Implementering av Dokumentdatabase og Design av Kolonnefamiliedatabase og Grafdatabase

Gruppe 13: Mats Engelien og Lars Erik Faber

Contents

[Implementasjon av Dokumentdatabase 3](#_Toc86090989)

[Figur 1 3](#_Toc86090990)

[Figur 2 4](#_Toc86090991)

[Figur 3 5](#_Toc86090992)

[Figur 4 5](#_Toc86090993)

[Figur 5 6](#_Toc86090994)

[Design av Dokumentdatabase 8](#_Toc86090995)

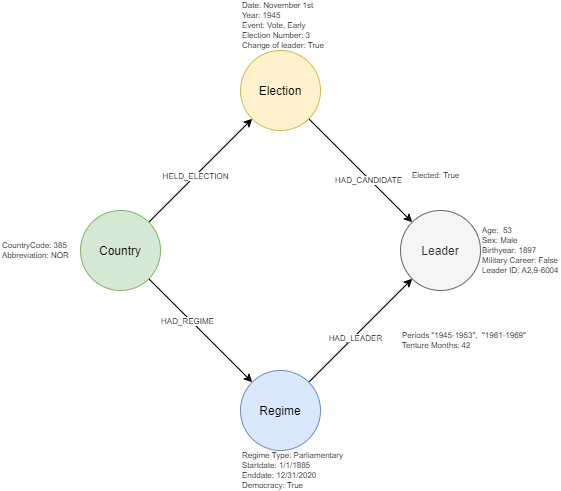
[Begrunnelse bak valgene rundt kolonnefamilie- og grafdatabase 8](#_Toc86090996)

[Hvordan eksisterende data oppdateres 8](#_Toc86090997)

[Aggregeringer 8](#_Toc86090998)

[Design av Grafdatabase 10](#_Toc86090999)

[Begrunnelse bak valgene rundt kolonnefamilie- og grafdatabase 10](#_Toc86091000)

[ 10](#_Toc86091001)

[Hvordan eksisterende data oppdateres 10](#_Toc86091002)

[Aggregeringer 10](#_Toc86091003)

# Implementasjon av Dokumentdatabase

I denne milepælen har vi implementert noen komponenter fra forrige milepæl via Scala og SBT.

## Figur 1

Gjentakende kode/importer, db init og async fix

import org.mongodb.scala.\_

import org.mongodb.scala.model.Aggregates.\_

import org.mongodb.scala.model.Filters.\_

import org.mongodb.scala.model.Projections.\_

import org.mongodb.scala.model.Sorts.\_

import org.mongodb.scala.model.Updates.\_

import org.mongodb.scala.model.UpdateOptions

import org.mongodb.scala.model.\_

import scala.concurrent.Await

import scala.concurrent.duration.Duration

import scala.collection.JavaConverters.\_

import org.mongodb.scala.bson.BsonObjectId

import java.io.\_

import java.util.List

import java.util.Arrays

import java.util.ArrayList

import java.util.concurrent.TimeUnit

object GetCostComparison extends App {

    implicit class DocumentObservable[C](val observable: Observable[Document]) extends ImplicitObservable[Document] {

        override val converter: (Document) => String = (doc) => doc.toJson

    }

    implicit class GenericObservable[C](val observable: Observable[C]) extends ImplicitObservable[C] {

        override val converter: (C) => String = (doc) => doc.toString

    }

    trait ImplicitObservable[C] {

        val observable: Observable[C]

        val converter: (C) => String

        def results(): Seq[C] = Await.result(observable.toFuture(), Duration(10, TimeUnit.SECONDS))

        def headResult() = Await.result(observable.head(), Duration(10, TimeUnit.SECONDS))

        def printResults(initial: String = ""): Unit = {

            if (initial.length > 0) print(initial)

            results.foreach(res => println(converter(res)))

        }

        def printHeadResult(initial: String = ""): Unit = println(s"${initial}${converter(headResult())}")

    }

    val mongoClient: MongoClient = MongoClient();

    val database: MongoDatabase = mongoClient.getDatabase("soci\_econ\_country\_profiles");

    val collection: MongoCollection[Document] = database.getCollection("countryProfiles");

}

## Figur 2

Innsetting av data fra CSV til database

val source = io.Source.fromFile("datasets/soci\_econ\_country\_profiles.csv");

    source.getLines.drop(1).foreach { line =>

        val row = line.split(",").map(\_.trim)

        def parseFloat(value : String) : Option[Float] = if (value == "") None else

        Some(value.toFloat)

        def parseInt(value : String) : Option[Int] = if (value == "") None else

        Some(value.toInt)

        def isEmpty( input : Option[String]) : Boolean =

            input match {

            case None    => true

            case Some(s) => s.trim.isEmpty

        }

        //parses incorrect values to a nothing or something value

        def isWrongData(value : String) : Option[Double] = if (value == "~0.0" || value == "-~0.0" || value == "...") None else

        Some(value.toDouble)

        val document = Document(

            "country" -> row(1),

            "populationInThousands" -> row(4).toDouble,

            "gdp" -> row(7).toDouble,

            "gdpPerCapita" -> row(9).toDouble,

            "unemployment" -> row(16).toDouble,

            "labourForceGender" -> row(17),

            "popGrowthRateAnnual" -> isWrongData(row(24)),

            "urbanPop" -> isWrongData(row(25)),

            "urbanPopGrowth" -> isWrongData(row(26)),

            "healthTotalExpenditure" -> row(32).toDouble,

            "educationTotalExpenditure" -> isWrongData(row(34)),

            "internetUsersPerOneHundred" -> row(41).toDouble,

            "qualityOfLifeIndex" -> row(50).toDouble,

            "purchasingPowerIndex" -> row(51).toDouble,

            "safetyIndex" -> row(52).toDouble,

            "healthCareIndex" -> row(53).toDouble,

            "propertyPriceIncomeRatio" -> row(55).toDouble,

            "affordabilityIndex" -> row(64).toDouble,

            "costOfLivingIndex" -> row(65).toDouble,

            "costOfLivingPlusRentIndex" -> row(67).toDouble,

            "lifeExpectancy" -> row(89).toDouble,

            "militaryExpenditure" -> {if (!row(90).isEmpty) row(90).toDouble else 0},

            "taxRevenue" -> {if (!row(93).isEmpty) row(93).toDouble else 0}

        );

        collection.insertOne(document).results();

    }

    println("All documents inserted")

## Figur 3

enkel aggregering som lister land etter gdp og viser snitt skatt

collection.aggregate(Seq(

        Aggregates.group("$country",

        Accumulators.sum("score", "$gdp"),

        Accumulators.avg("average\_taxes", "$taxRevenue")),

        Aggregates.sort(orderBy(descending("score"))),

        Aggregates.limit(10)

    ))

    .printResults()

## Figur 4

Henter to land fra input, viser billigst øverst (skal være sammenligning av costOfLivingPlusRentIndex)

println("Which Country would you like to compare?")

    println("Capital letter first!")

    val userInput = scala.io.StdIn.readLine();

    println("Next one: ")

    val comparison = scala.io.StdIn.readLine();

    collection.find(in("country", userInput, comparison)).sort(descending("costOfLivingPlusRentIndex"))

    .projection(fields(include("country"), include("costOfLivingPlusRentIndex"), excludeId())).printResults();

## Figur 5

post ny landsprofil

println("--- Fill In The Following Fields ---");

    print("Country (String: name): ");

    val country = readLine();

    print("populationInThousands (Int): ");

    val populationInThousands = readDouble();

    print("GDP (Double): ");

    val gdp = readDouble();

    print("GDP per capita (Double): ");

    val gdpPerCapita = readDouble();

    print("Unemployment (Double): ");

    val unemployment = readDouble();

    print("labourForceGender(String: f%/m%): ");

    val labourForceGender = readLine();

    print("popGrowthRateAnnual (Double): ");

    val popGrowthRateAnnual = readDouble();

    print("Urban pop (Double): ");

    val urbanPop = readDouble();

    print("Urban pop growth (Double): ");

    val urbanPopGrowth = readDouble();

    print("Total health expenditure (Double): ");

    val healthTotalExpenditure = readDouble();

    print("Total education expenditure (Double): ");

    val educationTotalExpenditure = readDouble();

    print("Internet users per 100 inhabitants(Double): ");

    val internetUsersPerOneHundred = readDouble();

    print("Quality of life index value (Double): ");

    val qualityOfLifeIndex = readDouble();

    print("Purchasing power index value (Double): ");

    val purchasingPowerIndex = readDouble();

    print("Safety index value (Double): ");

    val safetyIndex = readDouble();

    print("Health Care index value (Double): ");

    val healthCareIndex = readDouble();

    print("Property price to income ratio (Double): ");

    val propertyPriceIncomeRatio = readDouble();

    print("Affordability index value (Double): ");

    val affordabilityIndex = readDouble();

    print("Cost of living index value (Double): ");

    val costOfLivingIndex = readDouble();

    print("Cost of living + rent index value (Double): ");

    val costOfLivingPlusRentIndex = readDouble();

    print("Life expectancy (Double): ");

    val lifeExpectancy = readDouble();

    print("Total Military expendature (Double): ");

    val militaryExpenditure = readDouble();

    print("Tax revenue(% of GDP)(Double): ");

    val taxRevenue = readDouble();

    val document = Document(

        "country" -> country,

            "populationInThousands" -> populationInThousands,

            "gdp" -> gdp,

            "gdpPerCapita" -> gdpPerCapita,

            "unemployment" -> unemployment,

            "labourForceGender" -> labourForceGender,

            "popGrowthRateAnnual" -> popGrowthRateAnnual,

            "urbanPop" -> urbanPop,

            "urbanPopGrowth" -> urbanPopGrowth,

            "healthTotalExpenditure" -> healthTotalExpenditure,

            "educationTotalExpenditure" -> educationTotalExpenditure,

            "internetUsersPerOneHundred" -> internetUsersPerOneHundred,

            "qualityOfLifeIndex" -> qualityOfLifeIndex,

            "purchasingPowerIndex" -> purchasingPowerIndex,

            "safetyIndex" -> safetyIndex,

            "healthCareIndex" -> healthCareIndex,

            "propertyPriceIncomeRatio" -> propertyPriceIncomeRatio,

            "affordabilityIndex" -> affordabilityIndex,

            "costOfLivingIndex" -> costOfLivingIndex,

            "costOfLivingPlusRentIndex" -> costOfLivingPlusRentIndex,

            "lifeExpectancy" -> lifeExpectancy,

            "militaryExpenditure" -> militaryExpenditure,

            "taxRevenue" -> taxRevenue

    );

    collection.insertOne(document).results();

}

# Design av Dokumentdatabase

Her brukte vi datasettet «World University Rankings» med Times Higher Education World University Ranking metodikken fra Kaggle:

<https://www.kaggle.com/mylesoneill/world-university-rankings?select=timesData.csv>

## Begrunnelse bak valgene rundt kolonnefamilie- og grafdatabase

Raske queries, vi vil gjerne vite om alle skoler basert på faktorer

## Hvordan eksisterende data oppdateres

Ting tang walla walla bing bang

## Aggregeringer

Rådata (1)

[Beskrivelse]

Dette dataobjektet representerer rådata brukes til å oppdatere dataen i databasen. Dette objektet opprettes når ny data blir lest inn, eller når bruker legger inn selv,

[Pseudo-Kode]

[Bilde]

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Column** | **Value** | **Timestamp** |
| 1 | World Rank |  |  |
| 2 | University Name |  |  |
| 3 | Country |  |  |
| 4 | Teaching Score |  |  |
| 5 | International Student Score |  |  |
| 6 | Research Score |  |  |
| 7 | Citation Score |  |  |
| 8 | Income Score |  |  |
| 9 | Total Score |  |  |
| 10 | Number of Students |  |  |
| 11 | Students by Staff Ratio |  |  |
| 12 | International Students |  |  |
| 13 | Female to Male Ratio |  |  |
| 14 | Year |  |  |

# Design av Grafdatabase

Her brukte vi datasettet «Government Types of the World» fra Kaggle:

<https://www.kaggle.com/janzasadny/rulers-elections-and-irregular-governance>

## Begrunnelse bak valgene rundt kolonnefamilie- og grafdatabase

Ting tang walla walla bing bang

## 

## Hvordan eksisterende data oppdateres

Ting tang walla walla bing bang

## Aggregeringer

[Navn på aggregering]

[Beskrivelse]

[Pseudo-Kode]

[Bilde]

