SYSTEM DESIGN

Dette projekt bruger UP model, som er et krav for dette semesters projekt.

Unified Process (UP) er en iterativ og gradvis softwareudvikling proces ramme, der bruges til at modellere *hvad*, *hvem* og *hvornår* af software udviklingsproces. Det bruges til at definere roller af enkeltpersoner eller teams i et projekt og de opgaver der skal udføres af disse enkeltpersoner eller teams. Det hjælper til at sikre kvalitet, minimere risici og reducere omkostningerne. UP er arkitektur centreret og use-case driven. UP definerer fire faser*: Inception, Elaboration, Construction* og *Transition*.

Milestone / Fase	Inception	Elaboratoion	Construtuion	Transition

Mål	- System Vision (As-is, To-be, feature list: funktionelle krav, ikke funktionelle krav)
	- Use-Cases (Brief beskrivelser, Prioritering af use-cases, Fully dressed beskrivelser af de mest komplekse use-cases)
	- Mock Ups
	- Domæne Model
	- Kvalitets planlægning
	- Evaluerings plan
	  ○ Evaluerings kriterier
	  ○ Godkendelse
	- Iterations planer
	Iterations plan:
	- Se Trello
		- Arkitektur (3 eller 4 lags arkitektur)
		- Database design (Normalisering, Transformation, Tabel)
		- Den mest kritiske use-case
		  ○ SSD: accept test
		  ○ Interaktions diagram
		  ○ Design klasse diagram
		  ○ Implementering + test
		- Kvalitets planlægning
		- Iterations planer
			- SSD (define accept test)
			- Interaction diagram
			- Design class diagram
			- Implementing +test (unit tests)
			- Evaluation plan
			  ○ Evaluering kriterier
			  ○ Godkendelse
			- Iteration planer	

Antal Iterationer	1	2		
Færdigdato	17-12-2012			

INCEPTION FASE

SYSTEM VISION

Kjeld er tilknyttet Kopenhagen Fur, som er en international pels leverandør. Kopenhagen Fur er anerkendt for levering af høj kvalitet pels. De danske minkavlere er kendt for at producere en af de bedste pelskvalitet i verden, som kunderne er villige til at betale mere for. Derfor vil Kjelds mink-farm gerne forsætte med at forbedre deres produktion kvalitet og skræddersy deres produktion efter efterspørgslen på markedet. Kjeld. vil gerne være mere konkurrence dygtig fordi hans omsætning ikke kun er afhængig af pelsens kvalitet, med også efterspørgslen på markedet.

Formålet med denne systemvision er at beskrive de overordnede krav til Kjeld. Mink farm. Systemet skal primært hjælpe ham til at holde styre på information omkring de mink i hans farm. Disse informationer kan hjælpe Kjeld. til at forbedre kvalitet på mink produktion, samt at hjælpe ham med at imødekomme efterspørgslen på markedet og derved øge hans rentabilitet.

SITUATIONSANALYSE

Nu situationen (AS IS): Aktivitetstabel før IT

Hændelse	Aktivitet	Step i aktivitet	Aktør
Ny mink modtaget	Registrere mink	Find bur til minken Skrive arve information, farve, føde år … på papir og hæng det på buret	Kjeld/medarbejder
Tage blodprøve	Flytte syge mink	Tage blodprøve fra mink, Sende prøve til laboratorium, Modtage resultatet af blodprøve (brev), flytte syge mink. Når de bliver raske igen, så er de klar til at blive flyttet tilbage.	Kjeld/medarbejder
Tjek for bidesår	Flytte mink med bidesår	Kontrollere tilstanden af hver mink for bidesår, Adskille mink med bidesår Når de bliver raske igen, så er de klar til at blive flyttet tilbage.	Kjeld/medarbejder
Kontrollere min-kens kvalitet	Kvalitet kontrol	Mål pelsen Kontrollere farven Kontrollere arven Beslutte hvilken mink skal pelses eller gemmes til avl	Kjeld
Mink klar til salg	Levere mink	Pels mink Registrere mængden af pels Levere pels til Kopenhagen Fur	Kjeld Medarbejder

Tekstuel beskrivelse af Tabel 1 i trin:

- *Kjeld modtager nye mink(enten ved fødsel eller er købt)*
  - *1: Kjeld finder et bur til minken og skriver minkens oplysninger såsom fødselsår, arv, farve,… på et kort, der hænger på buret*

- *2: Dyrlægen kommer to gange om året for at tage blodprøver og tjekker for sygdom, han tilbagemelder resultater fra blodprøven(i et brev)*
- *3: Medarbejderne går rundt og tjekker hvert bur for bidesår. Hvis de finder nogle mink med bidesår, adskiller de mink med sår og flytter minken til en ny lokation (bur)*
- *4: En gang om året, laver de kvalitets kontrol for at hold styre på information såsom hvilken arter, længden af pelsen, parringsinformation omkring mink, samt hvor de befinder sig. Information bliver skrevet ned*
- *5: Efter omkring halvandet år og minken har været gennem kvalitetskontrol, er minken klar til pelsning. De pelser minken, registrere mængden af pelsen, samt pelsens type og levere pelsen til Kopenhagen Fur.*

Problemer (AS IS) - Nu situationen

- **Ineffektivitet**

  Det tager tid at gå rundt for manuelt at udfylde og hænge minkens information på buret

  Det tager tid at gå rundt for at læse eller opdatere oplysninger på burene

  Der er svært at huske de nye bure lokationer i hovedet, når mink bliver flyttet fra et bur til en anden
- **Fejl**

  Der kan opstå fejl på grund af manuel nedskrivning og opdatering af data
- **Information tab**

  Muligheden for at glemme ikke nedskrevne informationer, da de kun er i deres hoveder.

  Det er svært at få et hurtig overblik af hvilken mink racer, der sidder hvor og hvilken kvalitet de hver især har.

Forbedringsforslag (To Be)

- Integreret IT system som kan gøre det muligt at gemme, bruge og opdatere data fra en database
- Integreret IT system som kan gøre det mulige for at se up-to-date antal of mink, deres arv, og hvor de befinder sig, dvs. hurtig adgang til Farm data
- IT system til registrering af mink, bur, lokation og til håndtering af minkens kvalitet samt arv information
- Information skal ikke længere skrives manuelt, men i et IT system for at spare tid, undgå redundans og undgå information tab.

Forbedringsforslag (TO BE): Aktivitetstabel efter IT

Hændelse	Aktivitet	Step i aktivitet	Aktør
Ny mink modtaget	Registrere mink	Find bur til minken Opret bur Registrere arve information, farve, føde år … Gem information	Kjeld/medhjælper
Tage blodprøve	Registrere blod test information	Tage blodprøve fra hver mink Sende prøve til laboratorium Modtage blodprøve resultat Registrere blod prøve resultat i systemet Flytte mink og registrere sygdom information og deres lokation i systemet	Kjeld/medhjælper
Tjek for bidesår	Registrere bidesår information	Kontrollere tilstanden af hver mink Adskille mink med bidesår Registrere minkens sygdom information og deres lokation i	Kjeld/medhjælper

		systemet	
Syg mink er blevet rask igen	Flytte rask mink tilbage til deres gamle bur	Check på status på mink med bidesår i systemet. De raske mink blandt dem bliver flyttet tilbage til deres oprindelige bur, ved at bruge sygedom og lokation information fra systemet	Kjeld
Mink klar til salg	Levere mink	Efter at minken er blevet flået Registrere mængden og kvalitet af pels i systemet Levere pels til Copenhagen Fur Modtager meddelelse om at ordre er modtaget og	Medhjælper/ Kopenhagen Fur
Se status på farmen	View farm information	Systemet henter info fra hvert bur og viser dem på en side.	

FORUDSÆTNINGER TIL PROGRAMMET

Det har en stor betydning for Mink Farm, at det nye system skal være brugervenligt.
Eftersom at personalet i virksomheden ikke har den største viden inden for IT, så skal systemet fungere på en måde, så de ansætte ikke skal bruge for meget tid til at bruge systemet.
De vil implementere nummer på deres bure, som vil gøre det nemmere for dem at holde styr på lige præcist hvad der står hvor i deres system.
Systemet skal også være sikkert og stabilt, da de ikke føler sig gode med en computer, og bare vil have at det virker som det skal.

En liste over de **Funktionelle krav** der er kritiske for Keld V. Larsens Mink farm:

Systemet skal kunne oprette nye burer med bur lokation.
Systemet skal kunne håndtere mink arter, aldre, farve, ...
Systemet skal kunne holde styr på syge mink information.
Systemet skal kunne holde styr på status på minks lokation, sygdom.. .

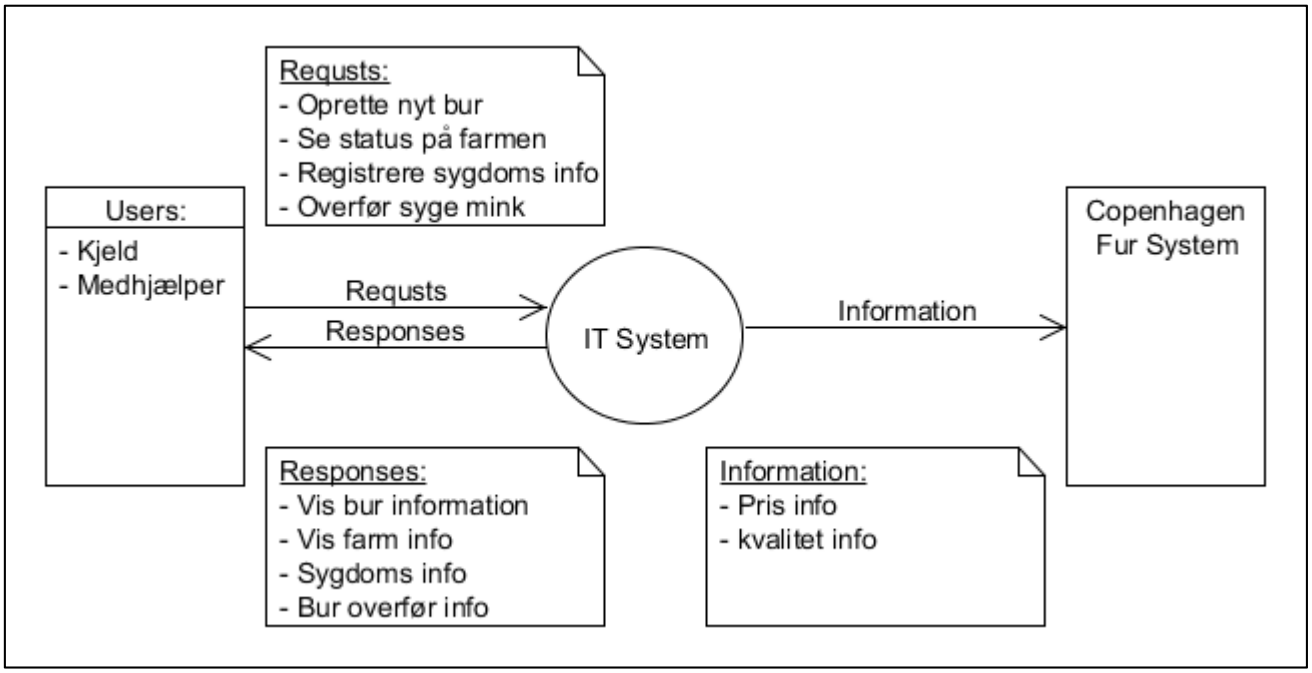
Ikke Funktionelle krav:
Systemet skal være brugervenligt.
Systemet skal være pålideligt.
Systemet skal fungere korrekt.
Systemet skal vise meningsfulde fejlbeskeder.

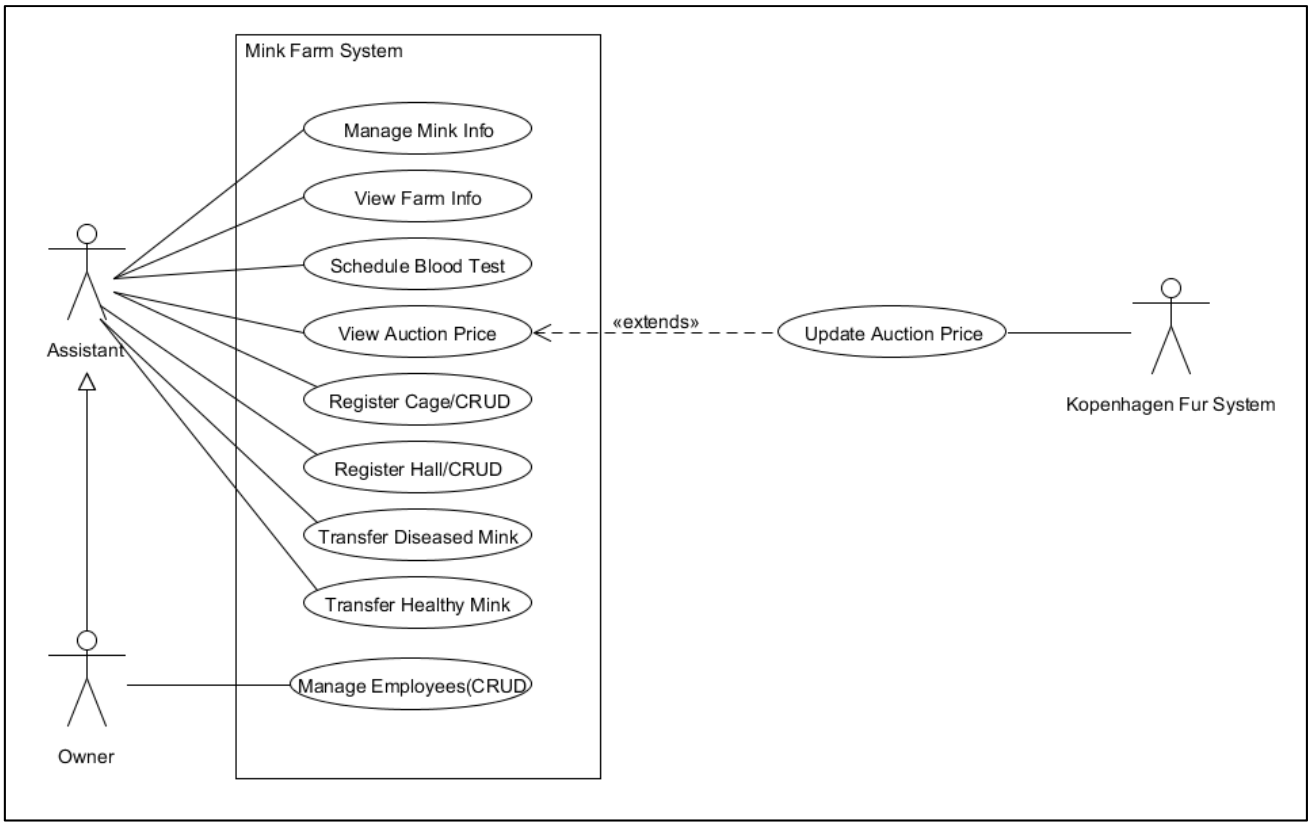
SYSTEM BRUGER BESKRIVELSE

Systemet vil have to brugergruppe: Ejer og Medhjælper.

Navn	Beskrivelse	Repræsentanter
Ejer	Har adgang til alle funktioner I systemet og håndtere med-hjælperne informationer.	Farm ejer (Kjeld). Har en okay god erfaring med brugen af en computer og IT System
Medhjælper	Ansvarlig for registeringen af informationer, så som at regi-strere nye mink, oprette nye bure og registrere overflyttede mink informationer I systemet	En medhjælper på Kjeld's farm – har lidt erfaring med brugen af computere og IT systemer

PRODUKT OVERSIGT





USE-CASE BRIEF BESKRIVELSER

Manage Mink Info:

- Medhjælperen registrerer minken og tilføjer den til systemet med div. informationen(hvilken farve minken er, længde på pels, hvor tæt pelsen er, gener, om den har Plasmacytose eller har haft, om der er bidsår).

View Farm Info:

- Systemet henter info fra hvert bur og samler info til en side, samt farm information

Schedule blood test:

- Hvilke bure der skal testes hvornår og af hvem.

Register Cage (CRUD):

- Systemet opretter buret og tilføjer dets status.

Register Hall (CRUD):

- Systemet opretter hallen og tilføjer hal nummer.

Blood test:

- Systemet modtager resultatet af blodprøverne og registrere flytning af mink.

Manage Employees (CRUD):

- Ejeren kan logge ind og rette hans medarbejders informationer.

Transfer Diseased Mink:

- Systemet modtager blodprøverne og finder et nyt but til de syge mink. Systemet registrere at minkene er blevet flyttet og flytter informationerne og husker det gamle bur.

Transfer Healthy Mink:

- Systemet modtager blodprøverne og udsender en påmindelse på at der er raske mink blandt de syge og er klar til at blive flyttet tilbage til det gamle bur.

View Auction Price:

- Systemet viser prisen på auktion priser

Update Auction Price:

- Kopenhagen Fur opdaterer den sidste auktions priser. Den er en ekstern system, udenfor vores system

USE-CASE PRIORITERING BASERET PÅ FORRETNINGSVÆRDI OG TEKNISK KOMPLEKSITET

Use-Case: Use-case nummer

Aktør: En rolle, som personerne på arbejdspladsen tager/er

Mål: Hvad aktøren skal have IT-systemet til at udføre

Forretningsværdi: Vigtighed af use-casen for forretningen; niveauet for værdi de får ud af denne use-case

Teknisk kompleksitet: Hvor svært mht. hvor meget tid det vil tage at løse opgave x

Estimering af tid: Hvor meget tid der er afsat til hver enkel use-case

Use-Case ID	Mål	Aktør		Forretningsværdi	Teknisk Kompleksitet
		Medarbejder	Ejer		
UC1	Manage Mink Info	X	X	3	2
UC2	View Farm Info	X	X	2	4
UC3	Schedule Blood Test	X	X	3	2
UC4	View Auction Price	X	X	1	2
UC5	Register Cage	X	X	3	2
UC6	Register Hall	X	X	3	2
UC7	Transfer Diseased Mink	X	X	4	4
UC8	Transfer Healthy Mink	X	X	3	2
UC9	Manage Employees		X	1	1

Værdierne er repræsenterede med 1-4, hvor 4 repræsenterer de mest værdifulde/kompleks use-case og 1 repræsenterer de mindre værdiful/kompleks use-case.

VALG AF USE-CASE

Vi har valgt at tage udgangspunkt i use-casen Transfer Diseased Mink

FULLY DRESSED USE-CASE BESKRIVELSER

TRANSFER DISEASED MINK

UC7: Transfer Diseased Mink

Omfang og niveau: Dette omhandler hvad der sker når der skal flyttes en mink fra et bur til et andet på grund af sygdom.

Primær aktør: Medhjælper

Pre betingelser: Der er tomme bure på minkfarmen

Mink er registret i et bur

Post betingelser: Minkene er blevet flyttet til et andet bur

Basis succes flow:

1. Systemet registrerer at blodprøven er afsendt
2. Medarbejderen for at vide at der er sygdom i buret
3. Medarbejderen flytter minkene til et andet bur
4. Systemet registrerer det gamle bur med alle oplysningerne
5. Systemet registrerer det nye bur og flytter oplysningerne
6. Systemet gemmer hvilken dato minkene er blevet flyttet

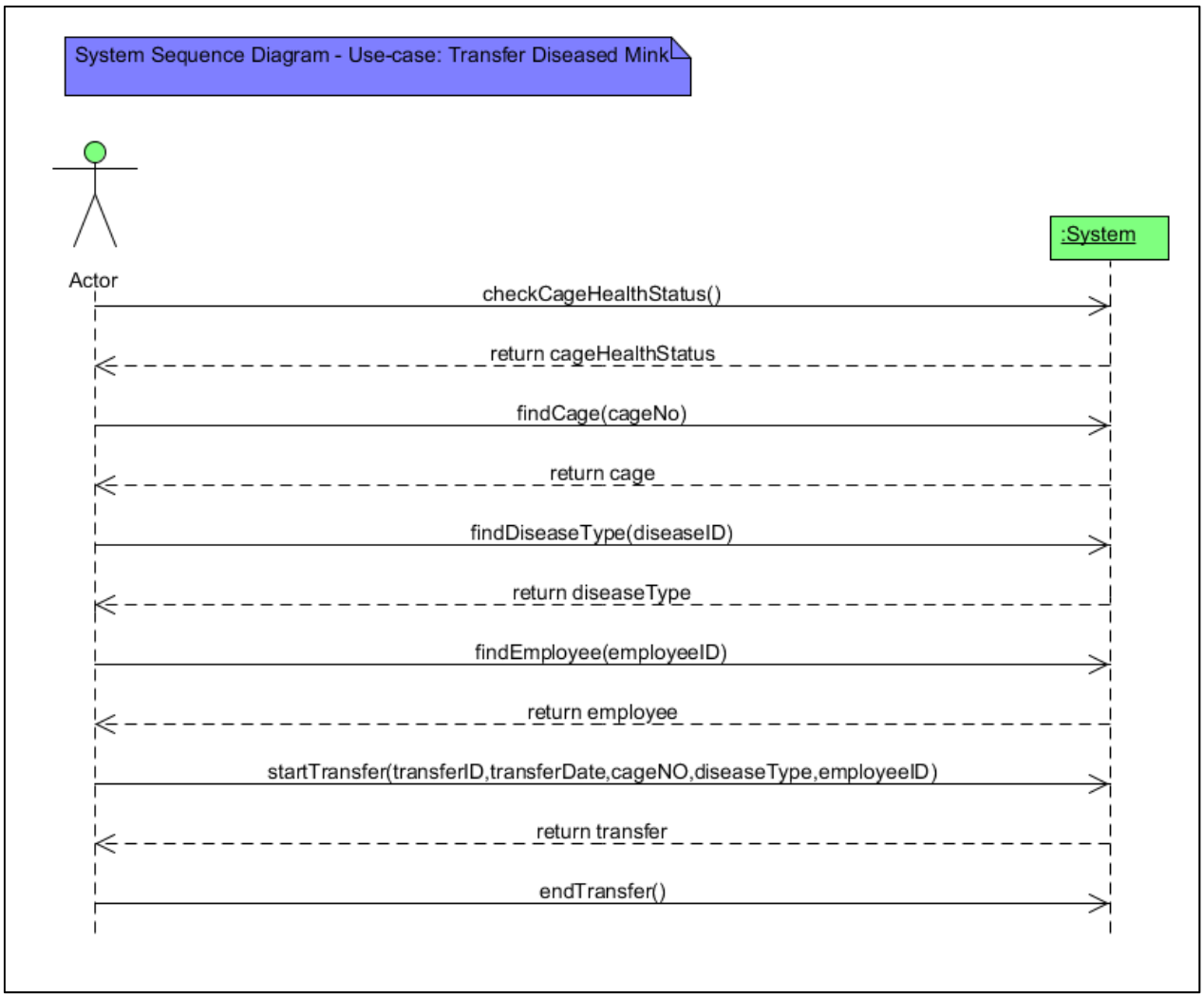
Alternativt flow:

- Systemet går ned
- Loginet virker ikke

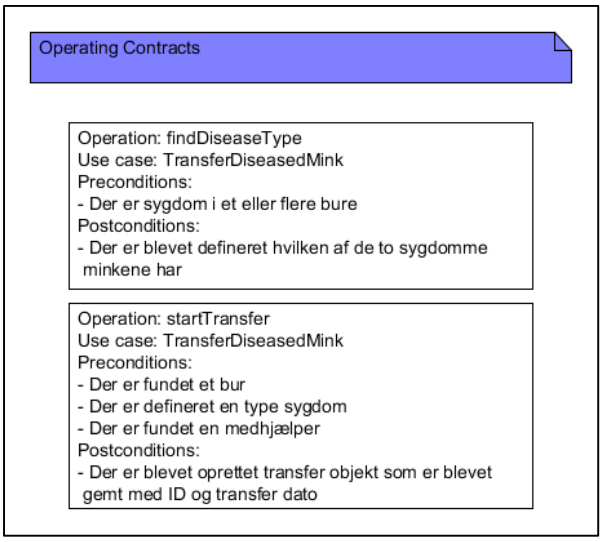
Mockups:

Se Bilag

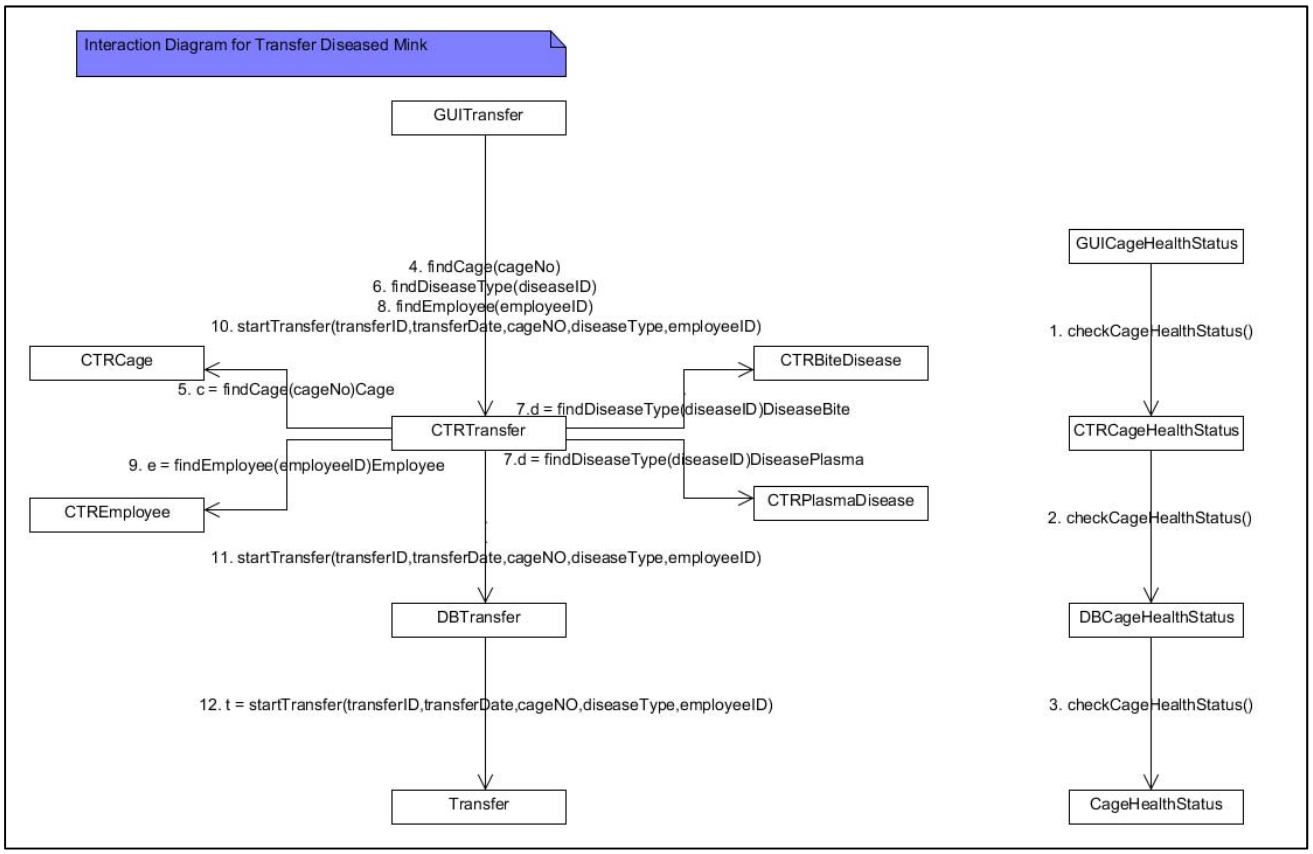
Følgende SSD er udarbejdet på baggrund af den ovenstående fully dressed use-case



Ud fra ovenstående SSD har vi de tilhørende operationskontrakter



Ud fra ovenstående SSD og tilhørende operationskontrakt, har vi udarbejdet følgende interaktions-diagram for use-casen Transfer Diseased Mink



!!! FIRST DOMAIN MODEL VERSION 1 without inheritance.+ incomplete attributes.

FASEPLAN

Phase	Number of Iteration	Start	End	Artifacts
Inception	1	Wk 49	Wk 50	• System vision • Use-Cases (Brief beskrivelser, Prioritering af use-cases, Fully dressed beskrivelse af de mest komplekse use-case) • Mock Ups • Domæne Model.

EVALUERINGSKRITERIER

De primære krav, samt systemets omfang er blevet identificeret

Sporbarhed mellem projektets vision, use-cases og domænemodel

GODKENDELSE PROCEDURE

Internt review af fase-artefakter.

ELABORATION FASE

FASEPLAN

Phase	Number of Iteration	Start	End	Artifacts
Elaboration	2	Wk 50	Wk 52	• Den mest kritiske use-case design(SSD: accept test, Interaktions diagram, operations kontrakt • Database design (Normalisering, Transformation, Tabel) • Arkitektur (3 eller 4 lags arkitektur) • Design klassdiagram-me,Implementering + unittests). • Kvalitets planlægning • Iterations planer

EVALUERINGSKRITERIER

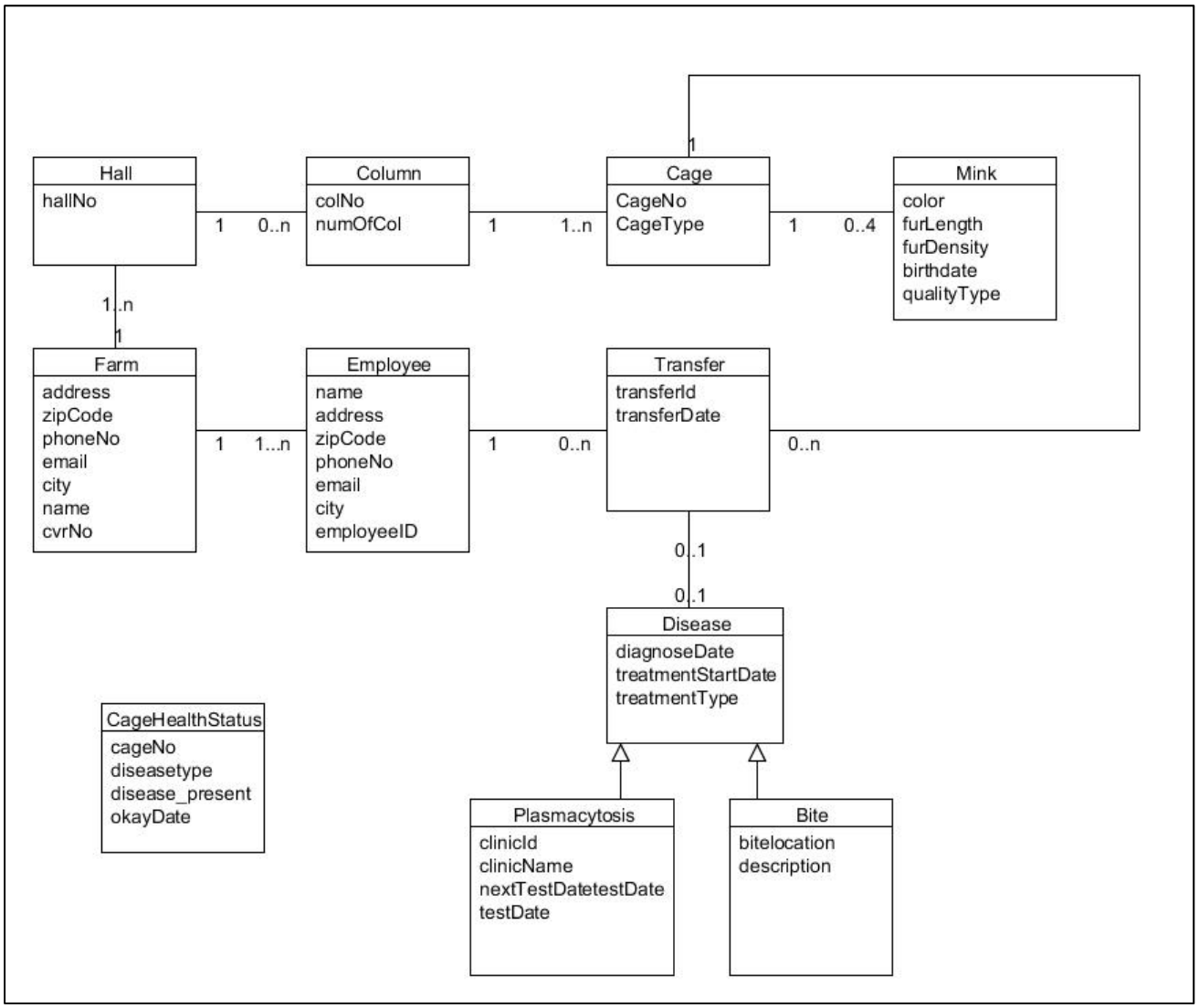
GODKENDELSE PROCEDURE

KVALITETS PLANLÆGNING

ITERATIONS PLANER

ITERATION 1

Domain Model: Mink Farm



TODO !! SSD, Operations kontrakt, domain model refinement (complete ordering of attributes), TESTS?

ITERATION 2

ARKITEKTUR

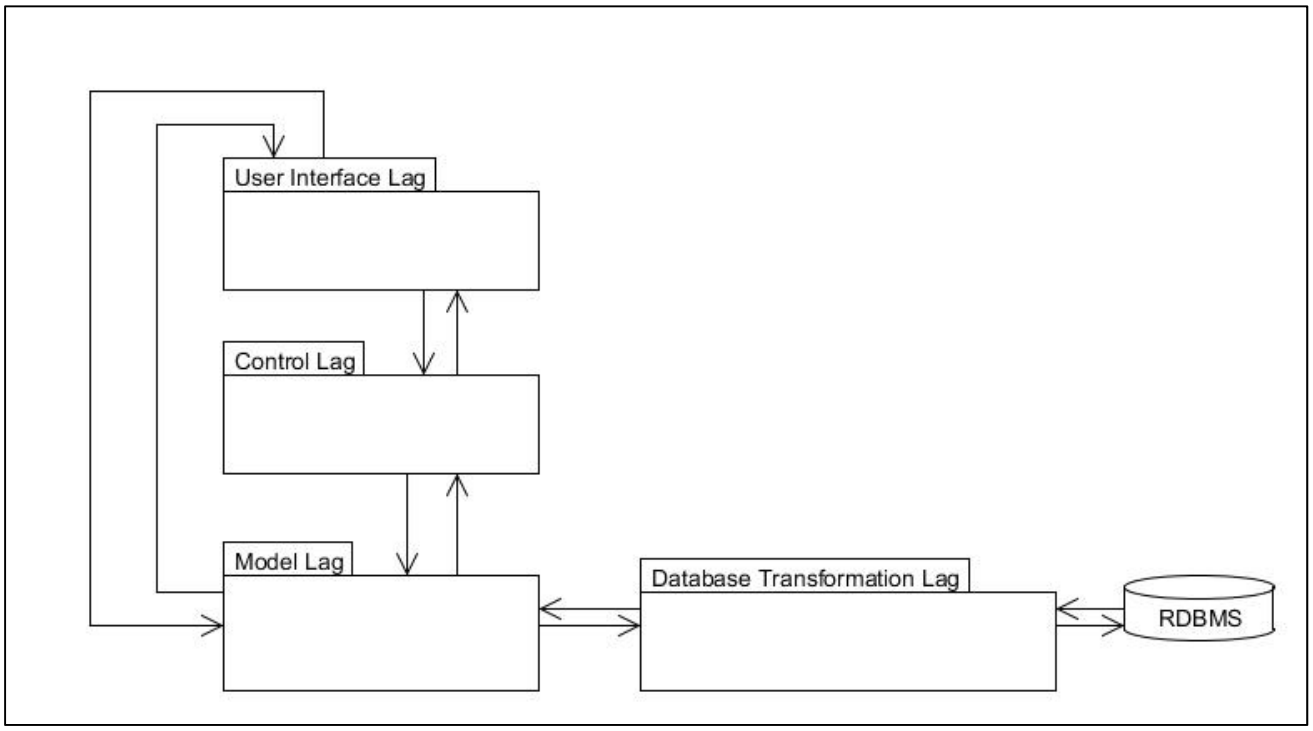
Patterns:

Patterns eller Mønstre er kendt løsninger på gentagne design problemer, som med mindre modifikationer kan anvendes i mange sammenhænge. Den fremmer udviklingen af cohesive moduler med minimal kobling. Vi vil referere til nogle design mønstre vi overvejede under vores design.

Facade Pattern

Facaden mønster er et design mønster, der bruges til at forenkle adgangen til funktionalitet i komplekse subsystemer. Facaden giver en enkelt interface, der skjuler implementeringsdetaljer i det underliggende subsystem.

Vi har valgt at strukturere vores IT-System ved at anvende 4 lags arkitektur Vi valgt at bruge et lag arkitektur, fordi det vil gøre det muligt at erstatte et lag med minimal indsats og uden bivirkninger i vores system. Denne arkitektur vil også gør det let at vedligeholde systemet på grund af de lave koblinger mellem lagene, samt at det vil gøre det muligt at genbruge lagene. Vores system er delt op i disse 4 forskellige lag: *User Interface Lag, Control Lag, Model Lag* og *Database Transformation Lag.*



!!TODO Rettes igennem

User Interface Lag: Dette lag er ansvarlig for håndtering af interaktion mellem aktøren og bruger-grænsefladen

Control Layer: Dette lag er limlaget mellem User Interface lag og Model lag. Håndtering af use cases sker i dette lag

ModelLayer: Dette lag er afledt fra domæne model. Den indeholder klasser fra vores domæne model

Database Transformation Lag: Dette lag indeholder klasser, som sikre for håndtering af persistence af objekter i model laget. Den har ansvar for kommunikation med databasen, samt bygning af objekter i model laget.

Database: Databasen gemmer objekter der skal holdes persistent.

PS: En arkitektur kan enten være **åben** eller **lukket**. Vores arkitektur anvender en åben/lukket arkitektur.

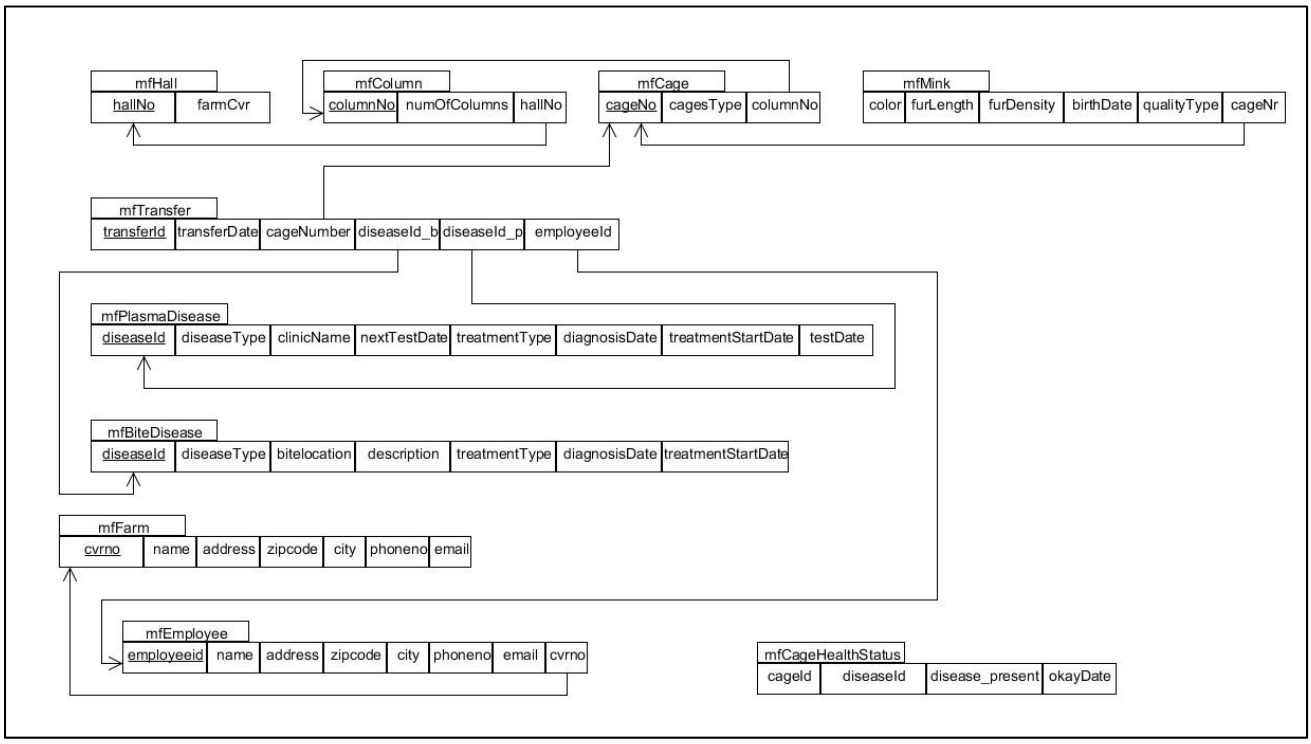
Åben arkitektur: Det betyder at … !todo

Lukket arkitektur: Det betyder at … !todo

DATA MODELING

Formålet med dette afsnit er at mappe vores domæne model til et relationelt skema, ved hjælp af mapning regler \add ref. Vi begynder med at beskrivelse vores databaseskema samt nogle integrity constraints vi har taget højde for, næste viser vi hvordan vi har valgt at mappe en generalisering struktur i vores domain model og til sidste har vi normalisering.

Databaseskema Et databaseskema bruges til at vise logisk design af en database. Dette databaseskema viser struktur af vores database, den viser alle vores tabeller og relationer mellem tabellerne. De primære nøgler er understreget.



De Integrity constraints vi har taget højde for:

- *NOT NULL constraint* hjælper til at sikre at et felt altid indeholder en værdi. Det for eksempel betyder, at man kan ikke indsætte et ny felt i eller opdatere employeeid felt i mfEmployee tabel uden at tilføje en værdi til dette felt.
- *Referential integrity constraint* gennemføres ved bruge af en kombination af en primær nøgle og en fremmed nøgle. For eksempel, skal man angive en gyldig cageNo for at tilføje et nyt række til mfTransfer tabel. Det er for at sikre at hver felt i en tabel, der er erklæret en fremmed nøgle, kun indeholde værdier fra forældrenes tabels primære nøglefelt.
- *Primære key constraint* hjælper til at entydigt identificerer hver række i en database tabel. Eksempel er primære nøglet cageNo mfCage tabel, som kan kun indeholde unikke værdier.
- *Foreign key constraint* tillader et nøgle fra en tabel (fremmed nøgle) til at pege på en primær nøgle i en anden tabel. Eksempel er fremmed nøgle cageNumber i mfTransfer tabel som peger på cageNo som er et primær nøgle i mfCage tabellen.

Vi har oprettet alle de tilsvarende databasetabeller i vores database. Vi vil få adgang til data i tabellerne ved at bruge SQL, som er både en DML (Data Manipulation Language) og en DDL (Data Definition Language). Følgende supplerende dokumenter er blevet tilføjet som bilag**: Database scripts** til

at oprette de **Tabeller** i databasen og **Data Dictionary** som indeholder navnene og typer af alle felt i tabellerne.

!!! TODO ADD DATA DICTIONARY, SCRIPTS FOR INSERT+DB CREATION IN BILAG

Mapping af Generalisering struktur i vores domain model

Problem:

Vi har en generalisering og specialisering struktur i vores domæne model (det er mellem Disease, Plasmacytosis og Bite). Vi har transformeret dette struktur til tabeller ved at bruge et af de 3 mulige alternativer (de 3 alternativer har hver deres fordele og ulemper).

Løsning:

Vi har valgt at lave 2 tabeller (*mfBiteDisease* og *mfPlasmacytosisDisease*).

Den primære grund for at vælge at bruge dette alternativ er, fordi de minker kan kun have en af de to typer af sygdomme ad gangen, som kan enten være plasmacytosis eller bidsår. Vi har derfor valgt, at hver specialiseringsklasse (bidesår og plasmacytosis) skal afbildes i en tabel, som også indeholder generaliseringsklassens(Disease) attributter.

Men dette betyder ikke, at denne metode ikke har nogle ulempe. En ulempe er, at det kræver rettelser i begge specialiseringsklasser når der sker ændringer i generaliseringsklassen, men det er noget vi ikke forventer at det vil forekomme tit.

Normalization

Formålet med normalisering er at dekomponere relationer (tabeller) med anomalier. Det hjælper os til at producere strukturerede relationer, der indeholder mindre eller ingen redundans. Dvs. den primær hjælper os til at:

- minimere lagerplads
- reducere redundans
- minimere insertion, deletion og update anomalies

Functional dependency bruges til at analysere design kvalitet af en relation(tabel). Anvendes af functional dependency regler kan hjælpe til at transformer en tabel til en tilstand som kaldes **Normal Form**. Normalformer sikre, at anomalier, redundans og inkonsistens er reduceret i en database.

En tabel kan være i en af følgende normale former: 1NF, 2NF, 3NF, ... eller BCNF. Vi anvender kun BCNF i dette projekt. Det er sådan at hvis en tabel er i BCNF, så er det også i 3NF, 2NF og dermed også i 1NF \ref.

BCNF i mfEmployee tabel

Ideen bag BCNF er, at hvis en tabel ikke er BCNF kompatibel, så skal de ikke-funktionelt afhængige attributter bruges til at danne nye separat tabeller. Vi vil vise hvordan vi har normaliseret mfEmployee tabel, ved at bruge BCNF. I øjeblikket er mfEmployee tabel ikke i BCNF.

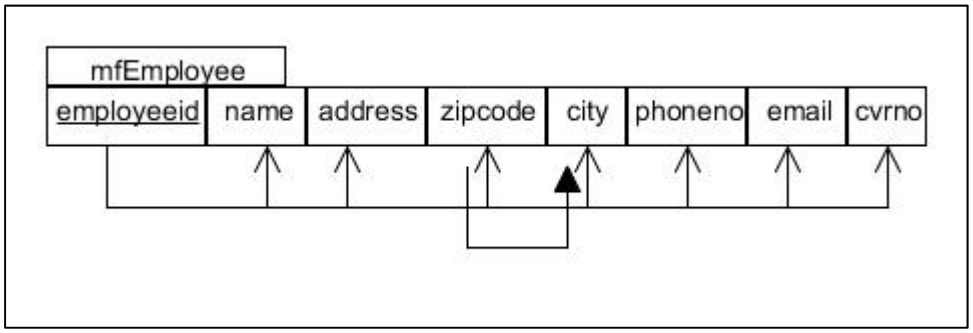
mfEmployee (employeeid, name, address, zipcode, phoneNo, city, email)

Kolonnen employeeid er en candidate nøgle, fordi den giver mulighed for unikt at afgøre værdien af de andre kolonner i mfEmployee tabel. Men, zipcode kolonne kan også afgøre værdier i city kolonne.

Selv om zipcode kolonne kan også afgøre værdier i city kolonne, kan det ikke bruges som en candidate nøgle fordi den kun kan afgøre værdier af city attributter og ikke alle de andre attributter.

Problem:

Vi kan konkludere, at mfEmployee tabel ikke er i BCNF, fordi city kolonne er funktionelt afhængig af både employeeid kolonne og zipcode kolonne - som ikke er en kandidat nøgle.

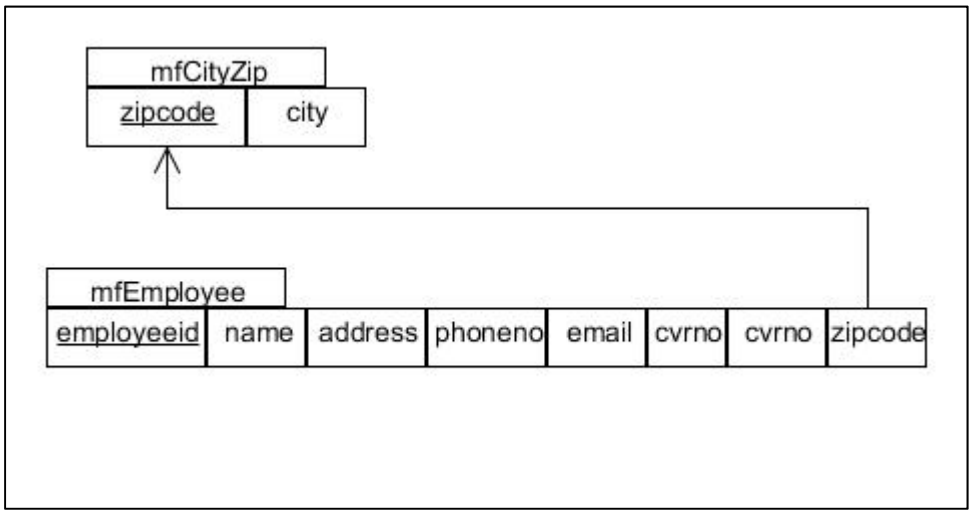


Løsning:

For at dekomponere denne tabel til BCNF, skal mfEmployee tabel derfor opdeles i to tabeller, hvilket resulterer i to nye skema, der ser sådan ud:

- mfEmployee (employeeid, name, address, phoneNo, email)

- mfcityZip (zipcode, city)



Den ny tabel mfCityZip bruger zipcode som primær nøgle.

Tabellen mfEmployee bruger fremmed nøgle zipCode til at pege på mfCityZips primær nøgle.

DESIGN MODEL

!!! TODO remember grasp pattern and patterns.

ARTIFACTS:

Interaktionsdiagram, designklassediagram, kode og test i kritisk use cases, Testing of model classes,

Testing of Database connection(patterns-singleton)

BILAG