Eksamensprojekt , forår 2015

02327 Indledende databaser og database programmering F15

Projektnavn: Eksamensprojekt
Afleveringsfrist: Mandag d. 11/05-2015 23:59
Gruppenummer: CDIO 16

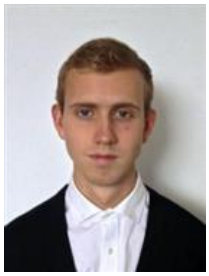
Gruppemedlemmer (studienummer, efternavn, fornavn):

s144858, Hansen, Nicklas

s123157, Adamsen, Frederik

s144846, Eriksen, Robert

s144845, Ørnby, Victor



s144844, Thomsen, Mads

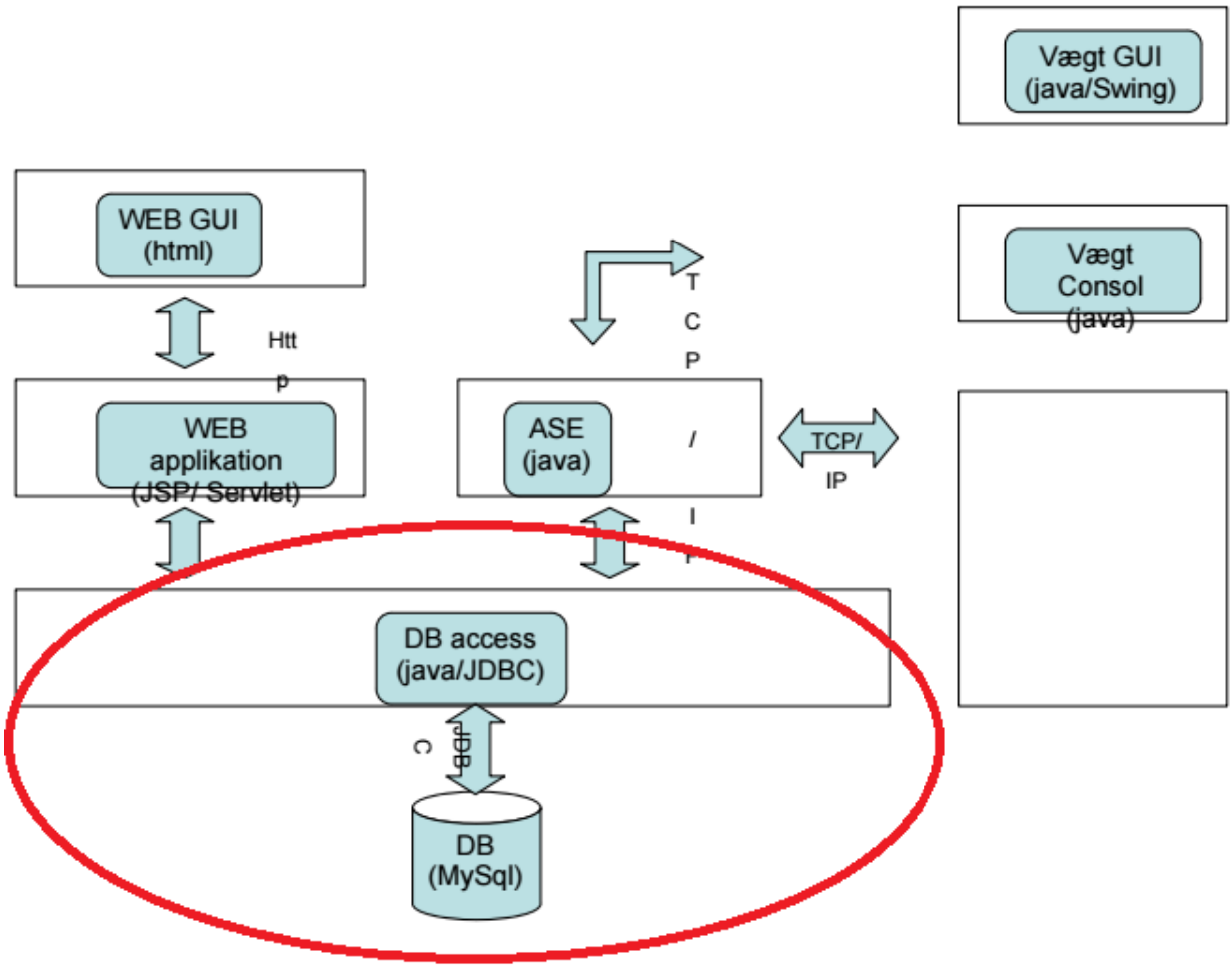
Rapporten er afleveret elektronisk via Campusnet og der skrives derfor ikke under. Rapporten indeholder 32 sider inkl. denne side og evt. bilag.

Indholdsfortegnelse

1. Indledning

Rapporten har til formål at dokumentere CDIO projektet i kurset 02327 der omhandler skabelsen af en robust database der skal tilknyttes et farmaceutisk vægtsystem. Projektet skaber en lang række udfordringer ift. udformningen af selve databasen, samt hvordan databasen og vægten skal kommunikere med hinanden. Til at lave databasen vil projektet gøre brug af MySQL. Kommunikationen mellem databasen og vægten vil blive etableret ved hjælp af JDBC (Java Database Connection) og DAO/DTO objekter til at kommunikere mellem database og applikation.

Forneden ses, markeret med rødt, den del af det overordnede CDIO system, som dette projekt beskæftiger sig med.



1.1 Timeregnskab

16_02327							
Time regnskab							
Dato	**Gruppefælle**	**Analyse og design**	**Implementation**	**Test**	**Dokumentation**	**Andet**	**I alt**
14/04/2015	Nicklas	2	1		2	1	6
21/04/2015	Nicklas	1	3	2	2		8
25/04/2015	Nicklas		2	1	2		5
28/04/2015	Nicklas	1	2	1	2		6
04/05/2015	Nicklas		2	1	1,5		4,5
05/05/2015	Nicklas	1		1	5	1	8
11/05/2015	Nicklas			1	5	1	7
14/04/2015	Victor	3	1				4
21/04/2015	Victor				4		4
28/04/2015	Victor	3					3
02/05/2015	Victor				2		2
05/05/2015	Victor	1		1	5	1	8
11/05/2015	Victor			1	4	1	6
14/04/2015	Robert	2	1		2	1	6
28/04/2015	Robert	1	2	1	2		6
04/05/2015	Robert		2	2	1,5		5,5
05/05/2015	Robert	1		1	3	1	6
11/05/2015	Robert			1	2	1	4
14/04/2015	Mads	1	1		2	1	5

25/04/2015	Mads		2	1	2		5
28/04/2015	Mads	1	2	1	2		6
11/05/2015	Mads			1	1	1	3
14/04/2015	Frederik	1	1		2	1	5
21/04/2015	Frederik	1	2		1		4
28/04/2015	Frederik	1	2	1	2		6
04/05/2015	Frederik		1	1	1,5		3,5
05/05/2015	Frederik	1		1	3	1	6
11/05/2015	Frederik			2	1	1	4
						Total timer:	146,5

Det kan ses af timeregnskabet foroven at alle gruppefæller har været en del af projektet og har arbejdet cirka lige meget på dette projekt og procentfordelingen er derfor nogenlunde jævn.

1.2 Motivation

Motivationen for projektet, er at vi som studerende opnår en tværfaglige forståelse for mulighederne ved databaser i sammenspillet med programmering. Oprettelsen af en velfungerende database er yderligere motiveret af at arbejdet fra dette projekt skal implementeres i 3. ugers kursus i Videregående Programmering i slutningen af semesteret.

1.3 Afgrænsning

Som tidligere nævnt indgår rapporten som led i et større CDIO projekt i samspil med 3 andre fag. Rapporten vil selvfølgelig bære præg af dette sammenspil (eksempelvis ved brug af JDBC), men vil som udgangspunkt afgrænse dokumentationen til at omhandle database relevant teori. Unit test, UML, java programming er alle discipliner der vil indgå i rapporten, men omfanget vil være afgrænset til bruge teorien i praksis frem for at uddybende at forklare den.

1.4 Problemformulering

Formålet er således både at kunne oprette, justere databaser ud fra teoretiske principper fra kurset forelæsninger, samt i praksis at implementere disse i selve projektet. Rapporten vil gennem E/R- diagrammer, dokumentere dennes proces, samt inddrage relevant teori til at belyse vores overvejelser i forbindelse med projektet. Projektet vil udarbejde datamodeller, designe entiteterne og deres relationer, samt angiver attributter til databasen der tilknyttes den farmaceutiske vægtsystemet. Yderligere vil vi anvende avanceret SQL til implementering af forskellige former for databasetilgang til at tilgå vægt databasen. Projektet vil dertil designe og implementere persistens mellem objektorienteret program og relationsdatabase via JDBC.

På baggrund af overstående skal følgende behandles:

- Analyser det udleverede materiale(opgaveformulering)
- kort beskrivelse af teknologier(mySQL, Java, JDBC)
- Fomulere kravspecifikation til systemet.
- Lave E/R Diagram for at skabe et overblik af databasen
- Normalisering af databasen til 3. normalform
- Implementering af JDBC til kommunikation mellem Java og databasen
- Implementere SQL til forskellige former for udtagning af data(funktioner).
- Teste databasen ved J-unit test.

1.5 Milestones

- Aflevering af kapitel 1. 17/3.
- Planlægning af projektforløb 17/3
- Lave opgave 1 og 2 18/3
- Lav E/R diagrammer af relationerne 18/3
- Design klassediagram 24/3
- Struktur (logisk skema) 1/4
- Normalisering 3. grad 7/4
- Test-cases 14/4
- J-unit test 14/4

2. Analyse

2.1 Udleveret materiale

I projektoplægget var der to opgaver der skulle fungerer som en analyse til projektet.

Opgave 1 var en øvelse i at forstå hvordan databasen er bygget op. Vi fik oplyst hvilke relationer der fandtes i databasen, f.eks. operatoer eller raavare, samt hvilke attributter disse havde. Derudover var datatyperne for disse attributter og beskrevet

Ligeledes var det oplyst forholdet mellem tabellernes relationerne ved at oplyse deres fremmednøgler. Efter at have forstået hvordan databasen var bygget op, begyndte vi at opsætte den udleverede database.

Da databasen kom op at køre, skulle der svares på en række spørgsmål hvor der skulle udtrykkes SQL forespørgsler til at hente svaret.

Opgave 2 gik ud på at få sat udviklingsmiljøet op således at man via koden kunne kommunikere med databasen. Da vi bruger Java som programmeringssprog, gøres dette med JDBC driveren. Dette er en standarddriver der er i stand til at fungere som “tolk” mellem database og applikation. Opgaven gik således ud på at downloade denne driver og få den implementeret korrekt i et Java-projekt.

2.1.2 Data Access Object

Vi fik yderligere udleveret en delvis implementation af databaseinterfacet. Dette indeholdt blandt andet en Connector-klasse som står for at oprette forbindelse til databasen og en Constant-klasse der indeholder IP, port, brugernavn, password og navn på database, som skal bruges af Connector-klassen til at forbinde til databasen.

I det udleverede materialer var der interfaces til alle DAO - Data Access Object. Disse DAO’er bruges til at tilgå data i databasen. Det vil sige, hvis man skal hente noget fra databasen ved et f.eks. et SELECT query, så skal det gøres gennem en DAO. Der blev ligeledes udleveret DTO-klasser for hver relation. DTO står for Data Transfer Object og disse bruges til at gemme data fra databasen. Altså bruges en DAO til at kommunikere med databasen og DTO til at gemme dataen i lokalt.

Opgaven gik ud på, at DAO interfacesne skulle implementeres. Det er i designfasen der blev implementeret disse ting og der kan læses mere om det i Design kapitlet.

2.1.3 Udleveret database

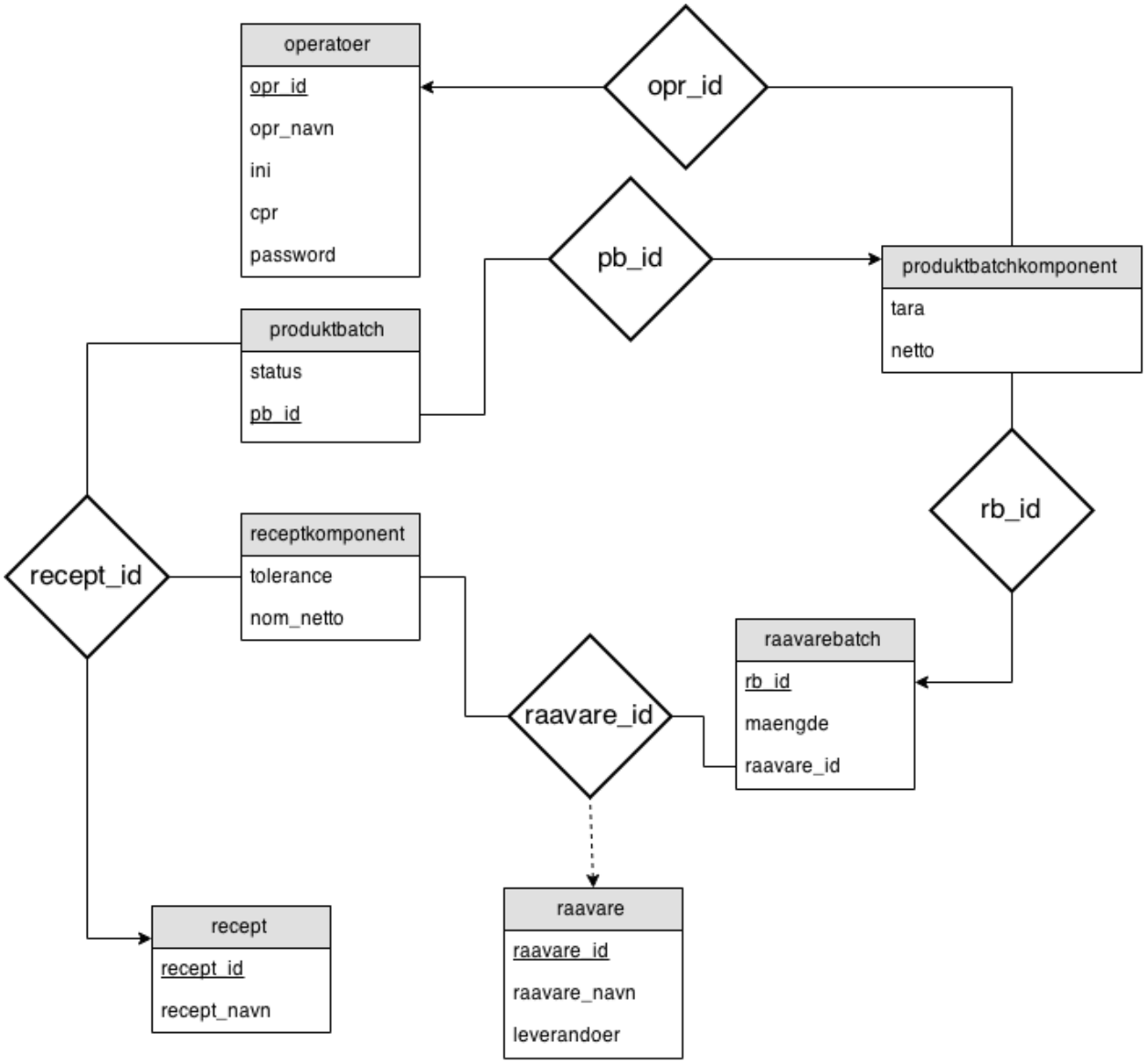
Vores database består af en række detaljer om, hvordan 3 forskellige pizzaer bliver dannet. Dette består bl.a. af hvilke råvare sådan en pizza består af, hvilke leverandøre varene er blevet leveret af osv. Hele vores database består af disse tabeller:

operatoer(opr_id : integer; opr navn : string; ini : string; cpr : string; password : string)
raavare(raavare_id : integer; raavare navn : string; leverandoer : string)
raavarebatch(rb_id : integer; raavare_id : integer; maengde : real)
recept(recept_id : integer; recept_navn : string)
receptkomponent(recept_id : integer; raavare_id : integer; nom_netto : real; tolerance : real)
produktbatch(pb_id : integer; recept_id : integer; status : integer)
produktbatchkomponent(pb_id : integer; rb_id : integer; opr_id : integer; tara : real; netto : real)

Man kan se et logisk diagram om, hvordan disse hænger sammen i kapitel 3.5

2.2 ER diagram

Forneden ses et Entity-Relationship (ER) diagram over den pizza database vi har fået udleveret.



Ovenfor ER-diagram (Entity Relationship) for databasen. Følgende afsnit vil fremhæve dele af diagrammet og beskrive relationerne imellem entities.

ER diagrammet er udarbejdet ved hjælp af det logiske diagram der kan ses senere i kapitlet. Diagrammet bruges til at forstå forholdet mellem entities og gør man relativt hurtigt kan danne sig et overblik at databasen og hvordan den fungerer.

Recept, Raavare og Operatoer er alle instanser af enkelte objekter, mens alle batchene og komponenterne er en samling af disse. Derfor vil disse enkelte instanser altid have en ‘mange-til-én’ relation mellem disse og komponenterne / batches. På samme måde er produktbatchkomponent en samling af andre komponenter / batches, og vi vil da få en ‘én-til-mange’ relation fra denne til andre komponenter / batches.

2.3 Platform/teknologier

I projektet vil gøre brug af MySQL som platform til databasen som er af typen relational database management system.
DB-access laget er skrevet i Java og vi har brugt Eclipse til at udvikle Java-kode i.
JDBC er et API (application program interface), der gør det muligt for programmøren via Java at forbinde og interagere med databaser. JDBC vil danne grundlag for at danne bro mellem Java-software og MySQL databasen.

Workbench til at lave det logiske skema. Det logiske skema har således været fundamentet for at vi kunne udarbedje ER diagrammet. Vi kunne have brugt en extension til Eclipse, men workbench blev valgt grundet det brugervenlige interface samt funktioner.

2.4 Kravspecifikation

Kravene for projektet er opstillet i projektoplægget, og disse vil ikke blive gengivet i dette afsnit.
Det overordnede krav er at få databasen op at køre så den kan bruges i 3 ugers projektet.
Systemet indeholder en database med følgende tabeller:

- operatør
- råvare
- råvarebatch
- recept
- receptkomponent
- produktbatch
- produktbatchkomponent

Projektet skal yderligere besvare de tilknyttede spørgsmål på tilfredsstillende vis.

2.5 Delkonklusion

I afsnittet om det udleverede materiale diskuteres vores overvejelser omkring databasen ift. datatyper og relationer mellem disse. Disse overvejelser ligger til grund for besvarelsen af opgave 1 der er vedlagt i bilag til rapporten. Afsnittet fungere yderligere som en introduktion

til JDBC (Java Database Connecter) og understreger vigtigheden i at få etableret denne. Dertil er enkelte tabeller og datatyper beskrevet i henhold til de udleverede materiale. Udover at analysere det udleverede materiale har vi også lavet et ER diagram. Af ER diagrammet kan multipliciteten aflæses, dvs. hvorvidt der er en til mange relationen af f.eks produktbatch og produktbatchkomponent. Diagrammet giver både et overblik over relationerne samt hvad tabellerne reelt indeholder.

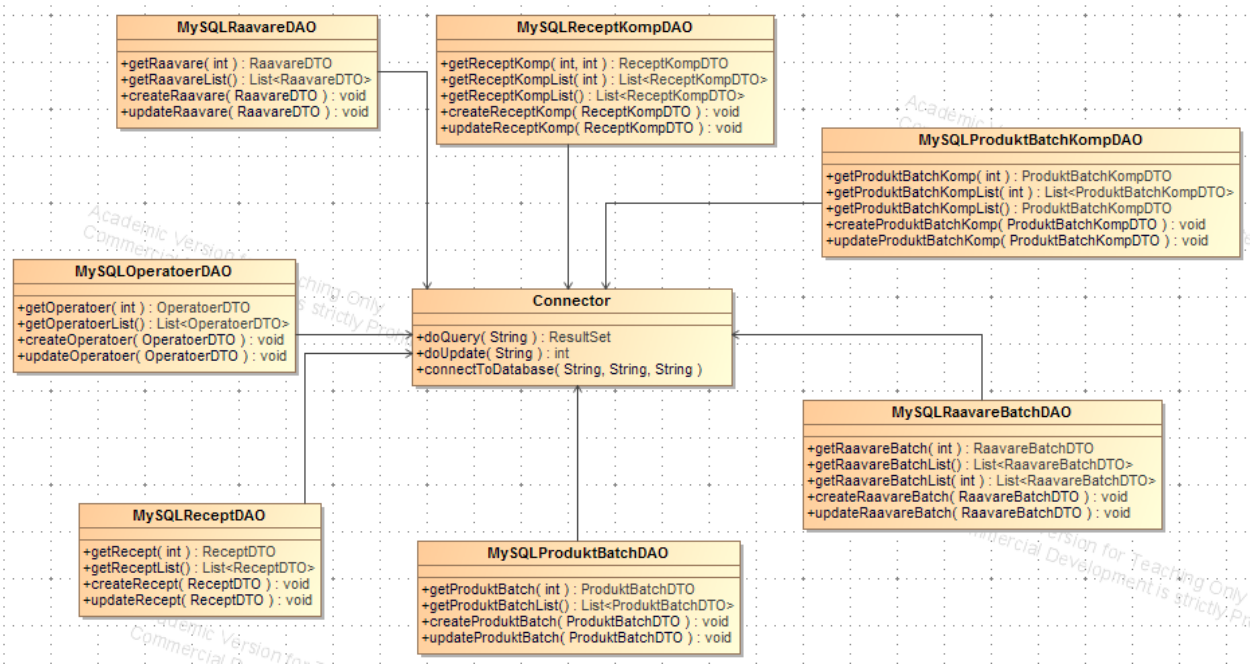
I afsnitet om platformsteknologier opsummere vi kort hvilke teknologier der er involveret i projektet. Overordnet set er der brugt følgende teknologier i projektet MySQL, JDBC, Workbench.Kapitel 2 afsluttes med opsummering af kravspecifikation til projektet.

3. Design / implementering

I dette kapitel kigger på hvordan selve databasen er opbygget, og hvordan vi har implementeret den i Eclipse. Den første del indebærer diagram af strukturen, og vi kigger på hvordan den opretholder reglerne for 3. normalform. Den anden del kigger på strukturen af vores program, og en test af en tabel for at vise at vores kode virker korrekt.

3.1 Design klassediagram

Forneden ses et design klassediagram over data-access objekterne i programmet. Det kan ses at alle DAO gør brug af Connector til at få forbindelse ned til databasen. Der findes en DAO for hver relation i databasen, som skal bruges når man vil hente data fra en bestemt tabel i databasen.



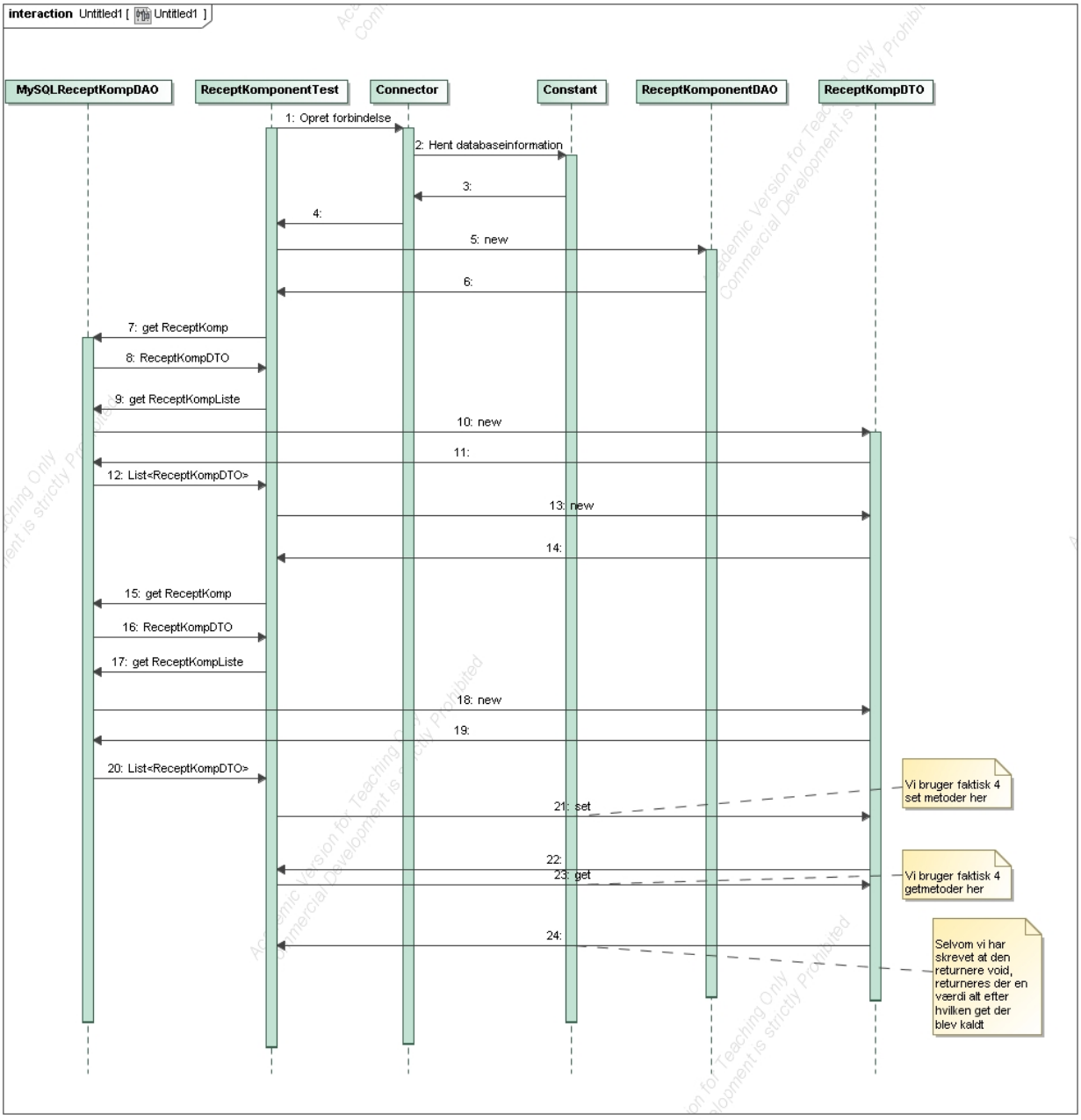
Vil man hente data fra operatør tabellen gør man det gennem MySQLOperatoerDAO. Der oprettes en instans af dette objekt og igennem dette kan tupler fra operatør tabellen i databasen udhentes. Dette gør sig gældende for hver tabel i databasen.

Hver DAO har metoderne get(), getList(), create() og update() som de i følge interfacet skulle have.

3.2 Sekvensdiagram

Et sekvensdiagram kan være med til at skabe et hurtigt overblik over hvordan et givent stykke kode forløber. Neden under ses et sekvensdiagram af vores test af Receptkomponent.

3.2.1 Forløb for ReceptKomponent test

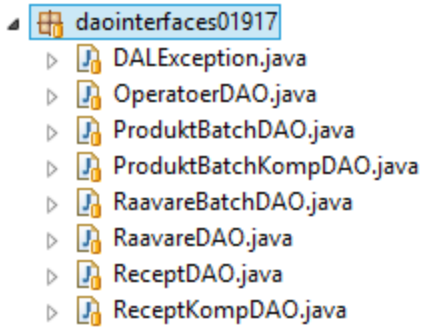


Det ses, at der bruges DAO objektet til at udhente data fra databasen. Dataen gemmes i et DTO objekt.

3.4 Implementation af funktioner

3.4.1 Access datalag (interfaces)

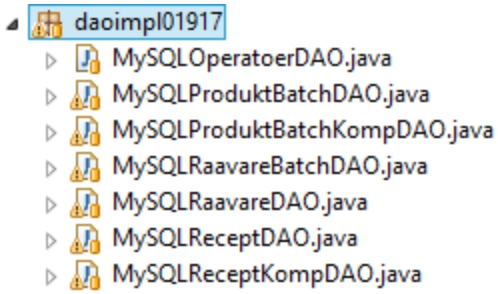
Vi fik, som tidligere nævnt, udleveret en del af implementationen af data access laget. Vi fik udleveret interfaces til alle DAO og vi skulle herefter selv lave implementationer til disse interfaces. Der skulle også laves DTO objekter for hver tabel. Forneden ses en oversigt over de udleverede interfaces:



I hver af disse interfaces var defineret metoder som skulle implementeres i vores implementation. Forneden ses et eksempel på et af interfaces, OperatoerDAO:

```
public interface OperatoerDAO {
    OperatoerDTO getOperatoer(int oprId) throws DALException;
    List<OperatoerDTO> getOperatoerList() throws DALException;
    void createOperatoer(OperatoerDTO opr) throws DALException;
    void updateOperatoer(OperatoerDTO opr) throws DALException;
}
```

Vi udviklede således Data Access Objekter implementationer til hvert interface, disse kan ses forneden:



3.4.2 SQL

Undervejs i implementationen af vores forbindelse mellem Java og databasen, har det selvfølgelig været nødvendigt med SQL forespørgsler. Disse bruges til at udhente eller indsætte data i databasen. Vi gør brug af Connector-klassen og JDBC-driver når vi skal kommunikere til vores MySQL database.

SQL står for Structured Query Language og er et stort sprog med mange "undersprog". Der findes flere typer af SQL queries, navnligt DDL (Data Definition Language), DML (Data Manipulation Language) og DCL (Data Control Language)[1].

DDL bruges til at modificere strukturen i databasen, såsom at tilføje, ændre eller slette definitioner af tabeller. Forneden ses de kommandoer vi har stiftet bekendtskab med i forbindelse med dette kursus:

CREATE
ALTER
DROP
RENAME

DML bruges til at vælge, indsætte, slette eller opdatere data i databasen. Det er hovedsageligt denne del af SQL vi har brugt i dette projekt. Vores DAO-implementationer gør brug af følgende DML kommandoer:

SELECT
INSERT
UPDATE
DELETE
CALL

DCL har vi ikke gjort brug af i dette kursus så derfor vil vi ikke komme nærmere ind på dette.

[1] http://www.w3schools.in/mysql/ddl-dml-dcl/

3.4.2.1 SQL sætninger i vores implementation

Vi har som sagt brugt SQL forespørgsler i vores Data Access Objects. I det følgende vil vises nogle af de forespørgsler vi har brugt i klassen OperatoerDAO og vi vil knytte et par ord til hver metode.

Forneden ses et udsnit af metoden getOperatoer i klassen OperatoerDAO. Metoden tager et operatør id som argument, som senere skal bruges til at selecte den rigtige operatør nede i databasen:

```
public OperatoerDTO getOperatoer(int oprId) throws DALException {
    ResultSet rs = Connector.doQuery("SELECT * FROM operatoer WHERE opr_id = " + oprId);
```

Når metoden kaldes, køres der via Connector-klassen en forespørgsel nede i databasen. Forespørgslen der køres er meget simpel og vælger blot den række fra operatoer tabellen hvor opr_id er lig med det angivne operatør id. Resultatet gemmes i et ResultSet. Hvis der findes en operatør, vil ResultSet have nøjagtig én row og hvis operatøren ikke findes vil ResultSet være tom og der kastes en DALException.

Den næste metode er createOperatoer. Denne tager et OperatoerDTO objekt som argument. Alle OperatoerDTO objektets felter indsættes i SQL forespørgslen som det kan ses forneden:

```
public void createOperatoer(OperatoerDTO opr) throws DALException {
        Connector.doUpdate(
            "INSERT INTO operatoer(opr_id, opr_navn, i.ni, cpr, password) VALUES " +
            "(" + opr.getOprId() + ", '" + opr.getOprNavn() + "', '" + opr.getIni() + "', '" +
            opr.getCpr() + "', '" + opr.getPassword() + "')"
        );
```

Denne forespørgsel forsøger at indsætte en ny række i tabellen operatoer med de givne værdier fra OperatoerDTO objektet. Hvis der i forvejen findes en operatør med det givne ID eller hvis der på anden måde opstår en fejl, kastes en DALException. Det bemærkes at der her bruges metoden doUpdate() hvor der i første eksempel blev brugt doQuery().

Connector.doQuery() bruges når man udhente data fra databasen, altså kun bruge SELECT statements. doUpdate() bruges til alt andet, altså når man vil ændre på data i databasen.

3.4.3 Stored Procedures

En stored procedure er en måde at gemme SQL statements i databasen. Ved at have stored procedures kan man blot kalde disse procedurer med et simpelt statement og få det samme resultat, istedet for at have et stort statement liggende i Java koden. Det kan være en god idé at have stored procedures hvis der er metoder man gør brug af tit og også fordi at man så har koden liggende centralt og ikke distribueret ud på forskellige applikationer. Skal man f.eks. fejlrette, skal man kun ændre ét sted i stored procedure istedet for i måske fem applikationer.

Vi har i projektet ikke gjort så meget brug af stored procedures i og med at vi har lagt det meste logik i vores Java applikation. Vi har dog for projektets skyld lavet en lille stored procedure som kan ses forneden, nemlig proceduren getOperatoer():

```
DELIMITER $$
CREATE DEFINER=`root`@`localhost` PROCEDURE `getOperatoer`(IN operatoerID INT)
BEGIN
SELECT * FROM operatoer WHERE opr_id = operatoerID;
END$$
DELIMITER ;
```

Når en stored procedure er oprettet, kan den kaldes på følgende simple måde:

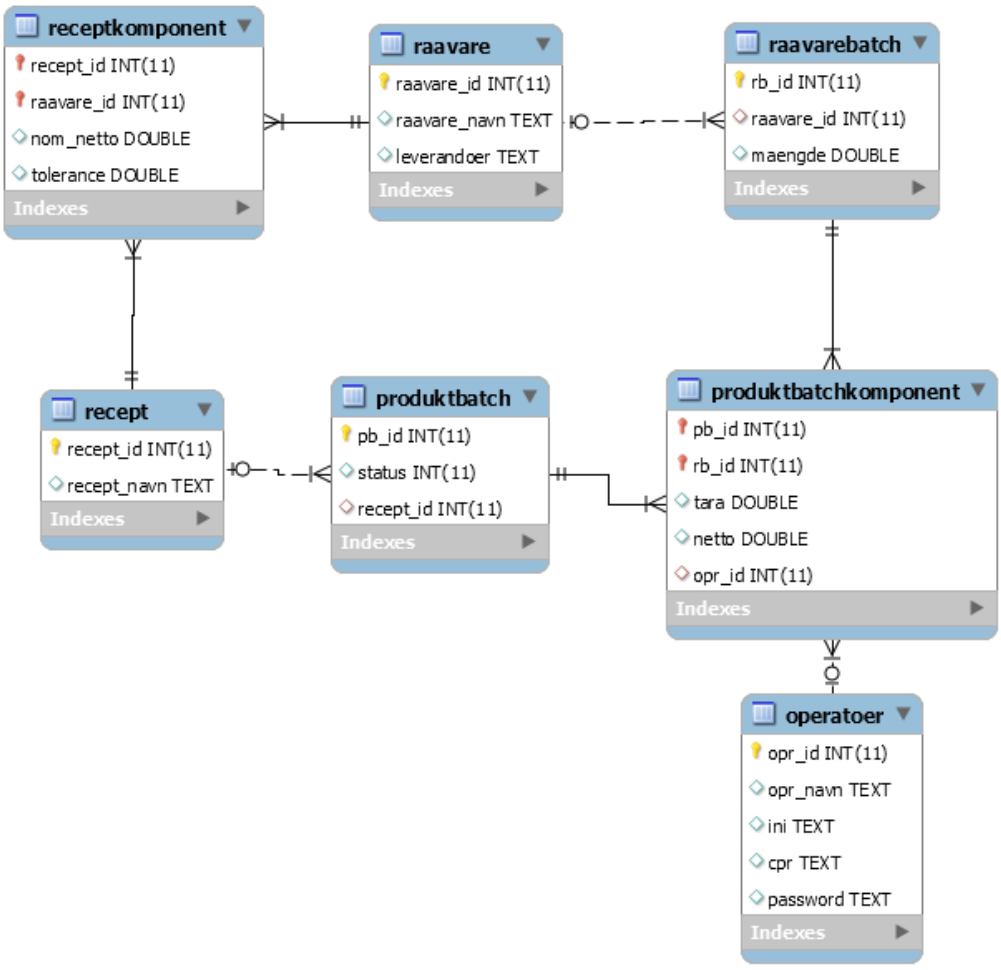
```
ResultSet rs = Connector.doQuery("CALL getOperatoer("+oprId+")");
```

Ovenstående query har samme funktion som hvis vi sendte følgende statement til SQL databasen:

```
ResultSet rs = Connector.doQuery("SELECT * FROM operatoer WHERE opr_id = " + oprId);
```

3.5 Struktur (logisk skema)

Det logiske skema for vores database er automatisk genereret ved hjælp af programmet MySQL Workbench. Det logiske skema giver et indblik i datatypen på de respektive attributer. Yderligere viser det, ligesom ER diagrammet, relationerne i mellem tabellerne. Forneden ses det logiske skema for vores database:



3.6 Normalisering til 3. grad

Følgende afsnit vil redegøre for hvordan databasen opfylder 3. normalform. Dette sker ved slavisk at opstille kravene til de forskellige normalformer og gennemgå hvorledes databasen opfylder disse krav. Optimering af databasen i henhold til normalform, sikrer en brugebar, stabil og konsistent datastruktur og minimerer redundant data.

1. Normalform

En tabel af 1. normal form betyder i praksis at:

- Hver celle i tabellen skal have en enkel værdi. Elementerne er atomiske.
- Alle entries i en kolonne skal have værdier fra samme domæne.
- Alle kolonner skal have et unikt navn.
- Hver række skal have en unik primær nøgle og alle rækker skal have primær nøgler.
- Rækkefølgen af kolonnerne skal ikke have nogen betydning.

I og med at Databasen der er blevet udleveret allerede er i 3. normalform, føler vi at det er nødvendigt at give eksempler for hver normalform:

raavare_id	raavare_navn	leverandoer
1	dej	Wawelka
2,3,4	tomat	Knoor, Veaubais, Franz
5	ost,skinke	Ost og Skinke A/S
null	champignon	Igloo Frostvarer

Denne følger ikke 1. normalform af 2 årsager:

- Hver celle i tabellen skal have en enkel værdi. Elementerne er atomiske.
- Som vi kan se er der flere tomater med forskellige leverandøre, og derfor indeholder raavare_id og leverandøre 3 elementer. Dette skal fikses så de kun fylder én.

- Hver række skal have en unik primær nøgle og alle rækker skal have primær nøgler.
- Champignon har 'null' som sin primære nøgle. Det er nu umuligt at finde denne eksakte champignon.

Disse problemer kan man løse ved at gøre dette table større:

raavare_id	raavare_navn	leverandoer
1	dej	Wawelka
2	tomat	Knoor,
3	tomat	Veaubais
4	tomat	Franz
5	ost	Ost og Skinke A/S
6	skinke	Ost og Skinke A/S
null	null	null

Vi har nu forlænget databasen, og helt fjernet raavaren champignon, da den indeholdte en null værdi.

2. Normalform

Kravene for 2. normalform for en tabel er givet ved:

- Tabellen skal opfylde kravene for 1. normalform
- Hver non-key attribut afhænger af hele den primære nøgle.

Vi vil gerne vise at vi har forstået dette. Hvis man forestiller sig en variation af 'raavarebatch':

rb_id	raavare_id	maengde	raavare_navn
1	1	1000	dej
2	2	300	tomat
3	3	300	tomat
4	5	100	ost
5	5	100	ost
6	6	100	skinke
7	7	100	champignon
8	7	2000	champignon
NULL	NULL	NULL	NULL

I raavarebatch er primærnøglen en superkey, bestående af rb_id og raavare_id. Vores nye indsatte attribut raavare_navn skal være afhængig af den fulde primære nøgle, og det er her det går galt. Raavare_navn er udelukkende afhængig af raavare_id, så vi splitter denne i 2 for at få 2 nye tables:

rb_id	raavare_id	maengde
1	1	1000
2	2	300
3	3	300
4	5	100
5	5	100
6	6	100
7	7	100
8	7	2000
NULL	NULL	NULL

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

Nu er ovenstående tabel på 2. normalform.

3. Normalform

Kravene for 3. normalform af databasen er givet ved:

- at opfylde 1. og 2. normalform.
- At der ikke er transitive/funktionelle afhængigheder.

På anden normalform blev eksemplet udført da tablet havde en superkey. Forestiller man sig et table med én attribut primary key:

raavare_id	raavare_navn	leverandoer	Leverandør CEO
1	dej	Wawelka	Thomas Wawelka
2	tomat	Knoor	Eneko Knoor
3	tomat	Veaubais	Tim Veaubais
4	tomat	Franz	Vanz Franz
5	ost	Ost og Skinke A/S	Hermann Blau
6	skinke	Ost og Skinke A/S	Hermann Blau
7	champignon	Igloo Frostvarer	Jonas Green
NULL	NULL	NULL	NULL

Denne er i 2. normalform, til trods for at Leverandør CEO faktisk afhænger enten af raavare_id eller Leverandør. Det er dog dette der separere 2. fra 3. normalform, alle attributter skal udelukkende afhænge af den primære nøgle raavare_id. I stedet burde man dele denne op i 2:

raavare_id	raavare_navn	leverandoer
1	dej	Wawelka
2	tomat	Knoor
3	tomat	Veaubais
4	tomat	Franz
5	ost	Ost og Skinke A/S
6	skinke	Ost og Skinke A/S
7	champignon	Igloo Frostvarer
NULL	NULL	NULL

leverandoer	Leverandør CEO
Wawelka	Thomas Wawelka
Knoor	Eneko Knoor
Veaubais	Tim Vaeaubais
Franz	Vanz Franz
Ost og Skinke A/S	Hermann Blau
Ost og Skinke A/S	Hermann Blau
Igloo Frostvarer	Jonas Green
NULL	NULL

Nu afhænger alle attributterne udelukkende af den primære nøgle.

Man kan derfor konkludere at databasen opfylder kravene for normalform 1-3. Dette skaber et godt udgangspunkt for den fremtidige brug af databasen i 3 ugers projektet med videregående programmering.

3.7 Test

Følgende afsnit vil gennemgå kort gennemgå hvad vi har testet. Koden til testen kan ses i projektet vi har uploadet, nedenfor ses consolen for vores test. Vi har brugt J-unit tests, som har den fordel, at hvis en fejl bliver fundet stoppes hele testen, og der returneres en fejl.

Vores test-case følger dette forløb:

1. Vi undersøger 'get'-metoden ved at forsøge at finde et element vi ved ikke findes, findes. Vi får en fejl her hvis vi ikke får beskeden at elementet ikke findes.
2. Vi undersøger metoden 'getlist' virker, ved at sammenligne listen vi har, med en identisk liste der også har dette element vi ikke har indsat endnu. Hvis listerne er identiske, returnere den en fejl.
3. Vi indsætter nu dette element
4. Vi undersøger om 'get' kan finde denne. Kan den stadig ikke, returnerer den en fejl
5. Vi undersøger om 'getlist' nu er identisk med den hypotetiske liste. Er den ikke dét, returnere den en fejl.
6. Vi ændrer alle værdierne med setmetoderne på vores indsatte element.
7. Vi undersøger om getmetoderne returnere de samme værdier som det vi ændrede dem til. Gør de ikke dét, returnere den en fejl.

Vi har implementeret denne test-case på ReceptKomponent delen af vores database og Java-applikation og vi har kørt testen fejlfrit gentagne gange.

3.8 Delkonklusion

Vi har fået lavet et design klassediagram over vores applikation som giver det fornødne overblik over hvordan applikationen hænger sammen på kryds og tværs.

I kapitel 3 har vi redegjort for hvorfor databasen opfylder 3. normalform. Denne konklusion er kommet i stand ved minutiøst at gennemgå kravene for normalform 1-3 og holde dem op imod den udleverede database. Bl.a. opfylder databasen at hver række har en unik primær nøgle samt at hver celle i tabellen har en enkel værdi. På bagrund af analysen kan vi konkludere at databasen er af 3 normalform.

Vi har testet alle funktionerne i Receptkomponent. Vi følte at det kun var nødvendigt at teste én tabel grundigt, da metoderne er meget ens.
Det skal nævnes at vi i vores test indsætter vi et element, og testen går ud fra at dette element ikke eksisterer inden testen begynder. Dog efter testen bliver dette element indsat indtil at det bliver manuelt slettet, så kører man testen 2 gange i træk uden at slette dette element, får man en fejl.

Det logiske skema er udarbejdet ved hjælp af workbench og viser relationerne og datatyperne af nogle af attributterne for databasen.

4. Konklusion

I løbet af projektet har vi oprettet og implementeret en grænseflade til pizza databasen, og vi mener selv at vi har gennemført denne opgave fyldestgørende. Vi har arbejdet med MySQL servere og SQL statements, og vi har implementeret en grænseflade i Java til MySQL serveren.

Vi har fået analyseret det udleverede materiale rigtig fint og det gav os en god base at bygge videre på.

Vi har fået bedre kendskab til de brugte teknologier. Vi har fået en større erfaring med Java men herunder mest hvordan JDBC driveren fungerer i sammenspillet mellem Java og database.

Vores viden om den relationelle database er også blevet betydeligt forøget, i og med vi har udarbejdet relevant dokumentation der understøtter vores arbejde og arbejdet med at få en MySQL database til at snakke sammen med et Java-program.

Vi har fået lavet et ER diagram over databasen og dette har givet os et vigtigt værktøj hvis vi i fremtiden skal arbejde med databaser, da vi har set hvor godt et overblik et ER diagram kan give en.

Vi har undersøgt den udleverede database og fundet frem til at den allerede opfyldte alle kravene for første, anden og tredje normalform.

Endeligt har vi lært mere om hvad SQL er for et sprog samt dets inddelinger i DML, DDL og DCL. Stored procedures er også et begreb vi nu kan kalde os bekendte med.

Databasen og vores applikation er blevet gennemtestet ved at vi har lavet J-unit tests som viser at databasen henter, indsætter og opdatere data korrekt og efter al forventning.

Alt i alt har det været et godt projektforløb med god hjælp og vejledning undervejs fra Bjarne Poulsen samt hjælpelærere. Det har hele tiden i gruppen været klart hvad vi skulle nå til det næste status-møde og gruppearbejdet har fungeret rigtig godt.

4.1 Perspektivering

Projektet kan siges at opfylde kravene opstillet i kravspecifikation, men efterlader stadigvæk en række uforløste initiativer der ville kunne gøre databasen og/eller applikationen endnu bedre. Følgende afsnit vil kort redegøre for disse mulige initiativer.

Blandt de mere oplagte uforløste initiativer er yderligere optimering af databasen. Databasen i projektet er af 3. normal form. Havde tiden været der ville vi prøve at implementere den næste normal form, Boyce-Codd normal form, til databasen. I praksis vil det betyde, at vi skulle identificere hvor der i databasen var flere funktionelle afhængigheder. Derefter ville man splitte tabellen op og lave to nye tabeller. Resultatet vil være en mindre grad af redundant data.

I projektet har vi ikke på tilfredsstillende vis arbejdet med sikkerhedshuller ift. at kunne tilgå databasen. I skrivende stund er der en kun en user der kan tilgå alt i databasen. Selvom der er et password knyttet til den ene bruger, så udgør det stadigvæk en betydlig sikkerhedsrisiko. Man ville kunne imødekomme denne risiko ved at implementere rights, der gav forskellige rettigheder til at kunne manipulere i databasen. Dertil kunne man stille højere krav til brugernavn samt kodeords sikkerhed. Hvis projektet skal fungere som et reelt produkt for en kunde ville disse sikkerhedsforanstaltninger være yderst vigtig at implementere, da flere mennesker ville skulle arbejde med databasen.

Overordnet kan man argumentere for at database projektet blot er et led i et større CDIO projekt der kulminerer med 3. ugers perioden. I denne periode vil databasen skulle stå mål for interaktion med den store, samlede applikation vi skal udvikle. Fremtidsudsigten og dermed perspektivet for databasen er at den kan fungere i dette samspil, hvilket understreger vigtigheden i at lave et solidt udgangspunkt.

5. Litteraturliste

http://www.w3schools.in/mysql/ddl-dml-dcl/
Dias-show slides fra forelæsninger af Bjarne Poulsen
Database System Concepts 6th. Edition

6. Appendiks

6.1 SQL script til vores database

SQL script til vores database er vedlagt i projektet i filen "**pizza_company_CDIO16_11-05-15.sql**".

6.2 Svar på opgaver i udleveret materialer

Opgave 1

 # 1.9.1 Find the name of all instructors who work in the Physics department:

select name from instructor where dept_name="Physics"

#1.9.2. Find the name of all instructors whose salary is greater than 60000:

select name from instructor where salary > 60000

#1.9.3. Find the name of all instructors who works in the Physics department and have a salary

greater than 60000:

select name from instructor where salary > 60000 and dept_name = "Physics"

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

select distinct(raavare_navn) from raavare where raavare_id in

(select raavare_id from raavarebatch group by raavare_id having count(raavare_id) > 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:

select recept_id,raavare_id,raavare_navn from recept natural join receptkomponent natural join

raavare;

#3.A Find navnene på de recepter som indeholder mindst én af følgende to ingredienser

(råvarer): skinke eller champignon:

```
select distinct(recept_navn) from receptkomponent natural join recept where raavare_id in
(select

raavare_id from raavare where raavare_navn = 'skinke' or raavare_navn = 'champignon');
```

#3.B Find navnene på de recepter som indeholder både ingrediensen skinke og ingrediensen champignon:

```
select r.recept_navn from recept r natural join receptkomponent natural join raavare where

raavare_navn = 'skinke' and r.recept_id in(

select r2.recept_id from recept r2 natural join receptkomponent natural join raavare where

raavare_navn = 'champignon');
```

#4 Find navnene på de recepter som ikke indeholder ingrediensen champignon:

```
select raavare_id from raavare where raavare_navn = 'champignon';

select recept_id from recept natural join receptkomponent where raavare_id in (select
raavare_id

from raavare where raavare_navn = 'champignon');

select distinct(recept_id) from recept natural join receptkomponent where recept_id not in
(select

recept_id from recept natural join receptkomponent where raavare_id in (select raavare_id
from

raavare where raavare_navn = 'champignon'));
```

#5. Find navnene på den eller de recepter som indeholder den største nominelle vægt af

ingrediensen tomat:

```
select MAX(nom_netto) from raavare natural join receptkomponent where raavare_navn =
'tomat';

select raavare_id from raavare where raavare_navn = 'tomat';

select recept_navn from receptkomponent natural join recept where raavare_id in (select

raavare_id from raavare where raavare_navn = 'tomat') and nom_netto in (select

MAX(nom_netto) from raavare natural join receptkomponent where raavare_navn = 'tomat');
```

#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;
```

#7 Find identifikationsnumrene af den eller de produktbatches som indeholder en størst nettovægt af ingrediensen tomat:

```
select MAX(netto) from raavarebatch natural join raavare natural join produktbatchkomponent

where raavare_navn = 'tomat';

select raavare_id from raavare where raavare_navn = 'tomat';

select rb_id from raavarebatch where rb_id in (select raavare_id from raavare where

raavare_navn = 'tomat');

select * from produktbatchkomponent

where rb_id in

(select rb_id

from raavarebatch

where rb_id in
```

```
(select raavare_id

from raavare

where raavare_navn = 'tomat'))

and netto in

(select MAX(netto)

from raavarebatch natural join raavare natural join produktbatchkomponent

where raavare_navn = 'tomat');
```

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

```
select recept_id from recept where recept_navn = 'margherita';

select pb_id from produktbatch where recept_id in (select recept_id from recept where

recept_navn = 'margherita');

select distinct(opr_id) from produktbatchkomponent where pb_id in (select pb_id from

produktbatch where recept_id in (select recept_id from recept where recept_navn =
'margherita'));

select opr_navn

from operatoer

where opr_id in

(select distinct(opr_id)

from produktbatchkomponent

where pb_id in

(select pb_id

from produktbatch
```

where recept_id in

(select recept_id

from recept

where recept_navn = 'margherita')));

#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 recepts 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 operatoer natural join raavarebatch

natural join raavare;

#Angiv antallet af produktbatchkomponenter med en nettovægt p˚a mere end 10. Hint : Brug

aggregatfunktionen COUNT.

select count(pb_id) from produktbatchkomponent where netto > 10;

#Find den samlede mængde af tomat som findes på lageret, dvs. den samlede mængde af tomat som optræder i raavarebatchtabellen. Hint : Brug aggregatfunktionen SUM:

select sum(maengde) from raavarebatch natural join raavare where raavare_navn = 'tomat';

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

select sum(maengde),raavare_navn from raavarebatch natural join raavare group by

raavare_navn;

#Find de ingredienser (navne på r˚avarer) som indg˚ar i mindst tre forskellige recepter:

select * from raavare natural join receptkomponent group by raavare_navn having count(*) > 2;

```
select operatoerraavare_navn, count(recept_id) from raavare natural join receptkomponent
group by raavare_navn having count(tolerance) > 2;
```