COPENHAGEN SCHOOL OF DESIGN AND TECHNOLOGY

01/03/2019

Modulopgave 1
Semester 2
Dat18C

Gruppe 4:

Nicklas Birkehøj List

Aleksandr Uktamovich Sorokin

Frederik Søndergaard Jensen

Esben Flagstad Johannsen

Undervisere:

Jarl Tuxen

Asger B. Clausen

David H. Ema

Indholdsfortegnelse

Problemformulering

Vi blev stillet den opgave at udforme en database samt en applikation der viser statistikken for bevægelse til og fra København og Frederiksberg. Dataen vi har brugt stammer fra Danmarks Statistik, som vi modtog i 3 forskellige .csv filer.

Requirements

- Lave søgninger baseret på brugerens kriterier.
- Returnere en liste baseret på brugerens kriterier.
- Returnere en kombineret liste af flere tabeller.

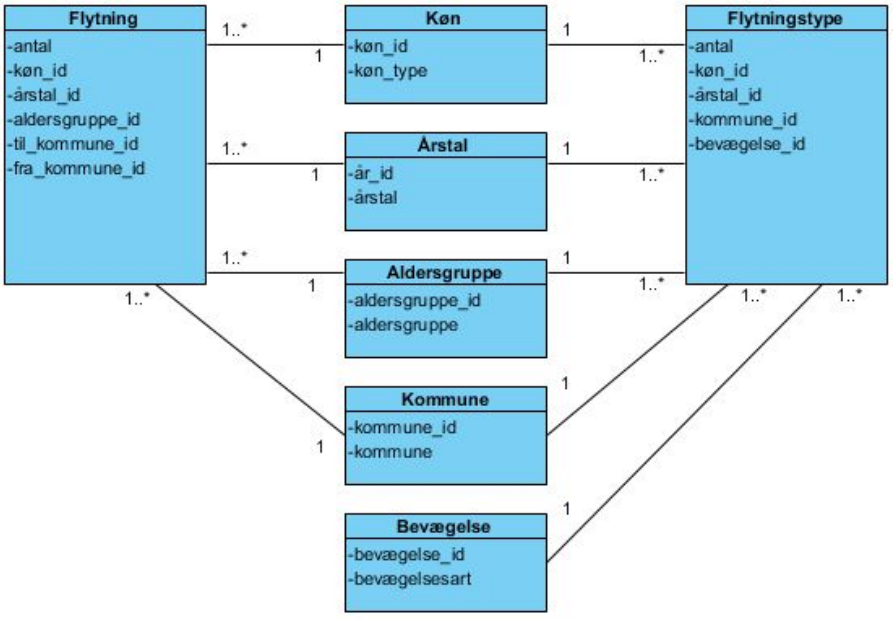
Afgrænsning

Søgning i bevægelsestyper håndteres efter helt samme princip som i til- og fraflytninger, og skaber derfor identiske problemstillinger og løsninger mht. implementering.

Vi vælger at fokusere udelukkende på søgning i til- og fraflytninger, og afgrænser os fra at understøtte søgning på tabellen der indeholder diverse bevægelsestyper (‘Flytningstype’ i EER diagrammet).

Domænemodel

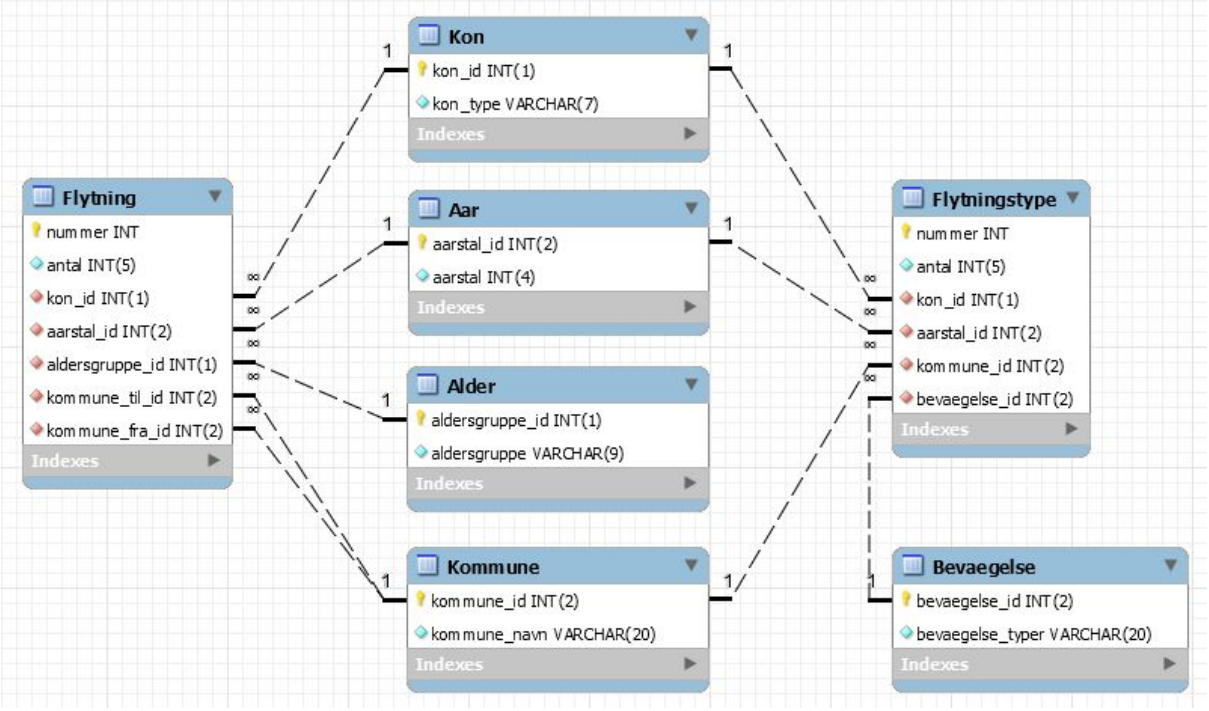
Vores domænemodel og EER diagram er stort set identiske. Primary og foreign keys er erstattet med associationer. Ligeledes er attributen ‘nummer’ ikke til stede i domænemodellen da auto increment relaterer sig specifikt til SQL/EER diagrammet.



EER Diagram

Data for til- og fraflytninger (Flytning) samt bevægelsestyper (Flytningstype) er lagt i hver sin tabel. Hvis vi ikke havde valgt, at afgrænse os til til- og fraflytninger (jf. afgrænsning), ville bevægelsestyper være blevet yderligere opdelt i flere tabeller, for dermed at isolere datamængden for hver type bevægelse.
Kolonnerne 'nummer' er af typen 'auto increment'. Ingen af kolonnerne i vores database accepterer 'null' som input.
Til de to hovedtabeller er der knyttet en række mindre tabeller der alle indeholder en primary key og en forklaringstekst. Der er to primære årsager til denne opdeling:

- Data i hoved tabellerne reduceres til en minimal størrelse og kan udvides ved at joine oplysningstype på til et specifikt udtræk.
- Indholdet af forklaring teksten optræder kun én gang i databasen og er derfor let at vedligeholde/opdatere.



Klargørelse og indlæsning af data

Efter at have valgt et endeligt design af databasen opstod et behov for, at ændre strukturen af rådata. Specifikt skulle de forskellige årstal lægges i samme kolonne. Til det formål, skrev gruppen et mindre java program (jf. bilag) der indlæser data fra en .tsv fil og outputter til en .csv fil. De øvrige tabeller, der for alles vedkommende består af et id med tilhørende forklaringstekst, blev lavet manuelt i excel.

På baggrund af vores EER Diagram, blev der dannet SQL DDL statements gennem forward engineering. Disse statements opretter tomme tabeller med ønskede data definitioner samt indbyrdes relationer. Import Wizard blev brugt til at fylde de tomme tabeller med data fra .csv filerne.
Redundant data i til- og fraflytning for hhv. København/Frederiksberg blev fjernet ved SQL/DISTINCT. Data for bevægelsestype indeholder en masse rækker, der kan dannes på baggrund af andre. Da vi har valgt at afgrænse os, blev der ikke gjort yderligere ved data efter indlæsning.

Normalisering

Vores database design overholder alle 3 normalformer.

Første normal form

- Alle kolonner indeholder atomare værdier. Der gemmes ikke flere værdier i de enkelte celler.
- Der er ingen repeterende kolonne grupper. År er lagt i én og samme kolonne og bryder derfor ikke længere med første normal form. De to kommune kolonner tilhører hver deres type - til og fra og kan derfor godt optræde i hver deres kolonne.

Anden normal form

- Samtlige kolonner afhænger af primærnøglen i deres respektive tabeller.

Tredje normal form

- Der er ingen transitive afhængigheder af primærnøglen i tabellerne.

Konklusion

Vi kan konkludere at vi har fået en bedre forståelse for databaser og deres opbygning, hvordan vi bruger forskellige SELECT statements og INNER JOINS. Vores samarbejde har været ret flydende uden mange skavanker, så vi har ikke været i meget tidspres og har haft tid til at udforme en god opgave efter vores egen mening.

Bilag

```
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.File;
import java.io.FileWriter;
import java.util.Scanner;

public class DataSanitering {

   public static void main(String[] args)
   throws FileNotFoundException, IOException {
      new DataSanitering().transposeData1("tilInput.tsv","tilOutput.csv");
      new DataSanitering().transposeData1("fraInput.tsv","fraOutput.csv");
      new DataSanitering().transposeData2("bevInput.tsv","bevOutput.csv");
   }


   public static void transposeData1(String inputFile, String outputFile)
   throws FileNotFoundException, IOException {
      Scanner scan = new Scanner(new File(inputFile));
      FileWriter output = new FileWriter(outputFile,false);

      boolean firstLine = true;
      while(scan.hasNextLine()) {
         String line = scan.nextLine();
         String[] tokens = line.split("\t");
         String columns = "";
         String delimiter = ",";

         for (int k=0;k<4;k++) {
            columns = columns + tokens[k] + delimiter;
         }
         if (firstLine) {
            output.write(columns + "aar" + delimiter + "antal" + "\r\n");
            firstLine = !firstLine;
         }
         else {
            for (int i=2006; i<2018; i++) {
               output.write(columns + i + delimiter + tokens[i-2002] + "\r\n");
            }
         }
      }
      output.flush();
      output.close();
   }
```

```
    public static void transposeData2(String inputFile, String outputFile)
    throws FileNotFoundException, IOException {
        Scanner scan = new Scanner(new File(inputFile));
        FileWriter output = new FileWriter(outputFile,false);

        boolean firstLine = true;
        while(scan.hasNextLine()) {
            String line = scan.nextLine();
            String[] tokens = line.split("\t");
            String columns = "";
            String delimiter = ",";

            for (int k=0;k<3;k++) {
                columns = columns + tokens[k] + delimiter;
            }
            if (firstLine) {
                output.write(columns + "aar" + delimiter + "antal" + "\r\n");
                firstLine = !firstLine;
            }
            else {
                for (int i=2006; i<2018; i++) {
                    output.write(columns + i + delimiter + tokens[i-2003] + "\r\n");
                }
            }
        }
        output.flush();
        output.close();
    }
}
```