

# 19\_DATABASE\_PROJECT

*02327 – Indledende Databaser Og Database Programmering*



02327 – Indledende Databaser Og Database Programmering

Gruppe nr.: 19

Afleveringsfrist: Fredag den 13/04-2016 kl. 23:59



Silas El-Azm Stryhn  
s143599



Ramyar Hassani  
s143242



Mikkel Hansen  
s143603



Frank Thomsen  
s124206



Martin Rødgaard  
s143043

Denne rapport er afleveret via Campusnet.

Denne rapport indeholder 30 sider inkl. forsider.

## TIMEREGNSKAB

Timeregnskabet er opgivet i procent og viser gruppemedlemmernes  
arbejdsindsats i projektet.

|             | Martin | Frank  | Silas  | Mikkel | Ramyar |
|-------------|--------|--------|--------|--------|--------|
| Indsats i % | 35%    | 16,25% | 16,25% | 16,25% | 16,25% |

## Indholdsfortegnelse

|  |    |
|--|----|
| Timeregnskab .....   | 2  |
| INDLEDNING .....   | 5  |
| Hvilket system skal databasen bruges med .....                         | 5  |
| Baggrund af motivation for projektet .....                             | 5  |
| Problemformulering .....   | 6  |
| Visionen for projektet.....  | 6  |
| Projektstyring af projektplan .....                                    | 6  |
| Rapportstruktur/Resume (hvad indeholder de forskellige kapitler) ..... | 6  |
| PROBLEMANALYSE.....  | 7  |
| Anvendte teknologier .....   | 7  |
| Krav .....   | 7  |
| Funktionelle krav .....  | 8  |
| Ikke-funktionelle krav .....   | 8  |
| E/R Diagram .....  | 8  |
| Normaliserings Analyse .....   | 9  |
| Første normalform .....  | 10 |
| Anden normalform.....  | 10 |
| Tredje normalform.....   | 10 |
| Views .....  | 11 |
| Transactions.....  | 11 |
| Software dokumentation.....  | 12 |
| Skemadiagram.....  | 12 |
| Roller i vægtsystemet .....  | 12 |
| Transaktionslogik .....  | 13 |
| Connector og Constant klasserne.....                                   | 13 |
| DTO klasserne.....   | 14 |
| DAO klasserne .....  | 14 |
| DAOImpl klasserne .....  | 14 |
| Strukturen af databasen .....  | 14 |

|                                     |    |
|-------------------------------------|----|
| Tabel 'raavare' .....               | 14 |
| Tabel 'raavarebatch' .....          | 15 |
| Tabel 'recept' .....                | 15 |
| Tabel 'receptkomponent' .....       | 15 |
| Tabel 'produktbatch' .....          | 15 |
| Tabel 'produktbatchkomponent' ..... | 16 |
| Tabel 'operatoer' .....             | 16 |
| Test .....                          | 17 |
| operatoerTest .....                 | 17 |
| createReceptTest .....              | 19 |
| KONKLUSION .....                    | 20 |
| Fremtidigt arbejde .....            | 20 |
| BILAG .....                         | 21 |
| Git: .....                          | 21 |
| Systemkrav .....                    | 21 |
| Databasen: .....                    | 22 |
| Opgave 1 SQL script .....           | 23 |
| Opgave 1 .....                      | 23 |
| Opgave 2 .....                      | 23 |
| Opgave 3 .....                      | 24 |
| Opgave 4 .....                      | 25 |
| Opgave 5 .....                      | 26 |
| Opgave 6 .....                      | 26 |
| Opgave 7 .....                      | 27 |
| Opgave 8 .....                      | 27 |
| Opgave 9 .....                      | 28 |
| Opgave Q1 .....                     | 28 |
| Opgave Q2 .....                     | 29 |
| Opgave Q3 .....                     | 29 |

## INDLEDNING

I denne rapport vil vi forsøge at give læseren et indblik i hvad vores tanker har været før, under og efter implementeringen af databasen. Den database som vi skal implementere er en database som vi senere skal bruge i projektet i 3 ugers perioden. Det vil sige at det er en database som blandt andet skal kunne opbevare data om brugere, recepter og råvare, men det kan der læses meget mere om i selve rapporten. I rapporten vil man også kunne læse om hvordan vi før implementationen analyserede opgaven, hvor vi blandt andet har lavet nogle diagrammer som kunne hjælpe os gennem implementationen, men også gøre det lettere for andre at kunne forstå hvordan vores database er skruet sammen.

### Hvilket system skal databasen bruges med

Denne database skal bruges til vores Afvejning System i 3-ugers perioden. Systemet er et system med nogle forskellige brugere, administrator, farmaceut, værkfører og operatører. Disse brugere, skal hver især have deres egne rettigheder, til hvad de må kunne oprette/se/gøre i systemet. De forskellige brugeres rettigheder er beskrevet længere nede i vores "Roller i vægtsystemet" afsnit.

### Baggrund af motivation for projektet

Motivationen for dette projekt er at vi skal have lavet en database, som skal være 90 % færdig, så vi er godt rustet til CDIO\_final i 3-ugers perioden.

## Problemformulering

På hvilken måde kan vi lave en database, som kan tages i brug til vores CDIO\_final projekt i 3-ugers perioden?

- Hvad skal inkluderes i databasen?
- Hvordan skal man kunne tilgå databasen?

## Visionen for projektet

Dette projekt skal være så færdigt, at vi kan tage det i brug til vores CDIO\_final, uden at lave nævneværdige ændringer. Der skal som sådan kun kunne tilføjes eller ændre mindre detaljer, så vores fokus i 3-ugers perioden ikke ligger på at færdig lave vores database.

## Projektstyring af projektplan

Vi har valgt at dele denne opgave op, så der er nogen der står for implementeringen og andre der står for opgaveskrivning og at lave diagrammer. Vi har derfor lagt projektet op på github, så det er muligt for hele gruppen at kunne være med i den helt opdaterede database.

Der er beskrevet hvordan man henter vores program over github i vores “Git”-afsnit under Bilag.

## Rapportstruktur/Resume (hvad indeholder de forskellige kapitler)

Vores rapport indeholder et afsnit om Problemanalyse, et om Software Dokumentation og så en konklusion.

I vores problemanalyse kommer vi ind på hvilke teknologier vi kommer til at benytte os af i vores database, samt analyseret, hvordan vores database skal sættes op, så den passer overens med vores CDIO\_final projekt. Vi har også lavet et E/R diagram, som viser hvordan relationerne er imellem vores klasser, og hvordan vi har tænkt os at opbygge vores database. Vi har forklaret lidt om normalisering og hvordan man

implementerer det i en database, på bedst mulig vis og så til sidst forklaret lidt om views og transactions.

Software Dokumentation's afsnittet viser vi hvordan vores databases er skruet sammen vha. et skema diagram, som viser forbindelserne tabellerne imellem samt de forskellige attributters nøgler og vigtighed for databasen. Vi har derefter beskrevet lidt om rollerne i systemet, hvem der skal kunne hvad af de forskellige brugere samt om strukturen i vores database. Til sidst i dette afsnit har vi lavet nogle test, som viser at vores implementation fungerer, og beskrevet disse. Herefter har vi så konkluderet og beskrevet hvad vi kan lave videre på i fremtiden.

## PROBLEMANALYSE

### Anvendte teknologier

For at løse opgaven, har vi blandt andet valgt at bruge views og transactions til at tilgå databasen, da det er en mere sikker måde at tilgå en database, fordi man undgår datatab.

Vi har valgt at lave de views og transactions, som vi har, for at vise at vi kan syntaxen, når vi i CDIO\_final, nok kommer på nogle flere vi skal lave. Så derfor har vi lavet nogle, som måske ikke giver så meget mening, da vi ikke vidste hvilke views og transactions vi ville få så meget ud af at lave og vise. Vores views og transactions vil blive beskrevet længere nede i dette afsnit.

### Krav

Vi har fået til opgave at lave en database, som kan bruges til et afvejningssystem, som kan bruges af forskellige type brugere. Systemet skal kunne tilgå databasen, så ens data er gemt, selvom programmet lukkes og startes op igen. Der er 4 forskellige slags brugere i systemet (Administrator, Farmaceut, Værkfører og Operatør). Disse fire type brugere skal kunne tilgå systemet, nogle med flere restriktioner end andre. Derfor skal man i vores database kunne oprette nye brugere, samt aktivere eller deaktivere brugerne, da man ved en sletning mister data andre steder i databasen. Man skal kunne opdatere alle brugerens informationer, og kunne skifte password. I



systemet skal det være muligt at oprette og ændre i recepter og råvarer, som bliver modtaget, og som skal lægges ind i systemet. Det skal også være muligt at oprette og ændre produkt batches og råvare batches. For produkter og recepter skal der ligeledes kunne blive lagt komponenter på i form af recept komponenter og produkt batch komponenter, som er med til at definere de forskellige recepter og produkt batches bedre. Vores database skal laves i tredje normalform, da man kun skal ændre data et sted, for at det bliver ændret i hele databasen.

### Funktionelle krav

Funktionelle krav er de krav som er nødvendige for at vores database og program kan fungere. For at vores program kan fungere bliver vi nødt til at kunne gemme data om vores operatører, vores recepter og vores råvarer. Det er også nødvendigt at vi kan hente information fra vores database samt tilføje og ændre i vores data.

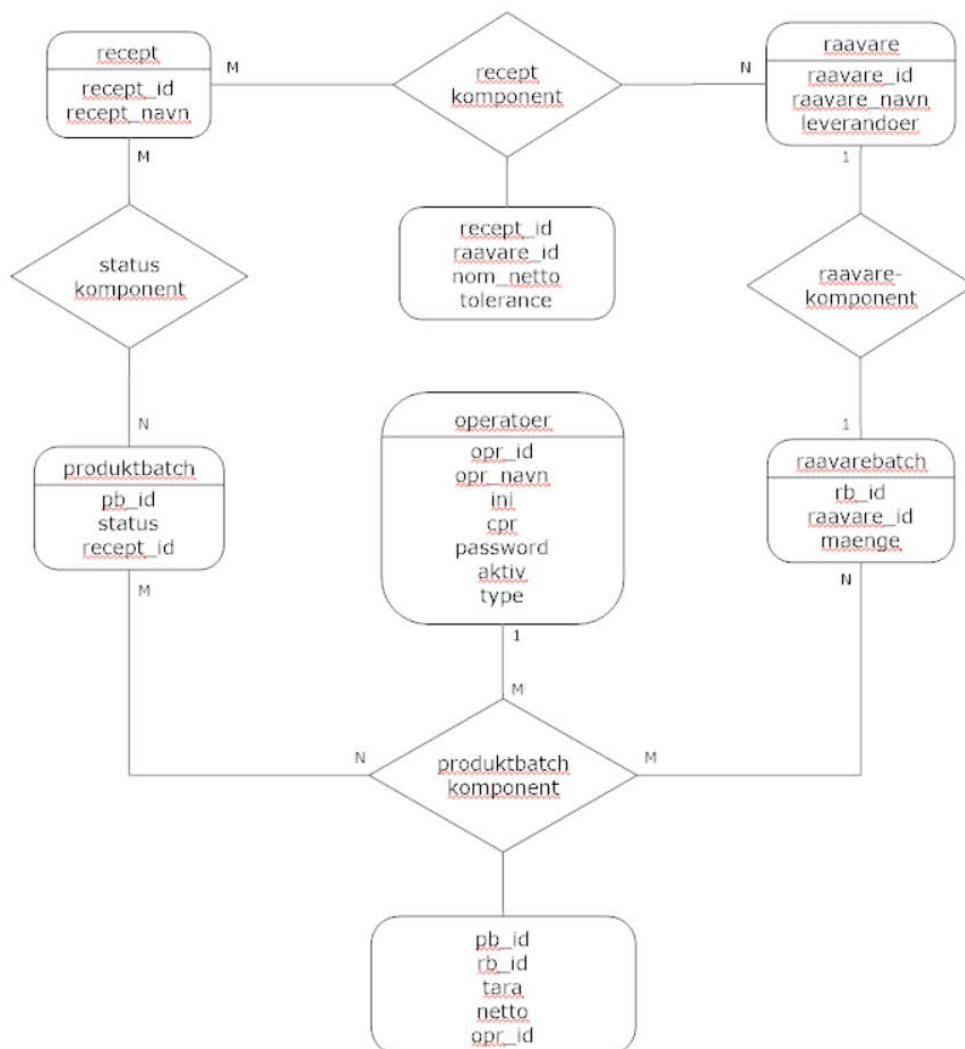
### Ikke-funktionelle krav

Modsat funktionelle krav er ikke funktionelle krav de krav som programmet ville kunne fungere uden. Dette er blandt andet at vi har vores database på 3. normalform. Programmet ville nemlig sagtens kunne fungere på første eller anden normalform. En anden ting som programmet ville kunne fungere uden er noget af dataen, på for eksempel operatøren. Her ville man kunne undvære initialerne, navn og cpr nummer. Da dette kun er krav som umiddelbart bruges til at gøre programmet mere brugervenligt.

### E/R Diagram

Vi har lavet et E/R diagram, som viser vores tanker bag, hvordan vores database skal bygges op. Vi har beskrevet de entiteter, som vi mener skal være med, samt deres attributter. Diagrammet viser så relationerne de forskellige entiteter imellem, og hvordan vores database skal struktureres. Relationerne, har forskellige navne, da en relation kan blive til en tabel senere i forløbet, og kan derfor også indeholde attributter. Som man kan se på vores diagram, har vi også beskrevet forholdet mellem entiteterne, som enten er beskrevet med et N, M eller 1, som betyder at det enten er et "mange til mange" forhold, som er vist med et M ved den ene og et N ved

den anden entitet. Der kan også være et “en til mange” forhold som er vist ved et M og et 1 tal. Til sidst er der et “1 til 1” forhold, som er vist ved et 1 tal ved hver entitet.



## Normaliserings Analyse

Normalisering er en process, som organiserer dataen i en database. Det er både oprettelse af tabeller og relationer imellem dem, som skal følge et sæt regler. Disse regler er lavet, for at beskytte data i databasen, og gøre databasen mere fleksibel, ved at fjerne data, som ikke bliver brugt. Overflødig data bruger unødvendigt plads. Hvis en data findes mere end et sted, og skal ændres, så skal de ændres på præcist samme måde alle steder. Derfor er det nemmere at ændre i databasen, hvis data er gemt få steder.

Reglerne der gælder for database normalisering kaldes en normalform. Hvis første regel er overholdt, så er databasen i første normalform. Hvis anden er overholdt, så er den i anden osv. Vi har i vores projekt brugt tredje normalform, da det anses for at være det højeste niveau inden for de fleste anvendelser. Normalisering kræver ofte flere tabeller og hvis du bryder en af de tre første normaliserings former, så skal dit program tage højde for evt problemer.

Vi har beskrevet de tre første normaliserings former herunder.

#### Første normalform

Det som sker når en tabel er i første normalform er at man fjerner gentagelser i kolonnerne samt at cellerne skal være atomic. Det vil sige at for eksempel hvis vi har en tabel med recepter og ingredienser, hvor en recept og alle ingredienserne er skrevet på en række laver man en ny række til hver ingrediens.

Hver kolonne skal samtidig have et unikt navn, og rækkefølgen af kolonner i en tabel er ligegyldig. Hver række i tabellen har en unik primærnøgle, som er forskellig for de andre rækker. Primærnøglen for en række kan være defineret af en eller flere kolonner.

#### Anden normalform

For at en tabel er i anden normalform, kræver det at tabellen allerede er i første normalform og at ikke nøgle attributter skal afhænge af hele primærnøglen, hvis primærnøglen er defineret af mere end en kolonne. Det er også på denne normalform at man begynder at opdele tabeller og forbinde dem med fremmednøgler.

#### Tredje normalform

For at en tabel er i tredje normalform, kræver det at den i forvejen er i anden normalform og derfor også i første normalform. Når en tabel er i tredje normalform betyder det at man opdeler data i tabeller med relevans til primærnøglen. Det vil sige at for eksempel i vores tabel samler vi alt relevant data for operatørerne i samme tabel og det samme for råvarerne og recepterne.

## Views

Et view er en ny tabel, som er lavet af andre tabeller eller dele af en tabel. Et view bruges til at vise et ønsket antal af kolonner enten fra en tabel eller fra flere tabeller. Dette gør at man fx vil kunne vise en bruger det man ønsker, og ikke alt der er i den givne tabel.

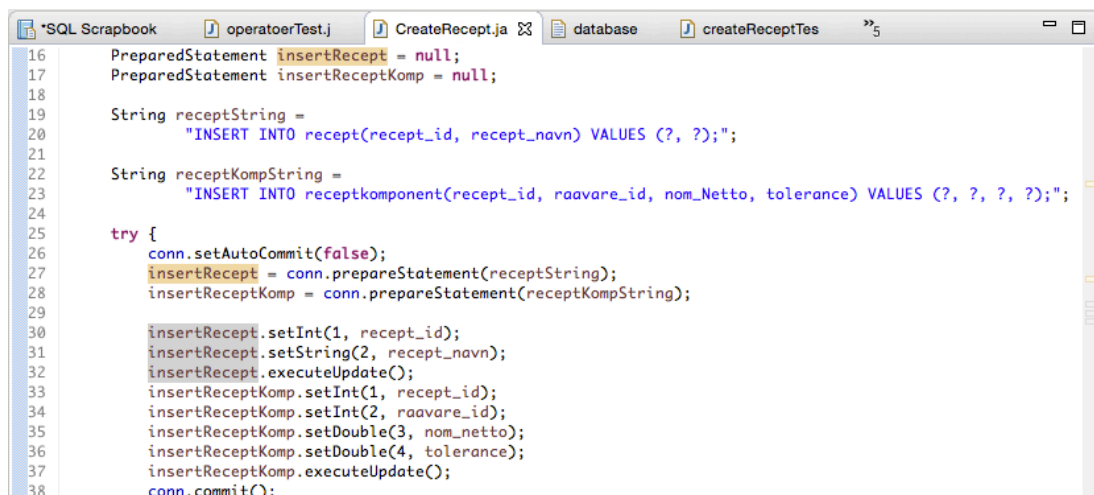
Vi har i vores database valgt at lave et view, som vi har kaldt showDisplayName, som kun viser opr\_id og opr\_navn fra tabellen operatoer, da det kan blive brugt til at blive vist på vores brugergrænseflade i CDIO\_final, så man kan se hvilken bruger det er man er logget ind som. Vi har lavet vores view samtidig med at vores database er blevet sat op.

SQL syntaxen ser således ud:

```
CREATE VIEW showDisplayName AS
SELECT opr_id, opr_navn FROM operatoer;
```

## Transactions

Transactions er en funktion som man bruger, hvis man vil køre flere statements uden at der bliver committed, hvilket er smart, hvis man fx overfører penge fra en tabel til en anden, så vil man skulle opdatere beløbet i to forskellige tabeller, og hvis man mister forbindelsen så vil der ikke blive mistet data, fordi man først committer, når begge statement er blevet kørt. I vores database har vi lavet en transaction, når der bliver oprettet en recept, for der skal der også oprettes en recept komponent til. Vi har så gjort at vi opretter begge dele, inden vi så committer som man kan se i eksemplet herunder.



```
16 PreparedStatement insertRecept = null;
17 PreparedStatement insertReceptKomp = null;
18
19 String receptString =
20     "INSERT INTO recept(recept_id, recept_navn) VALUES (?, ?);";
21
22 String receptKompString =
23     "INSERT INTO receptkomponent(recept_id, raavare_id, nom_Netto, tolerance) VALUES (?, ?, ?, ?);";
24
25 try {
26     conn.setAutoCommit(false);
27     insertRecept = conn.prepareStatement(receptString);
28     insertReceptKomp = conn.prepareStatement(receptKompString);
29
30     insertRecept.setInt(1, recept_id);
31     insertRecept.setString(2, recept_navn);
32     insertRecept.executeUpdate();
33     insertReceptKomp.setInt(1, recept_id);
34     insertReceptKomp.setInt(2, raavare_id);
35     insertReceptKomp.setDouble(3, nom_netto);
36     insertReceptKomp.setDouble(4, tolerance);
37     insertReceptKomp.executeUpdate();
38     conn.commit();
```



samme rettigheder som en værkfører og operatør. Til sidst er der en administrator, som er superbrugeren i systemet, som har rettighederne til at kunne oprette andre brugere, samt opdatere disse. En administrator har også rettigheder til det, som de andre brugere har rettigheder til.

Rollerne i systemet er også opdelt, da det primært er operatører, som skal afveje, selvom de andre brugere også har rettighederne til dette. En værkførers primære rolle er at administrere råvare batches og produkt batches. Selvom både en farmaceut og en administrator har de samme rettigheder, er det ikke deres rolle i systemet.

Farmaceutens rolle i systemet er at administrere råvare og recepter. Administratorens rolle er at oprette, opdatere, og aktivere/deaktivere og se de andre brugere. Dette er beskrevet ved CRUD, som står for “create, read, update and delete”. Her har vi dog erstattet delete med aktiver/deaktiver.

## Transaktionslogik

Vi har lavet noget java implementation til vores database, så man ikke skal direkte ned og rode i den, men der kommer et mellemlag, for hvis man piller direkte ned i databasen, vil der nemt kunne opstå fejl, og dele af databasens informationer vil kunne gå tabt. I de efterfølgende afsnit vil vi komme lidt ind på, hvad vi har lavet af java kode, og hvordan det fungerer.

## Connector og Constant klasserne

Connector klassen er den klasse hvor vi opretter forbindelse til databasen og derfor også en af de vigtigste klasser i hele projektet. Det er også i denne klasse vi har metoderne doQuery og doUpdate som gør at vi kan ændre i databasen som også er vigtigt del af dette projekt.

For at vi kan oprette forbindelse til vores database, kræver det at vi bruger vores Constant klasse, som indeholder konstanter som port nr, servernavn, databasen, username og password. Alle disse konstanter skal bruges, for at oprette forbindelse til databasen, og det bliver gjort i metoden “connectToDatabase(url, username, password)”. port nr, servernavn og databasen hører alt sammen ind under url.

### DTO klasserne

Vores DTO klasser, indeholder konstruktørerne for de forskellige tabeller, samt alle de funktioner som gør, at vi kan ændre i vores database. fx har vi i OperatorDTO klasse en konstruktør som stemmer overens med vores tabel i databasen, og metoderne, kan bruges til at ændre de forskellige attributter, som er i konstruktøren, efter at denne er oprettet.

### DAO klasserne

Vores DAO klasser, er interfaces, som vi bruger til vores klasser hvor vi har lavet vores implementation (DAOImpl klasserne)

### DAOImpl klasserne

Dette er de klasser, hvor vi har lavet vores implementation. De forskellige klasser indeholder metoder, som opretter de forskellige ting som operator, raavare, recepter, produktbatch osv.

De indeholder også metoder, som kan opdatere de forskellige tabellers indhold og til sidst en metode, som kan vise vores tabellers indhold.

### Strukturen af databasen

Hele vores database hænger sammen og er struktureret på en måde, at man kan finde alle informationer, ved at søge efter dem. De forskellige tabeller hænger sammen på forskellige måder, som vil blive beskrevet herunder.

#### Tabel 'raavare'

Denne tabel indeholder information om en råvare og hvilken leverandør som leverer råvaren.

raavare\_id → Holder styr på hvilket ID den pågældende råvare har, og er samtidig primærnøglen.

raavare\_navn → Råvarens navn.

leverandoer → Holder styr på, hvilken leverandør råvaren har

## Tabel 'raavarebatch'

Denne tabel indeholder yderligere information om en råvare, såsom mængden af råvaren.

rb\_id → ID'et for råvarebatchen, samt primærnøglen for tabellen.

raavare\_id → Dette er den fremmednøgle for tabellen, som har en forbindelse til raavare.

maengde → Dette er mængden af en råvare.

## Tabel 'recept'

Denne tabel indeholder de recepter, som kan blive lavet i vores afvejningssystem.

recept\_id → ID'et for en recept, samt primærnøgle for tabellen.

recept\_navn → Dette er navnet af recepten.

## Tabel 'receptkomponent'

Denne tabel indeholder komponenterne for recepterne.

recept\_id → En del af primærnøglen sammen med raavare\_id. Samt en fremmednøgle, som har en forbindelse til recept.

raavare\_id → En del af primærnøglen sammen med recept\_id. Samt en fremmednøgle, som har en forbindelse til raavare.

nom\_netto → Vægten af råvaren.

tolerance → Hvor meget vægten må afvige med.

## Tabel 'produktbatch'

Denne tabel indeholder de forskellige produktbatches.

pb\_id → ID'et for et produktbatch, samt primærnøglen for tabellen.



status → hvert produkt batch har en status, om den er oprettet / under produktion / afsluttet.

recept\_id → Fremmednøgle, som har forbindelse til recept og som viser hvilken recept den er med i.

#### Tabel 'produktbatchkomponent'

Denne tabel indeholder komponenterne til produkt batchene.

pb\_id → En del af primærnøglen sammen med rb\_id. Samt fremmednøglen, som har forbindelse med produktbatch.

rb\_id → En del af primærnøglen sammen med pb\_id. Samt fremmednøgle, som har forbindelse med raavarebatch.

tara → Råvarens emballage vægt.

netto → Råvarens vægt uden emballage.

opr\_id → Fremmednøgle, som har forbindelse til operatoer.

#### Tabel 'operatoer'

Denne tabel indeholder alt information om en operatør, som id, navn, initialer, cpr-nummer og passwordet.

opr\_id → Primærnøglen for tabellen, som viser det unikke ID for en operatør.

opr\_navn → Navnet på en operatør.

ini → Initialerne for en operatør.

cpr → Cpr-nummeret, som er en personlig del af en operatørs informationer.

password → Det kodeord, som skal bruges til at logge ind i vores afvejning system.

aktiv → Fortæller hvorvidt en operatør er deaktiveret eller ej. Dette er en del af vores tabel, som ikke var med fra start, men vi har valgt at indsætte denne, da vi ikke mener man skal kunne slette en bruger, men at man kun skal kunne deaktivere, og selvfølgelig kunne aktivere igen.

type → Viser hvilken type vores operatør er.

(Administrator/Farmaceut/Værkfører/Operatør) Dette er endnu en attribut, som vi selv har sat ind, da vi i vores CDIO\_final skal kunne definere hvilken type en bruger er.

## TEST

Vi har valgt at lave de test, som vi har gjort, for at tjekke om man kan indsætte informationer i tabellerne, vha metoder i java. Vi har lavet test til hver af tabellerne, samt testet om vores transaktion createRecept() virker. Vi har dog valgt, ikke at ville vise alle vores test, men kun 2 af disse, da de fleste af dem foregår på samme måde. operatoerTest og createReceptTest er de to test, som vi beskriver i dette afsnit.

### operatoerTest

Vi har lavet en operatør test, hvor vi opretter en bruger, viser en bruger, og opdatere hans initialer, som man kan se i testen nedenunder.

```

1 package test01917;
2
3 import daoimpl01917.OperatorImpl;
4
5
6
7
8
9
10
11 public class operatorTest {
12     public static void main(String[] args) {
13         try { new Connector(); }
14         catch (InstantiationException e) { e.printStackTrace(); }
15         catch (IllegalAccessException e) { e.printStackTrace(); }
16         catch (ClassNotFoundException e) { e.printStackTrace(); }
17         catch (SQLException e) { e.printStackTrace(); }
18
19         System.out.println("Operator nummer 1:");
20         OperatorImpl opr = new OperatorImpl();
21         try { System.out.println(opr.getOperator(1)); }
22         catch (DALEException e) { System.out.println(e.getMessage()); }
23
24         System.out.println("Indsaettelse af ny operator med opr_id = 2");
25         OperatorDTO oprDTO = new OperatorDTO(2, "Don Juan", "DJ", "000000-0000", "iloveyou");
26         try { opr.createOperator(oprDTO); }
27         catch (DALEException e) { System.out.println(e.getMessage()); }
28
29         System.out.println("Operator nummer 2:");
30         try { System.out.println(opr.getOperator(2)); }
31         catch (DALEException e) { System.out.println(e.getMessage()); }
32
33         System.out.println("Opdatering af initialer for operator nummer 2");
34         oprDTO.setIni("DoJu");
35         try { opr.updateOperator(oprDTO); }
36         catch (DALEException e) { System.out.println(e.getMessage()); }
37
38         System.out.println("Operator nummer 2:");
39         try { System.out.println(opr.getOperator(2)); }
40         catch (DALEException e) { System.out.println(e.getMessage()); }
41
42         System.out.println("Alle operatoerer:");
43         try { System.out.println(opr.getOperatorList()); }
44         catch (DALEException e) { System.out.println(e.getMessage()); }
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100

```

Testen forløb godt, vi fik en ekstra operatør ind i vores database, samt opdateret hans initialer og derefter udskrevet alle operatører som er i databasen. Her forventer vi at der er 2 operatører, da sysadmin er hardcoded ind i databasen.

```

<terminated> operatorTest (10) [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_65.jdk/Contents/Home/bin/java (04/05/2016 16.25.00)
Operator nummer 1:
1   Sys-Admin      SA      000000-0000   Test1234
Indsaettelse af ny operator med opr_id = 2
Operator nummer 2:
2   Don Juan      DJ      000000-0000   iloveyou
Opdatering af initialer for operator nummer 2
Operator nummer 2:
2   Don Juan      DoJu    000000-0000   iloveyou
Alle operatoerer:
[1   Sys-Admin|    SA      000000-0000   Test1234, 2   Don Juan      DoJu    000000-0000   iloveyou]

```

Som man kan se på vores test resultat, så gik det som vi forventede, og fik oprettet, samt opdateret vores operatør med ID 2 og udskrevet begge to i en liste til sidst.

## createReceptTest

I vores createReceptTest har vi testet om vores createRecept transaktion virker. Vi kører vores createRecept transaktion med de nødvendige input, og tjekker om resultatet stemmer overens, ved at udskrive vores resultat i konsollen.

```
public class createReceptTest {
    public static void main(String[] args) throws SQLException, InstantiationException {
        try { new Connector(); }
        catch (InstantiationException e) { e.printStackTrace(); }
        catch (IllegalAccessException e) { e.printStackTrace(); }
        catch (ClassNotFoundException e) { e.printStackTrace(); }
        catch (SQLException e) { e.printStackTrace(); }

        CreateRecept re = new CreateRecept();
        ReceptImpl recept = new ReceptImpl();
        ReceptKompImpl receptKomp = new ReceptKompImpl();

        re.createRecept(3, "Test1", 1, 105.0, 0.6);

        System.out.println("Recept nummer 3:");
        try { System.out.println(recept.getRecept(3)); }
        catch (DAException e) { System.out.println(e.getMessage()); }

        System.out.println("Receptkomponenter til recept nummer 3:");
        try { System.out.println(receptKomp.getReceptKompList(3)); }
        catch (DAException e) { System.out.println(e.getMessage()); }
    }
}
```

Som man kan se i vores test script, indsætter vi en recept med recept\_id 3, kalder den Test1, for den til at linke til en råvare med raavare\_id 1, giver den en netto vægt på 105.0 og en tolerance på 0,6. Vores output bliver så udskrevet som en recept og en receptkomponent, og her forventer vi i recept at der kommer til at være følgende output: 3, Test1

for vores receptkomponent forventer vi følgende output: 3, 1, 105.0, 0.6

```
<terminated> createReceptTest [Java Application] C:\Progran
Recept nummer 3:
3      Test1
Receptkomponenter til recept nummer 3:
[3      1      105.0    0.6]
```

Som man kan se, så stemmer vores resultater, som er udskrevet i konsollen, overens med de resultater som vi forventede for denne test, og derfor fungerer vores transaktion createRecept optimalt.

## KONKLUSION

Vi har lavet en database, som lever op til de krav, som der er blevet stillet til os, om at den skal kunne passe til vores 3-ugers projekt CDIO\_final. Vi har inkluderet views og transactions, da det er den bedste måde at tilgå en database, da man på denne måde undgår data spild og sikrer sig at en række statements bliver kørt, før de bliver gemt i databasen. Hvis der så sker en fejl, eller forbindelsen ryger midt i at et statement bliver kørt, så vil der ikke mangle data.

### Fremtidigt arbejde

I fremtiden vil vi gøre de sidste detaljer færdige, så vores database kommer til at passe 100% til vores CDIO\_final i 3-ugers perioden. Vi har som sagt lavet nogle views og transactions, som ikke er helt optimale, men for at vise at vi kan syntaxen og ved hvordan de fungerer, og derfor nemt kan lave nogle, når vi kommer bedre i gang med vores 3-ugers projekt. Så både views og transactions kommer vi til at arbejde videre på, når vi er kommet lidt længere med projektet.

## BILAG

### Git:

For at få fat i vores projekt, kan man importere det direkte til Eclipse vha. linket [https://github.com/MartinR1993/Database\\_projekt.git](https://github.com/MartinR1993/Database_projekt.git). Dette gøres ved at trykke File → Import → Git → Projects som Git → Clone URI → Indtaster linket i URI-feltet → Next → Finish. Når dette er gjort henter Eclipse selv hele projektet ned. Når først man har hentet projektet ned til Eclipse, skal man blot åbne klassen “main” og køre denne, for at starte spillet.

Man har dog også mulighed for at hente projektet ned som en .ZIP fil, der kan importeres i Eclipse som et eksisterende projekt. Dette gøres ved at trykke File → Import → General → Existing projects into workspace.

### Systemkrav:

Vores spil kan køres på alle maskiner der som minimum kører med Windows XP. Det eneste systemkrav der er, er at man skal have installeret JAVA 1.8, for at kunne bruge det library som vi har anvendt (jre 1.8.0\_60). Man kan dog godt bruge en nyere udgave, så skal man blot ind og ændre i, hvilke libraries der er tilknyttet projektet. Det er ikke noget større problem, men hvis man ikke føler sig sikker på hvordan man tilføjer / fjerner libraries vil vi ikke anbefale det.

## Databasen:

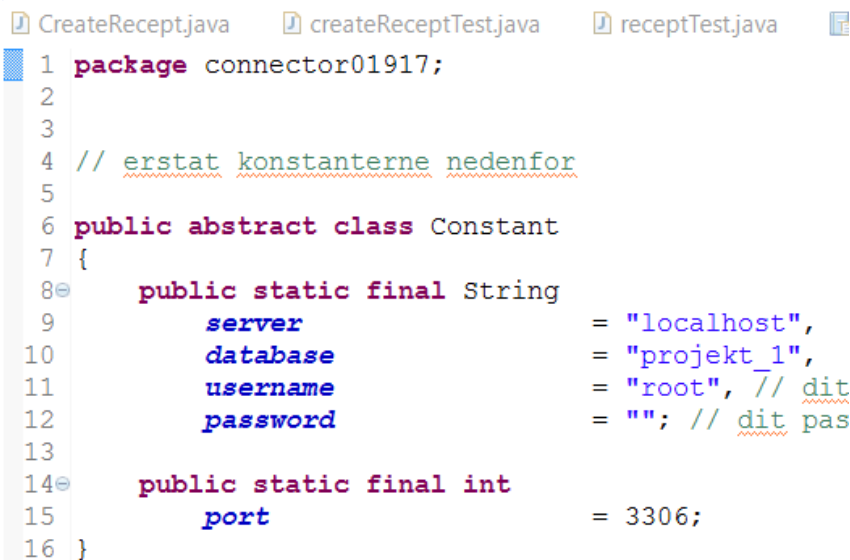
For at kunne benytte vores database og lave vores test, kræver det adgang til vores database. For at få adgang til den, kræver det for det første at den er oprettet, og sat op som den skal.

Vi har i vores rapport lavet lidt om i databasen end det script som vi fik udleveret fra "Pizza" databasen, og den har vi lagt op som en .sql fil inde under mappen "src" i vores projekt.

For at kunne oprette den skal du gå ind under Database Development, og drag and drop database.sql filen. Derefter skal du execute all, ved at højre klikke på skærmen og vælge "Execute all", dernæst er databasen oprettet og sat op som ønsket.

Den sidste ting du skal være opmærksom på, før du kan connecte til databasen er at gå ind i vores Constant.java klasse og ændre "password" til dit password, hvis du har et for at komme ind.

Så det skulle komme til at se ud som på billedet nedenunder, bare med dit password.



```

1 package connector01917;
2
3
4 // erstat konstanterne nedenfor
5
6 public abstract class Constant
7 {
8     public static final String
9         server          = "localhost",
10        database         = "projekt_1",
11        username         = "root", // dit
12        password          = ""; // dit pas
13
14     public static final int
15         port             = 3306;
16 }

```

Derefter har du adgang til databasen, som bliver brugt i vores system.

## Opgave 1 SQL script

### Opgave 1

Bestem navnene på de råvarer som indgår i mindst to forskellige råvarebatches.

```
SELECT raavare.raavare_id, raavare.raavare_navn, raavarebatch.rb_id
FROM raavare NATURAL JOIN raavarebatch;
```

Ved at køre ovenstående kommandoer, får vi som output alle de råvarebatches der findes samt hvad de indeholder.

|   | raavare_id | raavare_navn | rb_id |
|---|------------|--------------|-------|
| 1 | 1          | dej          | 1     |
| 2 | 2          | tomat        | 2     |
| 3 | 3          | tomat        | 3     |
| 4 | 5          | ost          | 4     |
| 5 | 5          | ost          | 5     |
| 6 | 6          | skinke       | 6     |
| 7 | 7          | champignon   | 7     |
|   |            |              |       |

### Opgave 2

Bestem relationen som for hver receptkomponent indeholder tuplen (i,j,k) bestående af receptens identifikationsnummer i, receptens navn j og råvarens navn k.

```
SELECT recept_id, recept.recept_navn, raavare.raavare_navn FROM receptkomponent
NATURAL JOIN raavare NATURAL JOIN recept;
```

Ved at køre ovenstående kode, får vi som output



|    | recept_id | recept_navn | raavare_navn |
|----|-----------|-------------|--------------|
| 1  | 1         | margherita  | dej          |
| 2  | 1         | margherita  | tomat        |
| 3  | 1         | margherita  | ost          |
| 4  | 2         | prosciutto  | dej          |
| 5  | 2         | prosciutto  | tomat        |
| 6  | 2         | prosciutto  | ost          |
| 7  | 2         | prosciutto  | skinke       |
| 8  | 3         | capricciosa | dej          |
| 9  | 3         | capricciosa | tomat        |
| 10 | 3         | capricciosa | ost          |
| 11 | 3         | capricciosa | skinke       |
| 12 | 3         | capricciosa | champignon   |

### Opgave 3

3.1 - Find navnene på de recepter som indeholder mindst én af følgende to ingredienser (råvarer): skinke eller champignon.

```
SELECT DISTINCT recept_navn
FROM recept NATURAL JOIN receptkomponent NATURAL JOIN raavare
WHERE raavare_navn = 'skinke' OR raavare_navn = 'champignon';
```

Ved at køre ovenstående kommandoer, får vi som output de recepter der indeholder enten skinke eller champignon

|   | recept_navn |
|---|-------------|
| 1 | prosciutto  |
| 2 | capricciosa |

3.2 - Find navnene på de recepter som indeholder både ingrediensen skinke og ingrediensen champignon.

```
SELECT DISTINCT recept_id, recept_navn
FROM recept NATURAL JOIN receptkomponent rk1 NATURAL JOIN raavare
WHERE raavare_navn = 'skinke'
AND EXISTS
(SELECT recept_id, recept_navn
FROM recept NATURAL JOIN receptkomponent rk2 NATURAL JOIN raavare
WHERE raavare_navn = 'champignon' AND rk1.recept_id = rk2.recept_id);
```

Ved at udføre ovenstående kommandoer får vi som output de recepter, der indeholder ingredienserne skinke og champignon.

|   | recept_id | recept_navn |
|---|-----------|-------------|
| 1 | 3         | capricciosa |

#### Opgave 4

Find navnene på de recepter som *ikke* indeholder ingrediensen champignon.

```
SELECT DISTINCT recept_navn
FROM recept NATURAL JOIN receptkomponent rk1 NATURAL JOIN raavare
WHERE raavare_navn != 'champignon'
AND EXISTS
(SELECT DISTINCT recept_navn
FROM recept NATURAL JOIN receptkomponent rk2 NATURAL JOIN raavare
WHERE raavare_navn = 'champignon' AND rk1.recept_id != rk2.recept_id);
```

|   | recept_navn |
|---|-------------|
| 1 | margherita  |
| 2 | prosciutto  |

## Opgave 5

Find navnene på den eller de recepter som indeholder den største nominelle vægt af ingrediensen tomat.

```
SELECT recept.recept_id, recept.recept_navn, receptkomponent.nom_netto, raavare.raavare_navn
FROM recept NATURAL JOIN receptkomponent NATURAL JOIN raavare
WHERE (raavare_navn = 'tomat')
AND nom_netto = (SELECT max(nom_netto)
FROM recept NATURAL JOIN receptkomponent NATURAL JOIN raavare
WHERE raavare_navn = 'tomat');
```

|   | recept_id | recept_navn | nom_netto | raavare_navn |
|---|-----------|-------------|-----------|--------------|
| 1 | 1         | margherita  | 2.0       | tomat        |
| 2 | 2         | prosciutto  | 2.0       | tomat        |
|   |           |             |           |              |

## Opgave 6

Bestem relationen som for hver produktbatchkomponent indeholder tuplen  $(i, j, k)$  bestående af produktbatchets identifikationsnummer  $i$ , råvarens navn  $j$  og råvarens nettovægt  $k$ .

```
SELECT pb_id, raavare_navn, netto FROM produktbatchkomponent
NATURAL JOIN raavarebatch NATURAL JOIN raavare GROUP BY pb_id, raavare_navn;
```

|    | pb_id | raavare_navn | netto |
|----|-------|--------------|-------|
| 1  | 1     | dej          | 10.05 |
| 2  | 1     | ost          | 1.98  |
| 3  | 1     | tomat        | 2.03  |
| 4  | 2     | dej          | 10.01 |
| 5  | 2     | ost          | 1.47  |
| 6  | 2     | tomat        | 1.99  |
| 7  | 3     | dej          | 10.07 |
| 8  | 3     | ost          | 1.55  |
| 9  | 3     | skinke       | 1.53  |
| 10 | 3     | tomat        | 2.06  |
| 11 | 4     | champignon   | 0.99  |
| 12 | 4     | dej          | 10.02 |
| 13 | 4     | ost          | 1.57  |
| 14 | 4     | skinke       | 1.03  |

## Opgave 7

Find identifikationsnumrene af den eller de produktbatches som indeholder en størst nettovægt af ingrediensen tomat.

```
SELECT pb_id, netto
FROM produktbatchkomponent
NATURAL JOIN raavarebatch NATURAL JOIN raavare
WHERE raavare_navn = 'tomat'
AND
netto = (SELECT max(netto)
FROM produktbatchkomponent
NATURAL JOIN raavarebatch NATURAL JOIN raavare
WHERE raavare_navn = 'tomat');
```

|   | pb_id | netto |
|---|-------|-------|
| 1 | 3     | 2.06  |

## Opgave 8

Find navnene på alle operatører som har været involveret i at producere partier af varen margherita.

```
SELECT DISTINCT opr_navn
FROM operatoer NATURAL JOIN produktbatchkomponent NATURAL JOIN produktbatch
NATURAL JOIN recept
WHERE recept_navn LIKE 'margherita';
```

|   | opr_navn    |
|---|-------------|
| 1 | Angelo A    |
| 2 | Antonella B |

## Opgave 9

Bestem relationen som for hver produktbatchkomponent indeholder tuplen  $(i, j, k, l, m, n)$  bestående af produktbatchet's identifikationsnummer  $i$ , produktbatchets status  $j$ , råvarens navn  $k$ , råvarens nettovægt  $l$ , den tilhørende receipts navn  $m$  og operatørens navn  $n$ .

**SELECT** pb\_id, status, raavare\_navn, netto, recept\_navn, opr\_navn

**FROM** produktbatchkomponent **NATURAL JOIN** produktbatch **NATURAL JOIN** recept

**NATURAL JOIN** raavarebatch **NATURAL JOIN** raavare **NATURAL JOIN** operatoer;

|    | pb_id | status | raavare_navn | netto | recept_navn | opr_navn    |
|----|-------|--------|--------------|-------|-------------|-------------|
| 1  | 1     | 2      | dej          | 10.05 | margherita  | Angelo A    |
| 2  | 1     | 2      | tomat        | 2.03  | margherita  | Angelo A    |
| 3  | 1     | 2      | ost          | 1.98  | margherita  | Angelo A    |
| 4  | 2     | 2      | dej          | 10.01 | margherita  | Antonella B |
| 5  | 2     | 2      | tomat        | 1.99  | margherita  | Antonella B |
| 6  | 2     | 2      | ost          | 1.47  | margherita  | Antonella B |
| 7  | 3     | 2      | dej          | 10.07 | prosciutto  | Angelo A    |
| 8  | 3     | 2      | tomat        | 2.06  | prosciutto  | Antonella B |
| 9  | 3     | 2      | ost          | 1.55  | prosciutto  | Angelo A    |
| 10 | 3     | 2      | skinke       | 1.53  | prosciutto  | Antonella B |
| 11 | 4     | 1      | dej          | 10.02 | capricciosa | Luigi C     |
| 12 | 4     | 1      | ost          | 1.57  | capricciosa | Luigi C     |
| 13 | 4     | 1      | skinke       | 1.03  | capricciosa | Luigi C     |
| 14 | 4     | 1      | champignon   | 0.99  | capricciosa | Luigi C     |

## Opgave Q1

Angiv antallet af produktbatchkomponenter med en nettovægt på mere end 10. *Hint:*

Brug aggregatfunktionen COUNT.

**SELECT** COUNT(netto)

**FROM** produktbatchkomponent

**WHERE** netto > 10;

|   | COUNT(netto) |
|---|--------------|
| 1 | 4            |

## Opgave Q2

Find den samlede mængde af tomat som findes på lageret, dvs. den samlede mængde af tomat som optræder i raavarebatch-tabellen. *Hint*: Brug aggregatfunktionen SUM.

```
SELECT SUM(maengde)
FROM raavarebatch NATURAL JOIN raavare
WHERE raavare_navn LIKE 'tomat';
```

|   | SUM(maengde) |
|---|--------------|
| 1 | 600.0        |

## Opgave Q3

Find for hver ingrediens (råvare) den samlede mængde af denne ingrediens som findes på lageret. *Hint*: Modificér forespørgslen ovenfor. Brug GROUP BY.

```
SELECT raavare_navn, SUM(maengde)
FROM raavarebatch NATURAL JOIN raavare
GROUP BY raavare_navn;
```

|   | raavare_navn | SUM(maengde) |
|---|--------------|--------------|
| 1 | champignon   | 100.0        |
| 2 | dej          | 1000.0       |
| 3 | ost          | 200.0        |
| 4 | skinke       | 100.0        |
| 5 | tomat        | 600.0        |