UPPSALA
UNIVERSITET

Rapport NOSQL

Rapport om applikation och databas

Kursnamn: Databaser 2
Gruppmedlemmar: Jonas Örnfelt
Datum: 2020-02-20

Innehållsförteckning

Rapport 1 **6**

Källförteckning 6 6

Rapport

Denna rapport ämnar redogöra över skapandet av en databas som utformats efter en applikations-idé. I applikationen ska köp kunna genomföras av olika produkter och när detta sker behöver kunduppgifter registreras, såväl som ordern och vad den innehåller. Produkterna ska även tilldelas en eller flera kategorier i syfte att användaren ska kunna navigera sig fram i applikationen och hitta önskad produkt.

Följande *Collections* har implementerats i databasen: *Customers*, *Items*, och *Orders*. Namnen beskriver informationen som dem innehåller. Den förstnämnda omfattar kundens uppgifter i form av en unik id, namn på kunden, dennes mailadress samt fysisk adress som i detta fall är nästlad så att gatunamn, stad och postnummer är tillgängliga. Till sist lagras också id för kundens beställningar i en Array, vilket medför möjligheten att lagras flera id för beställningar.

Kollektionen *Items* erhåller det unika namnet på produkten, antal i lager, pris, och kategori(er). *Orders* innehåller en unik id, status för beställningen, datum för när den skapades, och de produkter som den innehåller. När en ny beställning läggs till är det viktigt att tidigare beställningar inte åsidosätts och försvinner från databasen. Istället ska nya id för *Orders* läggas till i den befintliga data-arrayen i *Customers*.

Ytterligare en viktig notering är att *Items* saknar ett id-värde. Detta kringgås genom att namnet för en produkt är unikt och därför är refererande möjligt genom just namnet. Per automatik lagrar databasen också ett s.k. objekt-id som också kan användas i nödfall för att komma åt en specifik produkt (MongoDB Docs, 2020).

Följande frågeställningar bör reflekteras över innan valet av nästlade dokument eller uppdelning genomförs:

- Kommer barndokumenten behövas varje gång föräldradokumentet anropas i en Query?
- MongoDB har en begränsning som gör att storleken på dokument behöver vara mindre än den maximala storleken för BSON-dokument, vilket är 16mb. Om dokumentstorleken inte är ett problem, är ett inbäddat dokument möjligt som dokumentstruktur.
- Borttagning av föräldradokumentet innebär att barndokumentet också tas bort, är det lämpligt för den data som dokumentet kommer innehålla?

Anledningen till att kollektionerna urskiljs och att inte information om produkter och order finns i samma kollektion som kunderna, är för att data snabbt kommer fyllas på i kollektionerna som nu istället är separerade. Det blir väldigt mycket information i en och samma kollektion. Som beskrivet ovanför stöter vi även på problem när ett föräldradokument tas bort, om dem också innehåller exempelvis information om produkter (Fowler & Sadalage, 2013).

Implementationen strider nu inte emot de tre frågeställningarna, även för den nästlade data som återfinns i *Customers*. Denna data har en stark koppling till kunden och kan därför tas bort i de fall där föräldradokument tas bort. Storleken bör inte heller växa på så sätt att det motstrider det definierade kravet. Det som skulle kunna argumenteras över är ifall den nästlade data alltid behöver anropas när övrig information behövs.

Med tanke på applikationens syfte anser jag att detta inte medför komplikationer i detta fall. Adressen hos kunden är direkt bunden till individen, vilket gör det lämpligt för lagring i den givna kollektionen. Uppdelning av detta hade orsakat en ökad komplexitet för databasen som vi nu kringgår.

När det kommer till valet av vilken typ av databas som är mest passande för applikationen, behöver användandet datamängd övervägas. I de fall där databasen blir snabbt växande är MongoDB

rekommenderat då det är mer kostnadseffektivt att skala upp. Dessa databaser är skalbara horisontellt på så sätt att dem kan utökas med fler servrar, medan SQL är beroende av utökandet av hårdvara.

Jag anser att MongoDB är ett bättre val för applikationen och grundar detta delvis i att databasen då redan har ett stöd för utökande, men främst för att den huvudsakligen kommer hantera enkla anrop. SQL är högst lämpligt i de fall där komplexa Queries kommer genomföras, tack vare standardgränssnittet för hanteringen av dessa. Införing av nya data, som kan tänkas ske frekvent i applikationen, är enkelt att göra och kräver inte tidigare steg (Fowler & Sadalage, 2013).

Nedan beskrivs och demonstreras olika Queries som kan användas i applikationen för olika ändamål:

- Visa alla orders med öppen status med följande Query:
  db.orders.find({status: "Open"}).pretty()
- Visa alla items med specifik kategori:
  db.items.find({categories: "Art"}).pretty()
- Visa alla items med någon av kategorierna i given array:
  db.items.find({categories: {$in: ["Essentials", "Bathroom"]}}).pretty()
- Visa alla items med lager över 50:
  db.items.find({stock: {$gt: 50}}).pretty()
- Sorterar items baserat på pris (lägst först):
  db.items.aggregate([{$sort: {price: 1}}]).pretty()
- Sortera customers baserat på hur många orders de lagt:

```
db.customers.aggregate([
        {$project: {
                        id: 1,
                        numbOfOrders: {$size: "$orderID"}}},
        {$sort: { numbOfOrders: -1 }}
]).pretty()
```

- Visa kundinformation med join för orders för kund med specifikt id:

```
db.customers.aggregate([
        {$match: {_id: 1}},
        {$lookup: {
                        from: "orders",
                        localField: "orderID",
                        foreignField: "_id",
                        as:"customer_order(s)"
}}]).pretty()
```

- Visa kundinformation med join för orders för kunder från Visby eller Göteborg:
```
db.customers.aggregate([
        {$match: {"address.city": {$in: ["Visby", "Gothenburg"]}}},
        {$lookup: {
                        from: "orders",
                        localField: "orderID",
                        foreignField: "_id",
                        as:"customer_order(s)"
}}]).pretty()
```

- Uppdatera customers med ny stad i den nästlade adressen:
```
db.customers.updateOne({_id: 6}, {$set: {'address.0.city': "Karlstad" }})
```

- Ta bort specifikt fält för samtliga värden i kollektion:
```
db.customers.update({}, {$unset: {email:1}}, false, true)
```

- Lägg till nytt item (används för nästkommande query):
```
db.items.insertOne({"name": "Towel", "stock": 10, "price": 79, "categories": ["Bathroom"]})
```

- Visa items som inte har beställts. Den nya produkten bör alltså dyka upp:
```
db.items.aggregate([
   {$lookup: {
        from: "orders",
        localField: "name",
       foreignField: "itemNames",
        as: "matched_order"
      }},
  {
    $match: { "matched_order": { $eq: [] } }
  }]).pretty()
```

- Visa items och orders dem är med i:
  db.items.aggregate([
  {$lookup: {
  from: "orders",
  localField: "name",
  foreignField: "itemNames",
  as: "matched_order"
  }},
  {$match: { "matched_order": { $ne: [] }}}]).pretty()

- Ta bort item med givet namn:
  db.items.deleteOne({name: "Towel"})

Källförteckning

Fowler, M & Sadalage, P. (2013). *NoSQL Distilled*.
MongoDB Docs. 2020. *Manual Reference*. Tillgänglig:
https://docs.mongodb.com/manual/reference/
[Hämtad 2020-02-20]