## Introduksjon

(Generell informasjon om nettsiden)

## Hvordan kommer data inn?

Databasen vi har valgt tar utgangspunktet i et datasett som bruker data fra en undersøkelse på studenter fra to skoler i Portugal. For at ny data skal bli lagt inn i databasen, må det gjøres enda en undersøkelse ved enten de samme skolene eller en ny skole. Dataen må deretter legges inn via nettskjemaet med følgende felt på nettsiden:

* Skole
* Kjønn
* Alder
* Mors-utdanning
* Fars-utdanning
* Reisetid
* Studietid
* Stryk
* Skolestøtte
* Ekstratimer (Betalt)
* Aktiviteter utenom faget
* Internett
* Romantisk forhold
* Familie Relasjon
* Fritid
* Gå ut med venner
* Alkohol i ukedagene
* Alkohol i helgen
* Helse
* Fravær
* Karakter semester 1
* Karakter semester 2
* Sluttkarakter

Totalt er det 22 felt som må fylles ut, men når dette er sagt kommer mange av feltene til å ha standardverdier. I tillegg til nettskjemaet vil det være mulig å laste opp datasett i form av CSV eller JSON som har de samme kolonnene. Rader som er duplikater, vil ikke bli tatt hensyn til siden det er mulig å produsere de samme verdiene. Dette vil si at noen må manuelt legge til data via nettsiden for at dataen skal dukke opp i databasen, den vil ikke lytte etter endringer i datasettet via API-er.

# Riak eller ETCD?

I prosjektet vårt har vi bestemt oss for å bruke ETCD, siden ETCD prioriterer konsistent data over tilgjengelighet. Dette er viktig, siden for dette prosjektet er det mye viktigere at dataen stemmer, enn å ha rask respons-tid. Hele hensikten med nettsiden er å forstå studiene som er tatt ved de forskjellige skolene, derfor må dataen være oppdatert. Dessuten gjelder CAP-teoremet i praksis kun når det oppstår en nettverksfeil, så det vil ellers være rask responstid.

ETCD ytelse - <https://etcd.io/docs/v3.4/op-guide/performance/>

# Design av nøkkel

På grunn av måten vi velger å oppdatere eksisterende data i databasen, blir nøklene designet på en måte som skiller mellom komponenter og ny studentdata. Komponentene er JSON objektene som brukes til å vise data på nettsiden, mens studentdataen er hver individuell student. Hvis en student legges til eller oppdateres, oppdateres også komponentene i databasen. Den generelle strukturen på nøkkelen vil derfor være et prefiks for inndeling, etterfulgt av kolon, etterfulgt av navn eller id.

Verdier relatert til komponenter vil ha en nøkkel med prefiks «component» etterfulgt av navnet på komponenten. Eksempel på en nøkkel for en komponent i vår KV-store:

**component:studentsByFreeTime**

Verdier relatert til studenter vil ha nøkler med prefiks «student» etterfulgt av en 8-sifret id av typen Integer. Eksempel på en nøkkel for en student i vår KV-store:

**student:02381225**

KV-store vil dermed følge en lignende struktur:

|  |  |
| --- | --- |
| **Nøkkel** | **Verdi** |
| component:averageGradeByStudyHours | Json {…} |
| component:studentsByFreeTime | Json {…} |
| component:averageGradesByParentEducation | Json {…} |
| component:averageGradesByEducationalAid | Json {…} |
| component:freeTimeByStudyHours | Json {…} |
| component:agesByAlcoholWorkDaysAndWeekends | Json {…} |
| component:familyRelationAndAverageGradesByAlcohol | Json {…} |
| student:01238918 | Json {…} |
| student:54018230 | Json {…} |
| student:32195035 | Json {…} |
| student:10239585 | Json {…} |
| student:31230958 | Json {…} |
| student:35102200 | Json {…} |
| student:10002395 | Json {…} |
| … | … |

# Design av dataobjekter og aggregeringer

## Dataobjekt 1

|  |  |
| --- | --- |
| Representasjon | Json objekt |
|  | *"averageGradesByStudyHours"*: {  *"lessThanTwo"*: {  *"male"*: 10,  *"female"*: 15          },  *"twoToFive"*: {  *"male"*: 15,  *"female"*: 15          },  *"fiveToTen"*: {  *"male"*: 14,  *"female"*: 15          },  *"greaterThanTen"*: {  *"male"*: 13,  *"female"*: 14          }      } |

Dette dataobjektet tar for seg den gjennomsnittlige karakteren for begge kjønn etter hvor mange timer de studerer i uken. Det er representert som et søylediagram i nettsiden.

### Pseudo-kode

for each studyTime value, create groups: gender,

create groups: ("<2", "2 - 5", "5 - 10", ">10"), get average(grades)

### Aggregering

Her samler vi 4 grupper, en for hver tidsbergening, som hver splittes i to basert på kjønn, så regnes snitt-karakter.

## Dataobjekt 2

|  |  |
| --- | --- |
| Representasjon | Json objekt |
|  | *"studentsByFreeTime"*: {  *"veryLittle"*: 11,  *"little"*: 20,  *"medium"*: 30,  *"much"*: 25,  *"veryMuch"*: 14      } |

Her viser dataobjektet hvor mye fritid studentene har

### Pseudo-kode

Get group size of students grouped by freeTime

Calculate percentage

### Aggregeringer

Studenter grupperes etter hvor mye fritid de har, henter kun ut hvor stor hver gruppe er, så regnes det ut hvor stor hver gruppe er av totalen og prosent-verdien deres vises

## Dataobjekt 3

|  |  |
| --- | --- |
| Representasjon | Json objekt |
|  | *"averageGradesByParentEducation"*: {  *"none"*: {  *"fatherEducation"*:10,  *"motherEducation"*:15          },  *"primaryEducation"*: {  *"fatherEducation"*:10,  *"motherEducation"*:15          },  *"fifthToNinthGrade"*: {  *"fatherEducation"*:10,  *"motherEducation"*:15          },  *"secondaryEducation"*: {  *"fatherEducation"*:10,  *"motherEducation"*:15          },  *"higherEducation"*: {  *"fatherEducation"*:9,  *"motherEducation"*:14          }      } |

Denne grafen tar for seg et dataobjekt som samler mors og fars utdanningsnivå og viser karakterene barna har

### Pseudo-kode

get all/updated students, foreach parent's education level, split on gender,

get grades, average with key = parentEducationAndGrades

### Aggregeringer

Her hentes studentenes karakterer ut og legges inn i en gruppe for mors utdanningnivå, og en gruppe for fars, så snittes karakterene innad i disse gruppene.

## Dataobjekt 4

|  |  |
| --- | --- |
| Representasjon | Json objekt |
|  | *"averageGradesByEducationalAid"*: {  *"internet-support"*: 15,  *"internet-extra"*: 14,  *"support-extra"*: 13,  *"all"*: 16,  *"internet"*: 12,  *"schoolSupport"*: 11,  *"extraCourses"*: 12      } |

Dataobjekt som viser snittkarakter med utgangspunkt i tilgang til internett, skole støtte og ekstra betalt kursing

### Pseudo-kode

get grades, for each grade,

check XAND(internet && extraCourses && schoolSupport),

add grade to appropriate variable, add +1 to appropriate ticker,

return array with averages tied to the string values they have

### Aggregeringer

Her grupperer vi studenters karakterer etter tilgang til internett, skole støtte og ekstra betalt kursing, de plasseres i eksklusive grupper etter hva slags midler de har tilgang på, og skal ikke dukke opp i mer enn én gruppe. Når gruppene er satt, regnes snitt karakter for hver gruppe.

## Dataobjekt 5

|  |  |
| --- | --- |
| Representasjon | Json objekt |
|  | *"freeTimeByStudyHours"*: {  *"lessThanTwo"*: 2.5,  *"twoToFive"*: 5.3,  *"fiveToTen"*: 7.5,  *"greaterThanTen"*: 2.5      } |

Dette objektet viser hvor mye fritid studenter har i forhold til hvor mye de studerer

### Pseudo-kode

Group students by Study Time

For each group - add together and average free time for each group

### Aggregeringer

I dette dataobjektet må vi hente ut studenter i grupper etter hvor mye de jobber med skole på egen tid, og finner gjennomsnittet av hvor mye fritid de har på skalaen.

## Dataobjekt 6

|  |  |
| --- | --- |
| Representasjon | Json objekt |
|  | *"agesByAlcoholWorkDaysAndWeekends"*: {  *"veryLow"*: {  *"workDays"*: 15,  *"weekends"*: 18          },  *"low"*: {  *"workDays"*: 18,  *"weekends"*: 19          },  *"medium"*: {  *"workDays"*: 22,  *"weekends"*: 18          },  *"high"*: {  *"workDays"*: 18,  *"weekends"*: 22          },  *"veryHigh"*: {  *"workDays"*: 20,  *"weekends"*: 21          }      } |

I dette dataobjektet vises gjennomsnitt alder for hvor mye alkohol som drikkes i ukedager og helg

### Pseudo-kode

Get students age grouped by 1: weekend and 2: weekday alcohol consumption

(the same student can have two different values for each column and should exist in both)

Get the average age for each group

### Aggregeringer

Her hentes det ut en liste med studenter, så sjekkes mengden alkohol de drikker i helg og i ukedagene. Deretter blir alderen deres lagt inn på gruppa de hører hjemme hos i hver av kategoriene. Så regnes gjennomsnittsalder for gruppen ut.

## Dataobjekt 7

|  |  |
| --- | --- |
| Representasjon | Json objekt |
|  | *"familyRelationAndAverageGradeByAlcohol"*: {  *"veryLow"*: {  *"workDays"*: {  *"familyRelation"*: 3.4,  *"grade"*: 12              },  *"weekends"*: {  *"familyRelation"*: 4.9,  *"grade"*: 14              }          },  *"low"*: {  *"workDays"*: {  *"familyRelation"*: 4,  *"grade"*: 14              },  *"weekends"*: {  *"familyRelation"*: 3.6,  *"grade"*: 11              }          },  *"medium"*: {  *"workDays"*: {  *"familyRelation"*: 3.1,  *"grade"*: 11              },  *"weekends"*: {  *"familyRelation"*: 4.1,  *"grade"*: 15              }          },  *"high"*: {  *"workDays"*: {  *"familyRelation"*: 3.1,  *"grade"*: 8              },  *"weekends"*: {  *"familyRelation"*: 2.8,  *"grade"*: 12              }          },  *"veryHigh"*: {  *"workDays"*: {  *"familyRelation"*: 0.8,  *"grade"*: 7              },  *"weekends"*: {  *"familyRelation"*: 1.9,  *"grade"*: 9              }          }      } |

Dataobjektet viser hvor mye alkohol en student drikker(delt etter helg og hverdag), hvordan forholdet studenten har til familien sin er, og hvordan dette påvirker karakterene i gjennomsnitt

### Aggregeringer

I dette objektet henter vi ut grupper på lignende måte som i dataobjekt 6. Vi skaper da også 5 hovedgrupper med 2 subgrupper hver, en for helg og en for ukedag, og i hver av disse gruppene legges karakter og familie forhold inn og det utregnes gjennomsnitt.

Denne aggregeringen vil ha mange operasjoner og vil være treg, men dataen har ikke noe særlig krav om å være lett tilgjengelig, eller at den må lastes spesielt fort, da det heller er viktig at dataen er korrekt.

## Dataobjekt 8

|  |  |
| --- | --- |
| Representasjon | Json objekt |
| Ingen representasjon på nettsiden | {  *"school"*: "GP",  *"gender"*: "female",  *"age"*: 18,  *"motherEducation"*: 4,  *"fatherEducation"*: 4,  *"travelTime"*: 4,  *"studyTime"*: 4,  *"failures"*: 0,  *"schoolSupport"*: true,  *"extraPaidCourses"*: true,  *"extraCurriculars"*: true,  *"internet"*: true,  *"romanticRelationship"*: true,  *"familyRelation"*: 1,  *"freeTime"*: 1,  *"goOut"*: 3,  *"weekdayAlcoholConsumption"*: 1,  *"weekendAlcoholConsumption"*: 2,  *"health"*: 3,  *"absences"*: 3,  *"midtermOne"*: 15,  *"midtermTwo"*: 13,  *"finals"*: 16  } |

Dette dataobjektet er en en-til-en representasjon av en rad i datasettet og representerer en student.

# Oppdatering av data

recieve new/updated student-object

motta oppdatert student-objekt

hent data fra gammelt objekt i KV-store

hent data fra query-objekt fra KV-store

store oppdater query-objekt i KV-store

store oppdatert student-objekt i KV-store

Oppdatering av data vil kreve få direkte/enkle kall, men pga konsistens kriterier for vår data, må read-kallene gjøres lineært som vil ta noe mer tid enn serialiserte kall

En av grunnene til at dataen vil være lett tilgjengelig med få kall er at den ikke nødvendigvis vil endre seg spesielt ofte (mange endringer heller i et kort tidsrom), men aggregeringene som gjøres med dataen er heller det som tar lang tid. Derfor er konsistens på dataen mye viktigere. Vi er avhengig av at dataen er korrekt, og blir sjekket mot klynga mer enn vi er at uthenting av ny data er rask.