NoSQL databaser inom webbapplikationer

Jonathan Wiklund, 33264

Kandidatavhandling I datateknik

Åbo Akademi

Institutionen för informationsteknologi

Handledare: Mats Aspnäs

2012

Referat

Denna avhandling undersöker hur webbsidor kan utnyttja sig av NoSQL-databaser för att uppnå högre skalbarhet och prestanda, samt vad som kan vara bra ur en utvecklares synpunkt.

Webbsidor som får miljoner träffar per dag (sociala medier, Google, Amazon mm.) måste leverera stora mängder data åt användarna hela tiden. Utöver detta måste transaktioner mellan användare och webbsidor gå snabbt och vara korrekta.

Först kommer NoSQL-databasers gemensamma egenskaper att beskrivas och jämföras kort, varefter det kommer presentera mera ingående Googles BigTable och deras plattform Google App Engine. Sist diskuteras hur och när det passar sig att använda NoSQL-databaser och hur framtiden kan komma att se ut.

Sökord: NoSQL, Google BigTable, Google App Engine

Innehållsförteckning

1. Introduktion

1.1 Relationsdatabaser

Relationella databaser har sedan 1970-talet varit den regerande typen av databas i användning. Med uppkomsten av internet och webbsidor som lagrade sina innehåll i databaser blev relationsdatabaser snabbt standard och MySQL blev det mest populära databassystemet [1] . Andra kända relationsdatabaser är Oracle, Microsoft SQL Server, IBM DB2, PostreSQL och SQLite.

Internet har växt mycket under 2000-talet och t.e.x. Facebook besöks dagligen av över 400 miljoner användare [2]. Detta innebär att de underliggande databaserna för de populära sidorna måste kunna servera mera förfrågningar hela tiden.

1.2 Behov av alternativa databaser

Webb 2.0 är webbsidor och applikationer, där innehållet till stor del är genererade av användarna. Användare vill dela med sig av bilder och videor, kommentera nyheter, spara filer till moln-baserade lagringssystem, eller bara annars kommunicera med varandra i diverse forum.

De populäraste webbsidorna får miljontals visningar per dag. Användare på dessa sidor laddar upp videor och bilder, delar kommenterar på nyheter, söker videor eller annan relevant information. Detta sätter hård press på servrarnas databaser, som hela tiden måste uppdatera, radera eller hämta ut data. Säkerheten måste också beaktas så att inte obehöriga kommer åt användares delningar eller i värsta fall lösenord. Utöver detta så skall responstiden från att en användare gör en förfrågan till att innehållet visas skärmen, vara så kort som möjligt för att ge användaren en bra upplevelse.

Eftersom relationella databaser är svåra att skala horisontellt (att sprida ut databasen på flera servrar) så är det inte kostnadseffektivt att utöka databasernas kapacitet och prestanda efter en viss punkt [3]. Detta har lett till att aktörer som Amazon, Google,

Facebook m.fl har utvecklat och flyttat över till icke-relationella databaser, som skalar bättre horisontellt[3]. Dessa icke-relationella databaser kallas ofta för NoSQL databaser.

1.3 Benämningen NoSQL

Carlo Strozzi som 1998 döpte sin relationella databas till NoSQL (eftersom den inte använde sig av det strukturerade frågespråket SQL), har uttalat sig om att dagens så-kallade NoSQL databaser egentligen borde kallas för NoREL-databaser eller dylikt, eftersom de ofta inte använder sig av den relationella modellen.[4] Då NoSQL används i något sammanhang i denna avhandling kommer det att referera till de icke-relationella databaser som uppkommit under mitten av 2000-talet och framåt.

NoSQL står för Not only SQL[5] och är ett samlingsbegrepp för de icke-relationella databaser, som har blivit allt mera populära i och med prestandakraven som uppstått då antalet användare på webben har ökat.

2. NoSQL

2.1 Vad är NoSQL?

Kännetecknande för NoSQL-databaser är att de oftast inte lagrar data i relationstabeller. De flesta NoSQL-databaserna är skräddarsydda för att uppfylla speciella krav, tex snabba svar på förfrågningar, stora mängder data, varierande typer av data eller skalbarhet. Detta har lett till att det i dagsläget finns över hundra olika NoSQL-databaser att välja mellan beroende på specifikationskrav. [6]

För att uppnå de olika krav som sätts på en NoSQL-databas så kan inte ACID-garantierna uppfyllas helt. [7]

ACID (Atomicity, Consistency, Isolation, Durability) är ett garantikrav för transaktioner inom databashantering. *Odelbarhet* (eng. Atomicity) innebär att en transaktion endera utförs helt, eller inte alls. *Konsistens* (eng. Consistency) innebär att en transaktion inte får bryta någon av databasens regler. *Isolation* betyder att varje transaktion utförs oberoende av varandra och att två transaktioner inte samtidigt arbetar på samma data. *Hållbarhet* (eng. Durability) står för att en transaktion består efter att den har utförts.

ACID-egenskaperna garanteras inte på alla NoSQL-databaser[8] och därför används även CAP för att modellera garantikrav. CAP (Consistency, Availability and Partition Tolerance) är ett teorem som säger att man i ett distribuerat system vid en viss tidpunkt endast kan välja två av garantierna *konsistens*, *tillgänglighet* och partitionstålighet.

För att uppnå hög tillgänglighet och partitionstålighet och även viss konsistens, använder sig vissa NoSQL-databaser av en konsistensmodell kallad BASE (Basically Available, Soft state, Eventual Consistency), vilket innebär att alla utförda transaktioner förr eller senare blir tillgängliga på alla noder i ett distribuerat system.

2.2 Uppkomsten av NoSQL och dess utveckling

Databaser hos sociala nätverk och sociala media-sidor såsom YouTube, Twitter, Facebook, Digg, samt andra webbsidor såsom Googles tjänster och Amazon måste hantera stora mängder data och servera den snabbt åt miljontals användare dagligen. [4] Webb 2.0 sidorna och applikationerna blev under mitten på 2000-talet mycket populära och användarna på internet blev flera. Detta ledde till att de relationella databaserna som de stora aktörerna använde sig av blev otillräckliga med avseende på storlek och responstider. Traditionella relationella databaser har också problem med att smidigt servera varierande filtyper (bilder,videor,textfiler etc.). [3]

Eftersom de traditionella relationella databaserna nådde sina begränsningar hos de mest populära sidorna så påbörjades utvecklingen av alternativa databassystem, som skulle kunna hantera de olika krav som ställdes av utvecklarna. Google tog fram BigTable[9], Amazon utvecklade DynamoDB[5] och Facebook skapade Cassandra [3].

BigTable och DynamoDB finns inte fritt tillgängliga. Därför har det utvecklats NoSQL-databaser med öppen källkod som emulerar beteenden hos de stora aktörernas databaser. Dessa är tex Hypertable och HBase som är kloner av BigTable eller Dynomite som är en implementation av Dynamo.[10]

Google och Amazon erbjuder sina användare såkallade molnplatformer, där man kan ha sina applikationer serverade från deras moln, och man behöver således inte bry sig om hur applikationen skalar, utan detta sköts automatiskt. Dessa heter Google App Engine och Amazon SimpleDB och de är gratis för användare upp till en viss gräns. Om man överskrider gränserna blir man tvungen att betala avgifter beroende på användning. Respektive begränsningar hittas på Googles[11] och Amazons[12] webbsidor

2.3 Olika typer av NoSQL

Det finns varierande typer av NoSQL databaser, men det existerar dock ingen officiell kategorisering av dessa. Wikipedia delar till exempel in NoSQL-databaserna i följande kategorier: *Document-store*, *Graph*, *Key-value Store* (med diverse underkategorier), *Multivalue databases*, *Object Database*, *Tabular, Tuple store* och *RDF database*. [13] Det finns även andra försök på att kategorisera NoSQL-databserna [7] . Tillnäst kommer kort att beskrivas hur följande databastyper är uppbyggda och vad som är kännetecknande för dom: *Key-Value store, Document-oriented, Graph, Object database.*

2.3.1 Key-Value store

Nyckel-Värde-databaser är uppbyggda så att data sparas med unika nycklar, som pekar på en eller flera värden. Dessa värden kan vara av varierande typ och behöver inte specificeras med ett schema, utan programkoden måste hålla reda på vilken typ av data som finns sparade i olika nycklar. Nyckel-värde databaser är snabba på att utföra förfrågningar eftersom de har så enkel struktur och de stöder oftast endast förfrågningar såsom ”lägg till” (eng. Put), ”hämta” (eng. Get) och ”ta bort” (eng. Delete). Pågrund av denna enkla struktur är nyckel-värde databaserna bra på att hantera stora datamängder och enkla att skala horisontellt.

Eftersom nyckel-värde databaser är så simpla så går det inte att utföra avancerade förfrågningar, och sökningar efter värden går endast om indexering av dessa har implementerats.

2.3.2 Document-oriented

Dokumentbaserade databaser sparar sina värden i såkallade dokument, som i sin tur kan innehålla varierande mängder data. Dessa dokument nås via unika nycklar som oftast är en enkel sträng. Dokumentbaserade databaser kan utöver att hämta dokument via nycklar, också hämta ut dokument utifrån de värden som dom

innehåller. Strukturen är ”semi-strukturerad”, vilket betyder att värden som sparas till samma typer av dokument kan vara av varierande storlek och längd.

2.3.3 Graph

Grafdatabaser är uppbyggda av punkter (även kallade noder eller hörn) som innehåller egenskaper och är sammanbundna med linjer (även kallade bågar eller kanter). Två noder är alltid sammankopplad med en linje och kan även innehålla attribut som beskriver relationen mellan de två noderna. Grafdatabaserna utnyttjar sig av grafteorin från matematiken för att göra sökningar och hitta samband mellan noder. På grund av strukturen i grafdatabaserna så är dessa väldigt bra på att hantera många-till-många förhållanden mellan enititeter (objekt som representerar något)

Grafdatabaser kan vara bra för e-handel sidor, där man enkelt med hjälp av dessa kan rekommendera liknande produkter eller tipsa om produkter som blivit köpta samtidigt som den produkt det är fråga om.

2.3.4 Object database

Objektorienterade databaser representerar data som objekt och ämnar sig bra för avancerade strukturer av data. Utvecklare som skriver objekt-orienterade applikationer kan enkelt spara objekt direkt till databasen. Med en relationell databas måste utvecklaren i applikationskoden översätta informationen före den sätts in i databasen. Objekt-orienterade databaser är oftast skrivna i specifika språk och har en tät integration med applikationskoden.

2.4 Fördelar och nackdelar med NoSQL

En av de största fördelarna med NoSQL i allmänhet är att de är snabba, kostnadseffektiva och skalar bra horisontellt.[14] Kostnadseffektiviteten beror på att man istället för att skaffa nya och bättre serverdatorer, skaffar flera serverdatorer och delar upp arbetsmängden på dessa.

Eftersom en stor del av NoSQL databaserna är schemalösa och har sin struktur definerad i applikationskoden, så kan migrering vid införning av nya funktioner gå smärtfritt och utan att man behöver ha databasen ur funktion en lång tid.[10]

De enkla strukturerna i NoSQL databaser har också lett till att man kan skala mycket förutsägbart, vilket betyder att man vet ganska exakt hur mycket processorkraft och arbetsminne som går åt för att utföra olika förfrågningar. Relationella databaser är oftast komplext uppbyggda och därför kan man aldrig med säkerhet förutspå hur förfrågningar kommer att skala. [10]

NoSQL har fått kritik för att det inte finns ett standard förfrågningsspråk, vilket kan skrämma bort nya utvecklare. 2011 uppkom dock UnQL (Unstructured Query Language), som är en specifikation av ett unifierat förfrågningsspråk för NoSQL databaser.[15]

Att det finns över hundra olika NoSQL-databaser [6] ger en stor valmöjlighet åt utvecklare, som således enkelt kan välja en databas som passar applikationens specifikationer.

3. BigTable

Som exempel på en NoSQL-databas presenteras BigTable, som är utvecklad av Google och som är används i en stor del av deras tjänster. BigTable är designad för att hantera petabyte mängder data[9], och man har som utvecklare tillgång till den via Google App Engine.[16] Google App Engine är bra dokumenterad och lätt att förstå även som ny utvecklare [17]

Google var en av de första aktörerna som började skapa sina egen NoSQL databaser. [8]

3.1 Historia

BigTable började utvecklas i början av 2004 och togs i aktivt bruk hos Google runt februari 2005.[18] Bigtable sparar sin data i rader och kolumner, där varje cell kan ha attribut om tiden då datan skapats. Cellerna kan även innehålla flera versioner av datan.

BigTable är skriven i C++ och arbetsmängden som gick åt för att skapa den beräknas till ungefär 7år. Att BigTable är skriven i C++ ger den bättre prestanda än tex HBase som är skriven i Java.

3.2 Struktur

I Googles vetenskapliga artikel om BigTable [9], så beskrivs Bigtable som en gles, distribuerad, beständig multi-dimensionellt sorterad map-struktur. Vad detta innebär kommer att beskrivas mera ingående i denna sektion.

3.2.1 Gles

I en traditionell databas lagras data till disken radvis, medan BigTable lagrar datan kolumnvis (se figur 3.2-1 och 3.2-2). Detta medför att rad-orienterade databaser måste spara NULL-värden för att hålla strukturen, och således ämnar sig sämre för varierande strukturer.

Id	Namn	Ålder	Intressen
1	Olle	19	Fotboll
2	Toni	25	NULL
3	Markus	NULL	Jakt

Figur 3.2-1: Data sparad radvis

Id	Namn
1	Olle
2	Toni
3	Markus

Id	Intressen
1	Fotboll
3	Jakt

Id	Ålder
1	19
2	25

Figur 3.2-2: Data sparad kolumnvis, inga NULL-värden som tar upp plats på disk

3.2.2 Distribuerad

Databasens underliggande filsystem är utspritt på flera maskiner och kan hantera att maskiner sätts till, tas bort eller faller bort. Det underliggande filsystemet för BigTable är GFS (Google File System). GFS är optimerat för att ha hög feltolerans, ta vara på systemresurser och kunna servera filer snabbt åt en stor mängd klienter. GFS används i en stor mängd av Googles tjänster som underliggande filsystem [19]

BigTable har sina tabeller partitionerade (delade) enligt rader på intervall av ungefär 200MB, dessa delningar kallas tablets och en maskin har hand om cirka 100 av dessa. Orsaken till denna uppdelning är för att få finkornig lastbalansering, till exempel kan en tablet som får många förfrågningar flyttas till en snabbare maskin eller så flyttas andra tablets bort från maskinen så att den kan fokusera mera resurser på den belastade tableten. Om en maskin går ner så går det snabbt att bygga upp tabellerna pånytt i och med att de tablets som maskinen hade hand om kan tas upp av andra maskiner i klustret.

3.2.3 Beständig

Data som sätts in i BigTable hålls kvar ända tills en transaktion tar bort eller ändrar på den.

3.2.4 Multi-dimensionellt sorterad map-struktur

Map (härefter karta) i detta sammanhang är en associativ räcka, en datatyp som innehåller samlingar av nycklar ihopparade med värden. Dessa nycklarna är unika inom samlingen (Se figur 3.2-3) [20]

```
{
   "x" : "2",
   "y" : "2"
}
```

Figur 3.2-3: En punkt beskriven som en karta

Multi-dimensionellt sorterad map-struktur betyder att kartan har flera dimensioner. Detta innebär att en nyckel i en karta kan peka på en annan karta. (se figur 3.2-4)

```
{
   "radie" : "2",
   "punkt" : {
      "x" : "2",
      "y" : "2"
   }
}
```

Figur 3.2-4: En cirkel beskriven som en karta med flera dimensioner, där ”punkt” är en karta över koordinaterna för mittpunkten i cirkeln.

I Bigtable är dimensionerna: *rader*, som är kartor över *kolumnfamiljer*, som i sin tur är kartor över *kolumner* som i sin tur innehåller olika versioner av datan.

3.3 Strukturen beskriven/Illustrerad

BigTable byggs i grunden upp av rader och kolumner. Radernas nycklar är strikt alfabetiskt ordnade och pekar på kolumnfamiljer som innehåller kolumner. Orsaken till att nycklarna är strikt alfabetiskt ordnade är att man vill ha liknande data nära varandra i och med partitioneringen (Sektion 3.2.2). I exempelvis Googles sökmotorindexering (WebTable), så sparas webbsidors domänadresser ”omvända” in i databasen. Detta betyder att adressen ”mail.google.com” sätts in i databasen som ”com.google.mail” och således finns den nära andra adresser på samma domän när liknande data ska hittas.[9]

Nedan kommer strukturen beskrivas stegvis börjandes med endast rader och kolumnfamiljer, vartefter kolumner sätts till och tillsist tidsaspekten.

```
{
   "Rad1" : {
      "KolumnFamilj1" : "x",
      "KolumnFamilj2" : "y"
   },
   "Rad2" : {
      "Kolumnfamilj3" : ”z”
   }
}
```

Raderna i BigTable består av godtyckliga strängar som kan vara upp till 64KB stora. Dessa rader består av kolumnfamiljer som kan ha varierande mängder kolumner i sig

Kolumnfamiljerna måste specificeras före kolumner sätts till dom. Helst ska inte kolumnfamiljer ändras på under körning, och antalet distinkta kolumnfamiljer ska vara max några hundra stycken. Kolumnnycklar sätts enligt följande: familj:kolumnnamn. All data i en kolumnfamilj ska helst vara av samma typ eftersom BigTable utför komprimering av data i kolumnfamiljerna.

Information om vem som har tillgång till data och hur datan ska sparas (till minne eller till disk) kan specificeras skiljt för varje kolumnfamilj.

```
{
      "Rad1" : {
            "KolumnFamilj1" : {
                  "Kolumn1" : "x",
                  "..." : "...",
                  ”KolumnN" : "N"
            },
            "KolumnFamilj2" : {
                  "" : "y"
            }
      },
      "Rad2" : {
            "Kolumnfamilj3" : ”z”
      }
}
```

För att nå en specifik cell används rad/familj:kolumn.

Eftersom rader kan innehålla godtyckliga mängder av kolumner, så finns det inget inbyggt sätt att se hur många kolumner databasen innehåller, utan man måste göra en full genomgång av tabellen.

Man kan dock få reda på hur många kolumnfamiljer en databas innehåller eftersom dessa sällan ändras.

Sista dimensionen i BigTable är tidstämplar (eng. timestamps). Dessa tidstämplar medför att man i en kolumn kan ha olika versioner av datan. Tidstämplarna är av typen 64-bitars heltal och om inget skilt specificeras så sätter BigTable den automatiskt till ”realtid” i mikrosekunder. Om skilt specificeras så måste applikationskoden som utför transaktionen se till att tidstämplarna är unika för att undvika konflikt. BigTable har även skräphanterings-funktioner som kollar upp cellers olika versioner och tar bort onödiga enligt specifikation. Klienten kan specificera att X antal senaste versioner av datan sparas, eller att endast versioner som är nyare än en viss tidpunkt sparas (tex spara senaste månadens data).

```
{
      "Rad1" : {
            "KolumnFamilj1" : {
                  "Kolumn1" : {
                        "1” : "x",
                        "4" : "y"
                  },
                  "..." : "...",
                  ”KolumnN" : "N"
            },
            "KolumnFamilj2" : {
                  "" : "y"
            }
      },
      "Rad2" : {
            "Kolumnfamilj3" : ”z”
      }
}
```

Om förfrågningar körs på en cell så ger databasen alltid den senaste versionen av datan. Om man specificerar tiden (tex Rad1/KolumnFamilj1:Kolumn1/tid) så ges den data som har samma tidstämpel, alternativt äldre tidstämpel Om ingen äldre version hittas så returnerar databasen NULL.

3.4 Google App Engine

BigTable finns inte tillgänglig utanför Google, men man har som användare möjlighet att spara data till en BigTable-databas via Google App Engine [16]

Google App Engine är en plattform för att servera och beräkna data från molnet. Tjänsten kan ses som ett simplifierat gränssnitt till BigTable och är gratis att använda till en viss gräns [10]. Då denna gräns överskrids så faktureras man enligt överskriden mängd.

App Engine ger en möjlighet att kommunicera med BigTable-databasen via Java, Python eller Go.[17]. Figurer 3.4-1 och 3.4-2 visar hur man tillexempel kan skriva till en databas med Java och Python. I efterföljande exempel kommer Python att användas som språk.

```
...
Entity greeting = new Entity("Greeting", guestbookKey);
greeting.setProperty("user", user);
greeting.setProperty("date", date);
greeting.setProperty("content", content);

DatastoreService datastore = DatastoreServiceFactory.getDatastoreService();
datastore.put(greeting);
```

Figur 3.4-1: Java exempel för att spara en hälsning till en gästbok [21]

```
...
class Greeting(db.Model):
      author = db.UserProperty()
      content = db.StringProperty(multiline=True)
      date = db.DateTimeProperty(auto_now_add=True)

class Guestbook(webapp2.RequestHandler):
  def post(self):
   guestbook_name = self.request.get('guestbook_name')
   greeting = Greeting(parent=guestbook_key(guestbook_name))
   if users.get_current_user():
     greeting.author = users.get_current_user()
   greeting.content = self.request.get('content')
   greeting.put()
   self.redirect('/?' + urllib.urlencode({'guestbook_name':
                                          guestbook_name}))
```

Figur 3.4-2: Python exempel för att spara en hälsning till en gästbok [22]

App Engine har ett förfrågningsspråk som heter GQL (Google Query Language). GQL är mycket likt SQL men stöder till exempel inte joins eftersom tabellerna i BigTable är delade och spridda på flera maskiner (sektion 3.2.2). Därför borde man, ifall man vill representera en-till-många eller många-till-många relationer, använda sig av ”referenceProperty” [23] Referenceproperty används för att referera från en entitet till en annan.

Expando-modellen [24] möjliggör att dynamiskt kunna spara attribut och egenskaper för en entitet. Det går även att specificera fasta attribut för entiteter av Expando-typ. Dynamisk skapade attribut kan innehålla varierande mängder av datatyper och är endast till för att spara värden av dessa typer.

3.5 Modellering med App Engine

3.5.1 Listor

App Engine har en list-datatyp, vilket innebär att man för en entitet kan sätta flera attribut under en nyckel. Detta är praktiskt om man vill spara information om tex flera e-postadresser, telefonnummer eller intressen (se figur 3.5.1-1). Eftersom datatypen är en lista kan man enkelt rangordna värdena (tex primära epostadressen i position 0 av listan). Man kan utöver att få listorna returnerade som strukturerad data, även göra förfrågningar på medlemmar i listorna (se figur 3.5.1-2) [10]

```
class User(db.model):
  name = db.StringProperty()
  emails = db.StringListProperty()
```

Figur 3.5.1-1: En Användar-entitet (eng. User) som har attributet namn av typen sträng, och attributet email av typen sträng-lista [10]

```
results = db.GqlQuery("SELECT * FROM User WHERE emails =
'homer@simpson.com'")
```

Figur 3.5.1-2: En förfrågning som returnerar de Användare som har emailen homer@simpson.com *i sin 'emails' attribut-lista[10]*

3.5.2 En till många samband

För att modellera en-till-många samband mellan entiteter kan referenceproperty() användas. Referenceproperty specificerar att en entitet refererar till en annan entitet. På den refererade entiteten skapas då en samling av de entiteter som refererar till den. [23]

Exempelkod för att skapa person-entiteter som kan ha flera telefonnummer-entiteter

```
class Person(db.Model):
	namn = db.StringProperty()

class TelefonNummer(db.model):
	Person = db.ReferenceProperty(Person, collection_name='nummer')
	Typ = db.StringProperty(choises=('hemma','jobb','fax','annan'))
	nummer = db.PhoneNumberProperty()
```

Definering av Person och TelefonNummer klasserna. TelefonNummer har en referenceproperty som pekar på en Person och samlingens namn är 'nummer'. Ifall collection_name inte specificeras så nås samlingen via 'Person.telefonnummer_set'

Skapa en person med två telefonnummer:

```
olle = Person(namn='Olle').put()
TelefonNummer(Person=olle, typ='hemma', nummer='123-145-432').put()
TelefonNummer(Person=olle, typ='jobb', nummer='123-145-267').put()
```

Skriva ut en lista över en persons telefonnummer:

```
for telefon in olle.nummer:
	print '%s: %s' % (telefon.typ, telefon.nummer)
```

3.5.3 Många till många samband

Relationer av typen många-till-många kan modelleras med att ha en lista av nycklar eller genom att sätta upp ett relations-objekt som refererar till två entiteter (likt relations-tabeller i en relationell databas). [23]

Lista av nycklar-modellen innebär att man på ena sidan av sambandet, skapar en lista innehållande nycklarna till de andra entiteterna. Listan sätts på den sida av sambandet som förväntas bli kortare eftersom sökningar kommer göras via listan. Exempelkod för personer och hobbier:

```
class Person(db.Model):
	namn = db.StringProperty()
```

```
        hobbies = db.ListProperty(db.key)

class Hobby(db.Model):
        namn = db.StringProperty()
        beskrivning = db.TextProperty()
```

I exemplet har listan satts på person-entiteten eftersom man kan anta att en person har ett fåtal hobbies, medan det kan finnas tusentals olika hobbies.

För att sätta att personen 'Olle' har fiske som hobby:

```
fiske = Hobby.gql("WHERE namn = 'fiske'").get()
olle = Person.gql("WHERE namn = 'Olle'").get()
if fiske.key() not in olle.hobbies
        olle.hobbies.append(fiske.key())
        olle.put()
```

Hämta alla med fiske som hobby:

```
fiske = Hobby.gql("WHERE namn = 'fiske'").get()
fiskare = Person.gql("WHERE hobbies = :1", fiske.key() ).get()
```

Alternativt kan man specificera en hjälpfunktion i Hobby-klassen som returnerar samma resultat som ovanstående:

```
class Hobby(db.Model):
        namn = db.StringProperty()
        beskrivning = db.TextProperty()

        @property
        def members(self):
          return Person.gql("WHERE hobbies = :1", self.key())
```

Man kan även modellera många-till-många relationer genom att skapa ett relations-objekt som har två stycken referenceProperty:n och eventuellt andra attribut. Denna typ av modellering passar bättre om man tillexempel vill specificera en beskrivning för relationen, alternativt i de förhållanden där båda sidorna av relationen innehåller stora samlingar relationer. [23]

3.6 Restriktioner med App Engine

App Engines förfrågningsspråk GQL tillåter inte att man har utför förfrågningar som har jämförelser på olikhet på mera än ett attribut. olikhets-operatorerna är: < , <= , >=, > och != .[25] Detta innebär att

```
förfrågan = Person.gql("WHERE ålder > 22 AND ålder < 30")
```

Är korrekt, medan:

```
förfrågan = Person.gql("WHERE ålder > 22 AND längd < 170")
```

inte kommer att fungera. En annan regel man även måste komma ihåg då man arbetar med GQL är att olikheter måste sorteras före andra attribut. Detta betyder att:

```
förfrågan = Person.gql("WHERE ålder > 22 ORDER BY name")
```

Inte kommer att fungera, medan:

```
förfrågan = Person.gql("WHERE ålder > 22 ORDER BY ålder,name")
```

är korrekt.

Likadana förfrågningar i SQL fungerar utan problem, och man har därför större frihet att göra avancerade förfrågningar.

3.7 Utvärdering

Ifall man har en applikation som får många användare, så kan det vara till stor fördel att servera applikationen från App Engine eftersom Google sköter om skalningen automatiskt. Man får även automatiskt tillgång till en stor del statistik via App Engines webbsidor.

Man kan inte utföra avancerade förfrågningar mot databasen, utan måste istället ändra på tankesättet när man modellerar databasen. Detta kan avskräcka nya användare och användare som är vana med den relationella modellen.

Om man går över gränserna för gratis användning av App Engine [11] så faktureras man enligt den datamängd som applikationen överskrider med, vilket är bra i den åsyn att man endast betalar för de resurser man använder.

4. Diskussion

Det finns för- och nackdelar med båda typerna av databaser. Relationella databaser klarar av avancerade förfrågningar och leverar data snabbt i de flesta situationer.[10] NoSQL:s största säljningspunkt är skalbarheten som behövas när stora mängder data behandlas och levereras under hög belastning, dock med försvagade garantier för konsistens.

Jag har fått den uppfattningen att NoSQL ämnar sig bättre för enkla strukturer och hög belastning. NoSQL databaser borde i mitt tycke inte ersätta relationella databaser utan istället komplettera, där relationella databaser inte presterar lika bra.

Enligt Neal Leavitt kommer inte NoSQL-databaser att ersätta relationella databaser, dock kommer de att ämna sig bättre för vissa projekt och ändamål.[3]

MySQL tillkännagav den 15.2.2012 ”MySQL Cluster 7.2” med ett NoSQL gränsnitt, Schemalös lagring mha MEMCACHED, Key-Value lagring, och upp till 70 gånger snabbare ”avancerade förfrågningar”.[26] Det blir intressant att se hur detta kommer att påverka marknaden ifall det presterar som utlovat. Jag tror att dessa utlovade prestandaökningar och funktioner kommer ge databasadministratörer enkla sätt att utöka på existerande MySQL-databaser, istället för att migrera till en NoSQL-databas.

Litteraturförteckning

[1] “MySQL :: Market Share.” [Online]. Available: http://www.mysql.com/why-mysql/marketshare/. [Accessed: 20-Mar-2012].

[2] “Facebook Newsroom - Fact Sheet - Facebook.” [Online]. Available: http://newsroom.fb.com/content/default.aspx?NewsAreaId=22. [Accessed: 11-Feb-2012].

[3] N. Leavitt, “Will NoSQL Databases Live Up to Their Promise?,” *Computer*, vol. 43, no. 2, pp. 12–14, Feb. 2010.

[4] C. Strozzi, “NoSQL Relational Database Management System: Home Page.” [Online]. Available: http://www.strozzi.it/cgi-bin/CSA/tw7/I/en_US/nosql/Home%20Page. [Accessed: 12-Feb-2012].

[5] C. von der Weth and A. Datta, “Multiterm Keyword Search in NoSQL Systems,” *IEEE Internet Computing*, vol. 16, no. 1, pp. 34–42, Feb. 2012.

[6] “NOSQL DATABASES.” [Online]. Available: http://nosql-databases.org/. [Accessed: 22-Feb-2012].

[7] B. G. Tudorica and C. Bucur, “A comparison between several NoSQL databases with comments and notes,” in *Roedunet International Conference (RoEduNet), 2011 10th*, 2011, pp. 1–5.

[8] L. Bonnet, A. Laurent, M. Sala, B. Laurent, and N. Sicard, “Reduce, You Say: What NoSQL Can Do for Data Aggregation and BI in Large Repositories,” in *2011 22nd International Workshop on Database and Expert Systems Applications (DEXA)*, 2011, pp. 483–488.

[9] 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,” *ACM Transactions on Computer Systems (TOCS)*, vol. 26, no. 2, p. 4, 2008.

[10] I. T. Varley, “No Relation: The Mixed Blessings of Non-Relational Databases.” [Online]. Available: http://ianvarley.com/UT/MR/Varley_MastersReport_Full_2009-08-07.pdf. [Accessed: 02-Nov-2012].

[11] “Quotas - Google App Engine - Google Code.” [Online]. Available: http://code.google.com/appengine/docs/quotas.html. [Accessed: 23-Feb-2012].

[12] “Amazon SimpleDB Pricing.” [Online]. Available: http://aws.amazon.com/simpledb/pricing/. [Accessed: 23-Feb-2012].

[13] “NoSQL - Wikipedia, the free encyclopedia.” [Online]. Available: http://en.wikipedia.org/wiki/NoSQL. [Accessed: 25-Feb-2012].

[14] Peng Xiang, Ruichun Hou, and Zhiming Zhou, “Cache and consistency in NOSQL,” in *2010 3rd IEEE International Conference on Computer Science and Information Technology (ICCSIT)*, 2010, vol. 6, pp. 117–120.

[15] “UnQL Specification - Confluence.” [Online]. Available: http://www.unqlspec.org/display/UnQL/Home. [Accessed: 03-Mar-2012].

[16] “BigTable - Wikipedia, the free encyclopedia.” [Online]. Available:

http://en.wikipedia.org/wiki/BigTable. [Accessed: 05-Mar-2012].
[17] "Developer's Guide - Google App Engine - Google Code." [Online]. Available: http://code.google.com/appengine/docs/. [Accessed: 20-Mar-2012].
[18] "Google's BigTable - Andrew's Website." [Online]. Available: http://andrewhitchcock.org/?post=214. [Accessed: 22-Mar-2012].
[19] S. Ghemawat, H. Gobioff, and S. T. Leung, "The Google file system," in *ACM SIGOPS Operating Systems Review*, 2003, vol. 37, pp. 29–43.
[20] "Associative array - Wikipedia, the free encyclopedia." [Online]. Available: http://en.wikipedia.org/wiki/Associative_array. [Accessed: 22-Mar-2012].
[21] "Using the Datastore - Google App Engine - Google Code." [Online]. Available: http://code.google.com/appengine/docs/java/gettingstarted/usingdatastore.html. [Accessed: 23-Mar-2012].
[22] "Using the Datastore - Google App Engine - Google Code." [Online]. Available: http://code.google.com/appengine/docs/python/gettingstartedpython27/usingdatastore.html. [Accessed: 23-Mar-2012].
[23] "Modeling Entity Relationships - Google App Engine - Google Code." [Online]. Available: http://code.google.com/appengine/articles/modeling.html. [Accessed: 23-Mar-2012].
[24] "The Expando Class - Google App Engine - Google Code." [Online]. Available: http://code.google.com/appengine/docs/python/datastore/expandoclass.html. [Accessed: 26-Mar-2012].
[25] "Queries and Indexes - Google App Engine - Google Code." [Online]. Available: http://code.google.com/appengine/docs/python/datastore/queries.html. [Accessed: 24-Mar-2012].
[26] "MySQL :: MySQL Cluster 7.2 (DMR2): NoSQL, Key/Value, Memcached." [Online]. Available: http://dev.mysql.com/tech-resources/articles/mysql-cluster-7.2.html. [Accessed: 25-Mar-2012].