

De olika NoSQL varianterna har skapats på grund av brister i relationsdatabaser och är ofta menade för ett eller flera specifika användningsområden och för att mängden data som måste sparas och hanteras ökar hela tiden. Ett annat viktigt designkrav är förmågan att skala databasen uppåt kostnadseffektivt och utan att skära ned på prestandan. I detta syfte har de olika NoSQL-databaserna lyckats bra, både kolumnbaserade Cassandra och dokumentbaserade MongoDB klarar av att lagra och hantera stora mängder data effektivt genom att använda en simplare datastruktur. Grafbaserade Neo4j presterar bra inom sitt specifika användningsområde när det gäller att hitta rutter eller mönster mellan noder. Största nackdelen är att det kan hända misstag när data lagras eller hanteras eftersom ACID stöd har lämnats för att förhindra att prestandan sjunker.

Arbetets mål att jämföra datastrukturen, samt hur data behandlas, av de olika varianterna har uppfyllts. Målet att jämföra prestandan visade sig vara komplicerat på grund av NoSQL-databasernas specifika användningsområden samt mängden data som behövs. Kapitel 4 har åtminstone visat att NoSQL-databaserna fungerar bra, och för det mesta bättre, när de används för det som de är menade för.

NoSQL-databaser har definitivt en plats i framtiden på grund av deras skalbarhet och kostnadseffektivitet men för kritisk data inom en traditionell servermiljö fungerar relationsbaserade varianter fortfarande väldigt bra.

KÄLLOR

About Apache Cassandra. 2016. Tillgänglig: http://docs.datastax.com/en/cassandra/3.0/cassandra/cassandraAbout.html
Hämtad 2.6.2017

Wu, C., Huang, Y., Lee, J., 2015, Comparisons between MongoDB and MSSQL Databases on the TWC Website, *American Journal of Software Engineering and Applications*, Vol. 4, No. 3

Hölsch, J., Schmidt,T., Grossniklaus, M., 2017, On the Performance of Analytical and Pattern Matching Graph Queries in Neo4j and a Relational Database, *Workshop Proceedings of the EDBT/ICDT 2017 Joint Conference*

Java Driver for Apache Cassandra 3.2. 2016. Tillgänglig: https://docs.datastax.com/en/developer/java-driver/3.2/manual/
Hämtad 5.12.2018

Mahmood, K., 2016, Performance Comparison of NOSQL Database Cassandra and SQL Server for Large Databases, *Journal of Independent Studies and Research – Computing,* Vol. 14, Nr. 2

Mongoid Manual. 2016. Tillgänglig: https://docs.mongodb.com/mongoid/master/
Hämtad 5.12.2018

Moniruzzaman, A. B. M. & Hossain, S. A., 2013, NoSQL Database: New Era of Databases for Big data Analytics - Classification, Characteristics and Comparison, *International Journal of Database Theory and Application*, Vol. 6, Nr. 4

MySQL 5.7 Reference Manual. 2017. Tillgänglig: https://dev.mysql.com/doc/refman/5.7/en/
Hämtad 1.6.2017

Rabl, T., Sadoghi, M., Jacobsen, H., Gomez-Villamor, S., Muntes-Mulero, V., Mankovskii, S., 2012, Solving Big Data Challenges for Enterprise Application Performance Management, *Proceedings of the VLDB Endowment*, Vol. 5, Nr. 12

Schlegel, D., Bona, J., Elkin, P., 2015, Comparing Small Graph Retrieval Performance for Ontology Concepts in Medical Texts, *VLDB Workshop on Big Graphs Online Querying*, s.32–44.

Aboutorabi, S. H., Rezapour, M., Moradi, M., Ghadiri, N., 2015, Performance evaluation of SQL and MongoDB databases for big e-commerce data, *2015 International Symposium on Computer Science and Software Engineering (CSSE)*

The MongoDB 3.4 Manual. 2016. Tillgänglig: https://docs.mongodb.com/manual/
Hämtad 2.6.2017

The Neo4j Developer Manual v3.2. 2017. Tillgänglig: https://neo4j.com/docs/developer-manual/3.2/
Hämtad 8.10.2018

MySQL Workbench 2018. Tillgänglig: https://www.mysql.com/products/workbench/
The Neo4j OGM Manual v3.1. 2017. Tillgänglig: https://neo4j.com/docs/ogm-manual/3.1/
Hämtad 5.12.2018

The Neo4j Operations Manual v3.2. 2017. Tillgänglig: https://neo4j.com/docs/operations-manual/3.2/
Hämtad 3.6.2017

Vicknair, C., Macias, M., Zhao, Z., Nan, X., Chen, Y., Wilkins, D., 2010, A Comparison of a Graph Database and a Relational Database, *Proceedings of the 48th Annual Southeast Regional Conference*

Yishan, L. & Sathiamoorthy, M., 2013, A performance comparison of SQL and NoSQL databases, *Conference: Communications, Computers and Signal Processing (PACRIM)*