Milepæl 2: Design av Nøkkel-Verdidatabase

Gruppe 13: Mats Engelien og Lars Erik Faber

Introduksjon

I denne milepælen har vi funnet et nytt datasett til nettsiden vår. Vi har gått bort ifra musikk-konseptet fra tidligere milepæl, da det ikke var gunstig for funksjonalitet på nettsiden. Datasettet vi har valgt for KV-store inneholder data fra undersøkelser gjort på to skoler i Portugal for å se hvordan studentene opptrer på skolen. Datasettet heter Student Performance Data Set og kan finnes her:

Kilde: https://www.kaggle.com/larsen0966/student-performance-data-set/version/2

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. Siden CAP-teoremet gjelder i praksis kun når det oppstår en nodefeil, betyr det ikke at vi ofrer tilgjengelighet fullstendig, men det er mindre viktig i det tilfelle enn konsistens. Til slutt er det merkverdig å nevne at Riak er mye mer omfattende enn ETCD, derfor blir det også foretrukket i prosjektet.

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 rådata som er nyttig for å holde dataen oppdatert. 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

Som tidligere nevnt har vi delt opp dataobjektene i to grupper: komponenter og student-data. Grunnen til denne oppdelingen er for å optimalisere ytelse ved uthenting og oppdatering av data. Oppdelingen gjør det mulig å oppdatere individuelle komponenter ved å kun lese fra student-dataen som endret seg. Da unngår man å hente alle student-dataene for å oppdatere alle komponentene, og som resultat bruker nettsiden mindre kall til databasen.

Ytelse og oppdatering av data

På grunn av konsistens-kriterier for vår data, må read-kallene gjøres lineært som vil ta mer tid enn serialiserte kall. Ofte vil vi kun bruke et kall for å hente ut data, og heller bruke ytelsen til systemet for operasjoner på denne dataen.

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

Ved første innlastning av nettsiden må den gjøre noen kall til databasen slik at den kan vise data på nettsiden. Siden det er 7 komponenter i denne nettsiden, har vi valgt å lagre hver komponent for seg i KV-store. Grunnen til dette er fordi hver komponent er ganske omfattende, og vi vil heller gjøre noen ekstra kall enn å laste et stort dataobjekt for hver oppdatering. Derfor blir det umiddelbart 7 kall mot KV-store når siden lastes inn.

Dersom vi ikke hadde designet dataen på denne måten, ville vi støtet på et problem dersom kun noen få studenter oppdateres. For eksempel med «averageGradesByStudyHours» komponentet i et datasett med 100 studenter, der 1 student endret verdi:

1. Hente alle studentobjekter fra databasen
2. Bygge et averageGradesByStudyHours objektet på nytt
3. Oppdater averageGradesByStudyHours i KV-store med den nye versjonen
4. Oppdater studenten i KV-store med den nye versjonen

Det første steget ville bruke 100 kall, en for hver student, det tredje og fjerde steget er to kall. Til sammen blir dette 102 kall. Men siden vi har strukturert databasen ettersom, vil vi kunne minske dette tallet betraktelig. Med samme eksempel som over, her er måten vi gjør det på:

1. Hente gammelt student-objekt fra KV-store
2. Hente komponent med key=component:averageGradesByStudyHours
3. Oppdatere dataen i komponentet i koden ved å gjøre beregninger
4. Lagre oppdatert komponentet i KV-store
5. Lagre oppdatert student-objekt i KV-store

Steg 1, 2, 4 og 5 gjør 1 kall hver, som betyr at her gjør vi kun 4 kall mot KV-store. Denne metoden vil også brukes for å oppdatere de resterende komponentene, som betyr at hver oppdatering er kun 4 kall.

Det er viktig å punktere at samme metode vil også bli brukt dersom man oppdaterer data ved å laste opp en ny CSV fil. Den eneste forskjellen her er at programmet må først dekonstruere CSV-en for å hente innholdet før man gjør kall mot databasen.

Hva om vi var dataeier?

Den største forskjellen for produktet vårt ville vært hvor korrekt dataen er, hvem som hadde kunne gjort endringer, og hvordan/når de endringene ville blitt gjort. Nå kan det legges til eller endres når som helst, og det brukes verdier som ikke helt passer opp mot verdier vi heller er vant med, eller verdier som ikke er dekker hele spekteret vi ser etter (foreldreutdanning skulle gjerne hatt 3 ekstra verdier for bachelor/høyere utdanning, master, og phd. f.eks.).

Datasettet hadde mest sannsynlig da sett noe annerledes ut, og vært tilpasset våre behov. Dette betyr i hovedsak noe sammenslåing av kolonner, samt verdi endringer fra strenger til tall, eller andre grupperingsmetodikker.

Siden dette er forskning/spørreundersøkelser og data som skal hentes inn fra sikre kilder, men i større volum av gangen, ville det da heller vært logisk å laste hele csv-filer av gangen kontra slik som det vil være nå som man kan legge inn en og en student i nettskjemaet. Selvfølgelig ville det også vært mulig dersom datasett endrer seg, men hele datasettet ville nok blitt lastet på nytt med sjekker om oppdatert data.

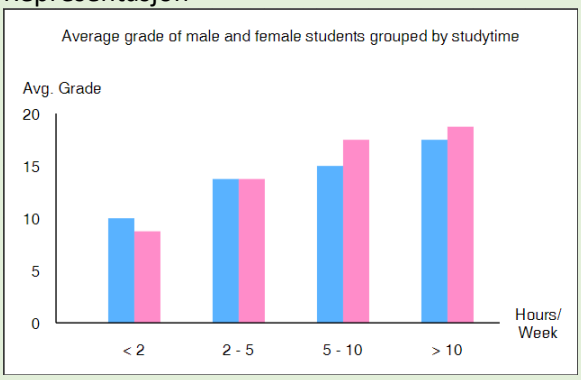
Vi ville fremdeles ikke byttet til RIAK da dataen vi har er mest avhengig av konsistens, og trenger ikke noen spesielt høy grad av tilgjengelighet. Pga. dette vil lineære lese-spørringer i ETCD være bedre enn serialiserte spørringer i RIAK.

Dataobjektene

Denne seksjonen viser fram de forskjellige komponentene/dataobjektene med tilsvarende Json objekt og representasjon på nettsiden. Dette er for å illustrere hvordan vi ønsker at dataobjektene skal se ut. For hvert dataobjekt Vil det stå beskrevet hvordan data aggregeres, samt pseudo-kode på hvordan dataen hentes ut.

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,

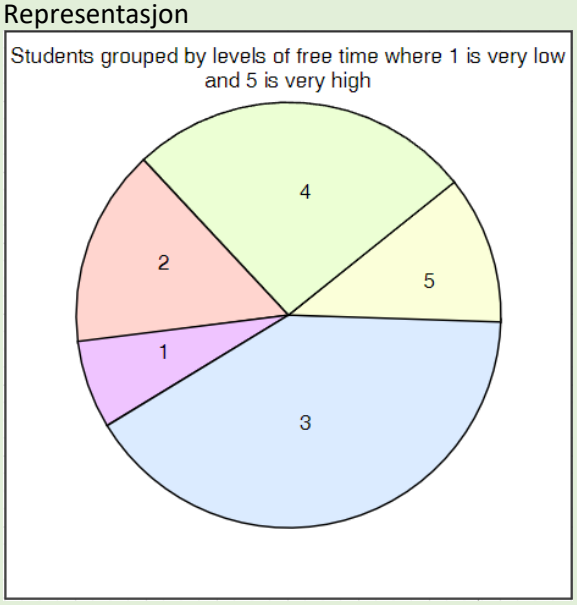
create groups: ("<2", "2 - 5", "5 - 10", ">10"), get average(grades)

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 prosentvis hvor mye fritid studentene har

Pseudo-kode

Get group size of students grouped by freeTime

Calculate percentage

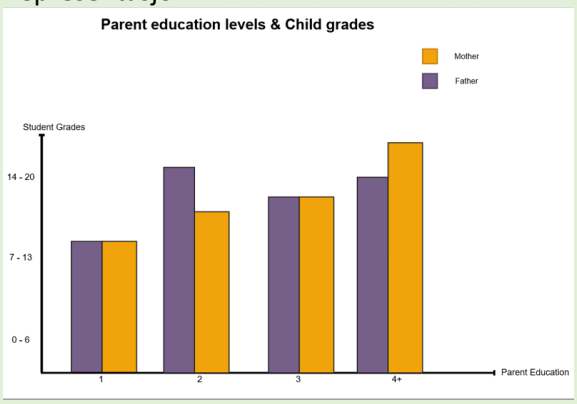
Aggregeringer

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

Gruppe 13

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,

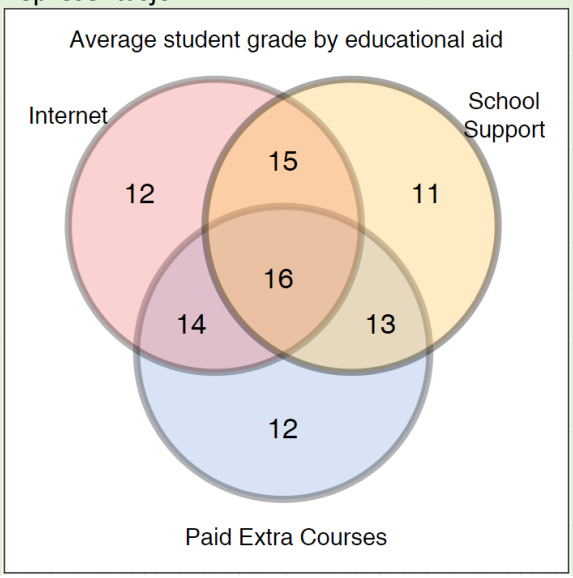
get grades, average with key = parentEducationAndGrades

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

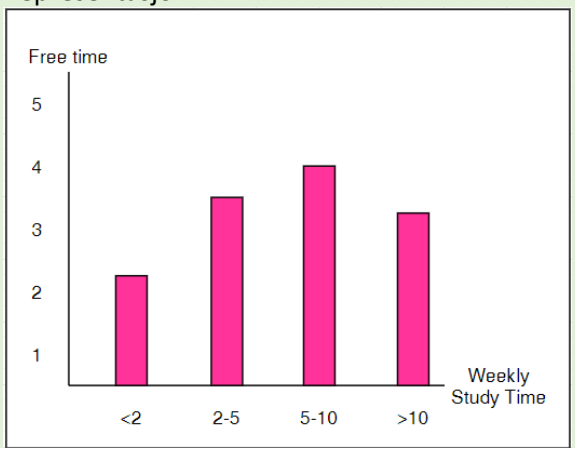
Aggregeringer

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.

Gruppe 13

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

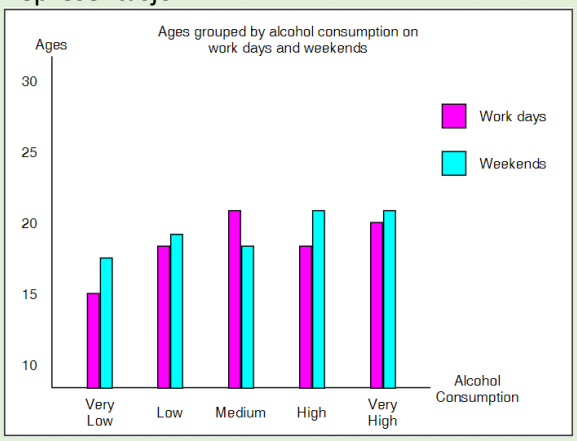
Aggregeringer

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.

Gruppe 13

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)

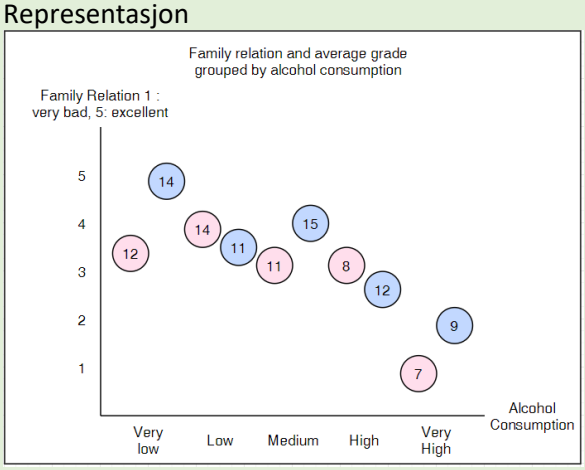
Get the average age for each group

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.