

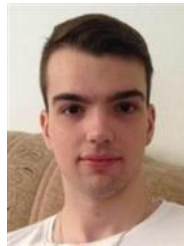
Database projekt



Mikkel
Leth
s153073



Sebastian
Hjorth
s153255



Senad
Begovic
s153349



Thomas
Madsen
s154174



Rasmus
Olsen
s153953



Nicki
Rasmussen
s15344

Afleveringsdato: 13.05.2016

Uddannelsessted: Danmarks Tekniske Universitet

Kursus: 02327

Lærer: Bjarne Poulsen

Indhold

Indledning.....	3
Formål.....	3
Problemformulering	3
Analyse	3
Database struktur	3
ER-diagram	4
Delkonklusion	5
Design og implementering	5
Forbedringer og ændringer	5
Databasen forbedrede struktur.....	6
Relationer	6
Normalisering	7
JDBC	10
Controller & backend	11
Transaktion-logik	11
Procedures.....	12
Authentication.....	12
Constraints.....	13
Views	13
Operatør	13
Værkfører	13
Farmaceut.....	14
Administrator.....	14
Test	14
Delkonklusion	18
Konklusion	18
Projektorienterede	18
Fremtidigt arbejde.....	19
Bilag	20
Bilag 1	20
Bilag 2	28

Bilag 3	29
Bilag 4	30

Indledning

Til denne opgave har vi fået udleveret en database, som indeholder information om et pizzarias produktionskæde. Til denne database har vi fået to opgaver, Database Projektet og CDIO, som vi skal løse med SQL og Java.

Formål

Formålet med dette projekt, er at udvikle videre på den database vi har fået, som giver os mulighed for at bruge den i kombination med faget 02324 videregående programmering. Databasen skal analyseres og forbedres inden at vi kan begynde at implementere vores CRUDL interface i Java.

Problemformulering

Den udleverede database indeholder syv tabeller. Databasen skal analyseres, derefter skal der implementeres interfaces som vi senere vil udvikle data access objekter og til sidst teste om det hele virker. Alle interfaces skal implementeres i Java med Eclipse Mars JDE. Med dette som udgangspunkt er vi kommet med følgende problemstillinger:

- Hvordan implementeres et database interface i java?
- Hvordan kan interfacet testes og bruges?
- Hvorfor er det gjort på denne måde?
- Kunne det gøres anderledes så det kunne hjælpe til det endelige projekt i 02324?
- Hvorfor laver vi interfaces, og hvad kan vi bruge dem til?

Analyse

Database struktur

Vi har fået udleveret en database i starten af projektet, for at få et bedre overblik over databasen har vi fået nogle små opgaver til at komme i gang. Løsningerne på disse opgaver kan ses i bilaget. Efter vi har fået et godt overblik over tabellerne har vi fundet frem til følgende ting om dem:

Databasen består af syv tabeller: Operatoer, Produktbatch, Produktbatchkomponent, Raavare, Raavarebatch, Recept, Receptkomponent.

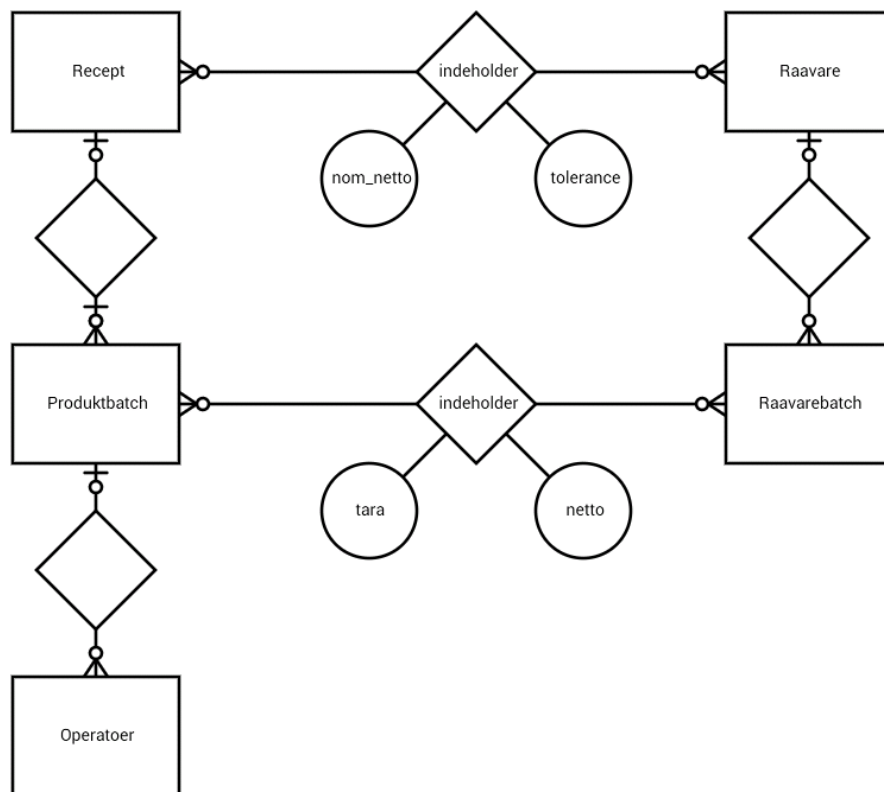
- Operatoer har ansvar for de ansatte der laver pizzaerne. Tabellen har operatørens navn (opr_navn), initialer (ini), password (password), deres cpr-nummer (cpr). cpr er primærnøglen i tabellen.
- Produktbatch holder styr på id'et af produktbatchet (pb_id), status over produktet i produktion vha. 0, 1 og 2 (status) og id'et til receptet hvor produktet bliver brugt (recept_id).
- Produktbatchkomponent har produktbatch id'et ligesom produktbatch (pb_id), råvareet's id (raavare_id), cpr nummeret for den ansat som har afvejet (cpr), samt tara (tara) og netto vægten af afvejning (netto).

- Raavare er de ingredienser pizzariaet bruger til deres vare. Tabellen indeholder råvarens navn (raavare_navn), leverandøren af råvaren (leverandoer) samt råvare id (raavare_id) som er primærnøglen i tabellen.
- Raavarebatch er pizzariaets lagerbeholdning. Her indeholder mængden af ingredienser (maengde), hvor primærnøglen er råvarebatch id'et (rb_id), samt hvor den har en fremmednøgle reference fra råvare (raavare_id).
- Recept er en liste af ingredienser som pizzariaet bruger til at producere varer. Denne tabel indeholder receptens navn (recept_navn) hvor den har primærnøglen recept id (recept_id)
- Receptkomponent er den tabel, som viser de enkelte ingredienser der er i recepterne. Tabellen indeholder en recept id (recept_id), råvare id (raavare_id), en nominelle vægt (nom_netto) samt en vægttolerance (tolerance). Primærnøglerne her er recept id og råvare id.

I kapitlet "Forbedringer & Ændringer" forklarer vi om hvordan vi har ændret databasen til at være bedre at arbejde med og matche de behov vi har i den fremtidige CDIO opgave.

ER-diagram

Efter vi har fået vores analyse af tabellerne på plads, har vi besluttet at lave et ER-diagram over strukturen i vores database:



På diagrammet kan vi se relationerne mellem tabellerne. Vi kan se sammenhængen mellem f.eks. recept og raavare, de har begge en mange-til-mange relation, da der er flere råvarer i en recept, men omvendt kan mange recepter indeholde den samme råvare. Man kan ikke have en mange-til-mange relation, og det er så derfor receptkomponent kommer ind mellem de to relationer. På den måde ville der blive oprettet en receptKomponent tabel hver gang der bliver oprettet en recept eller råvare tabel. Den samme relation gælder mellem raavarebatch og produktbatch med oprettelse af produktbatchkomponent.

Problemet med denne struktur, er at operatøren er bundet til hver råvare. Et eksempel, kunne være, hvis der var flere råvarer med forskellige leverandører. Der vil ske en fejl hvis den leverandørs råvare ikke har flere af slagsen tilbage, men de andre leverandører har den slags råvare til rådighed.

Delkonklusion

Efter at have analyseret databasen, kan vi så sige at den ikke er helt klar til de mål vi skal bruge den til. Der mangler bl.a. mulighed for rangorden til det endelige projekt, normalisering af databasen og generelt problemfri opdatering. Alle disse ting vil vi beskrive i de kommende design afsnit, hvor vi gennemgår vores ideer og ændringer af de forskellige dele af databasen.

Design og implementering

Vi har implementeret interfaces til databasen i Java, samt lavet ændringer i den udleverede database. Dette afsnit indeholder de tanker vi havde, da vi designede databasen og hvad vi endte med at implementere. Vi har vedlagt SQL scripts sammen med denne rapport som kan blive brugt til at gendanne den database vi har anvendt til projektet.

Forbedringer og ændringer

Vi startede ud med at ændre tabellen "raavarebatch" til at hedde "leverandoer". Da vi gjorde dette ændrede vi i tabellens indhold til at passe bedre, hvor vi fjernede raavarebatch id (rb_id) og tilføjede leverandørens navn (leverandor_navn).

Den næste ændring var en medfølge af skabelsen af "leverandoer", da "raavarebatch" havde relationer med andre tabeller, som derved skulle tilpasses til den nye. Vi har fjernet "leverandoer" kolonen i tabellen "raavare", så den kun indeholdte råvarens navn og id, som gjorde det muligt at bruge råvarens id i tabellen "produktbatchkomponent". Dette gjorde det uafhængigt af leverandøren i forhold til det forrige design, hvor hver ingrediens i en produktbatchkomponent specificerede hvilken leverandør råvaren skulle komme fra.

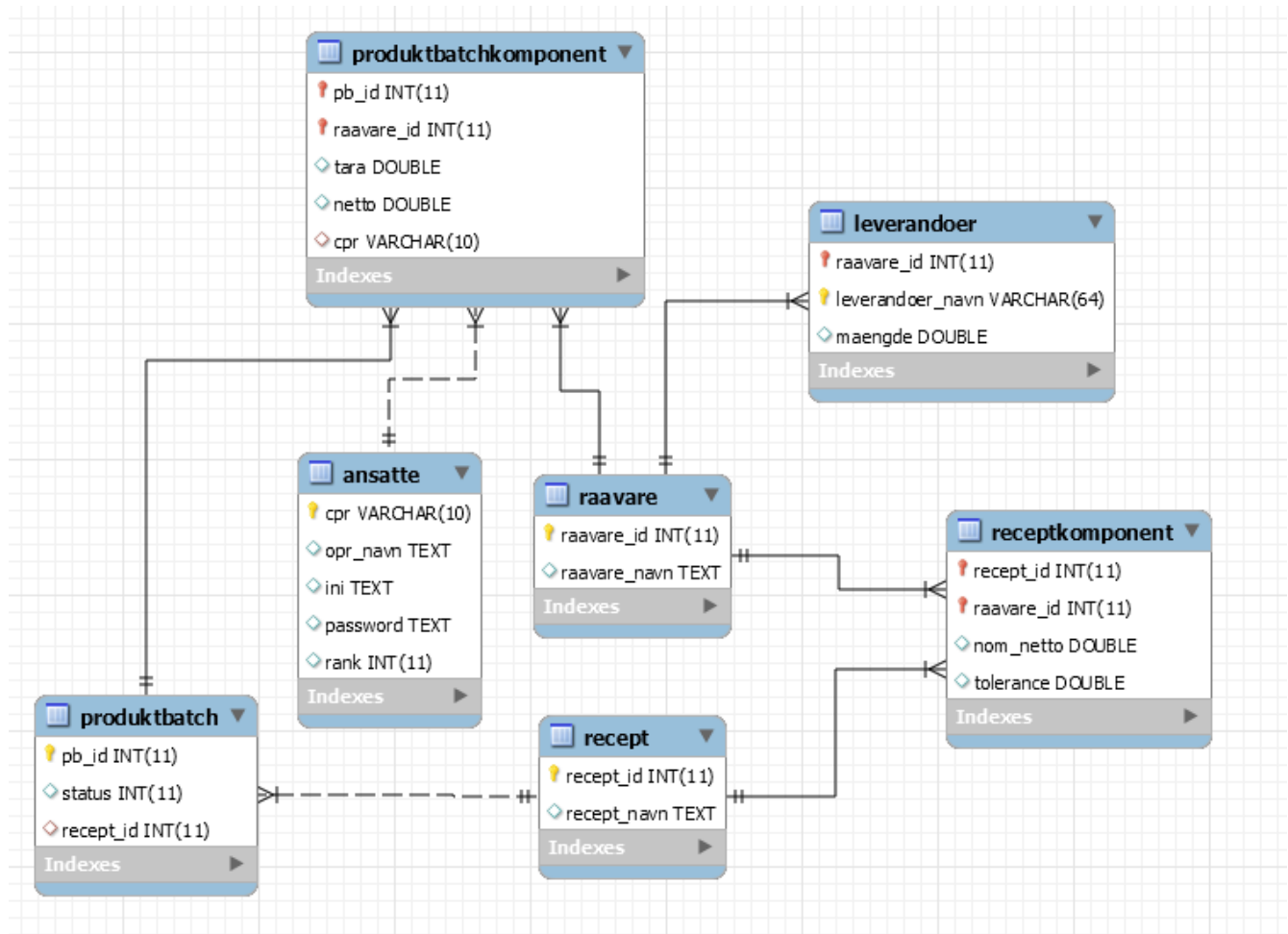
"Leverandoer" indeholder nu en foreign key til "raavare", samt leverandørens navn og hvor meget af den specificerede råvare fra leverandøren der er på lager.

Den sidste ændring vi lavede, var at ændre navnet i tabellen "operator" til "ansat" for at den skal kunne repræsentere ansatte af forskellige rank. Derudover har vi fjernet operatorID, da CPR er en bedre kandidat. Da vi skal arbejde videre med databasen senere hen i 02324 videregående programmering, har vi valgt at tilføje en

kolonne i tabellen, der hedder titel (titel). Titel tilføjes, da de ansatte som sagt kan have forskellig rank. Dette kommer til at være brugbart da ikke alle ansat kan have adgang til lige meget information, og dette ville rank systemet sørge for vha. views, som vi kommer til at beskrive mere senere i rapporten.

Databasen forbedrede struktur

Efter vores ændringer af den udleverede database, kan vi nu få et EER diagram genereret af MySQL Workbench, og så får vi følgende struktur:



Relationer

Dette EER diagramm viser strukturen af vores database, og hvordan denne er bygget op. Vi har ændret en smule i databasen, og dette diagram er med til at give et godt overblik over de præcise ændringer der er foretaget. Man kan bl.a. se at ansatte (tidligere operatør) ikke længere er afhængig af en anden tabel. Vi har derimod ændret i afhængigheden, så leverandør er afhængig af råvaretabellen. Grundet til dette er, at det er leverandøren, som leverer råvaren. Derfor har vi også fjernet forbindelsen mellem leverandøren og recepten.

Normalisering

Vi har hele tiden sat os selv det krav, at vores database skulle opfylde de tre normalformer. Første normalform opfylder vores database ved at have atomiske elementer, hver kolonne har et unikt navn, alle rækker har forskellige primærnøgler og rækkernes orden er ligegyldig.

Anden normalform er opfyldt, først og fremmest fordi databasen er på første normalform, men også fordi alle "ikke-nøgle" attributter er afhængige af hele primærnøglen, og ikke kun en delmængde af denne.

Databasen på tredje normalform, da den opfylder første og anden normalform, og attributterne er ikke afhængige af "ikke-nøgle" attributter, men kun primærnøglen.

Hvis vi starter med at kigge på den tabel, der hedder "raavare".

raavare_id	raavare_navn
1	dej
2	tomat
3	ost
4	skinke
5	champignon
NULL	NULL

Første normalform er opfyldt, da hver kolonne har et unikt navn, "raavare_id" og "raavare_navn". Alle tupler har deres egen unikke primærnøgle, i denne tabel er "raavare_id" primærnøglen, og der er ikke nogen tupler, som har samme "raavare_id". Elementerne er atomiske da der kun er en værdi i en kolonne for en tupel. Anden normalform er opfyldt da et "raavare_navn" er afhængig af hele primærnøglen, "raavare_id". Det giver lidt sig selv, når primærnøglen kun består af en kolonne.

Tredje normalform er opfyldt, lidt på samme måde som anden normalform, og i så lille en tabel som denne, så giver det sig selv. "raavare_navn" er fuldstændig afhængig af "raavare_id", som er primærnøglen.

Næste tabel vi kigger på er recept-tabellen.

recept_id	recept_navn
1	margherita
2	prosciutto
3	capricciosa

Tabellen "Recept" minder meget om tabellen "raavare". Første normalform bliver opfyldt på samme måde, hvor recept_id og recept_navn er unikke, og hvor recept_id udgør primærnøglen i tabellen.

Anden normalform bliver opfyldt hvor recept_navn er afhængig af recept_id.

Tredje normalform bliver også opfyldt, da det er sådan en lille tabel, kan den nærmest ikke andet end at være opfyldt, især efter den har opfyldt anden normalform.

Den næste tabel vi kigger på er receptkomponent.

	recept_id	raavare_id	nom_netto	tolerance
▶	1	1	10	0.1
	1	2	2	0.1
	1	3	2	0.1
	2	1	10	0.1
	2	2	2	0.1
	2	3	1.5	0.1
	2	4	1.5	0.1
	3	1	10	0.1
	3	2	1.5	0.1
	3	3	1.5	0.1
	3	4	1	0.1
	3	5	1	0.1

I denne tabel er det "recept_id" og "raavare_id", som tilsammen udgør primærnøglen. Dette medvirker at alle tupler har deres egen unikke primærnøgle. Igen er alle elementer atomiske, og hver kolonne har deres eget unikke navn, og vi kan derfor sige, at tabellen er på første normalform.

Anden normalform er opfyldt da "ikke-nøgle"-attributterne "nom_netto" og "tolerance" er afhængige af både "recept_id" og "raavare_id", og ikke kun den ene af dem. Hvis vi tager udgangspunkt i "nom_netto", så er "nom_netto"'s værdi 2, når primærnøglen er (1,2) og værdien er 1,5 når primærnøglen er (3,2).

Tredje normalform er opfyldt, "nom_netto" og "tolerance" er afhængige af primærnøglen og ikke en "ikke-nøgle"-attribut.

Nedenunder ses tabellen for produktbatch

	pb_id	status	recept_id
	1	2	1
	2	2	1
	3	2	2
	4	1	3
	5	0	3

Primærnøglen er her "pb_id". Tabellen her er på første normalform, da primærnøglen kun har unikke værdier. Elementerne er atomiske og hver kolonne har deres eget navn.

Tabellen er på anden normalform, da "ikke-nøgle"-attributterne er afhængige af hele primærnøglen, som i dette tilfælde kun er en attribut.

Den er på tredje normalform da "ikke-nøgle"-attributterne kun er afhængige af primærnøglen, og ikke andre attributter. F.eks. er "status" afhængig af "pb_id" og ikke "recept_id".

Her ses tabellen for produktbatchkomponent

pb_id	raavare_id	tara	netto	cpr
1	1	0.5	10.05	0707707007
1	2	0.5	2.03	0707707007
1	3	0.5	1.98	0707707007
2	1	0.5	10.01	0808808008
2	2	0.5	1.99	0808808008
2	3	0.5	1.47	0808808008
3	1	0.5	10.07	0707707007
3	2	0.5	2.06	0808808008
3	3	0.5	1.55	0707707007
3	4	0.5	1.53	0808808008
4	1	0.5	10.02	0909909009
4	3	0.5	1.57	0909909009
4	4	0.5	1.03	0909909009
4	5	0.5	0.99	0909909009

Hver kolonne har sit eget navn, og værdierne er atomare. Da primærnøglen her udgøres af "pb_id" og "raavare_id", så findes der heller ikke to ens primærnøgler, derfor er tabellen på første normalform.

Anden normalform er den også på, da "ikke-nøgle"-attributterne er afhængige af hele primærnøglen. F.eks. er "netto"=10.05 når "pb_id" og "raavare_id" begge er 1.

Når "pb_id" stadig er 1, men "raavare_id" ændres til 2, så er værdien for "netto" 2.03.

Hvis vi så ændrer "pb_id" til 2 og "raavare_id" tilbage til 1, så er værdien for "netto" 10.01. Det viser altså, at anden normalform er

opfyldt.

Tredje normalform er også opfyldt, "netto" er ikke afhængig af "tara" eller "cpr", men udelukkende primærnøglen. Det betyder altså, at "ikke-nøgle"-attributter ikke er afhængige af andre "ikke-nøgle"-attributter, men kun af primærnøglen.

Vi går videre til tabellen for leverandoer

raavare_id	leverandoer_navn	maengde
1	Wawelka	1000
2	Franz	0
2	Knoor	300
2	Veaubais	300
3	Ost & Skinke A/S	100
4	Ost & Skinke A/S	100
5	Igloo Frostvarer	100

Første normalform er opfyldt, af samme årsager som de andre tabeller, det giver sig selv, at en kolonne har sit eget unikke navn, og at elementerne er atomare. Primærnøglen er også unik, da en primærnøgle kun findes en gang. I denne tabel udgøres primærnøglen af "raavare_id" og "leverandoer_navn".

Anden normalform er opfyldt da "maengde" er afhængig af både "raavare_id" og "leverandoer_navn".

Tredje normalform giver sig selv, da der kun er en attribut, som ikke indgår i primærnøglen. Derfor er alle "ikke-nøgle"-attributter kun afhængige af primærnøglen og ikke af "ikke-nøgle"-attributter.

Til sidst har vi tabellen for ansat

cpr	opr_navn	ini	password	titel
0707707007	Angelo A	AA	IKje4fa	0
0808808008	Antonella B	AB	atoJ21v	1
0909909009	Luigi C	LC	jEfm5aQ	2

Igen er første normalform opfyldt, da hver kolonne har sit eget unikke navn og elementerne er atomare. Primærnøglen er også unik, og udgøres i dette tilfælde af "cpr".

Anden normalform er også opfyldt da ikke nøgle attributterne er opfyldt af hele primærnøglen, som kun udgøres af en attribut.

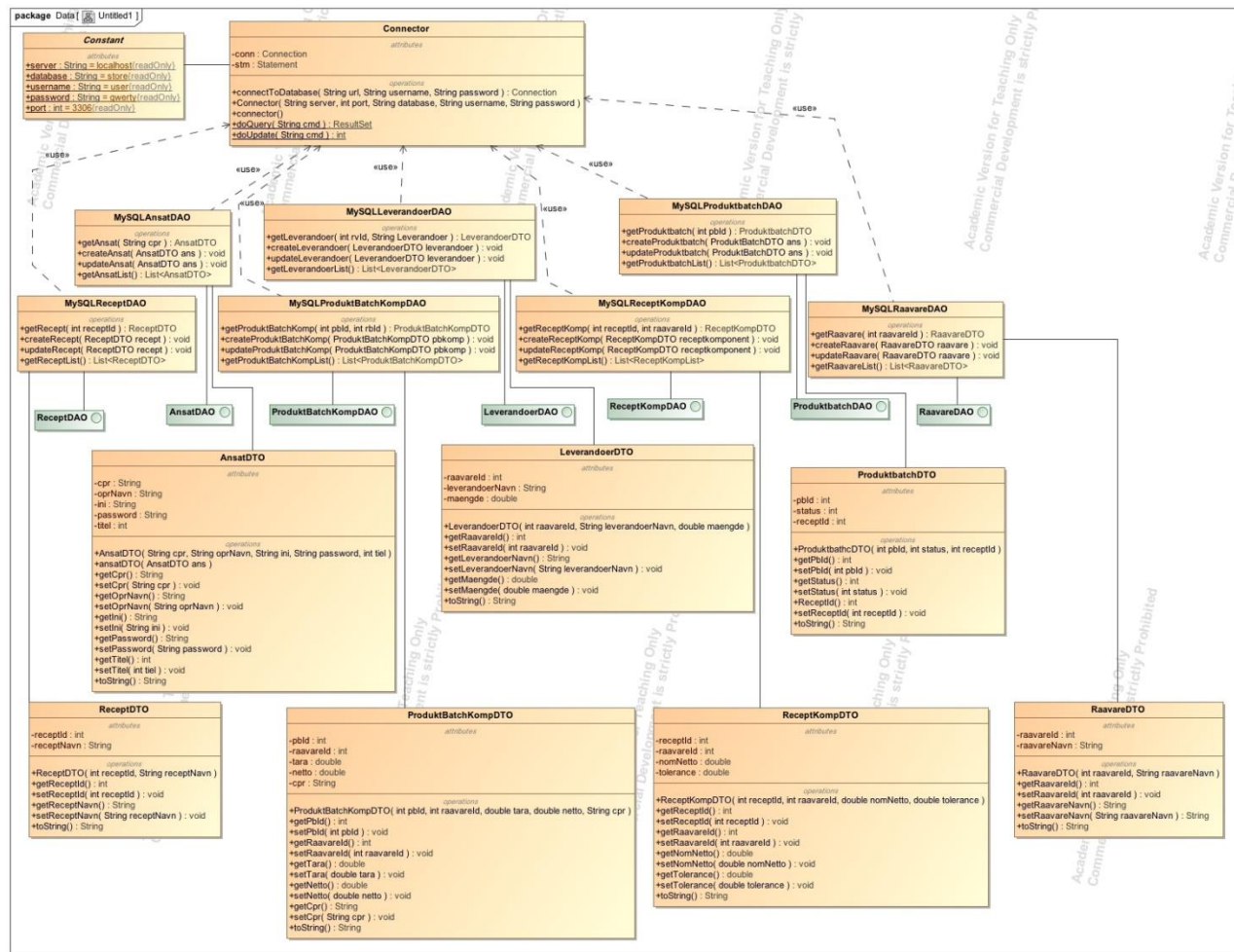
Anden og tredje normalform har vi med fuldt overlæg valgt at bryde med. "Ini" er afhængig af "opr_navn", som er en "ikke-nøgle" attribut, og dermed ikke af primærnøglen. Grunden til dette er, at vi vurderede et brud med disse to normalformer, som værende det bedste valg, hvor alternativet var, at fjerne "ini" fra denne tabel, lave en ny tabel ved navn "initialer", hvor kun "opr_navn" og "ini" ville indgå.

JDBC

Det er en god idé at bruge interfaces til at abstrahere database-kaldende fra koden, da det gør det langt nemmere at skifte det underliggende lag ud hvis man f.eks. senere bliver nødsaget til at skifte fra MySQL til MSSQL. Dette kan endda gøres ved runtime, hvilket er praktisk for visse programmer, såsom database editors der understøtter flere forskellige databaser. Vi har brugt interfaces i vores javakode ved at hver DAO klasser extender fra et DAO interface, som håndhæver hvilke metoder DAO klassen skal indeholde. Da klassen er tvunget til at integrere funktionerne i DAO interfacet, så kan vi tilgå DAO klassen gennem interfacet, da vi er sikrer på at metoder er implementeret.

Vi anvender JDBC til at oprette forbindelse til MySQL databasen. Hertil benytter vi os af filen Connector.java, som har ansvaret for at oprette forbindelse til databasen, samt at udføre queries på databasen via dens metoder. Connector klassen er også markeret som statisk, da det gør det muligt at tilgå klassen på tværs af projektet, hvilket reducerer antallet af forbindelser JDBC skal have til databasen (da hver klasse ellers skulle anmode om deres egen). Connector.java er dog ikke beregnet til at bliver brugt i multithreading applikationer.

Nedenunder kan ses et klassediagram over vores JDBC:



Vi har lavet et klassediagram til at give bedre overblik over det program vi har lavet og hvordan det hænger sammen.

Controller & backend

Vi har valgt at køre meget af vores logik på vores SQL backend frem for i vores JDBC controller, da vores database kommer til at blive brugt både fra en hjemmeside og en vægt.

Vi benytter os derfor af procedures, views og triggers til at have finere kontrol over hvad der kan blive tilføjet til vores database, samt at kunne opdatere vores views til at tilpasse sig ændringer i databasens design.

Transaktion-logik

Vi har valgt ikke at lave nogle transaktioner til denne rapport, da vi først kan bruge dem i CDIO final.

En transaktion er hvor en række SQL statements skal udføres fuldstændt eller slet ikke udføres. Dette kunne fx. være en bankoverførsel, hvor en person betaler penge til en anden person. Når sådan en transaktion tager plads, er der flere ting, som kan gå galt. Der kan være et brud på datastrømmen, som gør at pengene bliver trukket, men aldrig tilføjet til den anden konto. Så for at sikre sig en fuldstændig overførsel, kan man lave en transaktion. En transaktion er serie statements som sørger for, at hele overførelsen tager sted, så vi ikke mister noget data ved fejl. I CDIO kan vi bruge en transaktion til at sørge for at den rigtige mængde af en ingrediens bliver trukket fra vores varelager i leverandoer tabellen, samt at den rigtige mængde bliver registreret som brugt i en recept i vores produktbatchkomponent tabel.

Procedures

Vi har lavet to procedures til vores nuværende database. De er begge sat op sådan at vi kan modificere og opdatere efter behov, uden besvær. Vi bruger vores til procedures til at få data om vores leverandører fra databasen og til at tilføje bruger af databasen.

Vores første procedure er "addUser"¹, som tilføjer en bruger. Den kræver et cpr input på 15 tegn, vi har sat grænsen ved 15, fordi hvis det nu ikke passer med længden med et cpr nr, som er 10 tal, så vil vores trigger give en fejl. Grunden til at grænsen ikke er sat ved 10 tegn er, at vi så ville miste alle overskydende tegn, som blev skrevet, da programmet automatisk beskærer dets input. Så vi sikrer os altså, at der bliver givet en fejlmeddelelse, hvis inputtet er for langt. Vi har valgt at lave det til varchar input, da hvis man kaldte det som int, fjernede den det første tal, hvis det var et nul. Udover et cpr nummer, kræver den et nyt navn, nye initialer, nyt password, og at en rang bliver tilføjet brugeren, så vi brugeren har passende adgang til databasen.

Vores anden procedure er "getLeverandoer"², og den er et eksempel på hvordan, at man ville kunne lave et effektivt databasekald. Det er en simpel kommando, som henter en leverandør ud fra dens ID, og viser de attributter en leverandør nu har. Vi vil kunne arbejde videre i CDIO final projektet med denne type procedures til god effekt.

Authentication

Applikationen bruger kun én konto til at tilgå databasen, selvom det optimale ville være at have én konto pr. rank som kun var privilegeret til at tilgå de opgaver som ranken har tilladelse til. Vi valgte ikke at lave flere konti til dette projekt da det ville være overflødigt at skulle tilgå brugeres rank via én konti og derefter logge ind via en anden konti fra Java, som har de rette tilladelser. Det ville være overflødigt da brugeren stadig ville have adgang til de andre kontiers login-oplysninger da de ville være gemt i klienten. Vores CDIO projekt kan dog gøre brug af dette, da klienten forbinder til en server, som så henter dataene fra en database. Da brugeren ikke har direkte adgang til databasen, så vil det give mening at begrænse brugerens adgang ved at bruge en konti fra serveren, som er begrænset efter brugerens rank. Det bør nævnes at vi ikke ville lave en konti til hver bruger, men snarere en konti til hver rank, lidt ligesom roller i SQL. Det giver mening at gøre det på denne måde i et

¹ Se bilag 3 del 1

² Se bilag 3 del 2

server-client forhold, da det er serveren, og ikke klienten, der har adgang til autentificerings oplysningerne. Årsagen til at begrænse en brugers adgang til databasen på denne måde er at skærme sig bedre mod eventuelle SQL-Injections.

Constraints

Vi har tilføjet constraints til vores ansat tabel, da det var muligt at tilføje CPR numre, som ikke var præcist 10 tal i længden. Det kunne medføre ugyldige CPR numre hvis de blev indtastet forkert. Vi valgte derfor at tilføje en constraint i form af en trigger³, som bliver kaldt hver gang en ny ansat bliver tilføjet til databasen. Den tjekker så om den nye tuples CPR nummer er præcist 10 langt og afbryder indsættelsen med en fejl, hvis det ikke gælder. Vi valgte ikke at tjekke CPR når en tuple bliver opdateret, da CPR er den primære nøgle.

Til gengæld tjekker vi både ved insert og ved update af en tuple om Ini feltet er minimum langt, da Initialer bør være have minimum to karakterer.

En ansats titel kan heller ikke blive sat til andet end tallene fra og med 0 til og med 3 . Dette bliver også fanget af en trigger.

Årsagen til at vi bruger triggers og ikke SQLs constraints er at vores underliggende database(MySQL) ikke supporterer constraints.

Views

I vores program, kommer fire brugertyper til at have adgang til databasen, som hver har deres egne adgangskrav til databasen. Grundet dette er netop fordi de forskellige bruger kun skal have adgang til de informationer, som tilhører deres rolle. De fire brugertyper kommer til at have en rangorden, hvor den laveste er operatøren, og den højeste er administratoren. Desuden vil en med højere rang have mulighed for at åbne de samme views, som dem med en lavere rang end dem selv.

Vi har nedenunder beskrevet hvilken slags adgang de fire forskellige bruger skal have, og hvordan vi har oprettet views til dem.

Operatør

Operatøren er den laveste i den industrielle veje-fødekæde. Operatøren skal kun lave afvejninger. Vi har lavet et view til operatøren som viser de data som er relevante for operatørens arbejde. Viewet viser et udvidet receptkomponent tabel, som også viser tara, netto og navne på råvarenavnene. Se Views tabellen i bilaget.

Værkfører

Værkføreren er over operatøren og derfor nedarver alle operatørens funktioner. Udover disse viser operatørens view også råvarebatches og produktbatches som ønsket. Se Views tabellen i bilaget.

³ Se bilag 4

Farmaceut

Farmaceuten nedarver operatørens og værkførerens rettigheder. Farmaceuten kan også se råvarer og produktbatches. Se Views tabellen i bilaget.

Administrator

Administratoren skal have alle de rettigheder som de andre, administratorens view viser hele databasen. Se Views tabellen i bilaget.

Test

Vi har gjort brug af dummy tests og JUnit tests. Vores dummy test dækker alle funktionskald. Vi er gået ind og direkte kaldt alle: updater, view og list funktioner sådan, at vi opfylder "CRUDL" (Create, read, update, delete & listing). CRUDL dikterer, at man altid bør kunne tilgå sin database og manipulerer sin database efter behov. Vi har ikke testet delete funktioner, da vi ikke har behov for dem i vores database. Vores unit tests har vi brugt til at tjekke om vores data kunne tilgås automatisk, fungerer fuldt igennem java og til at se om vores funktionskald vil gå ind og fylde dataen ind i databasen, og den nye data så kan tilgås ligesom den data som allerede eksisterer i databasen.

Test af produktbatchkomponent + JUnit:

```
ProduktBatchKomponent med pb_id = 4 og raavare_id = 5:  
4      5      0.5      0.99      909909009  
Indsaettelse af nyt produktBatchKomponent med pbId = 4, raavareId = 2, tara = 0.5, netto = 10.1, cpr = 8549003489:  
Opdatering af tara, netto og cpr p test produktBatchKomponentet  
Alle produktBatchKomponenter p en liste:  
[1      1      0.5      10.05      707707007, 1      2      0.5      2.03      707707007, 1      3      0.5      1.98      707707007, 2      1
```

Finished after 0,507 seconds

Runs: 4/4 Errors: 0 Failures: 0

jUnitTest.ProduktBatchKompJUnit [Runner: JUnit 4] (0,456 s)

- testUpdateProduktBatchKomp (0,187 s)
- testGetProduktBatchKomp (0,136 s)
- testGetProduktBatchKompList (0,021 s)
- testCreateProduktBatchKomp (0,112 s)

Test af raavare + JUnit:

```

Raavare med raavare_id = 2:
2      tomat
Indsaettelse af ny raavare med raavare_id = 6 og raavare_navn = Oregano
6      Oregano
Opdatering af navn paa raavaren med raavare_id = 6
6      ananas
Alle raavarer paa en liste:
[1      dej, 2      tomat, 3      ost, 4      skinke, 5      champignon, 6      ananas]

```

Finished after 0,261 seconds

Runs: 4/4 Errors: 0 Failures: 0

jUnitTest.RaavareJUnit [Runner: JUnit 4] (0,207 s)

- testCreateRaavare (0,119 s)
- testGetRaavare (0,046 s)
- testGetRaavareList (0,014 s)
- testUpdateRaavare (0,028 s)

Test af leverandoer + JUnit:

```

RaavareBatch med:
2      Franz      0.0
Indsaettelse af nyt RaavareBatch med rvId = 1, leverandoer = Franz, maengde = 200
Opdatering af rvId, maengde og leverandoer paa test RaavareBatch
Alle RaavareBatch paa en liste:
[1      Franz      200.5, 1      Wawelka 1000.0, 2      Franz      0.0, 2      Knorr      300.0, 2      Veaubais      300.0,

```

Package Explorer JUnit

Finished after 0,49 seconds

Runs: 4/4 Errors: 0 Failures: 0

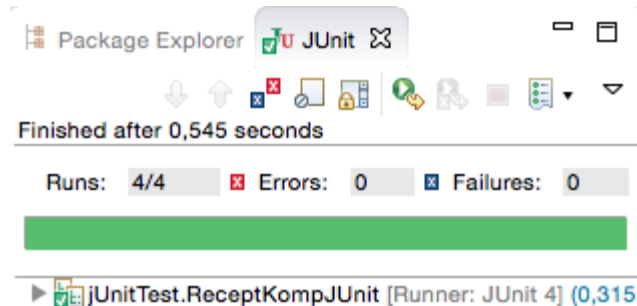
jUnitTest.LeverandoerJUnit [Runner: JUnit 4] (0,416)

Test af receptkomponent + JUnit:

```

Receptkomponent receiptId = 1, raavareId = 1:
1      1      10.0    0.1
Receptkomponent receiptId = 3, raavareId = 5:
3      5      1.0     0.1
Indsaettelse af ny receptkomponent med receiptId = 1, raavareId = 4:
1      4      3.0     0.1
Opdatering af nom_netto for receptkomponent receiptId = 1, raavareId = 1:
1      1      10.0    0.1
List af alle receptkomponenter:
[1      1      10.0    0.1, 1      2      2.0     0.1, 1      3      2.0     0.1, 1      4      5.0     0.1, 2      1      10.0    0.1,
List af alle receptkomponenter med receiptId = 3:
[1      1      10.0    0.1, 1      2      2.0     0.1, 1      3      2.0     0.1, 1      4      5.0     0.1, 2      1      10.0    0.1,

```



Test af ansat + JUnit:

```

Ansatz nummer 0707707007:
0707707007    Angelo A    AA    lKje4fa 0

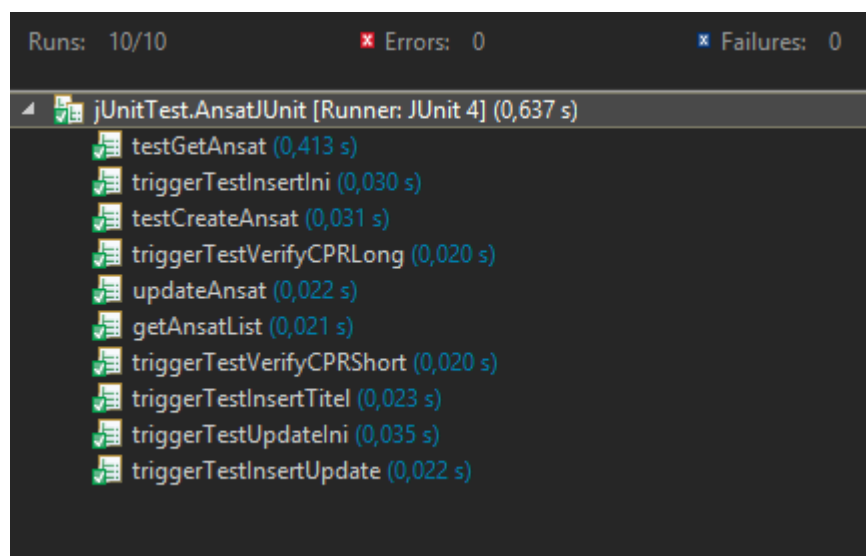
Indsaettelse af ny ansatz med CPR-nummer = 0000000002

Ansatz nummer 0000000002:
0000000002    Don Juan    DJ    iloveyou    1

Opdatering af initialer for ansatte nummer 0000000002
0000000002    Don Juan    DJ    iloveyou    1
0000000002    Don Juan    klmw    iloveyou    1

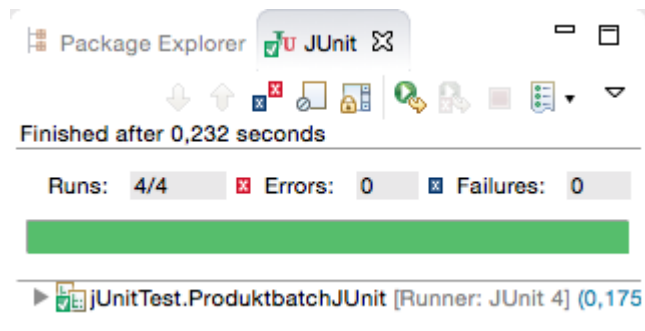
Alle ansatte:
[0000000002    Don Juan    klmw    iloveyou    1, 0707707007    Angelo A    AA    lKje4fa 0, 0808808008    Antonella B

```



Test af produktbatch + JUnit:

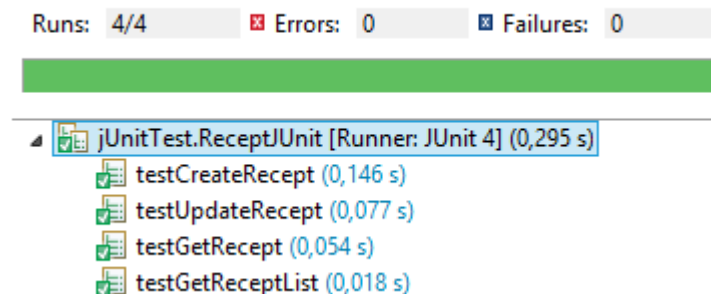
```
Produktbatch med pb_id 3:
3      2      2
Indsaettelse af ny produktbatch med id = 5:
Opdatering af produktbatch med id = 5:
Liste med alle elementer i ProduktBatch:
[1      2      1, 2      2      1, 3      2      2, 4      1      3, 5      2      1]
```



Test af recept + JUnit:

```
Recept med id 1:
1      margherita
Oprettelse af recept med navn kage og id'et 5
Recept med id'et 5:
5      kage
Opdaterer recepten med id'et 5
5      lagkage
Alle ansatte:
[1      margherita, 2      prosciutto, 3      capricciosa, 5      lagkage]
```

Finished after 0,344 seconds



Delkonklusion

Vi har sørget for at vores database er normaliseret efter de 3 første former og ændret vores database og dokumenteret ændringerne ved brug af et EER diagram.

Constraints på databasen blev implementeret for at sikre at der ikke kommer input i databasen som ikke passer ind.

Der blev lavet to nye procedures til brug af fremtidigt arbejde med CDIO. Fremtidigt arbejde i CDIO indebærer også forskellige brugere med forskellige rettigheder i fht. hvad de kan se. Vi har derfor lavet fire views som begrænser hvad hver bruger-type kan se af databasen.

Vi prioriteret en stor backend frem for en stor kontroller i java. Hertil implementerede vi et simpelt CRUD interface i Java ved brug af JDBC, som vi testede ved brug af JUnit og manuel input-tests.

Konklusion

Til allersidst kan vi konkludere, at vi nu har en database og et program, som er klar til at blive arbejdet videre med til CDIO final i faget 02324

Projektorienterede

Ved analyse af databasen fandt vi tidligt ud af hvilke ting vi ville ændre i databasen. Vi ville have de første tre normalformer indført på vores database. Det skulle være nemmere at opdatere information i databasen, hvilket hang sammen med normalformerne. Ansæt blev udvidet til at inkludere en rang, en forberedelse til det kommende CDIO projekt. Disse små ændringer ville være til hjælp senere i udviklingen.

Måden vi har implementeret vores database interface i java er ved at benytte os af den allerede givne connector klasse. Denne connector klasse forbinder jakoden med MySQL databasen, så det fra java af er muligt, at lave MySQL-statements.

Derudover benytter vi os også af et interface, som vi extender i vores DAO'er. I DAOerne laver vi passende kommandoer der kan kalde queries databasen gennem connectoren. Vores implementation er udviklet med CRUD i tankerne, vi har dog ikke implementeret delete funktionalitet da der specifikt blev sagt at de ikke skulle laves.

For at teste at vores interfaces virker som de skal, har vi lavet tests som laver forespørgslerne gennem interfacet og printer resultatet i konsollen. Vi lavede også automatiserede JUnittests. JUnittestene foregår på samme måde, men kontrollerer også at resultatet er hvad man kan forvente. Alle vores tests har været succesfulde og så vidt vi ved er der ikke nogen problemer med interfacet eller tilhørende DAO klasser. Det gode ved interfaces er at man kan indsætte nye klasser der virker med det samme uden, at man behøver at lave nogle ændringer i koden.

Meget af udviklingsprocessen har haft fokus på at man nemt skulle kunne videreudvikle det til CDIO. Vi har f. eks. allerede lavet views og procedures klar til at blive implementeret i CDIO. Her er det også meget praktisk at vores interfaces gør det nemt for os at arbejde videre med og ændre på, uden at der er behov for at gå tilbage for at ændre.

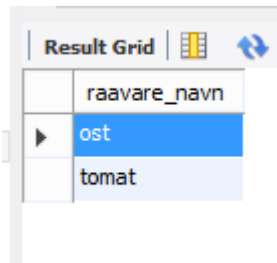
Fremtidigt arbejde

Vi kan bruge vores projekt til fremtidige CDIO projekter. Vi har udviklet en god solid grundstruktur at arbejde videre på og tilpasse det til de potentielle nye behov der dukker op.

Bilag

Bilag 1

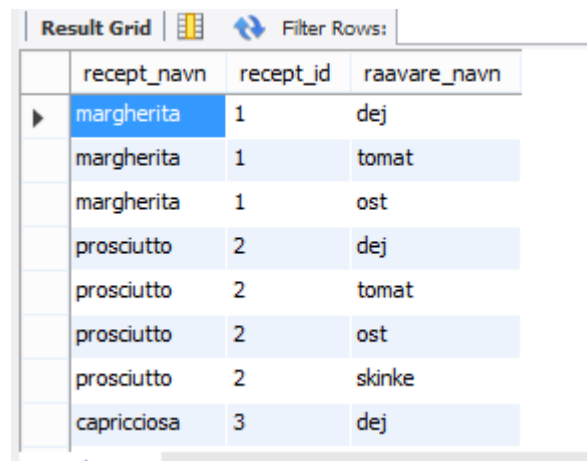
Opgave 1.1 - Bestem navnene på de råvarer som indgår i mindst to forskellige råvarebatches.



raavare_navn
ost
tomat

```
select raavare.raavare_navn from raavare
  join raavarebatch on raavare.raavare_id = raavarebatch.raavare_id
group by raavare_navn having count(raavarebatch.raavare_id)>1;
```

Opgave 1.2 - Bestem relationen som for hver receptkomponent indeholder tuplen (i, j, k) bestående af receptens identifikationsnummer i , receptens navn j og råvarens navn k .



recept_navn	recept_id	raavare_navn
margherita	1	dej
margherita	1	tomat
margherita	1	ost
prosciutto	2	dej
prosciutto	2	tomat
prosciutto	2	ost
prosciutto	2	skinke
capricciosa	3	dej

```
create view opskrift as
```

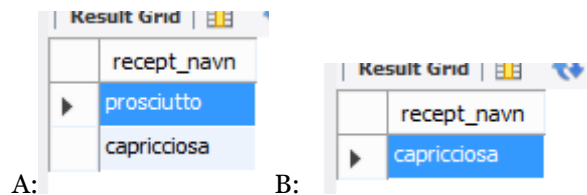
```
select recept.recept_navn, receptkomponent.recept_id, raavare.raavare_navn
from recept
  join receptkomponent
```

```

on recept.recept_id = receptkomponent.recept_id
join raavare
on raavare.raavare_id = receptkomponent.raavare_id;

```

Opgave 1.3 - Find navnene på de recepter som indeholder mindst én af følgende to ingredienser (råvarer): skinke eller champignon. Find navnene på de recepter som indeholder både ingrediensen skinke og ingrediensen champignon.



A:

```

select recept_navn from recept where recept.recept_id in
(
  select receptkomponent.recept_id from receptkomponent where
    (
      receptkomponent.raavare_id in
      (
        select raavare.raavare_id from raavare where raavare.raavare_navn
like 'skinke' or 'champignon'
      )
    )
);

```

B:

```

select recept_navn from recept
join receptkomponent as skinke on skinke.raavare_id in
(
  select raavare_id from raavare where raavare.raavare_navn like 'skinke'
)
join receptkomponent as champignon on champignon.raavare_id in
(

```

```

        select raavare_id from raavare where raavare.raavare_navn like 'champignon'
    )
    and champignon.recept_id=skinke.recept_id where recept.recept_id=skinke.recept_id;

```

Opgave 1.4 - Find navnene på de recepter som *ikke* indeholder ingrediensen champignon.

Vha. to selects (med tilhørende 'where' statements,) inde i vores 'where' condition, specificere vi hvad det er vi leder efter, indtil vi kan vælge alle recepter hvor champignon ikke indgår.

Result Grid	
	recept_navn
▶	margherita
	prosciutto
	capricciosa

```

select recept_navn from recept where recept.recept_id in
(
    select receptkomponent.recept_id from receptkomponent where
        NOT receptkomponent.raavare_id in
        (
            select raavare_id from raavare where raavare_navn like 'champignon')
);

```

Opgave 1.5 - Find navnene på den eller de recepter som indeholder den største nominelle vægt af ingrediensen tomat.

Result Grid	
	recept_navn
▶	margherita
	prosciutto

```

select recept_navn from (recept natural join receptkomponent) where nom_netto=
(
    select max(nom_netto) from receptkomponent where raavare_id in
        (
            select raavare_id from raavare where raavare_navn like "tomat"

```

```

    )
)
and raavare_id in (
    select raavare_id from raavare where raavare.raavare_navn like "tomat"
);

```

Opgave 1.6 - Bestem relationen som for hver produktbatchkomponent indeholder tuplen (i, j, k) bestående af produktbatchets identifikationsnummer i , råvarens navn j og råvarens nettovægt k .

	pb_id	raavare_navn	netto
▶	1	dej	10.05
	1	dej	2.03
	1	dej	1.98
	2	tomat	10.01
	2	tomat	1.99
	2	tomat	1.47
	3	ost	10.07
	3	ost	2.06
	3	ost	1.55
	3	ost	1.53
	4	skinke	10.02
	4	skinke	1.57
	4	skinke	1.03
	4	skinke	0.99

```

select produktbatch.pb_id, raavare.raavare_navn, produktbatchkomponent.netto
from produktbatch, produktbatchkomponent, raavare
where produktbatch.pb_id = produktbatchkomponent.pb_id
and produktbatchkomponent.pb_id=raavare.raavare_id;

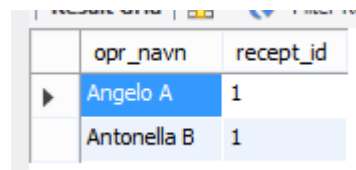
```

Opgave 1.7 - Find identifikationsnumrene af den eller de produktbatches som indeholder en størst nettovægt af ingrediensen tomat.

Result Grid	
	pb_id
▶	3


```
select pb_id from produktbatchkomponent
where (select max(netto) from produktbatchkomponent
      join raavare
      on produktbatchkomponent.raavare_id = raavare.raavare_id
      where raavare_navn like "tomat") = produktbatchkomponent.netto;
```

Opgave 1.8 - Find navnene på alle operatører som har været involveret i at producere partier af varen margherita.



	opr_navn	recept_id
▶	Angelo A	1
	Antonella B	1

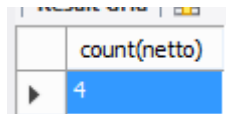
```
use project2;
select distinct ansat.opr_navn, recept.recept_id
from ansat
      join produktbatchkomponent
      on ansat.cpr = produktbatchkomponent.cpr
      join produktbatch
      on produktbatch.pb_id = produktbatchkomponent.pb_id
      join recept
      on produktbatch.recept_id = recept.recept_id
where recept.recept_id in (1);
```

Opgave 1.9 - Bestem relationen som for hver produktbatchkomponent indeholder tuplen (i, j, k, l, m, n) bestående af produktbatchets identifikationsnummer i, produktbatchets status j, råvarens navn k, råvarens nettovægt l, den tilhørende receipts navn m og operatørens navn n.

	pb_id	status	raavare_id	maengde	recept_id	opr_navn
▶	1	2	1	1000	1	Angelo A
	3	2	1	1000	2	Angelo A
	1	2	2	300	1	Angelo A
	1	2	3	100	1	Angelo A
	3	2	3	100	2	Angelo A
	2	2	1	1000	1	Antonella B
	2	2	2	300	1	Antonella B
	3	2	2	300	2	Antonella B
	2	2	3	100	1	Antonella B
	3	2	4	100	2	Antonella B
	4	1	1	1000	3	Luigi C
	4	1	3	100	3	Luigi C
	4	1	4	100	3	Luigi C
	4	1	5	100	3	Luigi C

```
select distinct produktbatchkomponent.pb_id, produktbatch.status, raavare.raavare_id,
leverandoer.maengde, recept.recept_id, ansat.opr_navn
from produktbatchkomponent
    join produktbatch
        on produktbatchkomponent.pb_id = produktbatch.pb_id
    join leverandoer
        on leverandoer.raavare_id = produktbatchkomponent.raavare_id
    join raavare
        on raavare.raavare_id = produktbatchkomponent.raavare_id
    join recept
        on recept.recept_id = produktbatch.recept_id
    join ansat
        on ansat.cpr = produktbatchkomponent.cpr
where leverandoer.maengde > 0;
```

Opgave Q1 - Angiv antallet af produktbatchkomponenter med en nettovægt på mere end 10. Hint : Brug aggregatfunktionen COUNT.

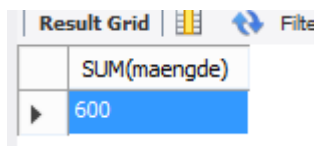


count(netto)
4

```
SELECT count(netto) FROM produktbatchkomponent
      where produktbatchkomponent.netto > 10;
```

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.

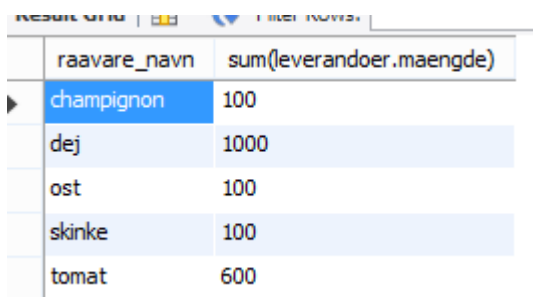
(I vores database har vi fjernet raavarebatch-tabellen, og derfor findes de dertil tilknyttede attributter i andre tabeller).



SUM(maengde)
600

```
SELECT SUM(maengde) FROM leverandoer
      WHERE raavare_id
            IN
            (SELECT raavare_id FROM raavare
              WHERE raavare_navn LIKE 'TOMAT')
```

Opgave Q3 - Find for hver ingrediens (råvare) den samlede mængde af denne ingrediens som findes på lageret. Hint : Modifiér forespørgslen ovenfor. Brug GROUP BY.

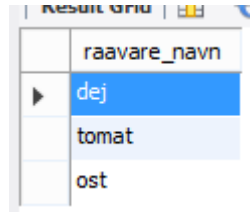


raavare_navn	sum(leverandoer.maengde)
champignon	100
dej	1000
ost	100
skinke	100
tomat	600

```
select raavare.raavare_navn, sum(leverandoer.maengde)
from raavare
```

```
join leverandoer
    on leverandoer.raavare_id = raavare.raavare_id
group by raavare.raavare_navn;
```

Opgave Q4 - Find de ingredienser (navne på råvarer) som indgår i mindst tre forskellige recepter.



A screenshot of a database result grid. The grid has a header row with the column name 'raavare_navn'. Below the header, there are four rows of data: 'dej', 'tomat', and 'ost'. The row containing 'dej' is highlighted in blue.

raavare_navn
dej
tomat
ost

```
SELECT raavare_navn from (
SELECT
    raavare_id as ID,
    COUNT(raavare_navn) AS num
FROM
    (select recept_id, raavare_navn, raavare.raavare_id from receptkomponent
    inner join raavare on raavare.raavare_id=receptkomponent.raavare_id) as amount
GROUP BY
    raavare_navn)
as amount inner join raavare on ID=raavare.raavare_id where amount.num >= 3;
```

Bilag 2

View tabel

View	MySQL Kode
OperatoerView	<pre>CREATE VIEW OperatoerView AS SELECT recept_id, receptkomponent.raavare_id, nom_netto, tolerance, raavare_navn FROM receptkomponent JOIN raavare ON receptkomponent.raavare_id = raavare.raavare_id JOIN produktbatchkomponent ON raavare.raavare_id = produktbatchkomponent.raavare_id GROUP BY recept_id;</pre>
VaerkfoererView	<pre>CREATE VIEW VAERKFOERER AS SELECT produktbatch.pb_id, produktbatch.status, recept_id, maengde FROM produktbatch JOIN produktbatchkomponent ON produktbatch.pb_id = produktbatchkomponent.pb_id JOIN leverandoer ON produktbatchkomponent.raavare_id = leverandoer.raavare_iddbafl1vaerkfoerer;</pre>
FarmaceutView	<pre>CREATE VIEW Farmaceut AS SELECT * FROM raavare JOIN recept;</pre>
adminView	<pre>CREATE VIEW admin AS SELECT * FROM ansat;</pre>

Her ses en tabel af den MySQL kode vi har lavet for at få de forskellige views.

Bilag 3

Procedures tabel

addUser	<pre>DROP PROCEDURE IF EXISTS addUser; DELIMITER // CREATE PROCEDURE addUser(in newcpr varchar(15), in newnavn text, in newini text, in newpwd text, in newrank int) BEGIN INSERT INTO ansat (`cpr`, `opr_navn`, `ini`, `password`, `titel`) values (newcpr, newnavn, newini, MD5(newpwd), newrank); END // DELIMITER ;</pre>
getLeverandoer	<pre>DROP PROCEDURE IF EXISTS getLeverandoer; DELIMITER // CREATE PROCEDURE getLeverandoer (in inID TEXT) BEGIN select leverandoer.leverandoer_navn from leverandoer, raavare where (leverandoer.raavare_id=raavare.raavare_id AND raavare.raavare_navn=inID); END// DELIMITER ;</pre>

Denne tabel viser de to procedures vi har lavet samt deres kode

Bilag 4

Trigger table

VerifyAnsatInsert	<pre> DROP TRIGGER IF EXISTS verifyAnsatInsert; DELIMITER \\\ create trigger verifyAnsatInsert BEFORE Insert ON ansat FOR EACH ROW BEGIN IF NEW.titel NOT BETWEEN 0 AND 3 THEN SIGNAL SQLSTATE '50000' SET MESSAGE_TEXT = 'Titel er ugyldig'; ELSEIF CHAR_LENGTH(NEW.cpr) < 10 THEN SIGNAL SQLSTATE '50000' SET MESSAGE_TEXT = 'CPR nummeret er for kort'; ELSEIF CHAR_LENGTH(NEW.cpr) > 10 THEN SIGNAL SQLSTATE '50000' SET MESSAGE_TEXT = 'CPR nummeret er for langt'; ELSEIF NEW.titel NOT BETWEEN 0 AND 3 THEN SIGNAL SQLSTATE '50000' SET MESSAGE_TEXT = 'Titel er ugyldig'; ELSEIF CHAR_LENGTH(NEW.ini) < 2 THEN SIGNAL SQLSTATE '50000' SET MESSAGE_TEXT = 'INI er for kort'; END IF; END \\\ DELIMITER ; </pre>
VerifyAnsatUpdate	<pre> DROP TRIGGER IF EXISTS verifyAnsatUpdate DELIMITER \\\ create trigger verifyAnsatUpdate </pre>

	<pre>BEFORE UPDATE ON ansat FOR EACH ROW BEGIN IF NEW.titel NOT BETWEEN 0 AND 3 THEN SIGNAL SQLSTATE '50000' SET MESSAGE_TEXT = 'Titel er ugyldig'; ELSEIF CHAR_LENGTH(NEW.ini) < 2 THEN SIGNAL SQLSTATE '50000' SET MESSAGE_TEXT = 'INI er for kort'; END IF; END \ DELIMITER ;</pre>
--	---