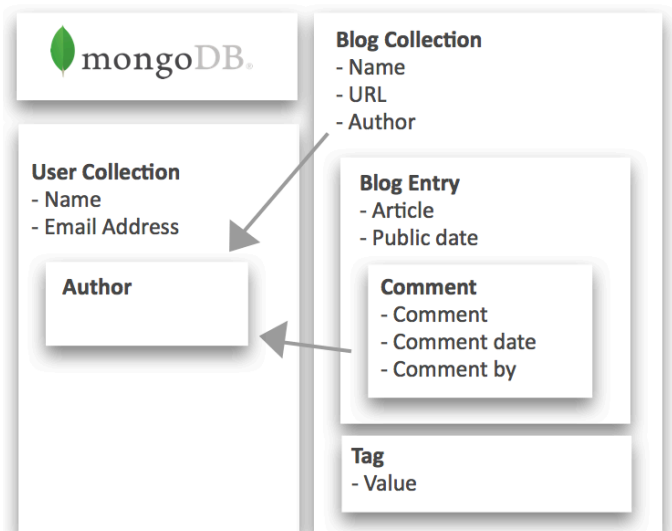
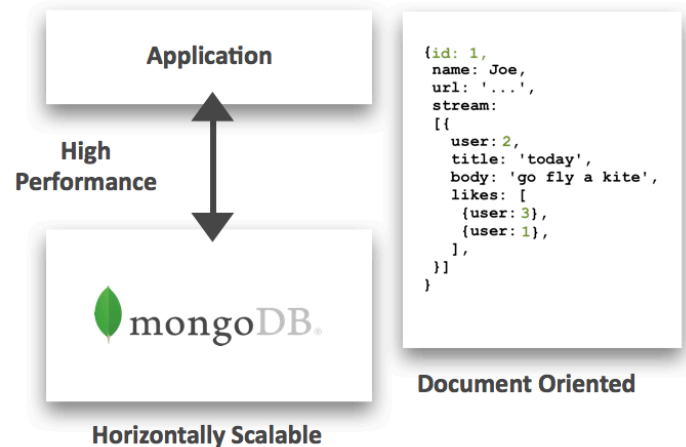


MongoDB and MySQL

Database Engines

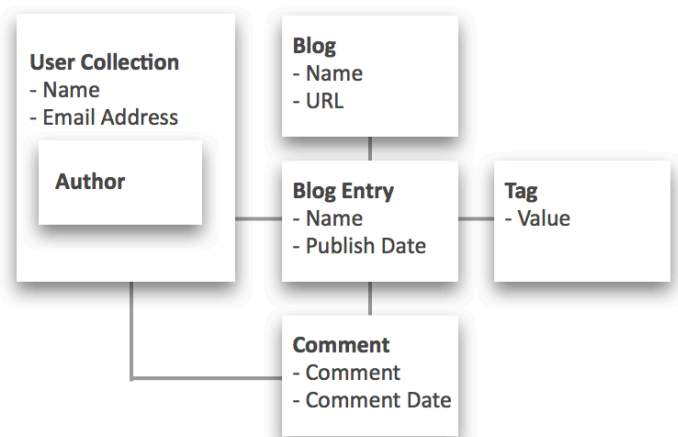
MongoDB

MongoDB er en cross-platform dokumentorienteret database, der bl.a. giver høj tilgængelighed, let skalerbarhed og høj ydeevne. I stedet for at bruge tabeller og rækker som i en relationel database, så er MongoDB bygget på collections og dokumenter. Dokumenterne omfatter sæt af "key-pair" værdier og er den grundlæggende dataenhed i MongoDB. Dokumentmodellen kortlægges til objekterne i applikationen, hvilket gør data simplere at arbejde med. Ligesom andre NoSQL-databaser understøtter MongoDB også dynamisk skema design, det gør det muligt at samle dokumenterne i en collection med forskellige felter og strukturer. Databasen bruger et dataudvekslingsformat kaldet BSON, hvilket minder meget om den binære repræsentation ligesom JSON. MongoDB giver også mulighed for at distribuere data i collection på tværs over flere systemer til horisontal skalerbarhed i takt med at datamængden øges.



MySQL

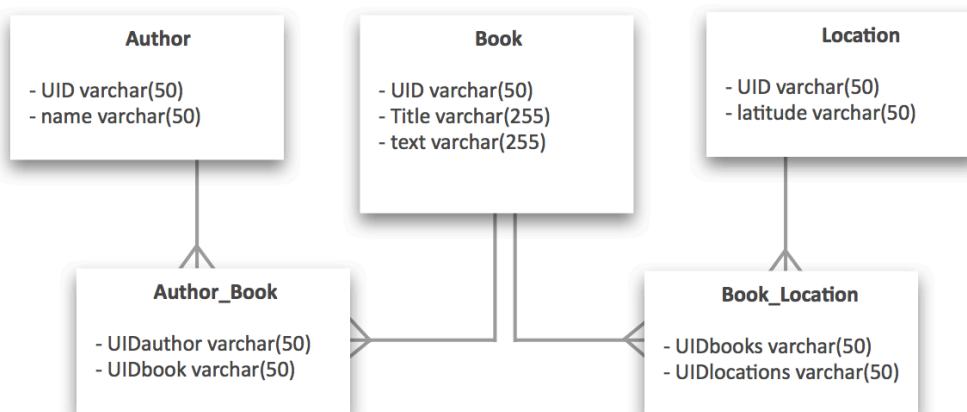
MySQL er en SQL databaseserver der understøtter mange samtidige brugere (Multithreaded). MySQL er bygget op omkring forskellige databaser på en server, hvor hver enkelt bruger som regel har adgang til en database. MySQL er en relationel database, hvilket betyder at dataene er organiseret som et sæt tabeller, hvorfra data kan hentes eller tilføjes uden at skulle reorganisere databasetabellerne. Relationelle databaser benytter SQL forespørgsler til at hente, opdatere, tilføje eller slette data fra en relationel database. Udover at være let tilgængelig er den også let at udvide.



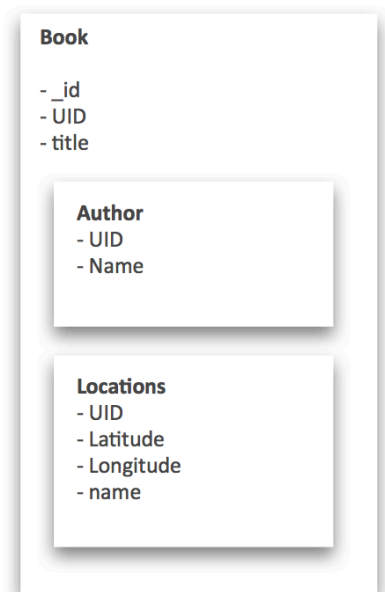
Data Modelling

Datamodellering beskriver hvordan den logiske struktur af en database er modelleret. Den definere også, hvordan data er forbundet til hinanden og hvordan det lagres og behandles i systemet. Entity Relationship (ER) model er baseret på virkelige elementer og forholdet mellem dem. Ved at formulere rigtige scenarier, kan man vha. ER-modellen skabe entiteter, relationer og generelle attributter. ER modellen er som sagt baseret på entiteter og deres attributter, og relationerne mellem disse entiteter. Entitet er en virkelig repræsentation af en enhed med egenskaber/attributter. Eksempelvis har vi i vores MySQL database en entitet der hedder "Book" med attributten "Title", og en "Book" kan have en "Author". Relationen mellem "Book" tabellen og "Author" tabellen er repræsenteret vha. en Pivot tabel. Pivot tabellen har til formål at vise hvilken "Author" har skrevet hvilken "Book". En af vores opgaver var at finde bøger og deres forfattere ud fra by navne. Derfor har vi lavet en relation mellem "Book" tabellen og "Location" tabellen vha. en Pivot tabel, som skal vise hvilke byer bogen nævner. Nedenstående er et ER-diagram af vores MySQL database.

MySQL ER-Diagram



MongoDB Model



I MongoDB har vi indlejret data modeller, så man kan integrere relaterede data i en enkelt struktur eller et dokument. I vores tilfælde har vi en Book data model, hvor Author og Locations er indlejrede sub dokumenter. Indlejrede data modeller tillader applikationen at gemme relaterede oplysninger i samme database og fordelen ved dette er at man ikke behøver lige så mange forespørgsler og opdateringer for at gennemføre almindelige operationer. En tommelfinger regel er, at man bør bruge indlejrede sub dokumenter, hvis man

```

{
  _id: <ObjectId>,
  username: "123xyz",
  contact: {
    phone: "123-456-7890",
    email: "xyz@example.com"
  },
  access: {
    level: 5,
    group: "dev"
  }
}
  
```

Embedded sub-document

Embedded sub-document

"contains" relation mellem entiteterne, hvilket vi har i vores tilfælde, da en bog har en forfatter og

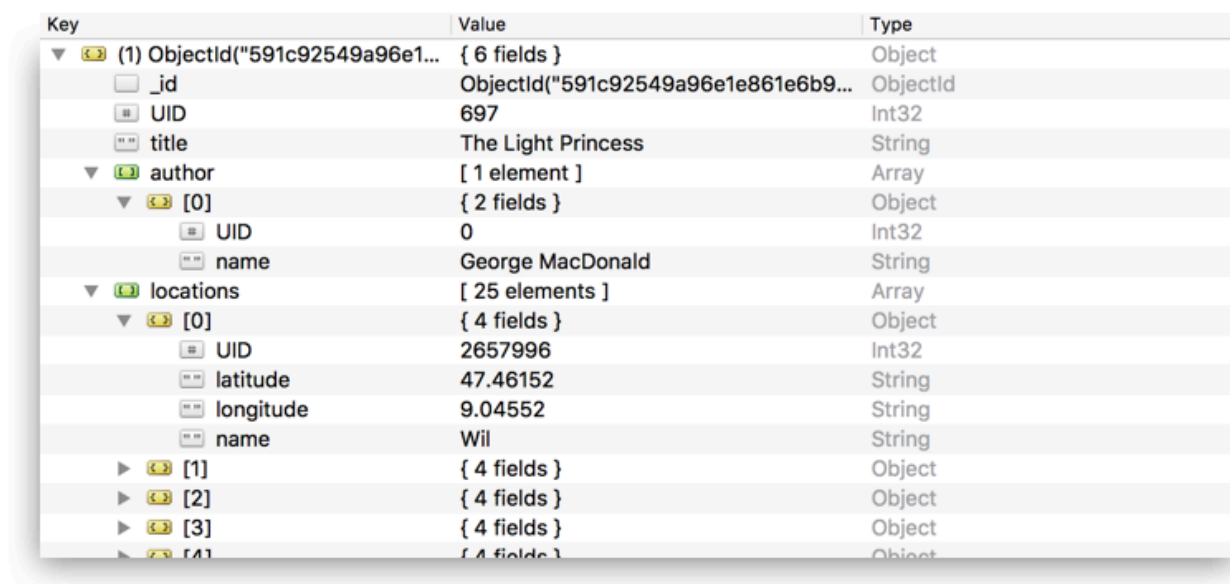
indeholder byer. Indlejret data modeller giver bl.a. bedre ydeevne for "read" og "request" operationer, og gør det muligt at opdatere relaterede data i en enkelt operation.

Data Import

Til import af data har vi gjort brug af Nicolai Bonderups "GutenbergToJsonTool" applikation. Hans applikation indeholder BookScanner, CityScanner og en SQLGenerator. Scanner klasserne scanner bøgerne og byerne. Herefter bruger man SQLGenerator til at generer SQL som vores MySQL database kan forstå. Når vi har kørt programmet igennem får vi BooksJson.txt, citiesJson.txt, locationSQL.txt, bookAuthorSql.txt og BooksJson.json. I vores MySQL database har vi først lavet alle vores tabeller, henholdsvis Books, Locations, Author, Book_Location og Author_Book. Herefter har vi importeret data ind i vores tabeller vha. de generede filer fra "GutenbergToJsonTool" ¹.

Til import af data i vores Mongo database, har vi gjort brug af Vagrant. Vi har først startet Vagrant SSH og har derefter tilføjet BooksJson.json filen til vores Vagrant server. Herefter har vi gjort brug af mongo import til at importere data ind i en "dbbooks" database med en collection ved navn "books".

"mongoimport --db dbbooks --collection books --drop --file booksJson.json --jsonArray". Nedenstående er opbygningen af vores Mongo database.



Key	Value	Type
(1)	Object	Object
_id	ObjectId("591c92549a96e1...")	ObjectId
UID	697	Int32
title	The Light Princess	String
author	[1 element]	Array
[0]	{ 2 fields }	Object
UID	0	Int32
name	George MacDonald	String
locations	[25 elements]	Array
[0]	{ 4 fields }	Object
UID	2657996	Int32
latitude	47.46152	String
longitude	9.04552	String
name	Wil	String
[1]	{ 4 fields }	Object
[2]	{ 4 fields }	Object
[3]	{ 4 fields }	Object
[4]	{ 4 fields }	Object

BooksJson.json filen som vi fik fra "GutenbergToJsonTool" applikationen er en JSON fil med indlejret data. MongoDB gør som sagt brug af BSON formatet hvilket minder meget om JSON. Derfor har importen denne fil været meget nem vha. Mongoimport forespørgslen.

¹ <https://github.com/NicolaiVBonderup/GutenbergToJsonTool>

Test Queries

Til vores test queries har vi gjort brug af "Parameterized.class", så vi ikke behøver at lave redundante "assertEqual" sætninger. @Parameters gør det muligt at køre testen med vores angivende parametre. Nedenstående er eksempel på et af vores test metoder.

Mongo Mock Testing:

```
@Parameters
public static Collection<String> data() {
    return Arrays.asList(new String[]{"test1", "test2", "test3", "test4", "test5"});
}

@Test
public void testGetLocationByTitle() {
    System.out.println("getLocationByTitle with title: " + _data);

    List<DTOLocation> listBooks = new ArrayList<>();
    for (int i = 0; i < 10; i++) {
        listBooks.add(new DTOLocation("title" + i, _data));
    }

    MongoMapping mocked = mock(MongoMapping.class);
    when(mocked.getLocationByTitle(_data)).thenReturn(listBooks);

    for (int i = 0; i < 10; i++) {
        assertEquals(listBooks.iterator().next().toString(), mocked.getLocationByTitle(_data).iterator().next().toString());
    }
}
```

Tests passed: 100,00 %

All 15 tests passed.(0,305 s)

- ✔ MockTesting.MongoMappingMockTest passed
 - ✔ testGetLocationByTitle[0] passed (0,187 s)
 - ✔ testGetBooksByAuthor[0] passed (0,0 s)
 - ✔ testGetBooksByCity[0] passed (0,0 s)
 - ✔ testGetLocationByTitle[1] passed (0,001 s)
 - ✔ testGetBooksByAuthor[1] passed (0,001 s)
 - ✔ testGetBooksByCity[1] passed (0,002 s)
 - ✔ testGetLocationByTitle[2] passed (0,001 s)
 - ✔ testGetBooksByAuthor[2] passed (0,001 s)
 - ✔ testGetBooksByCity[2] passed (0,001 s)
 - ✔ testGetLocationByTitle[3] passed (0,001 s)
 - ✔ testGetBooksByAuthor[3] passed (0,001 s)
 - ✔ testGetBooksByCity[3] passed (0,001 s)
 - ✔ testGetLocationByTitle[4] passed (0,001 s)
 - ✔ testGetBooksByAuthor[4] passed (0,002 s)
 - ✔ testGetBooksByCity[4] passed (0,002 s)

SQL Mock Testing

```
@Parameters
public static Collection<String> data() {
    return Arrays.asList(new String[]{"test1", "test2", "test3", "test4", "test5"});
}

@Test
public void testGetAllBookTitleWithAuthorByCityName() {
    System.out.println("getAllBookTitleWithAuthorByCityName: " + _data);

    List<DT0AuthorBook> listBooks = new ArrayList<>();
    for(int i = 0; i < 10; i++){
        listBooks.add(new DT0AuthorBook("title" + i, _data));
    }

    DBFacade mocked = mock(DBFacade.class);
    when(mocked.getAllBookTitleWithAuthorByCityName(_data)).thenReturn(listBooks);

    for(int i = 0; i < 10; i++){
        assertEquals(listBooks.iterator().next().toString(), mocked.getAllBookTitleWithAuthorByCityName(_data).iterator().next().toString());
    }
}
```

The screenshot shows a test runner interface with a green header bar indicating 'Tests passed: 100,00 %'. Below the header, it states 'All 20 tests passed.(0,29 s)'. A tree view shows the test suite 'MockTesting.SQLMappingTest' as passed. Underneath, 20 individual tests are listed, each marked with a green checkmark and 'passed' status, followed by their execution time in parentheses. The tests are grouped by author name (0, 1, 2, 3, 4) and include methods like 'TestGetAllBooksAndCitiesByAuthorName', 'testGetAllBookTitleWithAuthorByCityName', 'testGetAllLocationByBookTitle', and 'testGetAllBooksByGeolocation'.

Test Name	Status	Time
MockTesting.SQLMappingTest	passed	
TestGetAllBooksAndCitiesByAuthorName[0]	passed	(0,187 s)
testGetAllBookTitleWithAuthorByCityName[0]	passed	(0,0 s)
testGetAllLocationByBookTitle[0]	passed	(0,0 s)
testGetAllBooksByGeolocation[0]	passed	(0,0 s)
TestGetAllBooksAndCitiesByAuthorName[1]	passed	(0,0 s)
testGetAllBookTitleWithAuthorByCityName[1]	passed	(0,0 s)
testGetAllLocationByBookTitle[1]	passed	(0,001 s)
testGetAllBooksByGeolocation[1]	passed	(0,001 s)
TestGetAllBooksAndCitiesByAuthorName[2]	passed	(0,001 s)
testGetAllBookTitleWithAuthorByCityName[2]	passed	(0,001 s)
testGetAllLocationByBookTitle[2]	passed	(0,001 s)
testGetAllBooksByGeolocation[2]	passed	(0,001 s)
TestGetAllBooksAndCitiesByAuthorName[3]	passed	(0,001 s)
testGetAllBookTitleWithAuthorByCityName[3]	passed	(0,0 s)
testGetAllLocationByBookTitle[3]	passed	(0,001 s)
testGetAllBooksByGeolocation[3]	passed	(0,001 s)
TestGetAllBooksAndCitiesByAuthorName[4]	passed	(0,001 s)
testGetAllBookTitleWithAuthorByCityName[4]	passed	(0,001 s)
testGetAllLocationByBookTitle[4]	passed	(0,0 s)
testGetAllBooksByGeolocation[4]	passed	(0,0 s)

Conclusion

Vi synes det har været spændende at arbejde med forskellige databaser og prøve det af i praksis. Det er et svært valg at vælge hvilken database man vil arbejde med, men det afhænger af projektets størrelse og formål. Hvis vi skulle anbefale en database, ville det være MongoDB. Vi anbefaler MongoDB pga. fleksibiliteten af data modellen, skalerbarhed og den høje ydeevne. Modsat MySQL er det nemmere at kombinere og lagre strukturerede og ustrukturerede data struktur i en NoSQL database. I de seneste år er NoSQL databaser blevet mere anerkendt pga. deres ydeevne, skalerbarhed og fleksibilitet. Disse NoSQL databaser er bedre beregnet til krævende applikationer.