Gruppe 13
Implementasjon av KV-Database og Dokumentdatabase Design/Modellering

Mats Engelien og Lars Erik Faber

Implementasjon av KV-database

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

«Average Grades By Study Hours»

PostAvgGradesByStudyHoursFromCSV.scala:

```
import java.io._
import scala.collection.mutable.ArrayBuffer
import org.apache.commons._
import java.util.ArrayList
import org.apache.http.{ io => _, _ }
import org.apache.http.client._
import org.apache.http.entity.StringEntity
import org.apache.http.client.methods.HttpPost
import org.apache.http.impl.client.DefaultHttpClient
import org.apache.http.message.BasicNameValuePair
import org.apache.http.client.entity.UrlEncodedFormEntity
import com.google.gson.Gson

case class AvgGrades(
    female: Float,
    male: Float);
case class AvgGradesByStudyHours(
    lessThanTwo: AvgGrades,
    twoToFive: AvgGrades,
    fiveToTen: AvgGrades,
    greaterThanTen: AvgGrades);

object AverageGradesByStudyHours extends App {

    def calculateAverage(array:ArrayBuffer[Int]) : Float = {
        var average = 0f;
        var i = 0;
        array.foreach(x => {
            average += x;
            i += 1;
        });
        return average / i;
    }

    val source = io.Source.fromFile("datasets/student-performance.csv");
    var lines = source.getLines().map(line => line.split(",")).drop(1);

    // Temp ArrayBuffers
    // Female grades
    var gradesFLessThanTwo = ArrayBuffer[Int]();
    var gradesFTwoToFive = ArrayBuffer[Int]();
    var gradesFFiveToTen = ArrayBuffer[Int]();
    var gradesFGreaterThanTen = ArrayBuffer[Int]();

    // Male grades
    var gradesMLessThanTwo = ArrayBuffer[Int]();
    var gradesMTwoToFive = ArrayBuffer[Int]();
    var gradesMFiveToTen = ArrayBuffer[Int]();
    var gradesMGreaterThanTen = ArrayBuffer[Int]();
```

```
    // Fill ArrayBuffers with grades
    lines.foreach(line => {
        if (line(1) == "\"F\"") {
            if (line(13) == "1") gradesFLessThanTwo += line(32).toInt;
            if (line(13) == "2") gradesFTwoToFive += line(32).toInt;
            if (line(13) == "3") gradesFFiveToTen += line(32).toInt;
            if (line(13) == "4") gradesFGreaterThanTen += line(32).toInt;
        }
        if (line(1) == "\"M\"") {
            if (line(13) == "1") gradesMLessThanTwo += line(32).toInt;
            if (line(13) == "2") gradesMTwoToFive += line(32).toInt;
            if (line(13) == "3") gradesMFiveToTen += line(32).toInt;
            if (line(13) == "4") gradesMGreaterThanTen += line(32).toInt;
        }
    });

    val avgLessThanTwo = AvgGrades(
        calculateAverage(gradesFLessThanTwo),
        calculateAverage(gradesMLessThanTwo)
    );
    val avgTwoToFive = AvgGrades(
        calculateAverage(gradesFTwoToFive),
        calculateAverage(gradesMTwoToFive)
    );
    val avgFiveToTen = AvgGrades(
        calculateAverage(gradesFFiveToTen),
        calculateAverage(gradesMFiveToTen)
    );
    val avgGreaterThanTen = AvgGrades(
        calculateAverage(gradesFGreaterThanTen),
        calculateAverage(gradesMGreaterThanTen)
    );
    val avgGradesByStudyHours = AvgGradesByStudyHours(
        avgLessThanTwo,
        avgTwoToFive,
        avgFiveToTen,
        avgGreaterThanTen
    );

    val spock = new Gson().toJson(avgGradesByStudyHours);

    val post = new HttpPost("http://127.0.0.1:2379/v2/keys/component%3AaverageGradesByStudyHours");

    val nameValuePairs = new ArrayList[NameValuePair]();
    nameValuePairs.add(new BasicNameValuePair("value", spock));
    post.setEntity(new UrlEncodedFormEntity(nameValuePairs));

    // send the post request
    val client = new DefaultHttpClient
    val response = client.execute(post)
    println("--- HEADERS ---")
    response.getAllHeaders.foreach(arg => println(arg))
}
```

GetAvgGradesByStudyHours.scala:

```scala
import java.io._
import org.apache.commons._
import org.apache.http._
import org.apache.http.client._
import org.apache.http.client.methods.HttpGet
import org.apache.http.impl.client.DefaultHttpClient
import java.util.ArrayList
import org.apache.http.message.BasicNameValuePair
import org.apache.http.client.entity.UrlEncodedFormEntity
import com.google.gson.Gson


case class NodeResponse(key : String, value: String)
case class Node(key : String, value: String, nodes: Array[NodeResponse])
case class Message(action: String, node: Node)

case class AvgGrades(female: Float, male: Float);
case class AvgGradesByStudyHours(
    lessThanTwo: AvgGrades,
    twoToFive: AvgGrades,
    fiveToTen: AvgGrades,
    greaterThanTen: AvgGrades);

object HttpGetPostTest extends App {

    val url = "http://127.0.0.1:2379/v2/keys/component%3AaverageGradesByStudyHours";
    val result = scala.io.Source.fromURL(url).mkString;

    println(result);

    val messageParsed = new Gson().fromJson( result, classOf[Message] );
    val valueParsed = new Gson().fromJson( messageParsed.node.nodes(0).value,
classOf[AvgGradesByStudyHours] );

    println("Average Grades By Study Time:");
    println("< 2:");
    println("    female: " + valueParsed.lessThanTwo.female);
    println("    male: " + valueParsed.lessThanTwo.male);
    println("2 - 5:");
    println("    female: " + valueParsed.twoToFive.female);
    println("    male: " + valueParsed.twoToFive.male);
    println("5 - 10:");
    println("    female: " + valueParsed.fiveToTen.female);
    println("    male: " + valueParsed.fiveToTen.male);
    println("> 10:");
    println("    female: " + valueParsed.greaterThanTen.female);
    println("    male: " + valueParsed.greaterThanTen.male);
}
```

Terminal:

```
sbt:project> run
[info] compiling 1 Scala source to /home/student/Documents/BigData/project/target/scala-2.12/classes ...
[info] running HttpGetPostTest
{"action":"get","node":{"key":"/component:averageGradesByStudyHours","dir":true,"nodes":[{"key":"/component:averageGradesBy
StudyHours/00000000000000000009","value":"{\"lessThanTwo\":{\"female\":11.191011,\"male\":10.593496},\"twoToFive\":{\"femal
e\":12.19697,\"male\":11.897196},\"fiveToTen\":{\"female\":13.12,\"male\":13.590909},\"greaterThanTen\":{\"female\":14.1904
76,\"male\":11.357142}}","modifiedIndex":9,"createdIndex":9}],"modifiedIndex":7,"createdIndex":7}}

Average Grades By Study Time:
< 2:
    female: 11.191011
    male: 10.593496
2 - 5:
    female: 12.19697
    male: 11.897196
5 - 10:
    female: 13.12
    male: 13.590909
> 10:
    female: 14.190476
    male: 11.357142
[success] Total time: 5 s, completed Oct 11, 2021, 9:41:01 AM
sbt:project>
```

«Students By Free Time»

PostStudentsByFreeTimeFromCSV.scala:

```
import java.io._
import scala.collection.mutable.ArrayBuffer
import org.apache.commons._
import java.util.ArrayList
import org.apache.http.{ io => _, _ }
import org.apache.http.client._
import org.apache.http.entity.StringEntity
import org.apache.http.client.methods.HttpPost
import org.apache.http.impl.client.DefaultHttpClient
import org.apache.http.message.BasicNameValuePair
import org.apache.http.client.entity.UrlEncodedFormEntity
import com.google.gson.Gson

case class StudentsByFreeTime(
    veryLittle: Float,
    little: Float,
    medium: Float,
    much: Float,
    veryMuch: Float);

object StudentsByFreeTime extends App {

    val source = io.Source.fromFile("datasets/student-performance.csv");
    var lines = source.getLines().map(line => line.split(",")).drop(1);

    var veryLittle = 0f;
    var little = 0f;
    var medium = 0f;
    var much = 0f;
    var veryMuch = 0f;

    lines.foreach(line => {
        val freeTime = line(24).toInt;

        if (freeTime == 1) veryLittle += 1;
        if (freeTime == 2) little += 1;
        if (freeTime == 3) medium += 1;
        if (freeTime == 4) much += 1;
        if (freeTime == 5) veryMuch += 1;
    });

    val total = veryLittle + little + medium + much + veryMuch;

    veryLittle = (veryLittle / total) * 100;
    little = (little / total) * 100;
    medium = (medium / total) * 100;
    much = (much / total) * 100;
    veryMuch = (veryMuch / total) * 100;

    val studentsByFreeTime = StudentsByFreeTime(
        veryLittle,
        little,
        medium,
        much,
        veryMuch
    );

    val spock = new Gson().toJson(studentsByFreeTime);

    val post = new HttpPost("http://127.0.0.1:2379/v2/keys/component%3AstudentsByFreeTime");

    val nameValuePairs = new ArrayList[NameValuePair]();
    nameValuePairs.add(new BasicNameValuePair("value", spock));
    post.setEntity(new UrlEncodedFormEntity(nameValuePairs));

    // send the post request
    val client = new DefaultHttpClient;
    val response = client.execute(post);
    println("--- HEADERS ---");
```

```
    response.getAllHeaders.foreach(arg => println(arg));
}
```

«Average Grades By Parent Education»

PostAvgGradesByStudentEduFromCSV.scala:

```
import java.io._
import scala.collection.mutable.ArrayBuffer
import org.apache.commons._
import java.util.ArrayList
import org.apache.http.{ io => _, _ }
import org.apache.http.client._
import org.apache.http.entity.StringEntity
import org.apache.http.client.methods.HttpPost
import org.apache.http.impl.client.DefaultHttpClient
import org.apache.http.message.BasicNameValuePair
import org.apache.http.client.entity.UrlEncodedFormEntity
import com.google.gson.Gson

case class AvgGrades(
    motherEducation: Float,
    fatherEducation: Float);
case class AvgGradesByParentEdu(
    none: AvgGrades,
    primaryEducation: AvgGrades,
    fifthToNinthGrade: AvgGrades,
    secondaryEducation: AvgGrades,
    higherEducation: AvgGrades);

object AverageGradesByStudyHours extends App {

    def calculateAverage(array:ArrayBuffer[Int]) : Float = {
        var average = 0f;
        var i = 0;
        array.foreach(x => {
            average += x;
            i += 1;
        });
        return average / i;
    }

    val source = io.Source.fromFile("datasets/student-performance.csv");
    var lines = source.getLines().map(line => line.split(",")).drop(1);

    // Temp ArrayBuffers
    // Female grades
    val gradesMotherNone = ArrayBuffer[Int]();
    val gradesMotherPrimary = ArrayBuffer[Int]();
    val gradesMotherFifthToNinth = ArrayBuffer[Int]();
    val gradesMotherSecondary = ArrayBuffer[Int]();
    val gradesMotherHigher = ArrayBuffer[Int]();

    // Male grades
    val gradesFatherNone = ArrayBuffer[Int]();
    val gradesFatherPrimary = ArrayBuffer[Int]();
    val gradesFatherFifthToNinth = ArrayBuffer[Int]();
    val gradesFatherSecondary = ArrayBuffer[Int]();
    val gradesFatherHigher = ArrayBuffer[Int]();

    // Fill ArrayBuffers with grades
    lines.foreach(line => {
        val mEdu = line(6).toInt;
        val fEdu = line(7).toInt;
        val finalGrade = line(32).toInt;

        if (mEdu == 0) gradesMotherNone += finalGrade;
        if (mEdu == 1) gradesMotherPrimary += finalGrade;
        if (mEdu == 2) gradesMotherFifthToNinth += finalGrade;
        if (mEdu == 3) gradesMotherSecondary += finalGrade;
        if (mEdu == 4) gradesMotherHigher += finalGrade;
```

```
        if (fEdu == 0) gradesFatherNone += finalGrade;
        if (fEdu == 1) gradesFatherPrimary += finalGrade;
        if (fEdu == 2) gradesFatherFifthToNinth += finalGrade;
        if (fEdu == 3) gradesFatherSecondary += finalGrade;
        if (fEdu == 4) gradesFatherHigher += finalGrade;
    });

    val avgGradesByParentEdu = AvgGradesByParentEdu(
        AvgGrades(
            calculateAverage(gradesMotherNone),
            calculateAverage(gradesFatherNone)
        ),
        AvgGrades(
            calculateAverage(gradesMotherPrimary),
            calculateAverage(gradesFatherPrimary)
        ),
        AvgGrades(
            calculateAverage(gradesMotherFifthToNinth),
            calculateAverage(gradesFatherFifthToNinth)
        ),
        AvgGrades(
            calculateAverage(gradesMotherSecondary),
            calculateAverage(gradesFatherSecondary)
        ),
        AvgGrades(
            calculateAverage(gradesMotherHigher),
            calculateAverage(gradesFatherHigher)
        )
    );

    val spock = new Gson().toJson(avgGradesByParentEdu);

    val post = new
HttpPost("http://127.0.0.1:2379/v2/keys/component%3AaverageGradesByParentEducation");

    val nameValuePairs = new ArrayList[NameValuePair]();
    nameValuePairs.add(new BasicNameValuePair("value", spock));
    post.setEntity(new UrlEncodedFormEntity(nameValuePairs));

    // send the post request
    val client = new DefaultHttpClient
    val response = client.execute(post)
    println("--- HEADERS ---")
    response.getAllHeaders.foreach(arg => println(arg))
}
```

«Student»

PostStudentFromCSV.scala:

```
import java.io._
import scala.collection.mutable.ArrayBuffer
import org.apache.commons._
import java.util.ArrayList
import org.apache.http.{ io => _, _ }
import org.apache.http.client._
import org.apache.http.entity.StringEntity
import org.apache.http.client.methods.HttpPost
import org.apache.http.impl.client.DefaultHttpClient
import org.apache.http.message.BasicNameValuePair
import org.apache.http.client.entity.UrlEncodedFormEntity
import com.google.gson.Gson

case class Student(
    sex: Char,
    motherEdu: Int,
    fatherEdu: Int,
    studyTime: Int,
    schoolSupport: Boolean,
    extraPaidCourses: Boolean,
    internet: Boolean,
    familyRelation: Int,
    freeTime: Int,
    weekdayAlcohol: Int,
    weekendAlcohol: Int,
    finalGrade: Int
);

object Student extends App {
    val source = io.Source.fromFile("datasets/student-performance.csv");
    val lines = source.getLines().map(line => line.split(",")).drop(1);

    val students = ArrayBuffer[Student]();

    // Read and add students
    lines.foreach(line => {
        val student = Student(
            line(1)(1),
            line(6).toInt,
            line(7).toInt,
            line(13).toInt,
            if (line(15) == "\"yes\"") true else false,
            if (line(17) == "\"yes\"") true else false,
            if (line(21) == "\"yes\"") true else false,
            line(23).toInt,
            line(24).toInt,
            line(26).toInt,
            line(27).toInt,
            line(32).toInt
        );

        students += student;
    });

    val post = new HttpPost("http://127.0.0.1:2379/v2/keys/student");

    students.foreach(student => {
        val spock = new Gson().toJson(student);
        val nameValuePairs = new ArrayList[NameValuePair]();
        nameValuePairs.add(new BasicNameValuePair("value", spock));
        post.setEntity(new UrlEncodedFormEntity(nameValuePairs));

        // send the post request
        val client = new DefaultHttpClient;
        val response = client.execute(post);
        println("--- HEADERS ---");
        response.getAllHeaders.foreach(arg => println(arg));
    });
}
```

AddStudent.scala:

Denne filen simulerer at en bruker legger til studenter til databasen gjennom konsollen. Dataen blir sendt og lagret i KV-store.

```
import java.io._
import scala.io.StdIn._
import scala.collection.mutable.ArrayBuffer
import org.apache.commons._
import java.util.ArrayList
import org.apache.http.{ io => _, _ }
import org.apache.http.client._
import org.apache.http.entity.StringEntity
import org.apache.http.client.methods.HttpPost
import org.apache.http.impl.client.DefaultHttpClient
import org.apache.http.message.BasicNameValuePair
import org.apache.http.client.entity.UrlEncodedFormEntity
import com.google.gson.Gson

case class Student(
    sex: Char,
    motherEdu: Int,
    fatherEdu: Int,
    studyTime: Int,
    schoolSupport: Boolean,
    extraPaidCourses: Boolean,
    internet: Boolean,
    familyRelation: Int,
    freeTime: Int,
    weekdayAlcohol: Int,
    weekendAlcohol: Int,
    finalGrade: Int
);

object AddStudent extends App {

    println("--- Fill In The Following Fields ---");
    print("sex (Char: F or M): ");
    val sex = readChar();
    print("Mother's Education (Int: 0-4): ");
    val motherEdu = readInt();
    print("Father's Education (Int: 0-4): ");
    val fatherEdu = readInt();
    print("Study Time (Int: 1-4): ");
    val studyTime = readInt();
    print("School Support (Boolean: true/false): ");
    val schoolSupport = readBoolean();
    print("Extra Paid Courses (Boolean: true/false): ");
    val extraPaidCourses = readBoolean();
    print("Internet (Boolean true/false): ");
    val internet = readBoolean();
    print("Family Relation (Int: 1-5): ");
    val familyRelation = readInt();
    print("Free Time (Int: 1-5): ");
    val freeTime = readInt();
    print("Weekday Alcohol (Int: 1-5): ");
    val weekdayAlcohol = readInt();
    print("Weekend Alcohol (Int: 1-5): ");
    val weekendAlcohol = readInt();
    print("Final Grade (Int: 1 - 20): ");
    val finalGrade = readInt();

    val student = Student(
        sex.toUpper,
        motherEdu,
        fatherEdu,
        studyTime,
        schoolSupport,
        extraPaidCourses,
        internet,
```

```
        familyRelation,
        freeTime,
        weekdayAlcohol,
        weekendAlcohol,
        finalGrade
    );

    println(student);

    val post = new HttpPost("http://127.0.0.1:2379/v2/keys/student");

    val spock = new Gson().toJson(student);
    val nameValuePairs = new ArrayList[NameValuePair]();
    nameValuePairs.add(new BasicNameValuePair("value", spock));
    post.setEntity(new UrlEncodedFormEntity(nameValuePairs));

    // send the post request
    val client = new DefaultHttpClient;
    val response = client.execute(post);
    println("--- HEADERS ---");
    response.getAllHeaders.foreach(arg => println(arg));

    println("--- Data Posted! ---");
}
```

Country-Dokument

Country: “CountryName”

Population In Thousands: float

GDP: float

GDP Per capita: %

Unemployment: percent

Labour Force Participation By Gender: percent

PopGrowthRate Annual: Percentage

Urban population: percent

Urban population growth: avg annual precent%

Health Total Expenditure: % of GDP

Education Total Expenditure: % of GDP

Education enrollment primary/secondary/tertiary: F/M per 100 pop

Individuals using the internet: per 100 inhabitants

CO2 emission estimates: million tons/tons per capita

Pop. Using improved drinking water: Urban/rural %

Quality of Life Index: float

Purchasing power index: float

Safety Index: float

Health Care Index: int

Cost of Living: float

Cost of living index: float

Property price to income Ratio: float

Affordability index: float

Cost of living plus rent index: float

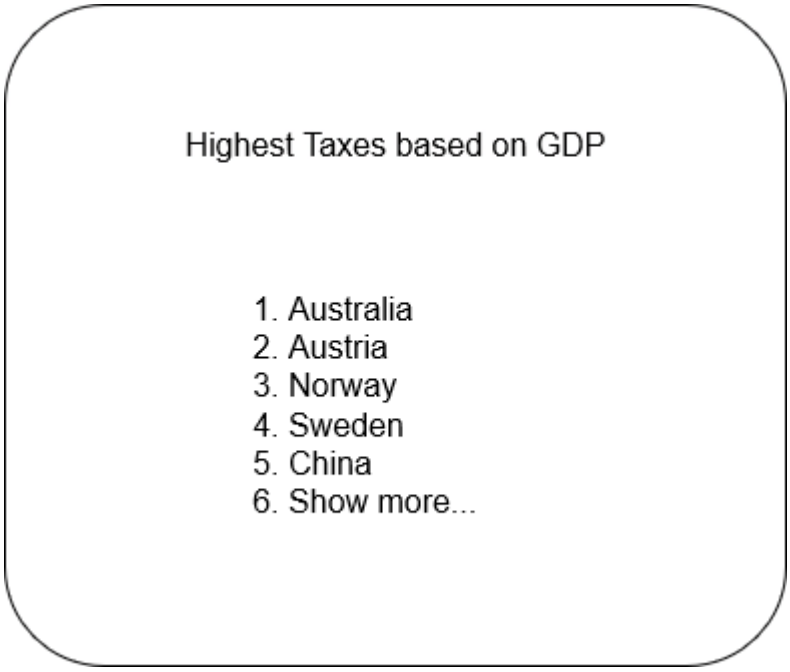
Life expectancy total: float

Military expenditure: float

Tax revenue: % of GDP

Aggregeringer

Hva er verdens høyeste beskattende land etter GDP



Pseudo-kode

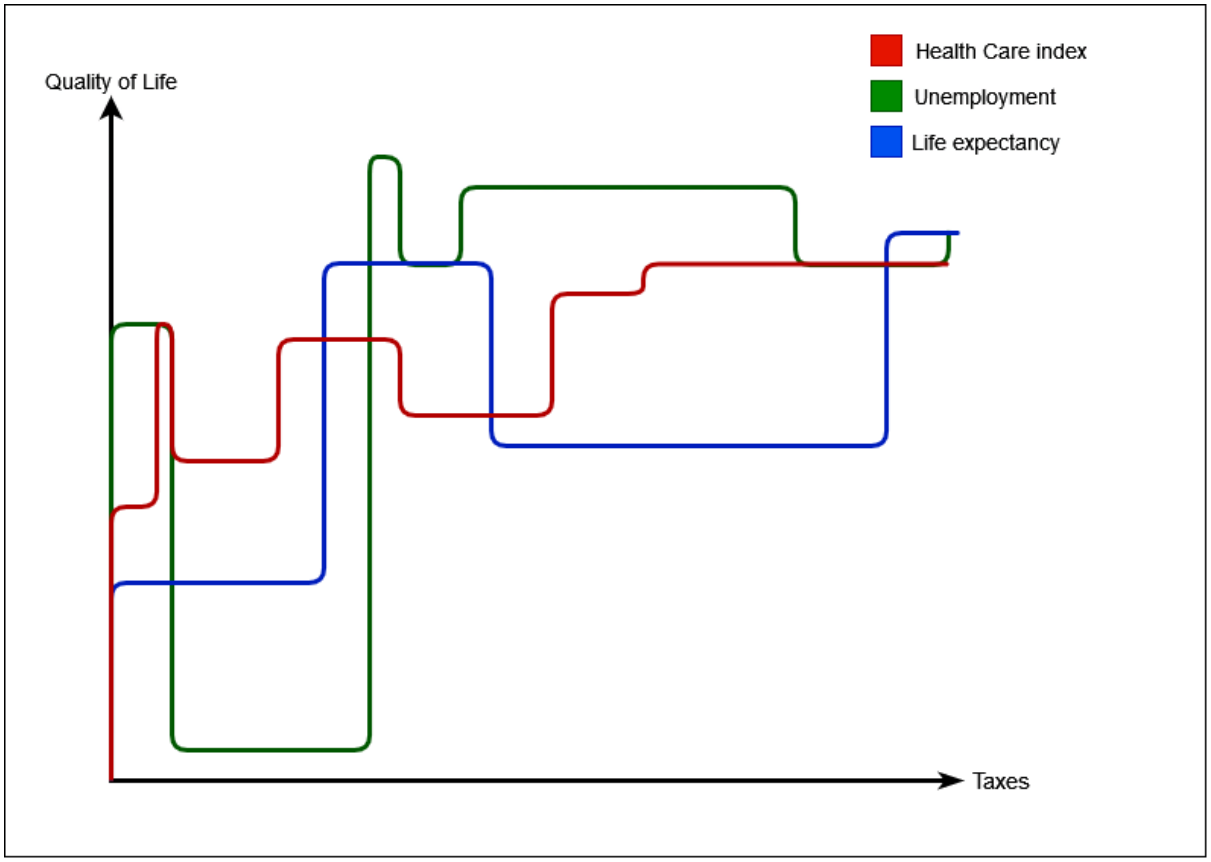
Hent ut alle land med skatt og GDP,

Sorter etter stor del av GDP som blir beskattet

Post (legg inn referanse til aggregering i Land dokumenter)

Denne aggregeringen kan bli lagret i databasen som et eget dokument da det ikke er noe som endrer seg ofte. Dette gjør det lett å hente ut tabellen, og heller flere redundanskopier av dokumentet gjør at det blir lett tilgjengelig selv ved høy trafikk i databasen.

Life-expectancy, Unemployment & Health Care index vs quality of life and taxes



Pseudo-kode

Lag indeks,

Hent alle land sine quality of life verdier,

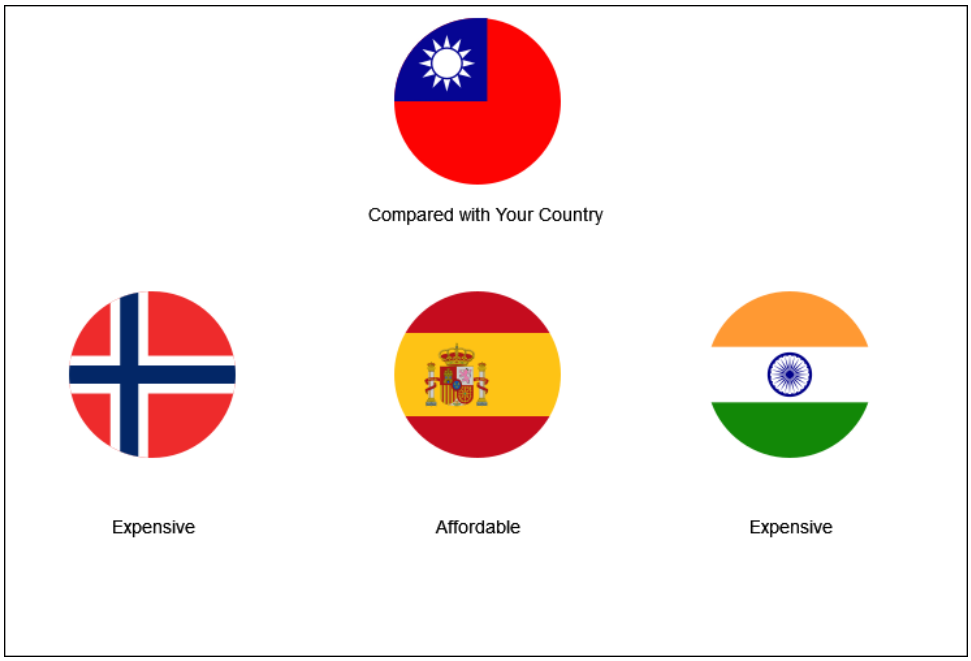
Lag objekter med disse verdiene,

og legg de sortert i en liste etter skatt i landet

post til dokument og legg referanse til referanse

Denne aggregeringen er en av de mer komplekse vi har, denne spørringen gjøres mot en Multikey Index i landsdokumentene som gjør at den kun trenger å gjøre denne spørringen når data endrer seg. Alternativt kan spørringen lagres som et eget dokument med referanse til landene. Å lagre til dokument kan potensielt være billigere da man heller kan hente ut dette ved hver read, men å indeksere vil gjøre at færre interaksjoner vil skje med unødvendige dokumenter.

Er det dyrt å flytte fra X land til Y, Z land?



Pseudo-kode

Hent alle land og legg de i en liste

Hent valgte lands navn og affordability index

Hvis X sitt land er innen samme verdisone som Y = affordable

Under = cheap

Mer= expensive

Post aggregering

Denne aggregeringen er veldig billig ytelsesmessig med kun databasekall for å hente ut landene som blir valgt i menyene. Vi kan da kun sette nødvendige dokumenter i aggregeringspipelinen. Disse landene (…Y) kan da sjekkes mot verdien satt i hovedland(X), og så postes med riktig streng verdi. Dette kan gjøres via On-Demand Materialized Views.

Her vil det bli kjørt da n+1 spørringer mot databasen hvor n = antall Y. Det vil så bli aggregert fra pipeline som en array.

For denne spørringen gir det mening å bruke en indeks som har referanse til alle landsdokumentene. Å bruke indekser gjør at vi heller ser kun på dokumentene vi ønsker, i stedet for at alle dokumentene i samlingen må sjekkes.

Andre mulige aggregeringer

Kart med prikker for hover/click viser info om f.eks. Cost of Living

Hva er arbeidsledigheten sammenlignet med skatt og Quality of life?

Hvordan påvirker kjønnsfordeling i arbeidslivet quality of life? Hvordan cost of living?

Hvordan påvirker skatt quality of life, og cost of living?

Hvordan påvirker skatten hvor mye penger som brukes på helse og utdanning i et land?

Urban populasjon og arbeidsledighet, hvordan påvirker de hverandre?

Cost of living mot Arbeidsledighet (graf I interval)

Hvorfor dokument database?

I hovedårsak er det gunstig å bruke dokument-orientert database med dette settet da vi kun trenger å organisere basert på land, og vil gjøre svært få kall mot databasen. Aggregeringene våre kan i flere tilfeller da ligge som referanser i hvert land, og vil heller eksistere som sine egne dokumenter. Ved endringer i datasettet vil aggregeringer kjøres på nytt, og endre verdiene i disse dokumentene. Grunnen til at dette kan være til fordel er hvor tungt det fort blir med mange kall mot dokument-orienterte databaser, så det viktigste blir konsistens på dataen. Når data oppdateres i her vil kun dokumentene som er påvirket av endringene ha operasjoner utført på de.

Mange aggregeringer er også mye raskere å få gjort i denne typen database da man henter ut et helt dokument ved spørringen, og alt informasjonen den inneholder. Dette gjør at man slipper å gjøre søk på tvers av tabeller eller må gjøre mere komplekse nøstede spørringer. I tillegg er det en veldig god modell for databaser som ikke gjør veldig mange endringer ofte, hvor tilgang til replika versjoner av data gir redundans ved mange spørringer og holder ytelsen oppe.

En av de aller mest interessante delene av denne typen database er indeksering, og hvordan det kan brukes for å gjøre at spørringer blir mye billigere. Indeksering tillater også da å lagre spørringer rett i RAM, men det krever at man har plass til hele settet man arbeider med også. I vår aggregering som tar inn brukerspesifiserte land og sammenligner kostnad kan potensielt være større da den kun trenger å holde det nyeste resultatet fra hver spørring.

Endring av data

1. Ny data kommer inn
2. Landsdokumentene som blir påvirket vil endre informasjon
   - Dokument-orienterte databaser vil kun gjøre endringer i dokumenter som blir påvirket av ny data, dersom det har skjedd endringer mot data som har vært indeksert vil det som regel være nødvendig å gjenskape indeksen
3. Aggregeringer og indekser som påvirkes og er lagret som dokument kjøres og lages på nytt
   - Dette vil da bli noe av det dyrere som skjer da aggregeringene kan gjøre mange databasekall, men dette vil være avhengig av hvor mange dokument som skal endres, og om endringene som er gjort påvirker de faktiske aggregeringene.
4. Id feltet vil måtte være det første feltet i en dokument, og rekkefølgen på feltene i objektet vil alltid være det samme så lenge en av endringene som skjer ikke omdøping av felt-navnene