Database Projekt
Forår 2017

Af Mathias Pedersen, Jonathan Carlsen, Nikolai Hansen og Joachim Dittman

Vores valg af database løsninger

Vi har valgt at prøve SQL og MongoDB af mod hinanden, da vi har arbejdet med dem i projekter før.

De har også forskellige fordele og ulemper som er med til at tilsikre et godt resultat af eksperimentet i denne opgave.

Som den ene database løsning i vores projekt, har vi valgt en ældre teknologi der understøtter et stort valg af opgave specifikke kald til databasen, der mindsker datamængden programmet skal lede igennem for at finde det korrekte resultat.

SQL er godt til at sætte relationer mellem de forskellige objekter og samtidig sørge for mindre redundant data med normalisering.

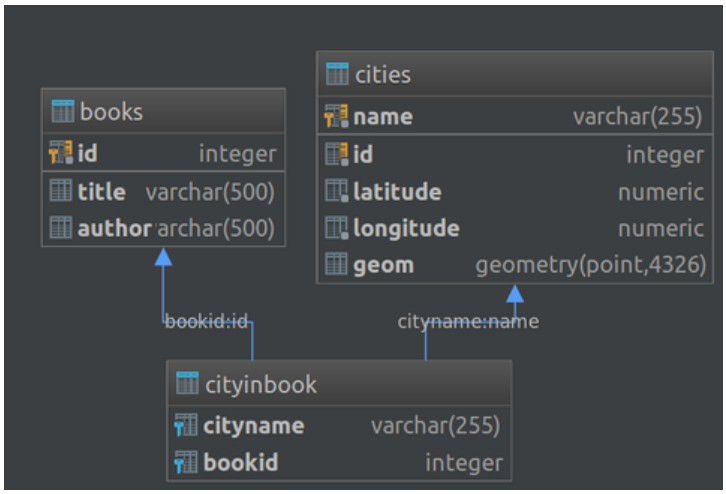
Den anden løsning vi har valgt er NoSQL MongoDB.

NoSQL er en database er bygget op som et JSON dokument, så har man typisk mere meta data i de forskellige objekter, for at formindske antal kald igennem JSON dokumentet.

F.eks. en ordre, der har et bruger Id, får også tilknyttet brugerens navn som meta data, så man kun skal hente et snapshot.

Data modeller

SQL database UML



Databasen er sat op men en mange til mange relation kaldet CityInBook, denne tabel er kun til for at vise at der er en relation mellem en specifik bog og en eller flere byer.
Tabellen kan også bruges til at finde ud af hvilke bøger en specifik by opstår i.

CityInBook er en weak entity, da den ikke kan eksistere uden de andre to.

MongoDB database

Databasen er sat op med to tabeller, en med alle bynavne, koordinator og en med bøgerne selv.
Under bøgernes meta data, er der koblet en liste af "cities" på, der opstår i den specifikke bog.
Dette array kan bruges til at søge efter bøger hvor en by opstår i.

```
{
  "_id" : ObjectId("591ca8463f931425eb2bc578"),
  "title" : "unknown",
  "author" : "unknown",
  "cities" : [
    "hampshire",
    "gettysburg"
  ]
}
```

I byernes tabel fremgår der en "location" med type: "point" der hjælper MongoDB frameworket med at finde lokationer i en specificeret omkreds af koordinatet.

Dataen bliver indekseret på "location" og "name" for at sikre en hurtigere behandlingstid når programmet kalder databasen.

```
{
  "_id" : ObjectId("591a25ca9b9601179f23febc"),
  "location" : {
    "coordinates" : [
      26.05806,
      44.29083
    ],
    "type" : "Point"
  },
  "name" : "1 decembrie"
}
```

Indsættelse af data

SQL

I Sprint 0 af vores SCRUM har vi gennemgået kravene til de forskellige kald og fandt frem til at sætte tre tabeller op i databasen.

En tabel med bøger, en med byer og en mange til mange tabel der viser de hvilke bøger der indeholder byer.

Dette er gjort for at opnå bedre tid på hvert kald. På denne måde skal kaldet ikke løbe hver bogs beskrivelse igennem med f.eks. LIKE eller CONTAINS.

Da vi skulle indsætte data, gik vi igang med at finde ud hvilke bøger der indeholdte hvilke byer og herefter oprettede vi hvert enkelt relation mellem bogens titel og bynavne i tabellen CityInBook.

Vores funktioner der indsætter dataen kan findes her:
https://github.com/jonathanec94/Gutenberg_Project/blob/master/src/main/java/DbInterface/Facade.java

Vi har valgt at bruge Postgresql, da vi har læst os frem til, at det skulle være godt til geolocation.

MongoDB

I Sprint 0 af vores SCRUM har vi gennemgået kravene til de forskellige kald og fandt frem til at sætte to collections op i databasen. En med bøger og en med byer.

Under vores Books collection har vi tilknyttet et array af byer der opstår i bogens beskrivelse. Da vi indsatte data gik vi igennem processen med at finde ud hvilke bøger der indeholdte hvilke byer og herefter oprettede hvert enkelt by mellem bogens beskrivelse og bynavne i den pågældende bogs cities array.

Vores funktioner der indsætter dataen kan findes her:
https://github.com/jonathanec94/Gutenberg_Project/blob/master/src/main/java/mongo/MongoImporter.java

Kald og statistikker

PostgreSql

Hvert query er kørt fem gange.

Query (from project)	Name	Avg	Median
1	Books by city	1061.8 ms	750 ms
2	Cities by title	956.4 ms	842 ms
3	Books by author	4899.2 ms	4571 ms
4	Books by location	1358.6 ms	1084 ms

MongoDB

Hvert query er kørt fem gange.

Query (from project)	Name	Avg	Median
1	Books by city	1236.6 ms	1149 ms
2	Cities by title	560.8 ms	518 ms
3	Books by author	3962.4 ms	3829 ms
4	Books by location	1923,6 ms	1951 ms

Som man kan se på resultatet, ligger svartiderne tæt på hinanden. Der dog lidt forskel på de forskellige svartider. Før projektet havde vi forventet at mongo ville have bedre performance. Vi tror at grunden til at svartiderne ligger så tæt på hinanden, skyldes Java driveren gør det svært at lave optimerede queries.

I nogle tilfælde kan man se at mongo er hurtigere fx opgave 4 'Books by location'. Grunden til dette kan være den måde indexes virke på i mongo. I vores table 'books' har vi et index på arrayet cities som indeholder cities name

```
{
  "_id" :
ObjectId("591ca8463f931425eb2bc578"),
  "title" : "unknown",
  "author" : "unknown",
  "cities" : [
    "hampshire",
    "gettysburg"
  ]
```

Mongo virker på den måde at den cacher alle index i maskinens ram hvilket gør det langt hurtigere at slå op på de forskellige cities, da det er indekseret i maskines ram i forhold til SQL som skal hente det fra disken.

Mongo er mere fleksibelt da det er schemaless frem for SQL som tvinger dig til at definere et schema før data kan indsættes i tabellerne. Mongo benytter som udgangspunkt ikke constraints hvilket også gør det nemmere at arbejde med, men kan også gå ud over data-integritet da man ikke kan være sikker på hvilke data er opbevaret i de forskellige objekter.

Udover Mongo har vi brugt PostgreSql som vores SQL database. I PostgreSql er det forholdsvis let at arbejde med geolocation. Man kan installere en extension til PostgreSql som hedder PostGis, i denne extension ligger der en del forskellige funktioner inde i databasen, som vi så kan kalde i vores query fx 'ST_DWithin()' se eksempel:

```
SELECT name FROM cities WHERE ST_DWithin(geom,
ST_GeographyFromText('SRID=4326;POIN(55.67594 12.56553)'), 20000);
```

Her kalder vi funktionen 'ST_DWithin()' hvor vi leder efter alle cities der er i en radius af 20 km = 20000m af København.

Vores anbefaling

Hvordan har vi gjort?

Vi har sat to database løsninger op, en med SQL og en med MongoDB.

SQL:

Hertil har vi udforsket forskellige muligheder for at strukturer databasen i SQL og her fandt vi frem til en struktur med et table til Books med deres data og en med Cities.

Da kaldende kræver at man kan finde bøger, der indeholder specifikke bynavne, så har vi valgt at oprette en relations tabel der sammenkobler de enkelte bynavne, med books tabellen.

Dette krævede dog en del arbejde ved indsættelsen af data, men alternativet var at løbe hver enkelt bog igennem for hver by og bruge LIKE eller CONTAINS til at finde resultaterne.

MongoDB:

På MongoDB siden kiggede vi på forskellige strukturer og fandt frem til en løsning med to collections, en kaldet Cities, med alle vores byer inklusiv koordinater og en med kaldet Books med alle vores bøger, der samtidig inkluderer et array med bynavne.

Vi har lavet arrayet med med byerne i hver bog Document, da vi ellers skulle hente og søge hele beskrivelsen igennem for hvert kald.
Dette krævede dog en del tid på struktureringen af data og indsættelsen af data, men man vinder det dog på de enkelte kald. Alternativet var at have hele bøger liggende i Books collectionen.

Hvad har vi fundet frem til?

Til dette projekt anbefaler vi at benytte mongo da vi arbejder med store datasæt hvor man ikke kan være sikker på kvaliteten og indholdet af det data der indsættes i databasen, f.eks. har alle bøger ikke nødvendigvis en author. Da mongo er schemaless er det nemmere at indsætte data i de forskellige collections. Som standard understøtter mongo også geospatial indexes hvilket er ideelt til dette projekt.

Bilag 1 - kildekode

https://github.com/jonathanec94/Gutenberg_Project