|  |  |
| --- | --- |
| DAT151DB – OBlig1  Høgskulen på Vestlandet | Adrian Mortensen  DAT151 |

Innhold

[Oppgave 1 2](#_Toc30678478)

[1. Oppsett av tabeller 2](#_Toc30678479)

[2. Legge til data i tabellene. 2](#_Toc30678480)

[3. Prøve kommandoer i MariaDB 2](#_Toc30678481)

[a. Endring av «engine» 2](#_Toc30678482)

[b. Vis index 3](#_Toc30678483)

[c. Analyser tabellen 3](#_Toc30678484)

[d. Sjekk av tabell 3](#_Toc30678485)

[e. Reparer tabellen 4](#_Toc30678486)

[f. Optimaliser tabellen 4](#_Toc30678487)

[g. Sjekksum av tabell 4](#_Toc30678488)

[h. Innochecksum program 4](#_Toc30678489)

[Oppgave 2 5](#_Toc30678490)

[1. Forståelse av oppgaven 5](#_Toc30678491)

[2. Besvarelse 5](#_Toc30678492)

[Oppgave 3 6](#_Toc30678493)

[1. Forståelse av oppgaven 6](#_Toc30678494)

[2. Besvarelse 6](#_Toc30678495)

[Oppgave 4 7](#_Toc30678496)

[a) Schema 7](#_Toc30678497)

[b) Implementering til MariaDB 7](#_Toc30678498)

[C) Optimalisering av spørringer 8](#_Toc30678499)

[SELECT COUNT(\*) FROM STUDENT WHERE FCODE = ‘FIN’; 8](#_Toc30678500)

[SELECT DISTINCT CYEAR FROM COURSE\_SCHEDULE; 8](#_Toc30678501)

[Referanser 9](#_Toc30678502)

# Oppgave 1

All kildekode output kan finnes på [github](https://github.com/H571531/DAT151) som sql filer.   
Samt kjørte kommandoer og print av output.

## Oppsett av tabeller

1. **DROP** **TABLE** **IF** **EXISTS** testTable**;**
2. **DROP** **TABLE** **IF** **EXISTS** person**;**
3. **DROP** **TABLE** **IF** **EXISTS** bedrift**;**
4. **CREATE** **TABLE** bedrift**(**
5. bedriftsId **varchar(**4**),**
6. navn **varchar(**8**),**
7. **CONSTRAINT** bedriftPK **PRIMARY** **KEY** **(**bedriftsId**)**
8. **);**
9. **CREATE** **TABLE** person**(**
10. enId **varchar(**4**),**
11. navn **varchar(**8**),**
12. alder **int,**
13. bedriftsId **varchar(**4**),**
14. **CONSTRAINT** bedriftFK **FOREIGN** **KEY** **(**bedriftsId**)** **REFERENCES** bedrift**(**bedriftsId**),**
15. **CONSTRAINT** personPK **PRIMARY** **KEY** **(**enId**)**
16. **);**
17. **CREATE** **TABLE** testTable**(**
18. testId **varchar(**4**),**
19. navn **varchar(**8**),**
20. **CONSTRAINT** testPK **PRIMARY** **KEY** **(**testId**)**
21. **);**

## Legge til data i tabellene.

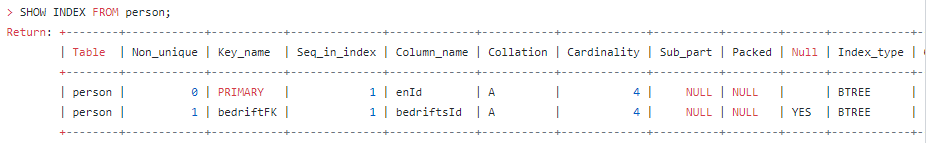
1. **INSERT** **INTO** bedrift **VALUES(**'abcd'**,**'Bedrift1'**);**
2. **INSERT** **INTO** bedrift **VALUES(**'acdc'**,**'Bedrift2'**);**
3. **INSERT** **INTO** person **VALUES(**'admo'**,**'Adrian'**,**24**,**'abcd'**);**
4. **INSERT** **INTO** person **VALUES(**'nono'**,**'Noen'**,**26**,**'abcd'**);**
5. **INSERT** **INTO** person **VALUES(**'pers'**,**'Person'**,**30**,**'acdc'**);**
6. **INSERT** **INTO** person **VALUES(**'andr'**,**'Andre'**,**60**,**'abcd'**);**
7. **INSERT** **INTO** testTable **VALUES(**'admo'**,**'Adrian'**);**

## Prøve kommandoer i MariaDB

### Endring av «engine»

Endrer først engine på «person» tabellen til «InnoDB» dette går fint og blir gjennomført.   
Prøver deretter og endre «person» tabellen til «MyISAM» her feiler operasjonen siden «person» inneholder foreign keys noe «MyISAM» ikke vil ta stilling til. Bruker derfor «testTable» og kjører endringen til «MyISAM» på den uten problemer.

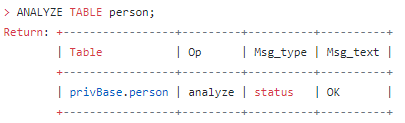
### Vis index



Output fra kjøring kan finnes på github og lagt ved. under "commands.sql"

Ting av interesse i denne outputen er spesielt rundt fremmed nøkkelen «BedriftFK» Den trenger ikke være unik og kan være «NULL».

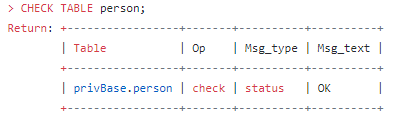
### Analyser tabellen



Output fra kjøring kan finnes på github og lagt ved. under "commands.sql"

Denne kommandoen analyserer tabellen for å finne nøkkeldistribusjon for tabellen (MariaDB, u.d.).

### Sjekk av tabell



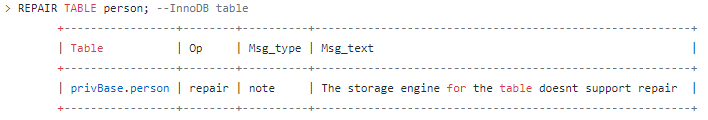
Output fra kjøring kan finnes på github og lagt ved. under "commands.sql"

Denne kommandoen sjekker for feil i tabellen (MariaDB, u.d.).   
Som vist ser alt greit ut i denne tabellen.

### Reparer tabellen

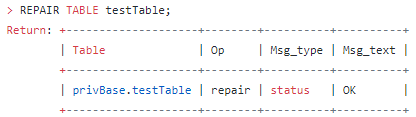
Denne kommandoen reparerer en muligens korrupt tabell. Men fungerer bare for noen av *«storage engine’ene»* (MariaDB, u.d.).

Prøvde først å kjøre på en *InnoDB*-tabell.



Output fra kjøring kan finnes på github og lagt ved. under "commands.sql"

Som vist vil ikke dette kjøre på denne *«enginen»* Prøvde dermed på *testTable* som er en MyISAM tabell fra tidligere kommandoer.



Output fra kjøring kan finnes på github og lagt ved. under "commands.sql"

På en MyISAM-tabell fungerte alt fint.

### Optimaliser tabellen

*OPTIMIZE TABLE* kommandoen har to hoved funksjoner. Enten brukes den til å defragmentere tabellen, eller for å oppdatere InnoDB fulltext indexen (MariaDB, u.d.).

Defragmentering burde brukes om man har slettet store deler av en tabell eller gjort store endringer på en tabell med variabel lengde på rekkene (MariaDB, u.d.).

Oppdatering av InnoDB fulltext indexen må spesifiseres at skal gjøres ettersom denne oppdateringen ikke skjer med engang ved endring; Siden denne operasjonen kan ta lang tid og er ressurskrevende (MariaDB, u.d.).

### Sjekksum av tabell

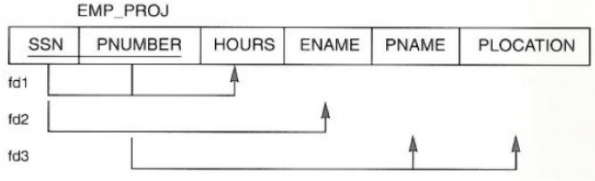
Denne kommandoen rapporterer på sjekksummen på en tabell, slik at man for eksempel kan sjekke om to tabeller er like eller om det er gjort en endring på tabellen (MariaDB, u.d.). Se output i «*commands.sql filen*»

### Innochecksum program

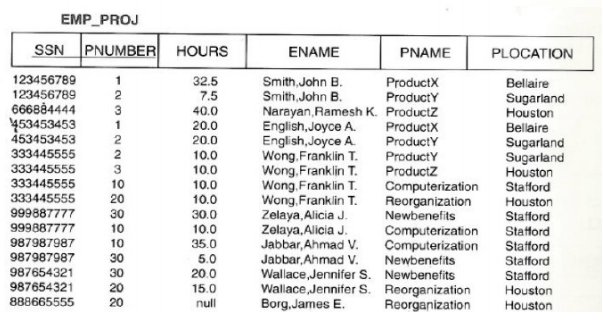
Innochecksum programmet vill man kjøre for å sjekke om tabell filen er korrupt eller om det er noe feil på den. Programmet sjekker om den nåværende checksummen mot en allerede lagret checksum.

# Oppgave 2

## Forståelse av oppgaven



Min forståelse fra dette diagrammet er at *HOURS* er avhengig av *SSN* og *PNUMBER*, ENAME er Avhengig av bare *SSN* og *PNAME, PLOCATION* er avhengig av *PNUMBER*.



I oppgaven er det beskrevet at ENAME kan ansees som en atomisk verdi. Dermed ser vi at dataen er atomisk.

## Besvarelse

Siden alle kolonnene er atomiske, er vi i første grad normalisering 1NF. Men siden *ENAME*, *PNAME* og *PLOCATION* bare er avhengig av 1 av halvdelene i primærnøkkelen er det ikke 2NF.

Lager derfor 2 tabeller ekstra og flytter disse kolonnene ut av EMP\_PROJ tabellen.

Ser nå slik ut:

|  |  |  |
| --- | --- | --- |
| SSN | PNUMBER | HOURS |

|  |  |
| --- | --- |
| SSN | ENAME |

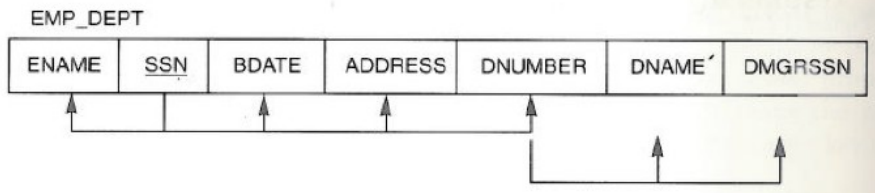
|  |  |  |
| --- | --- | --- |
| PNUMBER | PNAME | PLOCATION |

Dette oppfyller både 2NF og 3NF.

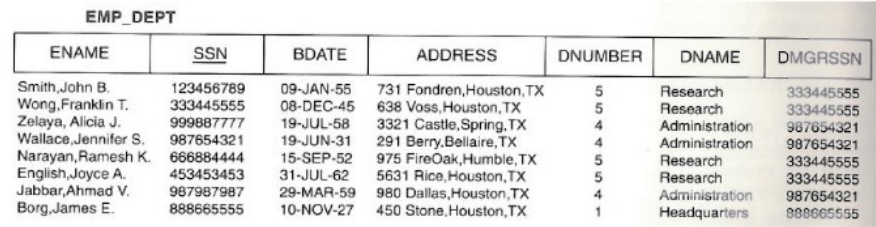
Siden alle attributtene som ikke er nøkler er direkte avhengig av hele primærnøkkelen og ingen av attributtene har en transitiv knytning til nøklene.

# Oppgave 3

## Forståelse av oppgaven



Jeg forstår denne oppgaven som at alt er avhengig av SSN men DNAME og DMGRSSN er avhengig av DNUMBER som da er avhengig av SSN



All data er atomisk ettersom vi skulle se på ENAME og ADDRESS som atomiske felt.

## Besvarelse

Ettersom all data er atomisk, er i hvert fall i 1NF. Siden all data er knyttet til primærnøkkelen er vi også i 2NF. Men siden DNAME OG DMGRSSN er knyttet transitivt til nøkkelen igjennom DNUMBER splitter vi disse ut av tabellen og lager en ny. Dermed vil modellen nå være:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ENAME | SSN | BDATE | ADDRESS | DNUMBER (FK) |

|  |  |  |
| --- | --- | --- |
| DNUMBER | DNAME | DMGRSSN |

Er nå i 3NF siden alle ikke nøkkel-attributter er nå ikke-transitivt bundet til primærnøkkelen i tabellen

# Oppgave 4

## Schema

Jeg gikk ut ifra at vi kunne lage flere tabeller enn de 3 navngitte. Dermed blir mine tabeller:

**GRADES**

|  |  |  |  |
| --- | --- | --- | --- |
| GradeID | YEAR | CourseID (FK) | StudentID (FK) |

**FACULTY**

|  |  |  |  |
| --- | --- | --- | --- |
| fCode | fName | fPhoneNr | fAddress |

**COURSE**

|  |  |  |
| --- | --- | --- |
| cCode | cName | cYear |

**DEPARTMENT**

|  |  |  |
| --- | --- | --- |
| DepID | DepName | fCode(FK) |

**COURSE\_SCHEDULE**

|  |  |  |  |
| --- | --- | --- | --- |
| SchedID | cYear | TeacherID(fk) | cCode(fk) |

**TEACHER**

|  |  |  |
| --- | --- | --- |
| TeacherID | Name | fCode(FK) |

**PROGRAM**

|  |  |
| --- | --- |
| ProgID | ProgName |

**STUDENT**

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| StudentID | BirthID | sName | CurAddress | sPhoneNr | homAddress | bDate | Gender | sYear | level | Faculty(fk) | Program(fk) | DepID(fk) |

## Implementering til MariaDB

Velger å skrive en .sql fil og source den til databasen kildekoden finnes på [github](https://github.com/H571531/DAT151) og lagt ved.

## C) Optimalisering av spørringer

#### SELECT s.SNAME, f.FNAME FROM STUDENT s, FACULTY f where s.FCODE=f.FCODE;

Denne vil da behøve en join for å finne fname fra student. Derfor kunne det vært ideelt å ha denormalisert student tabellen til å inneholde fname for å gjøre denne spørringen raskere.

### SELECT COUNT(\*) FROM STUDENT WHERE FCODE = ‘FIN’;

Denne spørringen vil være grei siden all informasjon finnes i student tabellen.

### SELECT DISTINCT CYEAR FROM COURSE\_SCHEDULE;

Denne spørringen vil også være rask siden alt som trengs er i tabellen.

# Referanser

MariaDB, u.d. *MariaDB Knowledge Base.* [Internett]   
Available at: https://mariadb.com/kb/en/  
[Funnet 2020].