IT-DIPLOMUDDANNELSEN

DATABASESYSTEMER OG WEB

Eksamensprojekt

af

Jan Schrøder Hansen

eFTERÅR 2010

**INDHOLD**

[1. Indledning 3](#_Toc278110191)

[2. Opgavebeskrivelse 3](#_Toc278110192)

[3. ER Diagram 3](#_Toc278110193)

[4. RI Diagram 6](#_Toc278110194)

[5. Normalisering 7](#_Toc278110195)

[6. Egenskabstabel 7](#_Toc278110196)

[6.1. Tabeloprettelse 9](#_Toc278110197)

[7. SQL udtræk og views 9](#_Toc278110198)

[8. Indeksering 10](#_Toc278110199)

[9. Sikkerhed og brugere 11](#_Toc278110200)

[10. Konklusion 11](#_Toc278110201)

[11. Bilag 12](#_Toc278110202)

[11.1. Brugervejledning 12](#_Toc278110203)

[11.2. Udviklingsmiljø 12](#_Toc278110204)

[11.3. Kode 12](#_Toc278110205)

[11.4. Indhold på vedlagte CD 12](#_Toc278110206)

# Indledning

Dette eksamensprojekt er lavet i forbindelse med faget Web og serverprogrammering på IT- Diplomuddannelsen, Ingeniørhøjskolen i København.

Faget har taget udgangspunkt i bøgerne ”Database Management Systems” af Raghu Ramakrishnan og Johannes Gehrke og ”SQL i praksis” af Ben Forta. Samt materiale udarbejdet af underviser Torben Lund.

# Opgavebeskrivelse

For at komme igennem så meget af materialet i faget som muligt, har jeg valgt at lave en webløsning til håndtering af ønsker og fejlreporter, til et eller flere softwareprodukter.

Der skal være mulighed for at følge status på fejl og ønsker. Er fejlen eller ønsket godkendt, afvist, er det under udvikling, under test etc. Derudover skal der være mulighed for at uploade forskellige filer såsom skærmdumps af fejl, eller prototyper på ny skærmlayouts og andre filer som kan hjælpe til at belyse en sag.

Programmets navn er BaRI, som står for ”Bugs and Request Interceptor”.

Det er en løsning jeg startede på under faget Web og Serverprogrammering, efteråret 2009, rapporten kan findes på den vedlagte cd. I denne opgave var mit fokus på et web framework kaldet Wicket[[1]](#footnote-2) . En lille del af opgaven skulle også demonstrere at løsningen kunne persistere data i en database. Så der blev lavet 2 tabeller. I denne opgave vil jeg forsætte på denne løsning, men med fokus på database delen.

Programmet udvikles i sproget Java[[2]](#footnote-3). Med Java følger databasen JavaDB[[3]](#footnote-4), derudover benyttes Hibernate[[4]](#footnote-5) som bro mellem den relationelle database verden og den objekt orienteret verden.

I de følgende afsnit, gennemgås de forskellige designe faser, som benyttes i forbindelse med design af en database. Startende med et *entity-relationship* digram[[5]](#footnote-6), herefter kaldet et ER diagram.

# ER Diagram

Følgende figur er et ER diagram bestående af entiteter (kasser), som er identificerbare selvstændige objekter, navngives ofte med et navneord. Samt relationer (kasser på spidsen) som beskriver relationer mellem entiteter, navngives ofte med udsagnsord. Og sidst men ikke mindst af attributter (cirkler), som kan knyttes til både entiteter og relationer. Attributter beskriver enkelte data som navn, adresse etc. På den eller de attributter, som entydig identificere en relation, er navnet på attributten understreget.

ER Diagram.wmf

Figur 1- ER Diagram

Som det fremgår af ovenstående figur så har alle entiteter en nøgle kaldet id (alle de gule attributter). Dette skyldes at jeg har valgt at benytte frameworket Hibernate, som er nemmere at arbejde med, når der bare benyttes et fortløbende tal som nøgle.

Som det fremgår af diagrammet er der et Produkt, som indeholder navne på de produkter som systemet skal kunne modtage fejl og ønsker til. Så er der en BariUser, som indeholder de brugere som har adgang til systemet. Mellem Produkt og BariUser er der en mange til mange relation kaldet userGroup som beskriver hvilke produkter de enkelte brugere har adgang til. En bruger har også en rolle, se userRole, som beskriver om en bruger er en udvikler, administrator eller en alm. bruger. En administrator kan alt i systemet, og som den eneste kan denne oprette nye brugere og produkter. En alm. bruger kan oprette fejl og ønsker, samt oprette diskussionsindlæg og uploade skærmdumps og andet. En udvikler kan det samme som en alm. bruger, samt ændre på en udviklingsstatus, som siger noget om hvor langt et ønske/fejl er mht. udvikling og test.

Centralt i systemet er der en BariCase som beskriver en fejl eller et ønske, til hvert BariCase kan der knyttes flere diskussionsindlæg (DiscussionMessage) og igen til hvert diskussionsindlæg kan der igen knytte flere filer (Attachement), som kan være skærmdumps eller andet, som kan benyttes til at uddybe et indlæg.

En BariCase kan være af type fejl eller ønske (Type), samt have en sagsstatus (CaseStatus), som kan være ”Ny”, ”Behandles”, ”Godkent”, ”Afvist” eller ”Afsluttet”, samt en udviklerstatus (DevStatus ) som kan være ”Ej begyndt”, ”I gang”, ”Klar til test”, ”Testet” og ”I produktion”.

ER diagrammet som sådan kan ikke direkte implementeres i en database, så næste design fase, er et *Referential integrity*[[6]](#footnote-7)diagram, herefter kaldet RI Diagram.

# RI Diagram

Et RI diagram benyttes til at beskrive de referentielle sammenhænge der skal være mellem tabellerne i en database. En database benytter referentiel integritet til at sikre, at data er konsistente. F. eks. at der ikke slettes rækker i en tabel, som der refereres til fra en anden tabel.

RI Diagram.wmf

Figur 2 - RI diagram

Som det fremgår af ovenstående RI diagram, så mappes næste alle entiteter fra ER diagrammet, over til en relation/tabel i RI diagrammet. Der er en ekstra relation som er UserGroup. I ER diagrammet var UserGroup en mange til mange relation. Og det mappes til en tabel i RI diagrammet. Denne tabel har så en ”1 en til mange” relation til Product og tilsvarende til BariUser. Alle de andre relationer fra ER diagrammet, beskrives har med 1 til mange relationer og fremme nøgler (FK – Foreign Key). F. eks. BariUser har en FK til UserRole. Derudover har alle tabeller en primær nøgle (PK), som alle er understreget i diagrammet. I ovenstående diagram kan også ses hvilke felter, som kan være tomme (null), de har en åben cirkel for felt navnet. F.eks. i BariCase, er der et felt som hedder conclusion. Som er en konklusion på en fejl eller et ønske. Og det kan i sagens natur først udfyldes til allersidst.

Efterfølgende skal ovenstående RI diagram efterprøves for nogle normaliserings regler.

# Normalisering

Normalisering er en form for database kvalitetssikring, som sikre et database design overholder visse regler. Bl.a. at samme data kun skrives en gang. Følgende er en kort beskrivelse af de 3 første normalforme.

**1. Normalform:**

* Tabeller skal have en primærnøgle
* Der må ikke være repeterende grupper.

**Alle tabeller har en primærnøgle som hedder id.**

**2. Normal form:**

* Tabellen skal være på første normal form.
* Man må ikke kunne udlede værdien af et felt uden for primærnøglen, af en del af nøglen.

**Alle tabeller har en primærnøgle på et felt.**

**3. Normal form:**

- Tabellen skal være på anden normal form.

- Man må ikke kunne udlede værdien af et felt uden for primærnøglen af et andet felt (eller en kandidatnøgle). F. eks. postnr. og by. Hvor by kan udledes af postnr.

**Der er ingen felter som kan udledes af andre (ikke nøgle) felter.**

Ovenstående RI diagram fra afsnit 4, kan hermed siges at være det endelige design. Efterfølgende skal de enkelte tabeller specificeres men hensyn til datatyper, størrelser, constraints, samt uddybende kommentere til de enkelte felter.

# Egenskabstabel

Da jeg har valgt at bruge JavaDB som database, vil det være datatype, som benyttes i denne database, som bruges i det følgende.

Fælles for alle mine tabeller er at de har et primærnøgle kaldet ”id”. Denne er af typen bigint, og det er en nøgle som database selv generere, når der oprettes en ny række. Derudover har alle tabeller et felt kaldet ”version”.

Version er ikke beskrevet tidligere, da det er Hibernate specifikt. Det bruges af Hibernate til optimistisk låsning[[7]](#footnote-8), som i korte træk består i, at systemet går ud fra, at de enkelte brugere ikke arbejder på samme data. Hvis der så er konflikter, så er det først til møllen princippet, der bestemmer, hvem der kommer af med sin opdatering.

F. eks. hvis bruger A læser en række med id = 1 og version = 1, og en bruger B læser den samme række. Bruger A opdatere rækken, hvor Hibernate øger version med 1, så den nu er 2. Nu vil bruger B også opdatere denne række. Men bruger B får en fejl. Fordi Hibernate prøver at lave en update med følgende where sætning: ”where id = 1 and version = 1”. Da denne række ikke mere findes, vil Hibernate returnere en exception, som fanges af systemet og fortælle brugeren at de data han forsøgte at gemme, er rettet af en anden bruger i mellemtiden.

For at afgrænse opgaven, har jeg valgt ikke at implementere følgende tabeller: UserRole, Type, CaseStatus, DevStatus samt Attachment.

UserRole, Type, CaseStatus og DevStatus er alle ”type” tabeller, dem har jeg valgt at implementer som java enums i stedet for. Det betyder at jeg har erstattet alle de fremmenøgler som pegede på disse, med varchars. Disse varchars har så fået nogle constraints, som siger hvilke lovlige tekster der kan skrives i disse felter. F.eks. for Type gælder at det kun er lovligt at bruge ”ERROR” og ”REQUEST”.

I det følgende vil de enkelte tabeller som implementeres, beskrives felt for felt, dog uden id og version, som er beskrevet tidligere og er fælles for alle. Null constraint undlades også, da det kun er 2 felter som kan være null. Og det er finished og conclusion i tabellen BariCase.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Tabel** | **Feltnavn** | **Datatype** | **Forklaring** | **Domain** | **Unique** |
| **Product** | name | varchar(50) | Produktnavn |  | Ja |
| **UserGroup** | bariUser\_id | bigint | FK til bariUser |  | Ja, sammen med product\_id |
|  | product\_id | bigint | FK til Procuct |  | Ja, sammen med bariUser\_id |
| **BariUser** | fullname | varchar(50) | Brugers fulde navn |  | Nej |
|  | login | varchar(20) | Bruges login kode |  | Ja |
|  | password | varchar(20) |  | Mindst 3 tegn | Nej |
|  | userRole | varchar(10) | Brugers rolle | En af følgende værdier: ADMIN, DEVELOPER, NORMAL | Nej |
| **BariCase** | title | varchar(50) | Overskrift til sagen. |  | Nej |
|  | created | timestamp | Dato/tid for oprettelse af sagen. |  | Nej |
|  | finished | timestamp | Dato/tid for afslutning af sagen. |  | Nej |
|  | description | varchar(400) | Beskrivelse af sagen. |  | Nej |
|  | conclusion | varchar(400) | Konklusion på sagen. |  | Nej |
|  | product\_id | bigint | FK id til Product tabellen. Til det produkt som sagen omhandler. |  | Nej |
|  | bariUser\_id | bigint | FK id til BariUser tabellen. Til den bruger som har oprettet sagen. |  | Nej |
|  | type | varchar(10) | Type som beskriver om sagen omhandler en fejl eller et ønske. | En af følgende værdier: ERROR, REQUEST | Nej |
|  | caseStatus | varchar(15) | Sags status. | En af følgende værdier: NEW, CONSIDERING, APPROVED, REJECTED, DONE | Nej |
|  | devStatus | varchar(15) | Udvikler status. | En af følgende værdier: NOTSTARTED, STARTED, READYTOTEST, TESTED, INPRODUCTION | Nej |
| **DiscussionMessage** | created | timestamp | Dato/tid for oprettelse af diskussionsindlægget. |  | Nej |
|  | message | varchar(400) | Selve indlægget. |  | Nej |
|  | bariCase\_id | bigint | FK id til tabellen BariCase. Til den sag som dette indlæg omhandler |  | Nej |
|  | bariUser\_id | bigint | FK id til tabellen BariUser. Til den bruger som har oprettet indlægget. |  | Nej |

## Tabeloprettelse

Selve oprettelsen af tabellerne, sker vha. af Hibernate. Hibernate har værktøjer til at danne java klasser ud fra et database skeme/ddl, og omvendt at skabe tabeller udfra java klasser med annotations. Jeg har valgt den sidste mulighed. De javaklasser som er brugt findes i pakken dk.jsh.itdiplom.dbsw.domain pakken (TODO: henvising til bilag), og resultatet kan ses af det skemafilen bari\_ddl.sql som oprettes. Se bilag TODO. Derudover har jeg filen bari\_init.sql som jeg vedligeholder manuel, som indeholder enklete constraings, jeg ikke har kunne lavet i java klasserne, samt enkelte system data, såsom 3 brugere og 3 produkter. Se

# SQL udtræk og views

Jeg har valgt ikke at bruge nogle views direkte i programmet, men at benytte mig af Hibernates Query Language (herefter bare HQL). HQL er i korte træk bare SQL hvor man benytter javaklasser i stedet for database tabeller. Et eksempel fra koden:

select bariCase

from dk.jsh.itdiplom.dbsw.bari.domain.BariCase bariCase

where bariCase.type = :type and product.id = :productid;

Som det ses kan HQL returnere hele objekter af javaklasser, her BariCase. Det betyder at man slipper for at mappe fra en tabel til et objekt. Dette gøres af Hibernate. ”:type” og ”:productid” er variabler som indsættes på kørselstidspunktet. Det er også muligt at benytte alm. SQL vha. Hibernate, fordi Hibernate brug af en JDBC[[8]](#footnote-9) driver. Flere eksempler på HQL kan ses i koden i pakken dk.jsh.itdiplom.dbsw.business. TODO henvisning.

Som eksempel på SQL sætninger som kunne oprettes som views (create view <navn>), har jeg lavet nogle statistik udtræk.

|  |
| --- |
| Find antallet af sager som har en given status. De skal grupperes efter type (ERROR, REQUEST) og status (NEW, CONSIDERING etc.). |
| select type, casestatus, count(\*) as number\_of\_cases  from baricase  group by type, casestatus; |
| Find antallet af diskussionsindlæg pr. sag. Grupperes efter type, titel. Antal udskrives i faldende orden. |
| select type, title, count(baricase\_id) as "No of discussion per case" from  baricase  left outer join discussionmessage on (baricase.id = discussionmessage.baricase\_id)  group by type, title, baricase\_id  order by count(baricase\_id) desc; |
| Find antallet af sager med en given kombination af type, status og bruger. |
| select bariuser.fullname, baricase.type, baricase.casestatus, count(baricase.bariuser\_id) as "No of cases"  from baricase  inner join bariuser on (baricase.bariuser\_id = bariuser.id)  group by baricase.type, baricase.casestatus, bariuser.fullname  order by fullname, type, casestatus; |

# Indeksering

Der er pt. ikke oprettet indekser. Men gode kandidater til felter som kunne indekseres, er fremmenøgler (FK), samt felter i som indgår i joins og where betingelser. I BaRI er der kun tre tabeller som kan vokse anseligt, det er BariCase, DiscussionMessage og evt. Attachment. Størrelsen på andre tabeller som f.eks. Product og BariUser, må anses for at være beskeden.

Så en god kandidat kunne være bariCaseId (FK til BariCase) i DiscussionMessage, så det er hurtigt at finde alle diskussionsindlæg til en sag. Og det samme vil gælde for discussionMessageId i Attachment, for hurtigt at finde attachemets til et diskussionsindlæg.

Hvis der mod forventning skulle komme tusindvis af bruger på systemet, så kunne der laves et index på feltet login i BariUser, for at finde en bruger lidt hurtigere i login processen.

Men min holdning er at der først skal oprettes indeks, når behovet opstår. Det betyder at man løbende skal holde øget med systemet, for at se om der er nogle SQL udtræk som begynder at tage langt tid. Dette kan gøres vha. logging, Hvor tiden for kritiske SQL udtræk logges. Men kan også lave en query execution plan[[9]](#footnote-10), som kan fortæller hvordan databasen udfører en givent SQL udtræk. Det siger noget om der benyttes index, eller om hele tabeller skannes igennem, samt tidsforbrug, IO resurser, CPU resurser m.m.

Indeks oprettes med ”create index on <table> (<column names>)”

# Sikkerhed og brugere

# Konklusion

TODO: Noget om hibernate, noget om start på en helt nyt projekt starte med UML clas diagram, som så kunne mappen til er diagram/egenskabstabel

# Bilag

## Brugervejledning

todo kort brugervejledning

## Udviklingsmiljø

todo: netbean, subversion/googlecode, tomcat

## Kode

Koden er delt op de enkelte i lag/områder vha. java pakker (package). Se følgende figur.

pakkediagram.wmf

Figur 3 - Pakkr diagram

De grønne pakker er udviklet i forbindelse med dette program. De 2 lyseblå pakker henviser til de 2 java frameworks som benyttes.

## Indhold på vedlagte CD

Indholdet på den vedlagte CD er inddelt i følgende 3 kataloger:

* Løsning – Indeholder java kode, htmlfiler m.m samt NetBeans projektfil.
* Program – Indeholder en bari.war samt JavaDB skemafil til oprettelse af databasen.
* Rapport – Indeholder denne rapport i Word 2007 og PDF format og diagrammer i Dia-format. Samt rapporten til faget Web og serverprogrammering, som denne opgave bygger videre på.

1. Wicket – se <http://wicket.apache.org/> [↑](#footnote-ref-2)
2. Java – se <http://www.java.com/en/> [↑](#footnote-ref-3)
3. JavaDB – se <http://www.oracle.com/technetwork/java/javadb/overview/index.html> [↑](#footnote-ref-4)
4. Hibernate – se <http://www.hibernate.org/> [↑](#footnote-ref-5)
5. ER diagram – se <http://en.wikipedia.org/wiki/Er_diagram>, samt kap. 2 i Database Management Systems. [↑](#footnote-ref-6)
6. Referential integrity – se <http://en.wikipedia.org/wiki/Referential_integrity> [↑](#footnote-ref-7)
7. Optimistisk låsning – se <http://en.wikipedia.org/wiki/Optimistic_concurrency_control> [↑](#footnote-ref-8)
8. JDCB – se <http://www.oracle.com/technetwork/java/javase/tech/index-jsp-136101.html> [↑](#footnote-ref-9)
9. query execution plan – se <http://en.wikipedia.org/wiki/Query_plan> [↑](#footnote-ref-10)