Abonnementssystem

Ornella Ardoino
Harry Ballesteros Petersen

Kgs. Lyngby 2003
IMM-THESIS-2003-1

ISSN 1601-233X

Forord

Dette projekt er resultatet af en forespørgsel fra finansverdenen og dens involverede parter. Projektet handler om at håndtere indkommende information fra finansverdenen og at videresende den til involverede bankkunder. For at kunderne kan få den information de ønsker har vi udbygget et såkaldt Abonnementssystem. Dette system vil forøge fleksibiliteten i de nuværende banksystemers servicering af kunderne.

Vi fik muligheden for at udarbejde projektet via SDC-udvikling, som er et dansk firma beliggende i Ballerup der leverer e-banking løsninger til mere end 80 banker i Danmark. Abonnementssystemet afspejler en fiktiv forbindelse mellem finansverdenen og kunderne samt den konkrete måde hvorpå kunderne bliver serviceret ved hjælp af dette system.

Abonnementssystemet er udarbejdet ved hjælp af nogle centrale metoder og nødvendige værktøjer fra programmeringsverdenen. Disse giver mulighed for at indlæse og udvælge informationer fra finansverdenen, som derefter sendes til de pågældende kunder.

Vi vil gerne takke vores vejleder på DTU, Jens Thyge Kristensen, og Jan Siegumfelt fra SDC-udvikling for deres støtte og vejledning. Desuden alle andre som har bakket os op undervejs med råd. Blandt disse vil vi specielt takke Ane og Karin.

....

Ornella Ardoino Harry Ballesteros Petersen

Abstract: The Subscription System

This project is the result of a request from the Financial World and its partners. When a potential bank client may have the need to have some information, when something is changing in the financial world, like for instance a balance of its account or an immediately stock information, without having a physically contact with a bank, it is now possible to receive information about it.

The main task of this project is to create a system that can handle a large amount of incoming information/data from an outside source. The treating and sorting of these incoming data, and afterwards the sending of those to the so-called subscriptors, will be handled by the implementation of a push-system.

First of all, it is necessary to have a database structure in order to save information concerning the subscriptors and the subscriptions. This Database is created using Microsoft Access Technology. This Technology allows manipulating Data easily because of the many features Access consists of. A possibility to implement the same Database is given with the use of MySQL.

Second, to deal with the incoming information from the financial World, it is necessary to know which kind of incoming file and the format of this. Since we ourselves had the possibility of creating this, we decided to do it in a XML format.
The Client Class processes the creation of this file. The file is created randomly as well as the data it contains. This means that the following output of information is never the same. In this way we approach the real world as much as possible.
The receiver of this data is a Server Class, which has an XML-Parser. This Parser parses the file in order to find informations of relevance for the subscriptor. As soon as the Parser reads the information it is supplied with the help of other Classes; These Classes searches into the Database to find the addresses of subscriptors in order to send the wanted informations later on.

At last the whole system has a need to have a website where the clients will have the possibility of turning into subscriptors.

Keywords:

Subscription system, push-system, database, database design, InnoDB table type, Financial world, Parser, parsing XML, Client, Server, homebanking, Website design.

Sammendrag: Abonnementsservice

Dette projekt er resultatet af en forespørgsel fra finansverdenen og dens partnere. I det øjeblik en potentiel bankkunde vil have et eventuelt behov for at få information om hvis noget forandrer sig i den financielle verden, som for eksempel en saldos værdi, eller en øjeblikkelig information fra børsen, er det nu muligt at få den pågældende information uden at have fysisk kontakt til en bank.

Hovedopgaven i dette projekt er at skabe et systen som kan behandle en stor mængde indkommende data fra en udenstående kilde. Behandlingen og sorteringen af disse indkommende data og den følgende afsendelse af disse til de respektive abonnenter vil blive foretaget via implementeringen af et push-system.

For det første er det nødvendigt at have en databasestruktur således at information vedrørende abonnenterne og abonnementet bliver gemt. Databasen skabes ved at anvende Microsoft Access Technology. Denne teknologi muliggør at manipulere data på en hensigtsmæssig måde på grund af de mange egenskaber, Access består af. En metode til at implementere den samme database gives ved brug af MySQL.

For det andet er det for at kunne behandle den indkommende information fra finansverdenen nødvendigt at have kendskab til den indkommende fil og dens format. Eftersom vi selv havde mulighed for at skabe denne, besluttede vi os for at udføre det i XML-format.
Klientklassen processerer dannelsen af denne fil. Filen er skabt tilfældigt på samme måde som den data den indeholder. Dette betyder at det følgende udtræk af information aldrig er det samme. På denne måde nærmer vi os den virkelige verden så meget som muligt.

Modtageren af denne data er en Serverklasse, som består af en XML-parser. Denne Parser parser filen således at den finder og udvælger information af relevans til den pågælende abonnent. Så snart Parseren læser informationen, hjælpes den af andre klasser; disse klasser søger ind i databasen for at finde adresserne på abonenterne, så den ønskede information afsendes på et efterfølgende tidspunkt.

Endelig har hele systemet behov for at have en webside hvor kunderne vil få mulighed for at kunne tegne et abonnenment.

Nøgleord:

Abonnementssystem, push-system, database, database design, InnoDB tabeller type, Finansverden, Parser, parsing XML, Klient, Server, homebanking, Website design.

Indhold

Introduktion

Det er almindelig kendt at der er sket en stor fremgang i selvbetjent homebanking og andre lignende banktjenester. Mange banker tilbyder deres service på Internettet. Betaling af regninger, overførsel af penge, etc. er blevet hurtigere, nemmere og billigere via nettet. På trods af den teknologiske fremgang, er der dog stadig behov for at forbedre måden hvorpå kunden serviceres.

Visse af de nuværende banksystemer i Danmark (Nordea, Jyske Bank, osv.) yder ikke den service, som Internettet reelt kan tilbyde. I dag er man udelukkende henvist til at benytte en computer der har installeret et sikkerhedsidentifikationsprogram for at kunne få oplysninger om bevægelser på ens konti. Man er med andre ord tvunget til fysisk at sidde foran en bestemt computer, at ringe til sin bank eller at finde en bankautomat for at få besked om sine saldi, hvilket ikke er særlig fordelagtigt, hvis man for eksempel er på ferie. Der er mangel på fleksibilitet og mobilitet, samtidig med at der er et stigende behov for en forbedring af bankernes mange tjenester både til den almindelige bruger og den erfarne finansmand. Det ville være mere profitabelt hvis disse kunne handle hurtigt, når der sker ændringer vedrørende deres egne forhold. Dette ville kunne lade sig gøre ved at give dem øjeblikkeligt besked via de allerede eksisterende kommunikationssystemer, for eksempel sms og e-mail.

Alle disse services kan gøres tilgængelige ved at tilbyde kunderne en måde at tegne et abonnement som vi vil opbygge. Vores projekt handler om at servicere dette abonnement, og at gøre et forsøg på at tilgodese ovennævnte efterspørgsel. Det udføres i samarbejde med SDC Udvikling, med det formål at pengeinstituttet vil tilbyde denne mulighed til kunderne.

Indledningsvis skal kunderne i det pågældende pengeinstitut blive abonnenter til vores service. De skal tilmelde sig og give relevante oplysninger, som bliver registreret og gemt. Via disse oplysninger vil vores system tilbagesende information der tilgodeser deres individuelle interesser. Informationen dækker aktier, nyheder, konti og valuta, og hentes fra forskellige kilder som banken, avisen og børsen.

Den vigtigste del i vort system er at samle alle de indsendte informationer fra kilderne og derefter videresende dem til de respektive abonnenter. Denne funktion skal nærmest ses som en igangsætter, en motor, hvortil data strømmer kontinuerligt og ubehandlet fra finansverdenen. Disse data dækker alle mulige transaktioner, overførsler af penge, stigende/faldende aktiekurser, og så videre. Motoren udvælger relevante oplysninger fra kilderne og puffer dem derefter ud til abonnenterne, som har tilmeldt sig en eller flere services alt efter interesse. De eventuelle ændringer indenfor de pågældendes interesseområder vil altså nå frem til abonnenterne via vores system. Leveringen af information sker via SMS eller e-mail, eventuelt ved hjælp af begge dele.

For at igangsætte systemet har vi konstrueret nogle fiktive abonnenter som efterligner de virkelige forhold. Ligeledes har vi oprettet en virtuel finans-, bank- og nyhedsverden. Oplysninger om abonnenterne og abonnementerne bliver gemt i en database.

Vi har opdelt projektet i følgende tre hoveddele:

- Konstruktion af databasen over abonnementsoplysninger.
- Implementering af motoren i programmeringssproget Java
- Brugergrænsefladen

Brugergrænsefladen giver kunderne mulighed for at tilmelde sig nogle services, og få en oversigt over disse.

Rapporten er struktureret således:

Kapitel 1-4 omhandler den indledende del af projektet og vores tanker og overvejelser før vi påbegyndte opbyggelsen af Abonnementssystemet. I første kapitel forklares betydningen af de forskellige anvendte begreber, og i kapitel 2 fremlægges projektets problemformulering. Følgende præsenterer kapitel 3 et forslag til hvordan problemet kan løses, og i kapitel 4 gennemgår vi diverse værktøjer vi anvender for at nå målet for dette projekt.

Kapitel 5-9 omhandler hoveddelen af processen, hvilket vil sige den konkrete installering og programmering af systemet. I kapitel 5 gennemgås strukturering af databasen og dens sammensætning, samt omsætning af databasens diagram (ER-diagram) til tabeller. I kapitel 6 behandler vi implementeringen af motoren bag projektet (*push-systemet)* og de forskellige programmer dette forudsætter og omfatter. Kapitel 7 beskriver brugergrænsefladen og implementeringen af de diverse elementer dette indebærer, og baggrunden for opbygningen af kildefilerne forefindes i kapitel 8. Kapitel 9 forklarer hvordan serveren og klienten er opbygget og deres indbyrdes samarbejde.

Kapitel 10 omfatter en separat del af projektet, nemlig testningen af systemet. Herunder er funktionaliteten og hastigheden særligt blevet testet. Desuden har vi også testet samarbejdet mellem websiden og databasen, og hver deres funktionalitet.

Kapitel 11 omhandler ligeledes en separat del, og her forefindes en brugsvejledning, som er en samling af projektets forskellige enheder, hvilket vil sige hardware, software og deres forbindelse med hensyn til projektet.

Kapitel 12-14 omfatter den evaluerende og afsluttende del af projektet. I kapitel 12 beskrives diverse problemer vi stødte på under udførelsen af projektet, og kapitel 13 giver et forslag på hvordan systemets hastighed og effektivitet kunne forbedres. Endelig følger der i kapitel 14 en konklusion af projektet

I appendikset findes alle anvendte kildekoder, som er delt i otte afsnit. Det første tre afsnit indeholder kildekoder for SQL-filerne, Javafilerne og ASP filerne. Fjerde afsnit

indeholder 4 eksempler på nogle mulige XML-kildefiler, og det femte afsnit indeholder en Javakode til opfyldning af en databasetest. I de sidste afsnit har vi en kildekode til et forslag til at forbedre Parseren og diverse koder for bat-filer og den vedlagte CD Roms indhold.

1. Definitioner

For en bedre forståelse af de begreber vi nævner og behandler, samt diverse applikationer vi anvender i vores projekt, vil vi i dette kapitel beskrive de mest anvendte begreber i projektet:

Abonnementssystem Er systemet hvor kunderne kan tilmelde sig for at blive serviceret med diverse oplysninger.

Abonnenter Kunder der tilmelder sig tjenester der tilbydes af abonnementssystemet.

Alert Inde i databasen sættes et *j* eller *n* for at vise om for eksempel en bestemt aktie skal serviceres til abonnenten. En alert vil blive sat op i databasen, når en abonnent vil have information om en bestemt information, dvs. kunden melder sig til Servicen. Et tegn der enten står som et j eller et n, *j* for ja og *n* for nej.

Bankverdenen Pengeinstitutter.

Brugergrænseflade Den udvendig side af abonnementssystemet. Hjemmesiden for tilmelding til forskellige services.

Buffer En midlertidig blok af hukommelse som opbevarer data indtil de kan processeres. Anvendes for at kompensere differencen i antallet af "flow" af informationer, når data bliver transmitteret fra en enhed til en anden.

Classpath Er en miljøvariabel i ens system. Variablen er meget vigtigt, da den viser hvor Java run-time systemet (jre'en) kan finde de nødvendige klasse definitioner der bruges i et Java-program. Classpath'en kan omfatte stien til diverse mapper eller andre filer, såsom *JAR*s.

Datafeed Fodring af data til systemet.

Database En database er en samling af oplysninger, som er systematiseret på en sådan måde, at oplysninger hurtigt kan genfindes. Formålet med en database er, at det er et "sted", hvor en oplysning gemmes så man på et senere tidspunkt kan genfinde og anvende den lagrede oplysning.

Driver Et modul som leverer en defineret grænseflade mellem et program og hardwaren.

Entitet En entitet repræsenterer en selvstændig helhed, en ting, et sted eller et fænomen, der kan beskrives enkelt, og som der er knyttet oplysninger til. En entitet gives et entitetsnavn. Er en repræsentation af et selvstændig fænomen, som har sine egne attributter tilknyttet. Benyttes i database sammenhæng.

ER-diagram (Entity-Relationship diagram) Er en grafisk illustration af den kommende databases indhold og samspil eller forbindelser heri. Relation og valg af relationstyper mellem databasens entiteter beskriver, hvordan en databasedesigner opfatter forbindelser eller sammenhænge mellem entiteterne eller bestanddelene i en kommende database.

IP-Adresse En IP adresse bestemmer et system i et netværk som opererer i forhold til Internetkonventioner. Ipv4 adresser er traditionelt skrevet som en sekvens bestående af 4 decimalnumre som er adskilt af punktummer, for eksempel 130.225.76.10, hvor hvert

nummer ligger i et interval [0..255] og som er den decimale værdi som svarer til 8 bits af adressen.

Kildefilen Mængde af informationer som abonnementssystemet bliver forsynet med fra bank-, finans- og nyhedsverdenen.

Konstraint eller **Begrænsningen**. I databasen er der mange gange behov for en unik betegnelse for genkendelse af den pågældende data blandt andre i databasen. Der er generelt forskellige typer af begrænsninger, blandt dem findes domæne begrænsninger, og begrænsninger af boolske typer, hvor værdierne forventes at være sand.

Kunde Er den der benytter systemet. Vedkommende har kun tilgangsrettigheder til egne data.

Netværk Det er almindeligt at kommunikere over netværket i et klient-server forhold. *Klienten* er den maskine som sender en forespørgsel ud på netværket, mens *Serveren* modtager forespørgslen og giver et svar. Klient og server er relative begreber således at datamaskinen (eller et program), som er klient i én sammenhæng, har mulighed for at være en server i en anden sammenhæng.

Netværksprotokoller Et regelsæt for hvordan systemer kommunikerer sammen over et netværk.

Parser Et computer program der bryder en tekst ned i mindre dele og udkaster kendte strenge, og tegn for videre analyse.

Port nummer Et port nummer er en måde at identificere en specifik proces, til hvilken en Internet- eller en anden netværksbesked skal videresendes til, når en server har modtaget denne.

Poster Entitets eller tabels indhold.

Push-systemet Er motor af systemet. Samlingen af metoder som vi opbygger til at udplukke informationer fra kildefilen og som puffer den pågældende information ud til kunderne (abonnenterne). Det er den interne del af projektet, der behandler data mellem database og omverdenen og vicevers.

SMS-bruger Den som har mobiltelefon og sender/modtager SMS-meldinger (forkortelse for Short Message Service).

Socket En socket er et endepunkt for kommunikation mellem 2 processer på hver sin eller samme maskine.

SQL Structurede Query Language (Struktureret forespørgselssprog).

Transaktioner Indkommende information fra vores kildefiler. En transaktion kan for eksempel være et aktiesymbol og dens kurs.

Triggers En trigger er et stykke af et program som venter på at en bestemt handling (event) skal ske.

2. Problemformulering

Overordnet er vort problem at finde en løsning på at indsamle de forskellige data, filtrere dem og videresende til abonnenterne alt efter hvilke interesser de har. Hvorledes behandles den tilstrømmende mængde af information fra kilderne, og hvordan sørger vi for at de rette informationer når frem til de rette abonnenter? Hvilken metode tilgodeser at dette udføres på den mest hurtigste og effektive måde? Hvilken teknologi kan klare at behandle 20 transaktioner per sekund til ca. 10.000 abonnenter? Hvordan kan man teste at alle forudsætninger er opfyldt?

De ovennævnte centrale spørgsmål vil føre os til videre uddybning af problemet med nogle underspørgsmål.

2.1 Lagring af data

Vi skal finde en måde at håndtere mulige abonnenter og deres data. Hvilken datastruktur er mest gavnlig til designet af dette projekt? Er denne datastruktur nem at arbejde med?

Kan man anvende den samme datastruktur til at forbinde dataerne fra kildefilen og dataerne fra kunderne side? Hvilket værktøj skal man anvende til implementering af den datastruktur, man vælger til at projektet kan anvendes for eksempel på SDC?

Hvilke objekter/entiteter skal vi tage højde for?

2.2 Implementering af motoren

Der skal implementeres programmer til at indhente data, filtrere disse og udplukke relevante oplysninger.

Hvilket programmeringssprog er både sikkert og anvendt både hos SDC og evt. pengeinstitutterne eller kilderne til systemet?

Kan det anvende triggers til behandling af data i databasen?

2.3 Dialog mellem maskinen og kunderne

Der skal være en tilpassende kommunikation mellem kunderne/abonnenterne til vores system. Hvilke teknologier er i dag den der vil give de bedste resultater, og som samtidig giver et godt indtryk til kunderne? Hvad med sikkerheden? Hvad med selve designet?

2.4 Sikkerheden i systemet

Sikkerheden i systemet spiller en væsentlig rolle i et sådan abonnementssystem, især hvis dette skal behandle informationer om transaktioner eller bankoplysninger vedrørende konti. Hvordan sikrer man sig at den fil, man modtager, kommer fra den rette kilde? Hvordan undgår man en såkaldt spoofing, hvilket vil sige at modtage en kildefil, der

giver sig ud for at være en anden kilde? Og hvorledes kan man undgå en erstatning eller læsning af filen og nogle af dens informationer, som kan blive foretaget af en indbryder?

2.5 Begrænsninger

Der er i det virkelige finansmarked mange aktier, hvor vi blot vælger en håndfuld af dem. Hvordan skal de grupperes eller betegnes i vores projekt således at vi stadig er nær virkeligheden? Hvilke af de services, vi kan vælge imellem blandt de mange muligheder, vil kunderne være mest interesserede i?

Med hensyn til sikkerheden i systemet (Klient/Server forhold) har vi således valgt at ikke beskæftige os med denne problematik da SDC har deres interne sikkerhedsforanstaltninger.

3. Forslag til problemløsningen

I det følgende beskrives vores overvejelser til løsningen af de anførte problemer. Dette skal forstås som en 'brainstorm' hvormed vi indledte arbejdet i dette projekt.

3.1 Overordnet fremgangsmåde

Da vi har en stor mængde af abonnenter og en tilsvarende stor mængde af informationer om de forskellige services, har vi behov for en måde at gemme alle disse data, hvilket fører til at vi skal have en datastruktur. Som datastruktur kan vi vælge en database.
Dernæst har vi behov for at implementere nogle programmer som skal behandle datastrømmen, det vil sige selve motoren bag projektet, som vi vil kalde et *push-system*. Dér skal også implementeres nogle algoritmer som kan holde rede på alle de informationer som bliver opdateret i databasen. Disse programmer kan laves i C, C++, PMP, Perl eller Java, idet de kan forbindes med en database.
Til sidst har vi behov for at implementere en brugergrænseflade for at sætte hele abonnementssystemet i gang, og få etableret den ydre præsentation af systemet. Det vil sige at der skal designes en web-side, hvor man kan abonnere på de forskellige tjenester man er interesseret i at få information om. Til at implementere web-siderne kan vi anvende PHP, HTML eller ASP.

3.2 Potentielle services i Abonnementssystemet

For en bedre forståelse af de mulige services abonnementssystemet vil levere, gennemgår vi i nedenstående linjer de mest oplagte muligheder. De fremhævede ord er de ord der benyttes på Figur 3.2.1.

- Fond omfatter værdier for alle de forskellige aktier man kan tilbyde til abonnenterne, og de kan resumeres på følgende måde:
    - **Europæiske aktier** DAX30, FTSE100, CAC40, MIBTEL30, IBEX35, EuroSTOXX 50, Europæiske kurslister.
    - **Amerikanske aktier** Dow Jones, Nasdaq100.
    - **Indekser** Skandinaviske, Europæiske, USA.
    - **Mest omsatte aktier**; Vindere, Tabere.
    - **Danske obligationer.**
    - **Toneangivende obligationer.**
- **Valuta** Omfatter forskellige valutaer man tilbyder abonnenterne, for eksempel Euro, Danske kroner, Dollar, Yen, og andre.
- Med **Kerne systemet** menes at det repræsenterer de forskellige bank-data, som man kan tilbyde kunderne. Information om ens kontobevægelser.

- **XXX** kan være alle forskellige nyheder som abonnenten kan være interesseret i. En idé kunne blandt andet være:
  Samlede finansnyheder, RB-Børsen, Reuters, Profit Warnings, Finanskalender, Ugefokus. Hvis en nyhed har en stor betydning for at en aktie stiger/går ned i pris, vil man få besked om det.

Det skal understreges at ovennævnte potentielle services kun er eksempler for mulige tjenester.

3.3 Datafeed

Finans-, bank- og nyhedsverdenen leverer alle ændringer til abonnementssystemet. Systemet sørger herefter for at finde relevante hændelser, eksempelvis hvis der er sket ændringer i aktiekurserne, og disse ændringer er relevante for abonnenterne.
Derefter sender systemet de udtrukne informationer til dem der abonnerer på servicen, ved brug af e-mail eller sms. Hændelsen og dens sammenhæng i systemet skematiseres i Figur 3.2.1
Der fødes data ind i systemet med omkring 20/30 transaktioner per sekund, og der er cirka 10.000 abonnenter. Strømmen af data kommer fra en kildefil, og denne afsendes fra bankverdenen til systemet. Filen indeholder en masse informationer om forskellige aktiebevægelser, kontobevægelser, og så videre.

3.4 Verbal beskrivelse af datamodel

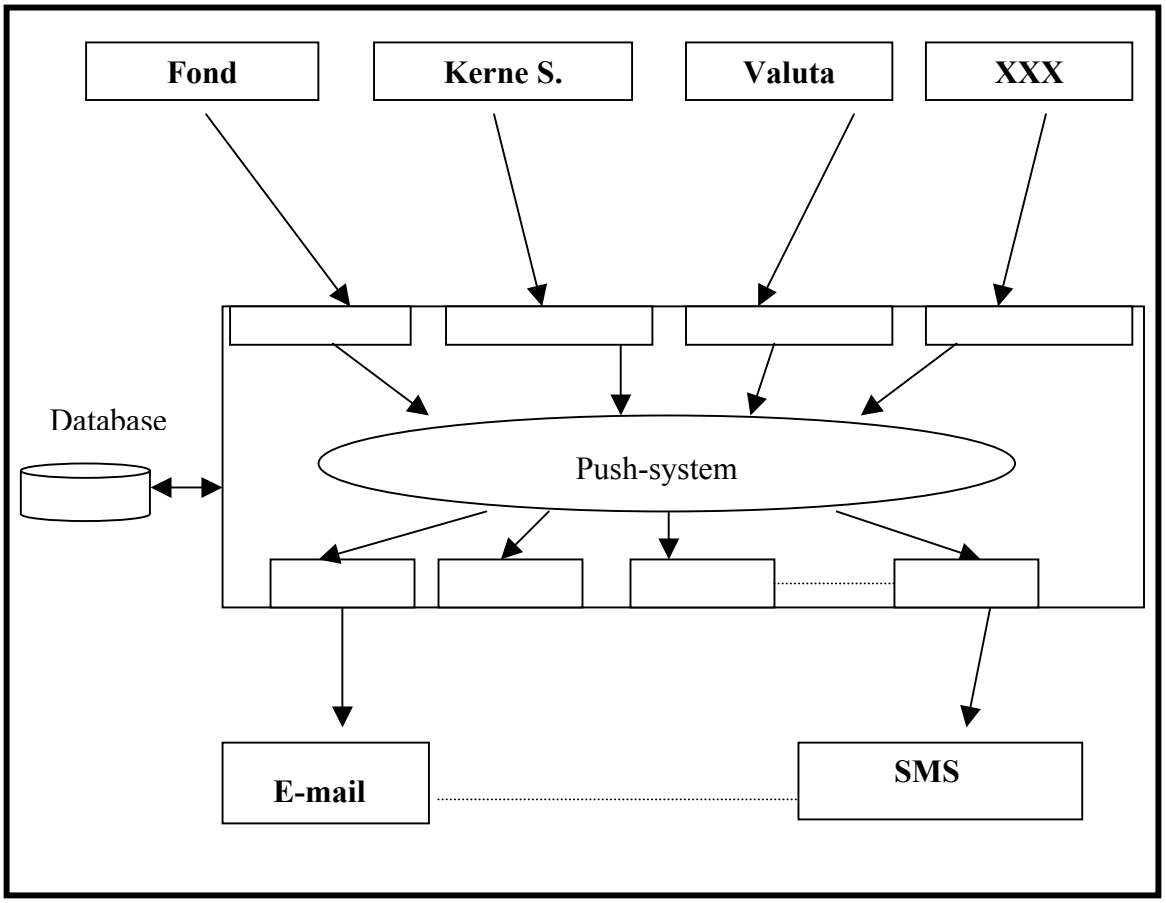
Vi vil konstruere en database, hvor man kan lagre alle informationer om abonnenterne og deres respektive abonnementer. Databasen skal struktureres således at den kan indeholde forskellige tabeller. Vi vil opbygge vores database til at servicere forskellige tjenester, i vores tilfælde aktier, konti, nyheder og valuta.

En tabel skal blandt andet indeholde informationer om abonnenterne, en anden om de forskellige aktier, en tredje om konti, etc. Dette betyder at der skal designes et E/R diagram, som derefter skal implementeres i SQL.

Hvis eksempelvis Niels Hansen vil være interesseret i at vide når Carlsberg B aktien har en kursværdi som er mellem 180 og 250 (og hvis man forestiller sig at denne får en kursværdi af 230 vil han få en besked via sms eller e-mail). Så kan disse informationer blive gemt i en tabel, hvor Niels' Cpr-nummer også er indsat. Hans data gemmes i en samlet profil sammen med hans mobilnummer eller e-mailadresse.

Efter at have foretaget denne abonnering vil alerten blive opdateret til værdien *j* på aktieposten, som indeholder Aktienavnet (*Carlsberg B*), for at angive at aktiesymbolet er markeret.

Når abonnementssystemet får strømmen af oplysninger fra kildefilen, skal det finde ud af om *Carlsberg B* aktien har fået ny værdi.



Figur 3.2.1: Skematisering af push-systemet

Kildefilerne som sendes fra finans-, bank- og nyhedsverdenen, kaldes henholdsvis Fond, Kerne System, Valuta og XXX. Som det ses, er der i midten af figuren placeret en stor firkant. Denne repræsenterer motoren i vores system, der har som opgave kun at udplukke de interessante informationer fra kildefilerne og bagefter at sende disse oplysninger til abonnenterne via sms og/eller e-mail. De små firkanter repræsenterer de indkommende og udgående data og måden hvorpå informationerne sendes til abonnenterne, altså e-mail og sms. Prikkerne angiver mulige veje i fremtiden hvor man kan benytte andre kommunikationsmidler, som for eksempel en TVSelector. Systemet benytter sig af databasen, hvorfra der udtrækkes/gemmes oplysninger om abonnementer med mere.

Hvis aktien har fået ny værdi tjekker systemet om det respektive aktiveringssymbol er markeret på aktiens post.

Såfremt dette er tilfældet, bliver informationen sendt til abonnenten (Niels), hvis informationen opfylder de opsatte krav (her kursværdi mellem 180 og 250).

Den samme fremgangsmåde vil gælde hvis man vil abonnere på andre services.

3.5 Systemgrænseflade eller *Push-system*

Med *push-systemet* menes et system der både behandler data fra kildefilen og videresender disse oplysninger med hensyntagen til de krav abonnenten stiller. Systemet omfatter motoren bag projektet, og er dermed den vigtigste del.

3.5.1 Baggrund for Push-systemet

For at forstå årsagen til at have et *push-system* skal man forstå hvordan en 'SAX-handel' foretages, hvordan aktiekurserne ændres og aktierne forhandles. Ved en 'SAX-handel' bliver handelen gennemført på børsen, når en anden kunde accepterer ordren. For at afgøre om handlen har fundet sted skal kunden selv undersøge sine ordrer eller depotbevægelser. Hvis en aktie bliver forhandlet, skal man ligeledes være ved telefonen eller i kontakt med sin bankrådgiver, eller man skal være tilstede på Børsen for at finde de rigtige priser, for enten at købe eller sælge.
Bedre vil det være hvis kunden ved hjælp af en besked til sin mobiltelefon eller en e-mail (elektronisk post) kan få beskeden kort efter handlen er gennemført, eller hvis aktien har den værdi kunden finder relevant.

3.5.2 Formål med Push-systemet

Idéen bagved *push-systemet* er aflevering af data som kunden på et vilkårligt andet tidspunkt har anmodet om via et såkaldt abonnementssystem. *Push-systemet* skal kunne køre kontinuerligt i baggrunden og levere/hente oplysninger til de pågældende abonnenter. *Push-systemet* skal også kunne håndtere afsending af de interessante data til kunderne. Dette kan foregå samtidig med andre hændelser og desuden skal *push-systemet* kunne håndtere mere end 20 transaktioner per sekund.
Ønskerne til systemets funktionalitet dækker flere forretningsområder, eksempelvis følgende:

- Push (Udplukning og afsending af data til abonnenterne) af alarm ved overskredne kursgrænseværdier.
- Push af alarm ved ændring af valutakurs.
- Push af alarm ved overskredne saldo-værdier.
- Push af nyheder.

*Push-systemet*s formål er (så) at indlæse strømmen af informationer som kommer fra pengeinstitutter eller aviser, det vil sige data fra kildefilerne. Disse informationer vil derefter blive behandlet på følgende måde:
De data som abonnenten er interesseret i at få vil være de eneste der vil blive udplukket.

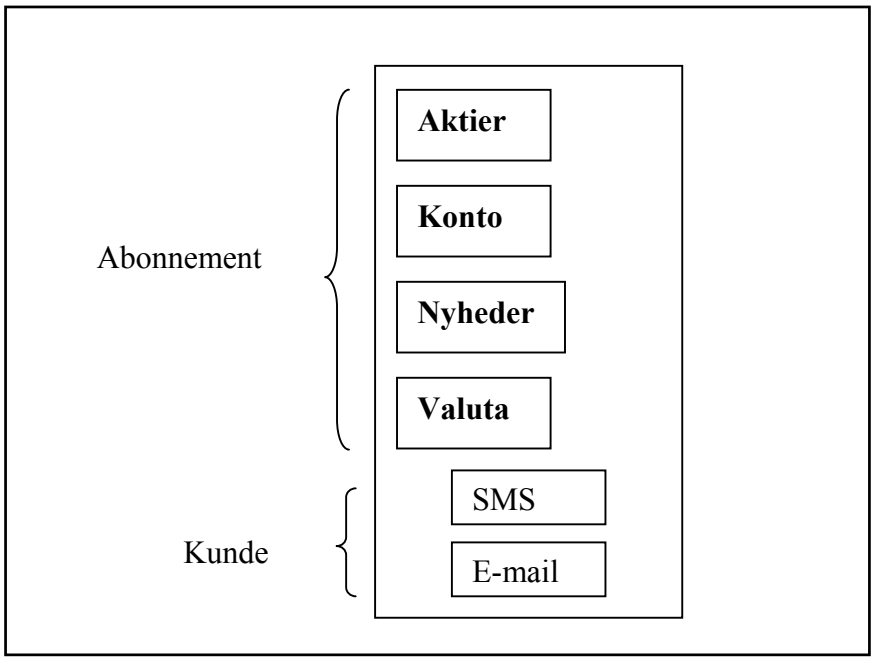
Først skal der dog laves et program der opdaterer *Alerterne*, på de enkelte poster, ud for abonnenternes ønsker. *Alerterne* aktiveres når en kunde tilmelder sig servicen.
Derefter kan abonnenten bestemme for hvilke informationer der skal sættes en *alert*. *Alerten* kan sættes, hvis man for eksempel er interesseret i at vide om en aktie har nået en bestemt værdi.
I Figur 3.6.1 beskrives den måde hvorpå abonnenter kan blive tilmeldt forskellige services og få de relevante informationer gennem disse to kommunikationsværktøjer: e-mail og sms. Herefter opdateres databasen øjeblikkeligt, og der viser sig et *j* eller *n*, henholdsvis for en tilmeldt eller ikke tilmeldt service.
Udplukningen af data til en abonnent kan gøres på forskellige måder:

- En metode kunne være at implementere et Java program der vil gennemgå databasen, for at undersøge den pågældende abonnents data og interesser, og derefter tjekke de respektive *alerter*. Derefter vil programmet udplukke de relevante informationer der har *alerter* på og videresende dem.
- En anden metode kunne være at indsætte informationer der har *alerter* på, fra databasen, i en liste. Denne kan blive implementeret i Java. Derfra kan de nyttige og relevante informationer udplukkes og tilsendes abonnenterne. Men med denne metode kan der opstå ulemper, da listen hele tiden skal opdateres, for eksempel hvis abonnenten tilmelder sig en ny aktie.

Forskellen på ovennævnte to metoder er at den første kan behandle data hurtigere end den anden, hvis der er mange informationer at behandle. Som nævnt kommer der konstant en stor mængde af transaktioner, og for hver af disse skal man tjekke om deres indhold har interesse. Dette er tidskrævende for eksempel når det gælder aktier, da det er almindelig kendt at der findes mange af disse, der er gemt i en liste, som automatisk får en stor længde. Men hvis disse er gemt i en database, vil søgningstiden blive formindsket.

3.6 Web-side

For at få abonnenter til servicen er det nødvendigt at have en brugervenlig grænseflade, hvor kunderne kan vælge imellem og tilmelde sig de forskellige services.
Vi overvejer at den første side, af denne web-side, kan vise alle forskellige services eller links til dem, logo og altså valgmulighederne systemet tilbyder. Herfra man kan vælge hvilke links man er interesseret i. Derefter sendes data fra abonnenterne til en tabel i databasen, hvor deres oplysninger gemmes.



Figur 3.6.1 Skematisering af Abonnementssystemets web-side.

I figur 3.6.1 er der fremvist en generalisering af den forventede web-side. Man kan se hvilke abonnementer man kan tilmelde sig: Aktier, Konto, Valuta, Nyheder. Og man skal angive hvilke kommunikations veje man vil have.
Denne web-side kan implementeres ved anvendelse af HTML, ASP værktøjer eller en anden teknologi som PHP eller XHTML.

4. Analyse af værktøjer

Til designet og implementeringen af dette projekt har det været nødvendigt at anvende nogle værktøjer. I udviklingen af databasen har vi overvejet at anvende MySQLversion 4.*, og Microsoft Access. Til udviklingen af web-siden overvejede vi at anvende ASP teknologien og HTML, samt JavaScript.
Desuden har vi behov for et Java-miljø der vil muliggøre implementeringen af Java programmerne. Java-programmerne skal give mulighed for at implementere *push-systemet*, og som tidligere nævnt, er *push-systemet* motoren i projektet. Ydermere er Java meget anvendt og det benyttes især i SDC-Udvikling.
For at kunne opbygge kildefilerne i XML format har vi behov for XML og eventuelt XML-relaterede applikationer.
Disse værktøjer skal kunne præstere at levere et effektivt og velfungerende system, og det er dette vi stræber efter i vores projekt.
En forklaring og beskrivelse samt vores vurdering af de foreslåede værktøjer, forefindes i det følgende. Vi vil fremhæve fordelene og ulemperne ved hvert værktøj vi beskæftiger os med.

4.1 MySQL kraftigt værktøj til implementering af databasen

MySQL baseres på Structured Query Language (SQL). SQL er det mest anvendte sprog hvis en database skal have data processeret, ændret og tilføjet. Hvis man samtidig vil opbygge web-baserede applikationer, er MySQL specielt egnet. MySQL kommunikerer også med programmeringssprog såsom C, C++, PMP, Perl og Java[1].

4.1.1 Fordele og ulemper ved MySQL

Fordele:

- Størrelsen af tabellen i databasen kan være den man ønsker.
- MySQL er specielt hurtig til læsninger, og langsommere (relativt set i forhold til andre databaser) til skrivninger (nye rækker samt opdateringer). Dette forhold er blandt andet det, der har gjort den meget brugt til web-løsninger, hvor man ofte har flere hundrede læsninger for hver skrivning. MySQL kan behandle mange data samtidigt og dette sker tilstrækkeligt hurtigt.
- MySQL håndterer op mod 100 database connections (skrivninger) på en enkelt maskine (en cpu, men en masse ram), og det bliver udført problemfrit.

Ulemper:

- Brugervenlighed er et minus ved MySQL. Det vil sige at alle kommandoer skal

[1] Matthew og Stones: [2]: 1

gives i interaktiv form, og man skal kende kommandoerne i forvejen.
- Programmet skal oprette sit eget miljø for at kunne få fuldt udbytte af faciliteterne.
- Det er meget svært at finde fejl ved indtastning af kommandoer, da programmet aldrig forklarer fejlmeldingen.

4.1.2 MySQL og InnoDB type tabeller

Der findes mange typer af tabeller i MySQL og én af disse kaldes *InnoDB*[2].

Tabellerne af type *InnoDB* er de eneste af MySQLs tabeltyper, der understøtter *FOREIGN KEY CONSTRAINT*.

Disse tabeller kan være af den størrelse man ønsker, selv i systemer, hvor deres operative system kun tillader filer med størrelser mindre end 2 GB.

InnoDB'en giver MySQL sikkerhed til at håndtere tabeller, det vil sige den umuliggør en gentagelse af nøglerne i tabellerne, hvis man ikke ønsker det.

4.2 MS Access

Microsoft Access' (MA) databasedesign-værktøj anvender man til oprettelse af små databasestrukturer for diverse applikationer og med mange formål. MA databasen er et optimalt system for databasedesign i specifikke miljøer. I det følgende gennemgås nogle af fordelene og ulemperne ved dette værktøj.

Fordele:

- **Meget nemt at arbejde med.**
  Det kan bruges af næsten alle, selv om man ingen erfaring har med databaser eller SQL. De forskellige forespørgsler er meget simple opgaver, især hvis man bruger værktøjslinier, wizards og grafiske interfaces leveret med Microsoft Access.
- **Data Flytning:**
  Access er et af de nemmeste databasedesign-systemer, når man taler om at dele og flytte data. Det unikke filsystem gør up- og download'ingen meget let, og ligeledes kopieringen og delingen af databasen til andre computere.
  Databasen kan nemt kopieres til at blive delt med andre afdelinger i for eksempel et firma, i stedet for at binde netværket med andre netværksforespørgsler.
  Således er det nemt og effektivt at udveksle prototyper af databasedesign uden at man skal installere en masse andet software.

[2] *InnoDB* begrebet står for en komplet database placeret i "enden" af MySQL og den accepterer fremmednøglens begrænsninger (FOREIGN KEY CONSTRAINT). For at benytte sig tabeller af type *InnoDB* i MySQL skal man specificere dette i filen *my.cnf* eller *my.ini*.(se Fig. 12.2). Man skal tilføje følgende linje i filen: *innodb_data_file_path=ibdata:30M*. For yderligere detaljer henvises til www.innodb.com eller/og www.mysql.com

- **Enkel backup/arkivering:**
  MA databasen har en meget enkel måde at lave en 'backup' på. I stedet for at gemme store mængder af filer er der kun en fil der skal gemmes. Databasens fil kopieres til det sted man vil gemme den og derfra er det nemt at opdatere, ændre eller gøre noget videre med databasen.

Ulemper:

- **Begrænset til små databaser:**
  MA er et begrænset databasesystem. Det er konstrueret for at designe og styre, og er meget effektivt når det gælder mindre antal af databaseregistre. Det manglende grundlag for at kunne håndtere en stor datamængde er på en måde kompenseret med Access' mange 'features'; hurtighed og flytbarhed. Udførelsen (performance) vil begynde at gå dårligt, i det øjeblik man når et antal af 25,000 registre i alt. Dette gælder også hvis man tilføjer flere registre og/eller flere parallelle brugere.
- **Begrænset til at lave et bestemt antal af transaktioner:**
  Microsoft Access er ikke et godt valg for høj produktion af ind- og udtagning af data i datamiljøer.
- **Begrænset antal samtidige databasebrugere:**
  MA kører bedst som et single-bruger system eller et lille multi-bruger system.50 eller flere kan bruge Microsoft Access samtidigt uden problemer, men det anbefales ikke. Som tidligere nævnt vil dette begrænse Access' udførelse/hastighed, hvor der er behov for en højere ind- og udtagning af data.

4.3 ASP teknologi til brugergrænseflader

ASP er en forkortelse for Active Server Pages. ASP er en teknologi som tillader dynamisk konstruktion af HTML sider, der skal sendes til en browser.

Med andre ord kan man skrive en mængde af instruktioner i ASP, som kan bruges til at generere HTML umiddelbart efter web-sidens forespørgsel er blevet sendt fra klienten (til serveren) og før HTML'en bliver sendt til browseren. Det vil sige at ASP er en teknologi som bruges til at bygge dynamiske og interaktive websider. (Dynamiske websider dækker over sider som arbejder interaktivt med brugerne.).

ASP er ikke et sprog (som Pascal eller C++)- men det bruger nogle af de eksisterende *scriptsprog* som VBScript eller JavaScript. Men man kan heller ikke påstå at det er en applikation (som for eksempel FrontPage og Word er). Derfor bruger man i stedet et ord som 'teknologi' til at definere ASP[3].

Fordele ved ASP

- Tillader kørsel af programmeringssprog, som ikke er støttet af ens browser.

[3] Sussman, Francis, Ulmann, Llibre, Kauffman, Duckett,Buser: [10] :13

- Muliggør programmering af dynamiske web-anvendelser uafhængigt af browsertypen, uden at omdirigere appletter, Dynamisk HTML, AktiveX kontroller til klientens side, da de er browser-specifikke.
- Kan levere klienten (browseren) data, som ikke forekommer eller findes hos klienten.
- Gør loadingens tid hurtigere end med klientsidens dynamisk applikationer (Dynamisk HTML, ActiveX, og andre), da man kun downloader en enkelt HTML side ad gangen.
- Forbedrer sikkerheden, da man kan skrive koder som aldrig kan ses fra browserens side.

Ulemper ved ASP

- ASP sider øger arbejdsbyrden på serveren. Hvis siden bliver populær er man nødt til at investere i mere hardware. Men dette må gøres på en hvilken som helst server-side applikation.

4.4 Java applikationer: JSDK, J2EE

For at benytte os af et Java-miljø er vi nødt til at beskrive Java-komponenterne.

4.4.1 JSDK

'Java Standard Development Kit' (JSDK) og 'Java 2 Standard Edition' (J2SE), er applikationer med de samme komponenter og det inkluderer: Javas API[4], JRE[5] og andre af SUNs[6] definerede faciliteter. Javas API er en samling af klasser som anvendes til at generere basale programmer i dette sprog. Disse klasser har samme funktionalitet som funktioner/klasser anvendt i andre programmeringssprog, for eksempel C, C++, osv.

Ved brug af disse API'er kan udvikles alle programmer, interfaces og andre elementer i Java, og herfra kan man definere andre specifikke klasser, der vil kunne blive anvendt for et program eller et produkt.

Fordele:

- Platform-uafhængigt, små filer for distribueringen af applikationer.
- God integration med de fleste web-servere.
- Objektorienteret.
- Muliggør brug af indlejret Java (Embedded Java).
- Velkendt for os.
- Anvendes af SDC og andre firmaer.

[4] Application Programming Interface.
[5] Java Runtime Environment.
[6] SUN Microsystems: Software leverandør på linie med Microsoft.

Ulemper:

- Har en noget højere hukommelsesanvendelse end for eksempel C/C++ baserede applikationer, og dermed kan applikationen blive lidt langsommere.
- Desuden bygges applikationer på runtimer oversættelse fra Java byte kode til maskinkode.

4.4.2 J2EE

Java 2 Platform Enterprise Edition, J2EE, er en af de mest nyttige platforme til et projekt[7] som dette. J2SDKEE er en platform der hovedsageligt distribuerer applikationer til et Server/Klient miljø, Java baserede systemer.

SUNs J2SDKEE er et værktøj til at programmere applikationer som bruger J2EE standarden. J2SDKEE er ikke en komplet J2EE platform og det er SDK heller ikke, da den for eksempel ikke indeholder XML-parser klasserne.

J2EE er en gruppe af specifikationer (drivere) som er designet af SUN og som tillader implementering af erhvervsagtige applikationer.

Denne platform har de rigtige 'drivere', der giver mulighed for at kommunikere og manipulere data i databasen ved hjælp af Java klasser, som for eksempel følgende drivere[8]:

- Java Database Connectivity (JDBC) som skaber forbindelse til databasen,
- Java Message Service (JMS) som hjælper med at sende og modtage data asynkroniseret
- JavaMail støtter Internet protokoller såsom IMAP4, POP3 og SMTP. Men JavaMail er lidt langsommere og mindre pålidelig end JMS.

Fordele:

- SUNs J2SDKEE er driftsikkert og kan øge udviklingshastigheden da der er mulighed for at genbruge allerede udviklede funktionaliteter.
- Det indeholder en køretidsinfrastruktur for værtapplikationer, hvilket vil sige at det benyttes som grund til flere andre applikationer.
- Er meget bredt anvendt blandt firmaer i hele verden, da det tillader Java udvidelse i form af API'er til dannelse af andre applikationer.
- Muliggør adgang til databasen (JDBC), samt anvendelse af distribuerede mapper (JNDI).

[7] J2EE er anvendt af SDC-Udvikling og mange andre firmaer.
[8] Nakhimovsky, Myers, Wilcox, Zeiger, Diamond, Griffin, Avedal, Holden, Tyagi, Damme, Allamaraju, Longshaw, O'Connor, Browet, Johnson, Karsiens, Kim, Huitzen: [8]: 25

- SUNs J2SDKEE har også adgang til distribuerede objekter og metoder (RMI/CORBA), elektroniske postfunktioner (JMS, JavaMail) og webbaserede applikationer (JSP og Servlets).

Ulemper:

- Værktøjet har ikke endnu mulighed for Parsing af XML dokumenter, hvilket vil sige at APIer som JAXP (s 266) ikke er inkluderet i J2EE-applikationen. Men XML er internt anvendt i J2EE[9].

4.5 XML

XML står for Extended Markup Language og er ligesom HTML et dokumentsprog baseret på tags[10]. Forskellen på XML og HTML er, at XML interesserer sig for selve indholdet i det pågældende dokument, hvor HTML mere er baseret på layout og præsentation.
XML´s formål er således at strukturere, gemme og videresende information.

XML teknologien er nu om dage meget udbredt for implementering af databaser på Internettet. Anvendelse af forskellige værktøjer for gennemførsel af forespørgsler er i høj grad ligeså effektivt som SQL statements.

XML indeholder ingen færdige elementer som HTML gør. Det er udvikleren/brugeren selv der bestemmer, hvad de forskellige tags betyder. Vi kan oprette så mange og forskellige tags vi vil, så længe vi overholder sprogets generelle syntaks. At overholde sprogets syntaks bliver også kaldt for "well formed".
Tags i XML vil typisk hedde noget som <*Aktier*> eller <*Kurs*> eller noget andet, der beskriver indholdet af det der står mellem et start- og et sluttag.

4.5.1 Grunden til at anvende XML

De fleste har oplevet Internettet og dets enorme vækst igennem en ganske kort årrække. I lang tid har HTML været det mest anvendte sprog til at viderebringe information på nettet.

Ved anvendelse af XML kan man benytte såkaldte DTD (DataType Definitions) eller XML Schema. Begge metoder giver mulighed for at definere en samling af ens egne tags som kan tilpasses et bestemt formål. XML Schema er en ny standard som har flere muligheder end DTD, og det anvender XML til at forklare tags'ene.

HTML indeholder et begrænset antal forhåndsdefinerede tags som man kan vælge imellem. XML derimod indeholder ikke forhåndsdefinerede tags, men blot et regelsæt for hvordan tags skal defineres og anvendes. Disse regler er derimod absolutte og kan ikke fravælges (I HTML kan man tage smutveje som for eksempel at lade være med at lukke

[9] Nakhimovsky, Myers, Wilcox, Zeiger, Diamond, Griffin, Avedal, Holden, Tyagi, Damme, Allamaraju, Longshaw, O'Connor, Browet, Johnson, Karsiens, Kim, Huitzen: [8]: 27
[10] Opbygges med to brackets ("<" og ">") og en titel der beskriver indholdet af den information, der står mellem en start- og sluttag. Et eksempel på tags: <tag> x </tag>.

alle tags). XMLs mulighed for at definere egne rammer giver langt flere anvendelsesmuligheder. XML kan anvendes til overførsel af alle tænkelige typer data.

Fordele:

- HTML er ganske udmærket til statisk information, såsom hjemmesider hvor indholdet ikke ændrer sig.
- Det er nemmere at validere data, fordi vi med XML har mulighed for at kontrollere, at de elementer der indgår i et dokument er gyldige. Vi kan indføre denne kontrol både før vi sender data, og når vi modtager data.
- Brugerne bestemmer selv, hvad de enkelte tags betyder. Man er ikke begrænset af et antal færdige tags ligesom i HTML.
- XML er blevet den globale standard til transport og udveksling af data via internettet. Alle de stor virksomheder i software industrien bakker den nye standard op. Det betyder blandt andet at XML dokumenter er uafhængig af platforme, systemer og applikationer. Når vi modtager XML data kan vi dybest set være ligeglade med om de oprindelige data kommer fra en Oracle database, MS Access eller en flad fil.

Ulemper:

- Denne statiske information kan derimod give problemer, hvis man skal vise indholdet af en database, der konstant ændrer sig. Det kan lade sig gøre, men det er nødvendigt at bruge ekstra software og muligvis også et par hacks hist og her for at konvertere databasen til HTML.

4.5.2 Parsing XML

Kildefilerne bliver bygget som et XML dokument. Dermed er der behov for en XML Parser. Vi analyserer her to XML parsere:

- **DOM** Parser læser XML filen og giver mulighed for at hente, med en enkel kommando, informationer fra XML-træet. Men denne Parser anvender stor hukommelsesplads og desuden bliver processeringen langsommere, da DOM's serializer går alle knuder igennem samtidig med at opbygge nogle af knuderne[11].
- **SAX** Parseren giver ikke mulighed for at hente informationer fra XML filen direkte med en kommando ligesom DOM gør. Men denne Parser er hurtigere og anvender meget mindre hukommelsesplads end DOM Parseren[12].

[11] Maruyama,Tamura, Uramoto,Murata, Clark, Nakamura, Neyama, Kosaka, Hada: [9]: 120
[12] Samme: 128

4.6 Valg af værktøj

På grund af ovennævnte analyse af værktøjer og gennemgangen af fordele og ulemper ved disse, har vi i hver del af projektet anvendt værktøjer. Vi begrunder vores valg i det følgende:

For at konstruere databasen har vi anvendt MySQL. Men til at kunne teste funktionaliteten af vores system har vi ment at det var fordelagtigt at implementere databasen i Access på grund af dettes brugervenlighed.

For at implementere vores motor har vi anvendt JSDK og J2SDKEE, da de indeholder de nødvendige klasser og drivere til formålet. Til konstruktionen af kildefilerne har vi valgt at anvende XML, på grund af dettes fleksibilitet.

Endelig har vi til opbyggelsen af vores brugergrænseflade anvendt ASP-, HTML- og JavaScript-teknologierne.

Desuden har vi valgt at anvende XML teknologien til dannelsen af kildefilerne, og til parsingen af disse har vi valgt at benytte SAX-Parseren.

5. Strukturering af Databasen

Da vi skal behandle en stor mængde data, er det nødvendigt at have en database. En vigtig del af vores projekt er derfor strukturering af databasen.

Databasen samler oplysninger systematisk. På den måde kan oplysningerne hurtigt genfindes og behandles. Databasen spiller en vigtig rolle for at servicere abonnenterne. Vi har valgt at kalde vores database for *AbService* (Abonnementsservice) og fra nu af refereres til denne som *AbService*.

AbService indeholder de informationer som websiden bruger til at servicere kunderne, samt abonnenternes profil/informationer.

For at designe og implementere databasen skal man igennem følgende fem trin[13]:

1. Man skal identificere alle entiteter som databasen skal indeholde.
2. Man skal finde hvilke relationer der er mellem entiteterne. Til dette formål benyttes E/R diagrammet.
3. Der udarbejdes en egenskabstabel, som viser entiteter, relationer, attributter og deres værdimængder.
4. For at teste om databasen er fornuftigt bygget anvendes normaliseringsmetoden.
5. Til at implementere databasen bruges SQL.

I de næste afsnit gennemgår vi opbygningen af tabellerne ved hjælp af de ovennævnte trin.

5.1 Identifikation af entiteter

De forskellige entiteter vi har i databasen er blevet valgt i forhold til hvordan de på et senere tidspunkt skulle blive behandlet. Vi skal behandle en stor mængde data ca. 10000 abonnenter, samt en mængde aktier som er væsentlig stor, og tilsvarende for konti, nyheder og valuta.

Da de valgte entiteter bliver oversat til tabeller, har vi valgt at referere til entiteterne som tabeller.
Som tidligere nævnt har vi en konstant strøm af informationer som kommer fra kilderne og derfor skal vi tilføje en attribut (*Alert*) til tabellerne for at hjælpe med at sortere de informationer kilderne indeholder fra dem vi ikke er interesseret i. Modsat skal behandles dem som har interesse fra kundernes side.
Som bekendt giver vi vores abonnenter mulighed for at tilmelde sig 4 forskellige tjenester vi tilbyder.

[13] Widow: [4]: 137-166

For en bedre forståelse af hvorfor vi har valgt at strukturere databasen på den måde som det er gjort, går vi igennem nogle eksempler i det følgende.

Eksempelvis har vi en kunde som er interesseret i at blive informeret via SMS, når vedkommendes konto går under et bestemt beløb. For at tilgodese dette skal vi først og fremmest have en tabel hvori alle forskellige kontonumre med de respektive ejeres cpr-numre er gemt. Derfor opretter vi en entitet, nærmere sagt en tabel som hedder *Konto*, hvor de vigtigste attributter vil være kontonummer, *Kontonr*, og Cpr-nummer, *CPR_NR*.

Endvidere repræsenterer disse to attributter også nøglen i tabellen fordi de gør indholdet i denne entydig. Desuden har vi også en attribut kaldet *Alert*. Denne benyttes kun til at gøre filtreringen af kildeinformationerne, i indgangen af vores system, mere hurtig. *Alerten* har en feltstørrelse af 1 tegn, fordi værdien af den skal kun være et *n*, der står for nej, og et *j* for ja. Når der i alertfeltet er et *j*, betyder det at alle informationer, som er tilknyttet dette kontonummer, har interesse fra abonnentens side. Derfor skal vi være opmærksomme, hvis der i kildefilen er informationer relateret til dette bestemte kontonummer, som har alert sat til *j*. Hvis *alerten* derimod er sat til et *n*, smider vi informationen væk uden at tage den i betragtning.

Som ekstra attribut har vi i tabellen *Konto* også en kontotype, *Typenavn*. Med denne vil det være muligt at se hvilke kontotyper der kan bruges til at udføre nogle handler på børsen. For eksempel, hvis en aktie har den ønskede kurs, kan man give besked til sin bank, at man vil sælge eller købe den. Dette er kun muligt på nogle bestemte kontotyper. Hvis man har en børneopsparing, kan man ikke gøre så meget med de penge man har på denne.

Kontotypen bliver i vores system ikke brugt til noget, men er tilføjet som en ekstra mulighed. Vi mener at denne mulighed vil blive anvendt hyppigere i fremtiden.

For at definere de begrænsninger kunden vil have til sin egen konto og på hvilken måde vedkommende vil informeres om de eventuelle ændringer på denne, tilføjer vi en ny tabel. Denne tabel skal profilere de specifikke slags abonnementer kunden kan have og denne kaldes *Abt_Konto*, for at minde om at den repræsenterer de forskellige abonnementer med ensidige konti. Her findes attributterne *CPR_NR*, som identificerer personen med interesse i oplysningerne. Mobilnummer, *Mobilnr*, og e-mail, *Email*, er de kommunikationsveje der bruges til at sende informationerne, og kunderne vælger selv hvilken. *Kontonr* står for det bestemte kontonummer abonnenten vil informeres over.

Slutteligt kan vi sige at der står en attribut, *Minval*, som repræsenterer en restriktion som vil blive sat op på den bestemte information. For eksempel en minimum værdi (kunden vælger selv) kontoens saldo skal være under for at kunden kan blive informeret om saldoen.

Således bliver kildeinformationerne også sorteret i forhold til den sidste information.

Sidst, men ikke mindst, skal vi have en tabel, hvori der er gemt abonnentens profil. Tabellen bliver døbt *Abonnent*. Her gemmer vi abonnentens fornavn, *Fornavn*, efternavn, *Efternavn*, cpr-nummer., *CPR_NR*, samt adgangskode, *Password*. Den sidste attribut bliver brugt således at abonnenten kan komme ind systemet og har den sikkerhed, at han/hun er den eneste som har adgang til sine oplysninger.

Med samme tankegang har vi også designet resten af tabellerne, som skal indeholde informationer relateret til valuta, nyheder og aktier. Der er også to ekstra tabeller som hedder *Gruppering* og *EmneA*. I tilfælde af at kunderne skal tilmelde sig bestemte aktier, skal de have muligheden for at gå alle aktienavne igennem. Derfor opdeler vi tabellerne efter gruppering eller efter det emne de tilhører. På den måde er det nemmere at finde rundt mellem de flere hundrede aktienavne, man kan få.

Vi har i *Nyheder* tabellen bestemt at nyhederne kun kan sendes via e-mail, da den kan indeholde en stor besked. Derfor findes attributten *Mobilnr* i *Abt_Nyheder* ikke. I Figur 5.3.1 findes en grafisk repræsentation af alle tabeller.

5.2 Design af E/R-Diagrammet

Et E/R-diagram er en grafisk illustration af den kommende databases indhold og samspil eller forbindelser mellem de forskellige entiteter. I diagrammet opstilles entiteter som firkanter og relationer som rhomber. Relationer og valg af relationstyper mellem en databases entiteter beskriver hvordan databasedesigneren opfatter forbindelser eller sammenhænge mellem entiteterne eller bestanddelene i en kommende database.

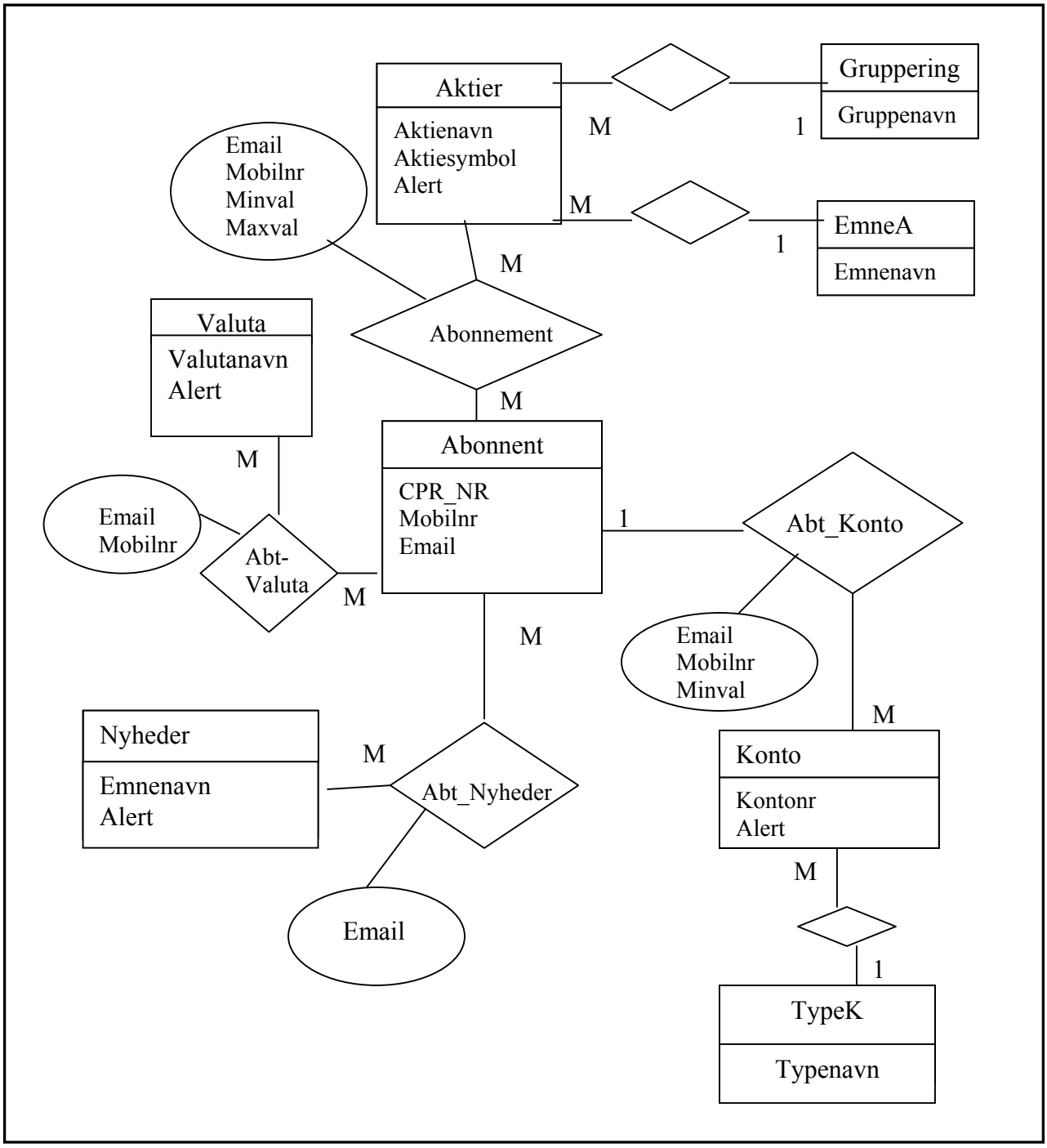
Når der er en relation mellem to entiteter, sondres mellem tre forskellige relationstyper, som har hver deres betegnelse der siger noget om hvordan forbindelsen eller sammenhængen mellem entiteterne er, eller hvordan de har noget med hinanden at gøre. Relationstypen kaldes også for kardinalitet, fordi typen siger noget om antallet af relaterede poster i to entiteter, og derigennem hvordan information i den ene entitet/tabel forholder sig til information i den anden entitet/tabel.

Vi har 3 forskellige relationstyper mellem to entiteter:

- En-til-en relation.
- En-til-mange-relation.
- Mange-til-mange relation.

For at kunne se E/R-diagrammets relationskardinaliteter sættes kardinaliteterne på linjer som forbinder de to tabeller med hinanden. For eksempel bliver en ”mange-til-mange” relation repræsenteret med et *M* på den ene side af rhomben og et *M* fra den anden side

I *AbService* har vi mange relationer imellem tabellerne. I figur 5.2.1 kan ses en repræsentation af *AbServices* E/R diagram. Dette vil blive gennemgået i det følgende



Figur 5.2.1: ***AbServices*** **E/R-Diagram.**

For en beskrivelse af AbService ER/diagram begynder vi med Aktie-tabellen. Man kan se i figuren 5.2.1 at *Aktie*-tabellen er relateret til tre andre tabeller: *Gruppering*, *EmneA* og *Abonnent*.

Relationen mellem dem er repræsenteret med en rhombe, mellem *Aktier* og *Gruppering* og mellem *Aktier*, og *EmneA*, er der den samme type relation som er en 'en-til-mange' relation. Det vil sige at en gruppe aktier kan have mange aktier, men én aktie er tilknyttet én enkelt gruppe. Det samme gælder for *EmneA*.

Imidlertid er relationen mellem *Aktier* og *Abonnent* en 'mange-til-mange' relation. Det vil sige at en kunde kan abonnere på forskellige aktier, og en aktie kan have forskellige abonnenter. En anden ting som bør bemærkes i denne relation er, at vi har knyttet flere oplysninger/felter til relationen, som *Mobilnr*, *Email*, *Minval* og *Maxval*. Disse profiler karakteriserer den type abonnement vores kunde har valgt. *Minval* og *Maxval* er den intervalinteresse, hvor abonnenterne informeres over de ændringer i den bestemte aktie der er valgt.

Fra diagrammet kan også ses de forskellige relationer tabellen *Abonnent* indgår i. Det er *Abonnent-Aktier, Abonnent-Konto*, *Abonnent-Nyheder* og *Abonnent-Valuta*. Dog afspejler relationen mellem *Abonnent-Konto*, ikke helt virkeligheden. Vi har beskrevet relationen som en 'en-til-mange' relation, hvilket betyder at en abonnent kan have flere konti, men en konto tilhører kun én abonnent. Dette sker ikke i virkeligheden, idet flere personer, eksempelvis et ægtepar, kan have fælles konto.

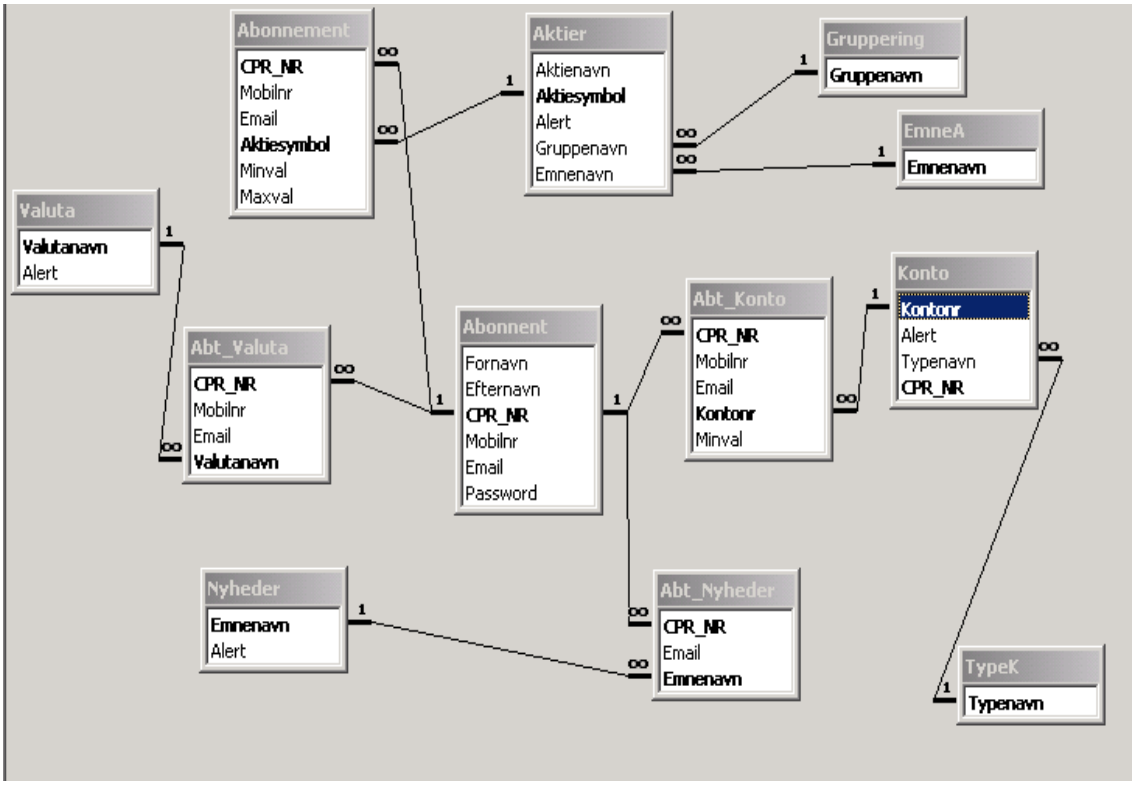
(Da vi ikke har den virkelige konto-tabel fra banken, har vi bestemt ikke at tage hensyn til denne detalje. Derfor er relationen ”en-til-mange” og dermed skal vi ikke have en ekstra tabel i vores database.)

Sammenhængen mellem de enkelte entiteter, og deres respektive attributter, samt relationernes kardinalitet (om det er ”en-til-mange”, ”mange-til-mange” eller ”en-til-en” relation) fremgår af ER-diagrammet, se Fig.5.2.1

I processen med struktureringen af tabellerne, omsætter vi hver 'mange-til-mange' relationer til 'en-til-mange' relationer, da databasen ikke accepterer 'mange-til-mange' relationer direkte. Omsætningen af det sidste medfører at der skal laves en *link-tabel*, der forbinder primærnøglefelterne fra de to tabeller ved hjælp af én til mange relationer, således at en post i link-tabellen svarer til mange poster i hver af de to forbundne tabeller. Link tabeller er et hjælperedskab og behøver ikke have deres egen primærnøgle. (Se figur 5.3.1).

5.2.1 Diagram over den referentielle integritet

Sammenhængen og forholdet imellem primærnøgle og fremmednøgle fremgår af tabel Fig.5.2.1.1, og her vises den referentielle integritet.



Figur 5.2.1.1Omsætning af ER diagram til endelig tabeller i *AbService*.

5.3 Tabellernes egenskaber

Som bekendt er dokumentation meget vigtigt ved al edb-udvikling, så man et år efter kan huske hvilke tanker man havde, da databasen blev designet, men også for at en anden person kan finde ud af, hvorfor databasen ser ud som den gør!

Dokumentationen for alle felter i en database kaldes samlet en *egenskabstabel*.

De entiteter, relationer samt attributter som skal anvendes, fremgår af nedenstående egenskabstabel. Endvidere forefindes i det kommende en kort forklaring til tabellerne.

Aktier Indeholder aktiernes oplysninger, det vil sige aktienavn *Aktienavn*, aktiesymbol *Aktiesymbol*, gruppen aktier hører til *Gruppenavn*, eventuelt en klassificering af aktier emnemæssigt, *Emnenavn*, samt en alert, *Alert*.

Vi kunne nøjes med kun at bruge aktienavn eller aktiesymbol for at genkende aktien. Fra abonnentens side er det vigtigt at have et aktienavn for at genkende aktien, men fra systemets side er der brug for en unik betegnelse, og det er aktiesymbolet. Dette vil gøre det lettere at orientere begge sider. *Alerten* her vil fortælle at der er en abonnent der vil have information om den pågældende aktie.

Gruppering Indeholder navn, *Gruppenavn*, hvilket vil sige gruppen aktierne tilhører.

Således kan abonnenten nemmere genkende en aktie ud fra dens tilknytning til en bestemt gruppe.

EmneA Deler aktierne i emnerne, *Emnenavn*. Begrundelsen er den samme som i *Gruppering*, men for aktieemne i stedet for *Gruppenavn*. Det er almindeligt at aktier i aviserne er ordnet efter emne eller grupper.

Abonnent Er en tabel som indeholder kundernes data såsom navn, *Fornavn*, *Efternavn*, telefonnummer, *Mobilnr*, e-mail, *Email*, Cpr-nummer, *CPR_NR*, samt kundens password, *Password*. Her vil alle abonnenter, der har tilmeldt sig en service, blive registreret.

Konto Indbefatter kontonummer, *Kontonr* og en alert *Alert* for at sætte kontoen i forbindelse med servicen. Tabellen har også kontotype, *Typenavn*, og kundens Cpr-nummer, *CPR_NR*.

Kontotabellen er en tabel for sig, da data normalt kommer via en direkte forbindelse med bankverdenen. Men i vores tilfælde har vi selv bygget den op. (Kontooplysningerne har vi ikke haft adgang til)

TypeK Typerne af konti, *Typenavn*. Denne tabel vil være nyttig, hvis der som førnævnt er behov for at bestemme hvilken kontotype der er bedst egnet til en bestemt service.

Nyheder Indeholder typer af nyheder, *Emnenavn* og en alert *Alert.*

Valuta Tabel rummer navnet, *Valutanavn*, på de forskellige valutaer der handles med, samt en alert *Alert*.

Abonnement er tabellen der repræsenterer relationen mellem tabellerne *Aktier* og *Abonnent*. Denne tabel indeholder alle de oplysninger om abonnementerne, aktiernes maksimum/minimum værdi, *Max*- og *Minval*, samt abonnentens Cpr-nummer *CPR_NR*, mobilnummer, *Mobilnr*, og e-mail, *Email*, hvortil oplysningerne sendes.

Abt_Valuta En tabel som repræsenterer relationen mellem *Abonnent* og *Valuta,* samt abonnentens Cpr-nummer *CPR_NR,* mobilnummer, *Mobilnr*, og e-mail *Email*, hvor oplysningerne vil blive sendt til.

Abt_Nyheder Indeholder abonnentens e-mail, *Email*, Cpr-nummer, *CPR_NR,* samt nyhedernes emne, *Emnenavn*. Tabellen er en relation mellem *Nyheder* og *Abonnent* tabeller.

Abt_Konto Indeholder abonnentens Cpr-nummer, *CPR_NR*, mobilnummer, *Mobilnr*, e-mail, *Email*, kontonummer, *Kontonr*, samt den værdi, *Minval*, der sættes som minimumgrænsen på den konto abonnenten har behov for at få besked om. Denne tabel er en relation mellem *Abonnent* og *Konto* tabeller

Tabellerne og deres attributter, værdimængde samt bemærkninger findes i fig. 5.3.1 og fig. 5.3.2.

Entiteter	Attributter	Værdimængde	Bemærkninger	Eksempler
Aktier	Aktienavn	Varchar(20)		CarlsberB
	Aktiesymbol	Varchar (20)	Primærnøgle	CARCb.Cob
	Alert	Char (1)		J
	Gruppenavn	Varchar (20)	Fremmednøgle	Danske aktier
	Emnenavn	Varchar (20)	Fremmednøgle	KFX
Gruppering	Gruppenavn	Varchar (20)	Primærnøgle	Europæisk
EmneA	Emnenavn	Varchar (20)	Primærnøgle	DAX30
Abonnent	Fornavn	Varchar (20)		Jens
	Efternavn	Varchar (20)		Jensen
	CPR_NR	Varchar (20)	Primærnøgle	301264-2337
	Mobilnr	Varchar (20)		26792589
	Email	Varchar (20)		jens@hotmail.com
	Password	Varchar (20)		jens
Valuta	Valutanavn	Varchar (20)	Primærnøgle	Euro
	Alert	Char (1)		N
Nyheder	Emnenavn	Varchar (20)	Primærnøgle	Blomberg
	Alert	Char (1)		J
Konto	Kontonr	Varchar (20)	Primærnøgle	12345678M
	Alert	Char (1)		J
	Typenavn	Varchar (20)		Mastercard
	CPR_NR	Varchar (20)		301264-2337
TypeK	Typenavn	Varchar (20)	Primærnøgle	Mastercard

Figur 5.3.1 Egenskabstabel del 1.

Tabellerne som opstår, efter omsætning af 'mange-til-mange' og 'mange-til-en' relationer til tabeller, refereres og beskrives i fig. 5.3.2.

Entiteter	Attribut	Værdimængde	Bemærkninger	Eksempler
Abonnement	CPR_NR	Varchar (20)	Primærnøgle	301264-2337
	Mobilnr	Varchar (20)		
	Email	Varchar (20)		jens@hotmail.com
	Aktiesymbol	Varchar (20)	Primærnøgle	ND
	Minval	Real		112
	Maxval	Real		150
Abt_Valuta	CPR_NR	Varchar (20)	Primærnøgle	301264-2337
	Mobilnr	Varchar (20)		26792599
	Email	Varchar (20)		jens@hotmail.com
	Valutanavn	Varchar (20)	Primærnøgle	Euro
Abt_Konto	CPR_NR	Varchar (20)	Primærnøgle	250167-2324
	Mobilnr	Varchar (20)		26367712
	Email	Varchar (20)		oa@mail.dk
	Kontonr	Varchar (20)	Primærnøgle	1111111111D
	Minval	Real		1000
Abt_Nyheder	CPR_NR	Varchar (20)	Primærnøgle	250167-2324
	Email	Varchar (20)		oa@mail.dk
	Emnenavn	Varchar (20)	Primærnøgle	Blomberg

Figur 5.3.2 Egenskabstabel del 2.

5.4 Normalisering

Ved normalisering skal hver entitet igennem en "kvalitetskontrol", hvor der ses nærmere på oplysningernes, eller rettere felternes indbyrdes forhold, afhængighed eller samspil. Det vil sige at der skal undersøges om et felt er afhængigt af et andet, eller om et felt bestemmer et andet felt indenfor en enkelt entitet. De indbyrdes afhængigheder og samspil blandt oplysningerne/felterne i en entitet er baseret på regler i den verden, hvor databasen skal fungere. Reglerne svarer til indholdsrestriktioner.

Kort fortalt handler normalisering om at effektivisere databasen:

- Undgå redundans i data, det vil sige at samme data er lagret forskellige steder i databasen.
- Undgå utilsigtede huller i databasen.
- Sikre sig mod utilsigtede "forældreløse" data.

Normalisering foretages ofte på et repræsentativt indhold i en tabel fra E/R diagrammet. Der eksisterer forskellige metoder til at normalisere en database. Metoden vi har valgt er den der går igennem de tre normalformer (NF) for alle tabeller i databasen, da denne er effektiv og metodisk. Med denne metode finder man de felter i tabellen, som bryder den tjekkede normalform, og udskiller dem i en ny tabel.

1NF: En tabel er på første normalform, når der ikke er repeterende grupper i tabellen. Hvis en tabel har en gentagende gruppe af felter indenfor samme primærnøgle, så skal denne gruppe af felter fjernes fra tabellen og lægges over i en ny tabel sammen med en kopi af primærnøglen.

Hvis der ikke findes nogen gentagende gruppe af felter i en tabel, opfylder tabellen allerede første normalform. I praksis skal man finde de kolonner, der ikke er entydigt identificeret af primærnøglen (repeterende grupper), og udskille disse i en ny tabel.

Man kan let tjekke om tabellen opfylder 1NF ved at sortere efter primærnøglen, og se efter om en værdi af primærnøglen optræder mere end én gang.

2NF: En tabel er på anden normalform, når den er på 1NF og der ikke er felter som kan identificeres entydigt af en del af en sammensat primærnøgle. Hvis en tabel har en sammensat primærnøgle (bestående af to eller flere felter), og der findes et felt i tabellen som ikke indgår i primærnøglen og er afhængigt af en del af den sammensatte primærnøgle (et eller flere felter i primærnøglen kaldet del-primærnøgle), så skal dette afhængige felt fjernes fra tabellen og lægges over i en ny tabel sammen med en ny kopi af del-primærnøglen, som bliver primærnøglen i den nye tabel.

Hvis primærnøglen i en tabel ikke er sammensat, det vil sige består af ét felt, opfylder tabellen allerede anden normalform. Hvis primærnøglen er sammensat af alle felterne i tabellen, det vil sige ingen felter, som ikke indgår i primærnøglen, opfylder tabellen igen anden normalform. Ligeledes hvis hver enkelt felt i tabellen uden for primærnøglen kun er afhængig af hele primærnøglen, opfylder tabellen anden normalform.

Man skal finde de kolonner, der kan identificeres entydigt ved kun en del af primærnøglen, og udskille dem i en ny tabel.

3NF: En tabel er på tredje normalform, når den er på 2NF og der ikke er felter i tabellen, som er entydigt afhængige af andre felter end primærnøglen.

Hvis en tabel har felter, der ikke indgår i primærnøglen, og der findes et felt i tabellen som ikke indgår i primærnøglen og er afhængigt af et andet felt (evt. flere felter tilsammen), der heller ikke indgår i primærnøglen, så skal dette afhængige felt fjernes fra tabellen og lægges over i en ny tabel sammen med en kopi af det/de felter, som feltet afhang af. Det bliver således primærnøglen i den nye tabel.

Hvis der er et felt i en tabel, som ikke indgår i primærnøglen, opfylder tabellen allerede tredje normalform.

Hvis primærnøglen er sammensat af alle felterne i tabel, det vil sige at ingen felter som ikke indgår i primærnøglen, opfylder tabellen allerede tredje normalform.

Hvis hver enkelt felt i tabellen udenfor primærnøglen kun er afhængig af hele primærnøglen, opfylder tabellen allerede tredje normalform.

Man skal finde kolonner, der entydigt identificeres af andre kolonner end primærnøglen eller -nøglerne og udskille dem i en ny tabel.

Da alle vores tabeller i databasen opfylder den sidste sætning i definitionen for tredje normalform, hvilket vil sige, at "hvis hver enkelt felt i tabellen uden for primærnøglen kun er afhængig af hele primærnøglen, opfylder tabellen allerede tredje normalform", da har vi alle vores tabeller på tredje normalform.

Vi vil ikke gennemgå alle tabeller for at bevise at de er på tredje normalform (3NF), men i det følgende vil vi ved et enkelt eksempel, tabellen *Abonnent*, gennemgå, hvorfor denne tilfredsstiller de tre normaliseringstrin.

Tabellen er på første normalform, fordi:

- Tabellen har en primærnøgle, CPR_NR, og alle de andre attributter afhænger af denne nøgle.
- Der er ingen flerværdi attributter. Det ville der have været, hvis attributterne Fornavn og Efternavn var slået sammen til ét attribut Navn, der bestod af medlemmets fornavn og efternavn i samme tekstfelt:

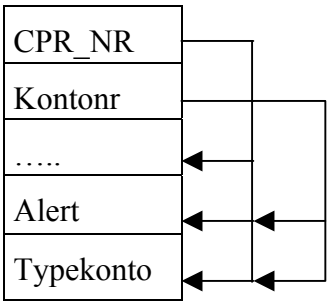
Navn
Jens Jensen

- Der er ingen sammensatte attributter. Man kunne forestille sig vi havde lavet en attribut, Aktieinteresse, hvor personens præference for de bestemte aktier blev nævnt i feltet, blot adskilt af et komma:

Aktieinteresse
Industri, Danske Aktier, ….

Tabellen er på anden normalform, fordi:

- Alle attributter afhænger af hele nøglen. Dette ville ikke været tilfældet, hvis vi for eksempel ikke havde lavet en separat tabel *Konto*. Vi ville i givet fald i *Abonnent* have haft en sammensat primærnøgle, bestående af *CPR_NR* og *Kontonr*. Men attributterne alert og typenavn ville være afhængige af *Kontonr* alene, og ikke af hele primærnøglen.



Tabellen er på tredje normalform, fordi:

- Alle attributter i tabellen er KUN afhængige af primærnøglen, og IKKE af andre felter. En mulig overtrædelse ville have været at tilføje en ekstra attribut i tabellen ved navn Mobilselskabet, eftersom den kun ville være afhængig af noget andet end primærnøglen – nemlig *Mobilnr*.

5.5 Implementering af databasen ved anvendelse af SQL

For at definere tabellerne i *AbService* benytter vi sproget SQL[14]. Manipulering, oprettelse og søgningen af data i databasen kan anvende SQL til alle SQL basererede værktøjer, såsom MA og MySQL. I kapitel 4 forefindes en nærmere forklaring af MySQL og M. Access.

Man skal være opmærksom på at tabeller skal bygges op i en bestemt rækkefølge[15]. Hvis en tabel refererer til en anden tabel, skal man først definere de tabeller som der refereres til. Man kan for eksempel ikke oprette tabellen *Aktier* uden først at have oprettet tabellerne, *Gruppering* og *EmneA*, da tabellen *Aktier* indeholder fremmednøglerne *Gruppenavn* og *Emnenavn*, der danner primærnøgle i deres respektive tabeller henholdsvis *Gruppering* og *EmneA*.

I det følgende beskrives hvilke principper man skal gå igennem for at implementere databasen. Vi gør dette ved hjælp af et eksempel.

Vi starter derfor med at oprette tabellen Gruppering. Dette gøres med kommandoen **CREATE TABEL**:

[14] SQL er et alment benyttet sprog til at oprette, søge og manipulere data i en database. Der er nogle der ikke kalder SQL et sprog, men et erklærende (deklarative) sprog, det vil sige SQL fortæller serveren hvad man vil gøre men ikke hvordan. Dette er op til serveren.

[15] Hvis man vil benytte sig af Begrænsninger (CONSTRAINT) i de forskellige database programmer (Acces el. MySQL).

```
CREATE TABEL Gruppering(Gruppenavn varchar(20) NOT NULL PRIMARY
KEY);
```

Her siges at kolonnen *Gruppenavn* skal være af typen *varchar(20)* og op til 20 tegn, det vil sige en attribut af denne type må indeholde strenge fra 0 til 20 karakterer, og at den skal være **NOT NULL**, som betyder at kolonnen skal have en værdi, det vil sige den skal ikke være tom, i hver af tabellens rækker.

Kolonnen danner samtidig primærnøglen (*PRIMARY KEY*), altså skal alle rækker i kolonnen have forskellige værdier.

Vi definerer i det følgende tabellen *EmneA*:

```
CREATE TABEL EmneA(Emnenavn varchar(20) NOT NULL PRIMARY KEY);
```

Det er den samme definition som for tabellen *Gruppering*.

Vi kan nu gå videre og definere tabellen *Aktier*:

```
CREATE TABEL Aktier(Aktienavn varchar(20),
   Aktiesymbol varchar(20) NOT NULL,
   Alert char(1) NOT NULL,
   Gruppenavn varchar(20) NOT NULL,
   Emnenavn  varchar(20) NOT NULL,
   PRIMARY KEY(Aktiesymbol, Gruppenavn, Emnenavn),
   CONSTRAINT Akt_Grup FOREIGN KEY(Gruppenavn)
   REFERENCES Gruppering(Gruppenavn)
   CONSTRAINT Akt_Emne FOREIGN KEY(Emnenavn)
   REFERENCES EmneA(Emnenavn));
```

Det fremgår af de sidste fire linier, at *Gruppenavn* og *Emnenavn* er fremmednøgler (*FOREIGN KEY*), og refererer (*REFERENCES*) til tabellerne *Gruppering* og *EmneA*. Med samme fremgangsmåde defineres resten af databasens tabeller. Alle tabellernes oprettelse i SQL findes i appendiks SQLfiler.

5.5.1 Begrænsninger (CONSTRAINT) i praksis i MySQL

For at definere tabeller i MySQL kan man anvende SQL koden fra foregående afsnit. Fra de sidste 5 linjer i koden man kan se at man i anvendelsen af *FOREIGN KEY* skal bruges *CONSTRAINT* (for eksempel *CONSTRAINT Akt_Grup FOREIGN KEY (Gruppenavn)*). Disse kan godt erklæres, men man skal huske at anvende MySQLs anden type af tabeller nemlig *InnoDB* (se afsnit 4.1.2).

For at MySQL kan tage imod de sidste linjer i den ovenstående SQL kode, skal *TYPE=InnoDB* tilføjes, dette er også kommenteret i det nævnte afsnit.

Herefter understøtter MySQL *'FOREIGN KEY CONSTRAINT'*.

For at anvende *FOREIGN KEY CONSTRAINT* i MySQL med *InnoDB* tabeltypen, er der et par forholdsregler man skal tage. For det første skal kolonnen man refererer til enten være en primærnøgle eller have et index skabt til det. For det andet skal kolonnen i tabellen til hvilken man tilføjer begrænsningen (*CONSTRAINT*) enten være en primærnøgle eller have et index at referere til.

Nedenstående viser hvordan tabeller implementeres med brug af MySQL ***TYPE=InnoDB***. Bemærk indsættelse af **INDEX**:

Følgende SQL kode opretter to tabeller med en *Foreign Key Constraint* på *Aktier*.

```
CREATE TABLE Gruppering(Gruppenavn varchar(20) NOT NULL PRIMARY
KEY) TYPE=INNODB;

CREATE TABLE EmneA(  Emnenavn varchar(20) NOT NULL PRIMARY KEY)
TYPE = InnoDB;

CREATE TABLE Aktier( Aktienavn varchar(20),
                     Aktiesymbol varchar(20) NOT NULL ,
                     Alert char(1) NOT NULL,
                     Gruppenavn varchar(20) NOT NULL,
                     Emnenavn varchar(20) NOT NULL,
                     PRIMARY KEY (Aktiesymbol),
                     INDEX (Gruppenavn),
                     CONSTRAINT Akt_Grup FOREIGN
                     KEY(Gruppenavn)
                     REFERENCES Gruppering(Gruppenavn),
                     INDEX (Emnenavn),
                     CONSTRAINT Akt_Emne FOREIGN
                     KEY(Emnenavn)
                     REFERENCES EmneA(Emnenavn)
                       ) TYPE = InnoDB;
```

Vi har defineret databasen i Access idet vi er af den opfattelse at det er mere anvendeligt, og det giver nemt en grafisk repræsentation af databasen sammen med de relationer som er mellem de forskellige tabeller. Man bruger samme SQL kode til at oprette tabellen. Men vi skal også nævne at den ikke er i stand til at indeholde større mængder data, som forklaret i kapitel 4.

5.6 Delkonklusion

Det skal understreges at vores database er et grundlæggende eksempel på hvordan det kunne have været i virkeligheden. Vi har begrænset den til kun at indeholde 4 services, og det kunne godt muligvis have været nogle andre typer services. Vi har også begrænset de enkelte muligheder indenfor de 4 services. For eksempel kunne vi have givet kunderne mulighed for at få at vide om der bliver sat et beløb ind på deres konto, og hvem det var fra, i stedet for at oplyse om at saldoen er gået under en bestemt værdi. Vores database skal derfor kun forstås som en prototype, et inspirerende eksempel, hvis struktur kan videredesignes.

6. Push-systemet

Push-systemets arbejde er at behandle alle indgående informationer fra kildefilerne. Disse er som nævnt før, input til systemet. Man skal frasortere informationer som ikke skal bruges til noget og behandle de andre. I systemet bliver informationer læst ind en ad gangen. Vi implementerer *push-systemet* i Java.
Da vi forklarede struktureringen af *AbService* databasen, nævnte vi at vi i nogle tabeller har oprettet en attribut med navnet *Alert,* til at skelne mellem indkommende informationer. Derfor skal man have en måde til at gå ind i databasen for at undersøge om *alerten* i korrespondancen med de bestemte informationer (*Aktiesymbol, Kontonr, Emnenavn, Valutanavn*) er sat op. Derefter skal man finde de abonnenter der er interesserede i denne information og bagefter hvilken kommunikationsvej man skal anvende til at sende informationen. Informationen sendes da og endvidere behandler systemet den næste information.

Under disse forudsætninger kan man dele push-systemarbejdet op i 5 store underordnede dele:

- Indlæsning af kildefiler
- Ud fra de forskellige *Alerter* bestemmes informationens vigtighed
- Finde ud af hvem der er interesseret i informationen
- Se om informationens værdi er inkluderet i abonnentens begrænsninger
- Udsendelse af informationen til de interesserede abonnenter.

For at kunne går ind og manipulere data i databasen er der behov for at etablere en forbindelse mellem *push-systemet* og databasen. Kommunikation mellem *push-systemet* og *AbService* er nærmere forklaret i afsnit 6.5.

Implementering af *push-systemet* bliver også behandlet senere i dette kapitel.

6.1 Indlæsning af kildefiler

I denne sektion vil vi ikke komme ind på hvordan filerne kommunikationsmæssigt modtages eller sendes. Vi antager at vi får filerne ind på en måde som beskrives i et andet kapitel (Kapitel Serveren/Klienten) og der forklares hvorledes kommunikationen foregår mellem vores system og kilde-verdenen.
For at kunne forstå de informationer der kommer fra input filen, skal man vide i hvilket format de er blevet skrevet, og hvordan informationerne er struktureret. Vi er blevet enige med SDC om at danne vores egen fil og vi har bestemt at informationerne bliver sendt i XML format. Dette betyder for eksempel at en aktiefil kan se ud som i fig. 6.1.1:

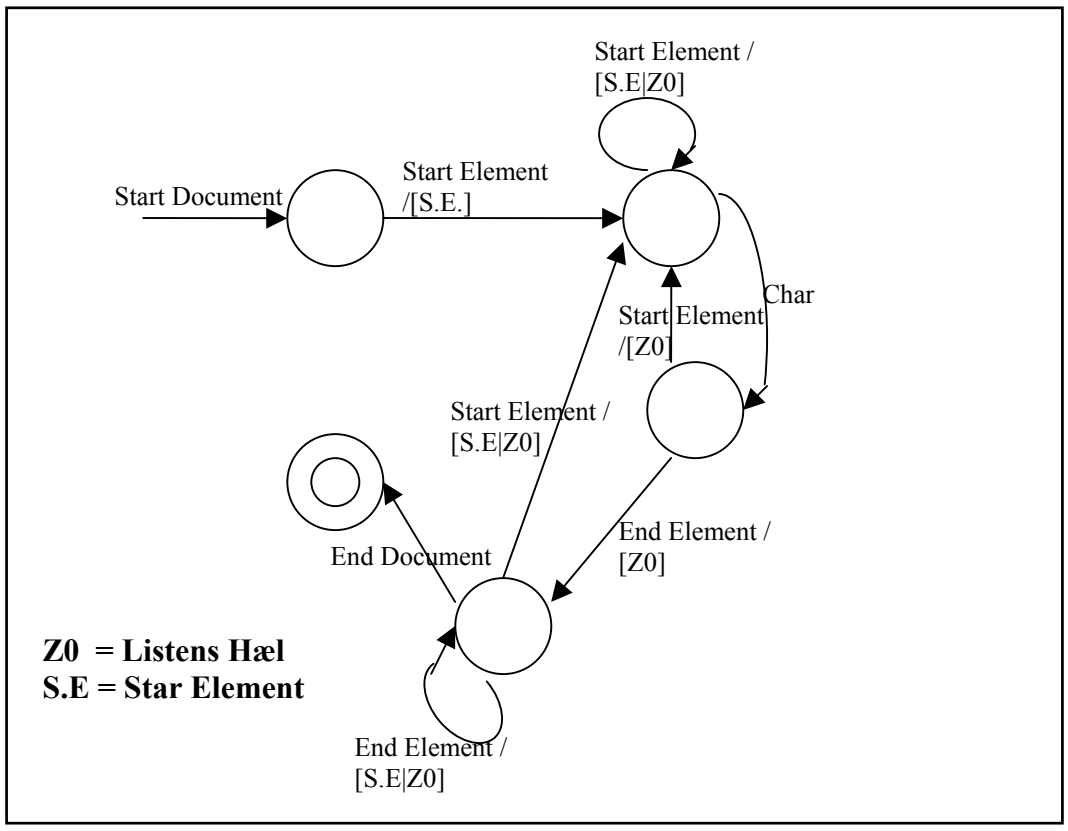
```
<KildeAktier>
<Aktier>
<Aktiesymbol>AAA</Aktiesymbol>
<Aktiekurs>950.3943</Aktiekurs>
</Aktier>
<Aktier>
<Aktiesymbol>DNW</Aktiesymbol>
<Aktiekurs>770.40314</Aktiekurs>
</Aktier>
<Aktier>
<Aktiesymbol>CHH</Aktiesymbol>
<Aktiekurs>70.13603</Aktiekurs>
</Aktier>
<Aktier>
<Aktiesymbol>DNS</Aktiesymbol>
<Aktiekurs>871.3117</Aktiekurs>
</Aktier>
<Aktier>
<Aktiesymbol>COL</Aktiesymbol>
<Aktiekurs>216.9818</Aktiekurs>
</Aktier>
</KildeAktier>
```

Figur 6.1.1

Det første tag angiver hvilken slags fil man får. For eksempel er det første tag i figur 6.1.1 *KildeAktier*, og det betyder at vi modtager aktieinformationer. Indlæsningen skal også give mulighed for at udplukke de informationer fra filen som bagefter skal hjælpe os med at servicere abonnenten. For at indlæse og udplukke informationen fra filen

anvendes en parser, en *SAX-Parser* – se kapitel 4, afsnit 4.5.2, for en nærmere forklaring på denne.

Parserens designet kan forklares ved hjælp af en tilstandmaskine som er repræsenteret i fig. 6.1.2.



Figur 6.1.2 Tilstandmaskinen for SAX-Parser

Figuren 6.1.2 fremhæver at filen skal have et format som skal respekterer det regulære sprog[16] som er symboliseret i figur 6.1.3

```
<Start Dokument>
              (<Start Element><Char><Slut Element>)*
<Slut Dokument>
```

Figur 6.1.3 Regulær udtryk for SAX-Parser.

Hele indlæsningen bliver således foretaget af parseren.

[16] Hopcroft, Motwani, Ullman: [13]: Kapitel 3 og 6.

6.2 Behandling af information

Når Parseren har udplukket de interessante informationer, evalueres deres vigtighed for abonnenterne. Dette udføres ved at tjekke *alerten* som relaterer til informationen. Kun de vigtige informationer behandles.

For at behandle informationerne skal vi finde de forskellige abonnementer som er blevet oprettet til de pågældende informationer. For hver af disse abonnementer skal der kontrolleres om værdien af informationen er inkluderet i den intervalbegrænsning som er fastsat i abonnementet. Hvis værdien er tilpasset abonnementets begrænsning, udtrækkes adressen på kommunikationsvejene (e-mail, telefonnummer) som er indbefattet i det pågældende abonnement. Efter dette formuleres en besked som indeholder den udplukkede information, og slutteligt afsendes den til gruppen af interesserede abonnenter.

6.3 Implementering af indlæsningen

SAX-Parser'en kan identificere et *start dokument*, *start element*, *characters*, *end element* og *end document*. S*tart document'*et identificerer begyndelsen af strømmen af informationer (kildefilen), og som nævnt skal den have et bestemt format (ellers vil Parseren udløse en fejlmeddelelse og stoppe parsingen af dokumentet) og ligeledes for *end document*'et. Formatet for begyndelsen af dokumentet er således:

```
<?xml version=\"1.0\" standalone=\"yes\"?>
```

*Start element'*et repræsenterer de forskellige tags, for eksempel *<KildeAktier>, <Aktier>, <Aktiesymbol>* og *<Aktiekurs>,* som kan ses i Fig. 6.1.1. Der er også en anden tag som ikke er vist i kildefilen (fig. 6.1.1), *<harry>,* denne bliver brugt til initialisering og afslutning af datastrømmen og som en genkendelse af afsenderen. Derfor er det vigtigt at tjekke om den er der.

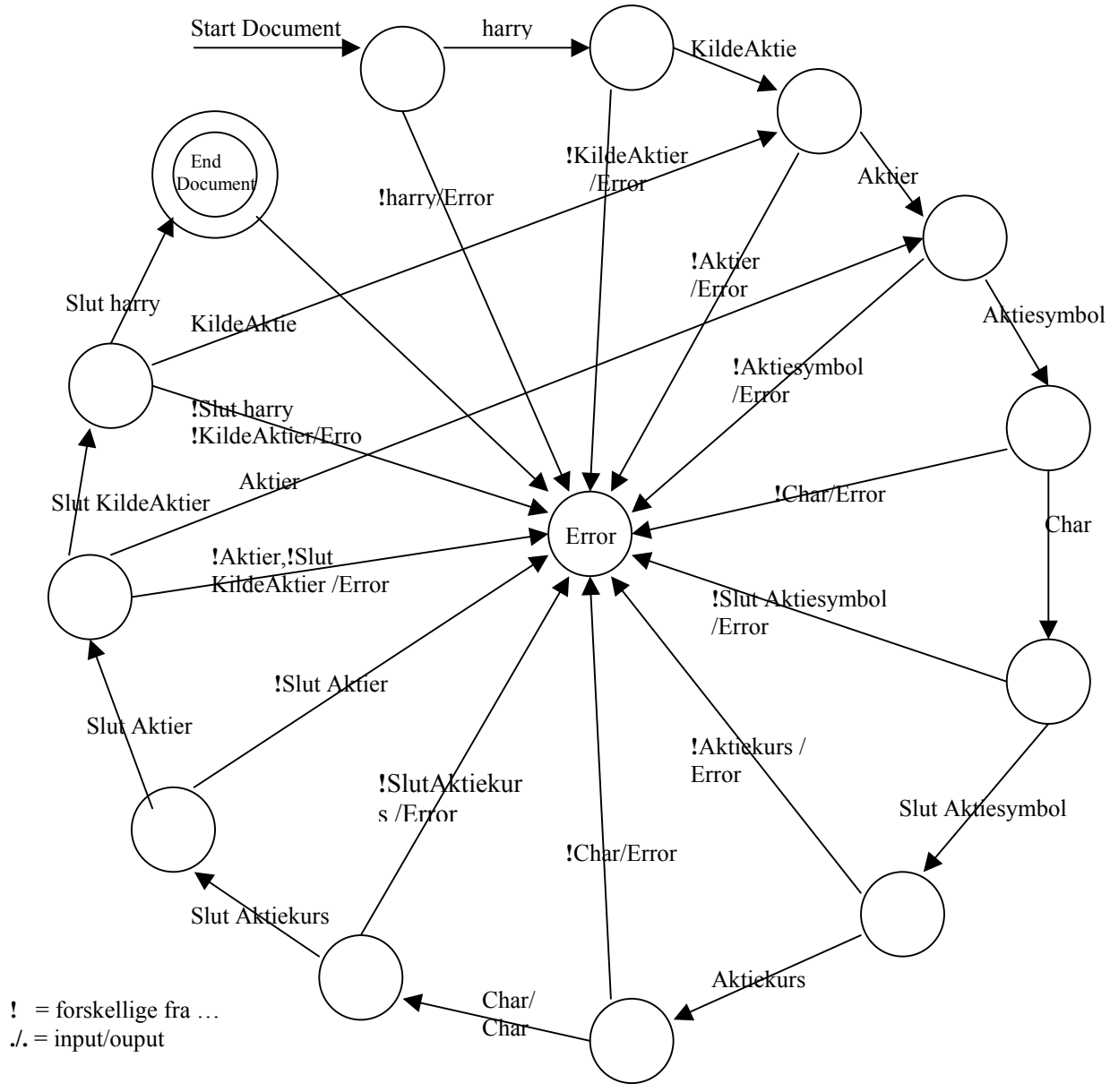
For at kunne anvende Parseren har vi lavet et Java-program: *ParserKildeFile,* og fra nu af refererer vi til dette program som Parser. For en bedre forståelse af Parseren har vi valgt at skematisere måden den arbejder på i en tilstandmaskine, Fig. 6.3.1.

I figuren kan ses den procedure Parseren går igennem for at læse fra kildefilen (KildeAktier). Parseren indleder læsningen af filen med at finde *start dokumentet* og går bagefter videre til næste tilstand, et *start element* ved navn *harry,* og det bliver læst. Hvis dette ikke sker producerer Parseren en fejl og slutter programmet; vi har valgt at kalde denne tilstand for *Error*. Der bliver igen skiftet tilstand, når man får noget andet i input end *KildeAktier*, og det giver fejl. Som nævnt genkender Parseren det sprog vi har i Figur 6.1.2. Når den til sidst læser *</harry>,* genkender den *End Dokumentet* og Parseren slutter.

Figuren 6.3.1 repræsenterer en specifik udgave af Fig. 6.1.2, og den behandler kun aktiefilen. Vores parser gør det samme arbejde for de andre tre kildefiler.

Parseren går igennem kildefilen, og hvis sekvensen ikke er som i vist i Figur 6.3.2, da meddeler den en fejl og slutter programmet.

Det første Parseren gør er at kategorisere informationerne fra kildeverdenen. Den skal genkende og sortere kildefilerne i aktie-, konto- , valuta- eller nyheds-filer.



Figur 6.3.1: Tilstandmaskine: Parsing af *KildeAktier d*okumentet

Parseren kategoriserer også kildefilerne i nogle niveauer, som er de forskellige tags kildefilen er bygget op omkring. Dette gør det nemmere at udplukke informationerne mellem niveauerne.

```
<Start Dokument>
              (<KildeAktier>
                     (<Aktier>
                            <Aktiesymbol>Character</Aktiesymbol>
                            <Aktiekurs>Character</Aktiekurs>
                     </Aktier>)*
              </KildeAktier>)*
<End Document>
```

Fig. 6.3.2 Repræsentationen af det sprog Parseren kan acceptere.

6.4 Implementering af behandlingen af information

Hvis vi for eksempel antager at vi modtager en kildeaktie kan vi ved hjælp af Parseren få fat på aktiesymbolet og aktiekursen, som er de informationer vi er interesseret i. Når Parseren genkender et *Aktiesymbol*, vil programmet kontrollere og bestemme informationens vigtighed.
Dette gøres ved hjælp af nogle metoder som kommunikerer og arbejder med databasen.
Forbindelsen etableres ved hjælp af nogle bestemte instruktioner, der understreges og fremhæves både i afsnit 6.5 og i selve koden ved hjælp af nogle kommentarer.
Når forbindelse med databasen etableres muliggøres en udtrækning af den post hvortil *alerten* er sat op, og som er tilknyttet det bestemte aktiesymbol.

For at se informationens vigtighed, har vi som sagt behov for at kigge ind i databasen i tabellerne. For at kunne gøre det i Java må vi anvende nogle metoder som er i klassen *java.sql*. Denne klasse giver mulighed for at bruge de forskellige SQL-forespørgsler (se afsnit 6.5) der anvendes i *doPreparedStatement* (som er en metode der er inkluderet i klassen *AlertForInformation*, implementeret af os til dette formål).
Hvis vi for eksempel skal finde ud af om Den Danske Banks aktier, som har symbolet DDB, er en interessant aktie, anvender vi følgende forespørgsel:

```
SELECT Alert FROM Aktier WHERE Aktiesymbol = DDB
```

Denne forespørgsel udtrækker det *Alert* felt fra tabellen *Aktier* og alerten er relateret med aktiesymbol DDB.
Vi kan hermed, via svaret fra select-forespørgslen der passeres til Parseren (og som fra dens værdi, *j* eller *n*), vurdere om aktien er interessant.

Kun hvis *Aktiesymbol*et er interessant, vil man hente aktiekursen og finde ud af hvem af abonnenterne der er interesseret i denne information. Er det ikke tilfældet går vi videre og finder næste *Aktiesymbol* i filen. For at se til hvem informationen skal sendes, må vi igen søge ind i databasen og dette arbejde udføres via nogle metoder. Vi anvender fra klassen *TakeInfoAbonnent* metoden *doPreparedstatement* til formålet.

I metoden anvender vi igen nogle SQL-forespørgsler. Dette gøres hvis man for eksempel vil finde ud af, hvem der er interesseret i en aktie, som har DDB som aktiesymbol og en aktiekurs på 400.

```
SELECT Mobilnr FROM Abonnement WHERE (Aktiesymbol = DDB AND
((400 >= Minval) AND (400 <= Maxval)))
```

I denne forespørgsel er vi interesseret i at få fat på alle de mobilnumre fra tabellen *Abonnement*, som har et aktiesymbol lig med *DDB* og en aktiekurs på *400*, og værdien er i den intervalbegrænsning der er bestemt af *Minval* og *Maxval*. Dette interval er bestemt af abonnenten ved tilmeldingen. Det samme gælder for de e-mails som skal sendes.

En anden information vi har behov for er, hvor mange sms og hvor mange e-mails der skal sendes. Denne information er vigtig, da den nødvendige metode, *showResults,* som også er i klassen *TakeInfoAbonnent*, beder om den. For at få antallet anvendes følgende SQL forespørgselsform, som i e-mailtilfældet ser således ud:

```
SELECT COUNT (Email) FROM Abonnement WHERE (Aktiesymbol = DDB
AND ((400 >= Minval) AND (400 <= Maxval
```

Når de nødvendige e-mailadresser og mobilnumre og antallet af disse er samlet, anvendes de af showResults-metoden. Som opgave har denne at sende de informationer via de to forskellige kommunikationsveje, som der spørges om. I *showResults* metoden bliver der også konstrueret den besked som skal sendes til kunderne, og den indeholder (i aktietilfældet) aktiesymbolet og dennes kurs.
For at sende e-mails, har vi anvendt metoden *postMail*, der er opbygget i klassen *SendMail,* og som via de forskellige metoder fra Javapakken, *javax.mail*, afsender de pågældende beskeder.

Efter dette er gjort, læser vi videre i filen og finder næste *Aktiesymbol*, og hele proceduren gentages, indtil man finder *End Document'et*. Dette sker kun når strømmen af informationer afbrydes.

Vi har behandlet afsendelsen af sms'er på samme måde som afsendelsen af e-mails. Dette var vi nødt til, da der i juli måned 2002 opstod en ændringspolitik, i teleselskaberne, med hensyn til afsendelse af sms fra Internettet. Fra denne måned skulle man betale for at kunne gøre det, og SDC kunne ikke give os deres licens til at gøre det på grund af deres

interne politik. Da metoden der skal afsende sms-beskeder ligner den der anvendes for afsendelsen af e-mails, blev der foreslået at vi skulle hente mobilnumre og formatere det som en e-mailadresse, og derefter sende beskeden til den fiktive e-mail adresse, som højst sandsynligt ikke eksisterer.

6.5 Kontakt til databasen

At få kontakt til databasen er en vigtig del af vores *push-system*. Det består af to del:

1. Indlæse databasedriveren
2. Etablere forbindelsen

Indlæsning af driveren sker ved at bede systemet indlæse den pågældende klasse, der derefter registrerer sig selv i JDBC-systemets driver-manager.

Med Java under Windows (for Access) følger en standard JDBC-ODBC-bro med, så man kan kontakte alle datakilder defineret under ODBC. Denne driver indlæses med:

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
```

Og den følgende sætning gælder for indlæsningen af driveren i MySQL.

```
Class.forName("org.gjt.mm.mysql.Driver");
```

Når forbindelsen oprettes, angiver man den ønskede databases navn. Databasens navn skal være defineret i Windows' Kontrolpanel under

```
DatabaseConnection=
DriverManager.getConnection("jdbc:odbc:AbService");
```

6.5.1 Kommunikation med databasen

Når vi har en forbindelse, kan vi oprette et *statement*-objekt, som vi kan sende kommandoer og forespørgsler til databasen med:

```
Statement myStatement = databaseConnection.createStatement();
```

Der kan opstå forskellige undtagelser af typen *SQLException*, der skal fanges.

Forespørgsler

SQL-forespørgslen SELECT udføres med metoden *executeQuery*:

```
ResultSet myResult = myStatement.executeQuery(<SQL
forespørgsel>);
```

Den giver et *ResultSet*-objekt, der repræsenterer svaret på forespørgslen. Data hentes fra objektet således:

```
while (myResult.next())
{
  String navn = rs.getString(<kolonne navn>);
  System.out.println(navn);
}
```

Man kalder altså metoden *next* for at få næste række i svaret, læser de enkelte celler ud fra kolonnenavnene (eller kolonnenumrene, regnet fra 1 og opefter), hvorefter man går videre til næste række med *next* osv. Når *next* returnerer false, er der ikke flere rækker at læse.

6.6 Anvendelse af SQL-forespørgsler

Kommandoen *SELECT* bruges i SQL til alle former for udtræk af data. Denne er en effektiv og bred kommando, og den indeholder mulighed for en række klausuler og variationer. Dog er det en meget simpel kommando, da alle *SELECT*-kommandoer antager samme form. For overblikkets skyld kan vi antage at alle *SELECT*-kommandoer har denne struktur:

```
SELECT  her kommer en liste over kolonner
FROM    her angives tabeller der skal bruges
[WHERE  eventuelle betingelser for udtrækket]
```

Formålet med *SELECT* er at opbygge en tabel med data udtrukket fra tabellerne i en relationsdatabase.

Af denne grund er første linje i en *SELECT*-kommando en liste over kolonner. Linjen beskriver de kolonner der skal være i resultatet af forespørgslen. Linjen *FROM* angiver den eller de tabeller, der kræves for at foretage forespørgslen. Der kan eventuelt være en *WHERE*-linje, som placerer en form for betingelse på udtrækket.

Der findes en række indbyggede funktioner i SQL, som bruges til udtræk af statistiske informationer fra en tabel, hvoraf én er *COUNT*-funktionen. *COUNT*-funktionen returnerer antallet af værdier i en kolonne, eller antallet af rækker i en tabel. Alle SQL-forespørgsler vi har anvendt, kan ses i nogle af koderne i Appendix B med Java-koder og Appendix C med ASP-koder.

6.7 Delkonklusion

Push-systemet behandler informationer ved at sortere dem efter vigtighed i forhold til abonnenternes interesser, og finder derpå de pågældende abonnenter i databasen. Derefter afsendes informationerne via sms eller e-mail. Formålet med *push-systemet* er hermed fuldstændig opnået. Herefter er *push-systemet* en del af Abonnementsservice, det vil sige at det inkluderer nogle Javaklasser, som giver mulighed for at etablere en forbindelse et samarbejde med databasen.

7. Brugergrænseflade

Vores brugergrænseflade er en del af et givent pengeinstituts netværk. Fra netbanken kan kunden, gennem et 'link', komme ind på vores system. Kundens identitet og andre sikkerhedsforanstaltninger bliver i forvejen kontrolleret, men det bliver alligevel gennemgået en ekstra gang fra Abonnementsservices side. Vores opgave er at designe og implementere de sider, som vil give kunden mulighed for at tilmelde sig abonnementssystemet nemt, hurtigt og effektivt. Disse sider (websider) bliver anvendt for at kunderne kan blive abonnenter i vores system og deres data bliver indsat i *AbService*-databasen henholdsvis for Aktier, Konti, Nyheder og Valuta abonnementer.

Vi har bygget systemets grænseflade i to skridt:

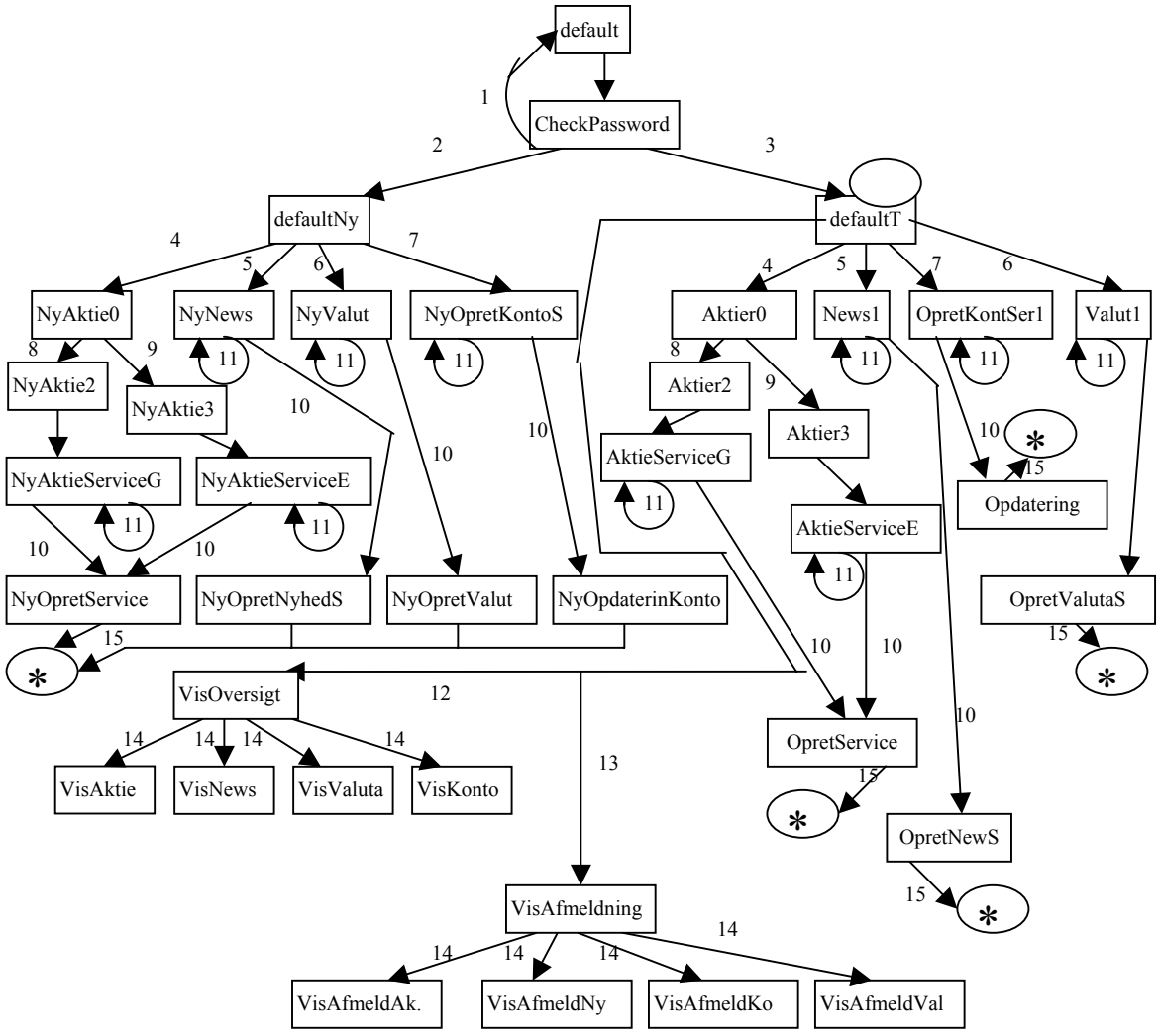
- Design af websiderne.
- Implementering af disse ved hjælp af ASP og HTML.

7.1 Design af websiderne

Vi begynder designet af web-siden ved at klassificere kunderne i to dele: *Ny tilmelding* og *Allerede tilmeldt*. Alle abonnenternes informationer gemmes i databasen, men er man ny i systemet skal man indtaste sine personlige oplysninger. Som allerede tilmeldt skal man altså ikke indtaste personlige oplysninger endnu engang.

Denne opdeling bliver udført på den første side. Radio-knapperne giver mulighed for en kunde at fortælle om han vil oprette *nytilmelding* eller om han *allerede* er tilmeldt.

Når man derefter kommer ind i servicen får kunden mulighed for at vælge hvilken service vedkommende vil abonnere på. Er man allerede tilmeldt skal man endvidere have mulighed for at se en liste over sine abonnementer og for at afmelde sig disse. Sammenhængen mellem alle websiderne vises i figur 7.1.1. Efter at kunden har valgt én af disse services og trykket på oprettelsesknappen, bliver oplysningerne gemt i databasen og er klar til viderebehandling. Der skal også tages hensyn til indtastningsfejl og kunden skal ikke kunne tilmelde sig samme service flere gange. Når kunden har valgt en service opstilles en alert for denne i databasen og når vedkommende afmelder sig skal der tjekkes om denne er den eneste der havde interesse for denne oplysning. I dette tilfælde skal *alerten* tages ned. Afmelder kunden sig alle services har vi valgt at gemme vedkommendes personlige oplysninger for at give mulighed at komme tilbage uden at indtaste sin profil igen. Vi har dog også et sekundært formål med at gemme disse oplysninger, i tilfælde af at vi skal tilbyde kunderne nogle andre services (for eksempel for at reklamere; afhængigt af firmaets politik).



Figur 7.1.1 ASP træ som viser sammenhængen mellem web-siderne

1: Fejl. Kunden har indtastet forkerte oplysninger. 2: Ny kunde som vil tilmelde sig Servicen. 3: Eksisterende kunde som vil logge sig ind. 4,5,6,7: Kunden vælger Aktier, Nyheder, Konto og Valuta. 8,9: Kunden vælger Grupper eller Emne af Aktier. 10: Kunden opretter servicen. 11: Slet indtastning. 12: Kunden kan vælge at få en oversigt over sine tilmeldinger. 13: Kunden kan vælge afmelding. 14: Til oversigt. 15 Tilbage til Service som "allerede tilmeldt"

Ovenstående figur viser vores tankegang bag designet af websiden. Roden af træet (default) er det *login* vi har og herfra bliver træet delt i to grene: En til venstre som er for de *nytilmeldte*, og den anden som er for de *allerede tilmeldte*. Som førnævnt fremhæver træet blandt andet at man som allerede tilmeldt har mulighed for at se en oversigt over sine abonnementer og at kunne afmelde sig. Endvidere viser det at man navigerer fra en side til en anden, efter at man har truffet nogle bestemte valg. Disse valg er repræsenteret ved de forskellige pile som vi har nummereret, og hvert tals betydning kan læses i kursivteksten nedenunder figuren.

7.2 Implementering af websiderne

Vi har implementeret alle websiderne i ASP-teknologien, JavaScript og HTML. I næsten alle sider har vi været nødt til at oprette en forbindelse til databasen. Forbindelsen etableres ved hjælp af nogle bestemte instruktioner, der understreges og fremhæves i koden ved hjælp af nogle kommentarer, koden kan ses i Appendix C.
Når forbindelsen er etableret kan vi gennem nogle SQL-forespørgsler gå til databasen for at udtrække og indsætte data. Det vil sige at alle handlinger mellem websiderne og databasen foregår ved hjælp af SQL-forespørgsler.

En af de første forespørgsler i databasen udføres fra *loginsiden* for at tjekke om informationerne kunne indtastes og om de passer med indholdet i databasen. Dette vil for eksempel sige, at hvis kunden oplyser at vedkommende *allerede* er tilmeldt, men i virkeligheden ikke er det, kan vi altså kontrollere dette. En *nytilmeldt* bliver kategoriseret som *allerede tilmeldt* så snart vedkommende har tilmeldt sig en service. Den personlige profil er allerede gemt i databasen ved første tilmelding.

Når kunden derefter vælger mellem de 4 forskellige services, giver vi kunden mulighed for at se en liste over de poster der forefindes i den bestemte service. Vi kunne have udført samme handling uden brug af databasen ved hjælp af HTML faciliteterne (OPTION), men dette ville have skabt et ufleksibelt system. Hvis vi for eksempel tager valuta i betragtning, er dette en post hvor der løbende kan ske forandringer. Ved hver ændring skulle man med HTML gå ind i selve koden for at opdatere samtlige ændringer, hvor vi med vores metode ved hjælp af databasen nemt og fleksibelt opdaterer uden nødvendigvis at have kendskab til HTML.

Efter kunden har valgt en post og indtastet oplysninger, skal alle informationerne gemmes i databasen. Når de skal gemmes skal man tage højde for at respektere databasens egenskaber, og for at undgå dubletter i databasen, tjekker vi om dette allerede er i tabellen. Databasen vil alligevel ikke acceptere det, og dermed meddeler vi dette til kunderne. Dette er også for sikkerhedens skyld, da browseren fortæller årsagen til fejlen, og dermed viser browseren navnet på tabellen og dens attributter, hvilket vi helst vil undgå.

Ved afmelding af en abonnent i en af de pågældende services opdateres oplysninger i databasen ved at slette vedkommende fra de relevante tabeller i databasen. Samtidig skal tælles hvor mange abonnenter der er interesserede i den bestemte post. Hvis optællingen viser at den pågældende abonnent var den eneste interesserede, skal *Alerten* tages ned.

For at fremvise oversigten over de forskellige abonnementer til abonnenten bruger vi pågældendes cpr-nummer til at finde de respektive postoplysninger. Herefter fremvises resultatet af denne udtrækning.

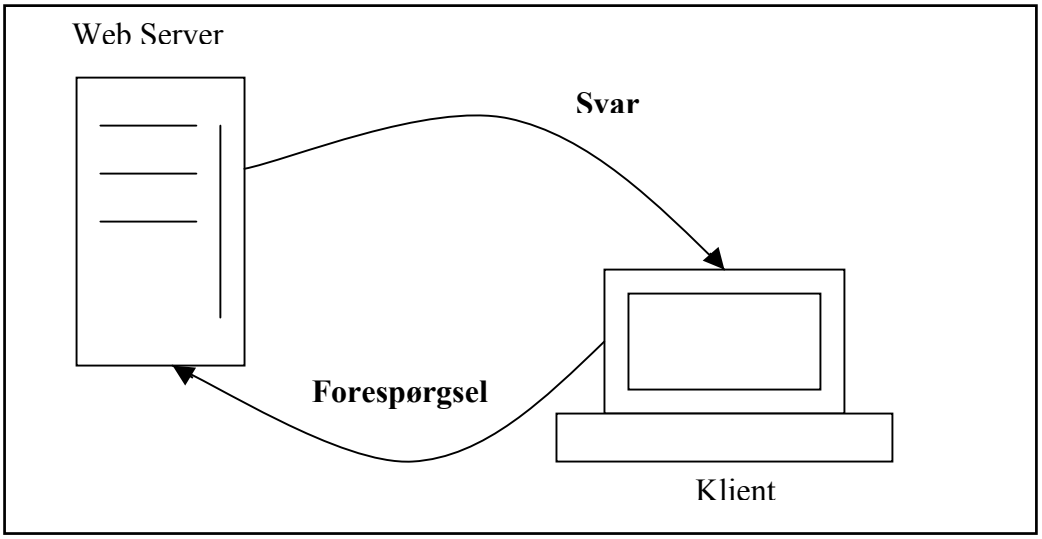
Ved implementering af web-siderne og for at beskytte mod uønskede manipulering af database har været ekstra opmærksomme på indtastningen af data i forskellige felter fra brugernes sider. Idet man kan slette en tabel, indsætte en række eller udføre en anden bestemt kommando i SQL, da kan man også skrive det i f.eks. *Fornavne*ts felt med det lille trick at man efter fornavnet indtaster et *semikolon*(;), derefter en SQL kommando. På

den måde vil også den sidste blive accepteret som en instruktion i SQL. For at undgå dette problem har vi kontrolleret at dette ikke sker i nogle af felterne.Det betyder at hvis kunden i et af felterne indsætter et semikolon, da udsender systemet en fejlmeddelse.

7.3 ASP processering af web-sider.

For at få en bedre forståelse af, hvordan siderne kommer i kundens browser, skal man igennem følgende 6 trin:

1. Der skrives en mængde af instruktioner i HTML , disse instruktioner/data bliver opbevaret i selve siden.
2. Derefter indtaster brugeren en forespørgsel i browseren og den bliver sendt til serveren.
3. Serveren finder filen med instruktioner.
4. Serveren følger instruktionerne og opretter en strøm af HTML.
5. Serveren sender den nyoprettede HTML data-strøm tilbage, gennem netværket, til browseren.
6. Browseren processerer HTML'er dataene og web-siden bliver vist.



Figur 7.3.1 Dynamiske websider.

Med en dynamisk web-side kan man få alle slags informationer, som ikke kendes i det øjeblik instruktionerne blev skrevet, for eksempel:

- Brugerens informationer og dennes personlige præferencer.
- Type browser kunden vil bruge.
- Andre informationer som kan blive hentet fra brugeren gennem forespørgsler.

- Informationer fra en database, tekst files, XML files, etc.

I vores tilfælde er vi interesseret i at få brugeroplysninger og at kunne arbejde med en database.

7.4 Delkonklusion

På vores webside tager vi højde for at man ikke kan indtaste nogle fejl i indtastningsfeltet. Hermed bliver databasens struktur ved indtastningen respekteret, da vi kontrollerer at oplysningerne ikke gentages, så dataene hører til samme gruppe. Således respekteres datatyperne af de forskellige poster også med det der er blevet defineret i databasen. Vi har forsøgt at gøre den så fleksibel som muligt, men websiden har ikke en komplet og finpudset form, da vi ved at den ikke skal sættes i produktion som den er. Den skal kun lægge en struktur for en fremtidig virkeliggørelse. Websiden er brugt i vores projekt til at skabe en kommunikation mellem kunderne og systemet. Endvidere fylder den dermed også databasen med information, og den tester anvendelsen af systemet.

8. Dannelse af kildefilerne

Vi skal til vores abonnementssystem bruge kildefiler, som skal indeholde oplysninger fra 'finansverdenen'. Disse filer bliver normalt afsendt fra 'bankverdenen' til SDC. Fra SDC's side blev det foreslået at vi selv implementerer kildefilerne (da deres kildefiler kun handler om bank-transaktioner).
Vi ved at kildefilerne er en kontinuerlig strøm af informationer. Disse informationer kommer ud fra banker, børsen eller avisen.
Vi opbygger selv vores egne kildefiler og simulerer deres opdatering, via et Java-program som generer tilfældige værdier. Det betyder, at hver information efter et ubestemt interval kan få en ny værdi. Det blev anbefalet at danne kildefilen i XML-format.
Behandling af kildefilerne er som bekendt gennemgået i de forrige kapitler.

8.1 Design af kildefilerne

Kildefilerne er blevet dannet i XML format. De repræsenterer de antagne kildefiler som pengeinstitutterne vil sende til abonnementssystemet. Ved hjælp af Java, har vi implementeret programmerne der genererer XML-filerne tilfældigt.
Disse programmer opretter fire filer :
Det vil sige at værdier og symboler ikke er fastsat i forvejen, men genereres tilfældigt. Dette gøres også for bedre at afspejle den virkelige verden, da man normalt ikke kan forudsige hvilke informationer der vil blive sendt, hvor mange og hvilke værdier disse indeholder. For at generere de forskellige aktiesymboler, kontonumre, nyhedsemner og valutanavne, hentes disse fra databasen.
Disse dannede filer indeholder oplysninger om aktiekursbevægelser, kontobevægelser, nyheder og valuta-kurser.
Der er heller ikke en fast frekvens for hvornår en bestemt aktie-, konti-, nyhed- eller valuta-fil, vil blive genereret.
For at benytte os af disse kildefiler bliver de dannet i Klientens side. Disse filer er dannet uafhængigt af Serveren, og klient-maskinen sender filerne til serveren via en socketforbindelse.
I kapitlet om Serveren er der en forklaring på hvordan disse kildefiler sendes eller modtages.

8.2 Implementering af Kildefilerne

Til dannelse af kildefilerne har vi implementeret en klasse, *MakingKildeFileXml*. Denne klasse genererer filerne tilfældigt ved hjælp af klassen *GemNavn* som henter oplysninger fra databasen og gemmer dem i nogle arrays. Den tilfældige generering sker på følgende måde: Først bestemmes en værdi tilfældigt som repræsenterer antallet af informationer filen skal indeholde. Derefter hentes hver information fra arrayet via et index som også genereres som en tilfældig værdi.

Kildefilerne skaber den strøm af informationer som er "inputtet" til *push-systemet*.

9. Klient og Server

For at repræsentere den virkelige verden har vi været nødt til at have en klient og en server. Klienten sender den fiktive strøm af information til serveren. Serveren er modtageren af strømmen og den servicerer den. For at programmere en klient eller en server som en del af en applikation, har vi behov for at have adgang til faciliteterne i programmeringssproget som gør os i stand til at opstille en kommunikationskanal mellem klient- og server-processerne i deres respektive systemer. Den almindelige abstraktion/kommunikation til dette formål er kendt som en socket-forbindelse. En *socket* er, groft sagt, endemålet i en kanal mellem to processer og den er identificeret af den relevante IP adresse og Port nummer. Vi har anvendt den såkaldte Java klientserver *socket*. *Javasockets* anvender, både for serveren og for klienten, nogle objekter fra klassen *socket* som er en del af *java.net* pakken.
For at opstille en *klientsocket* til en kommunikationskanal til port nummer *p* på serveren kaldet *s*, har klienten kun behov for at skabe et passende *socket* objekt med *p* og *s* som parametre.
For at opstille en *serversocket* til port nummer *p* på serveren *s*, skaber serveren et passende *ServerSocket* objekt og kalder acceptmetoden for at acceptere og lytte til et indkommende opkald fra en klient.

9.1 Klienten

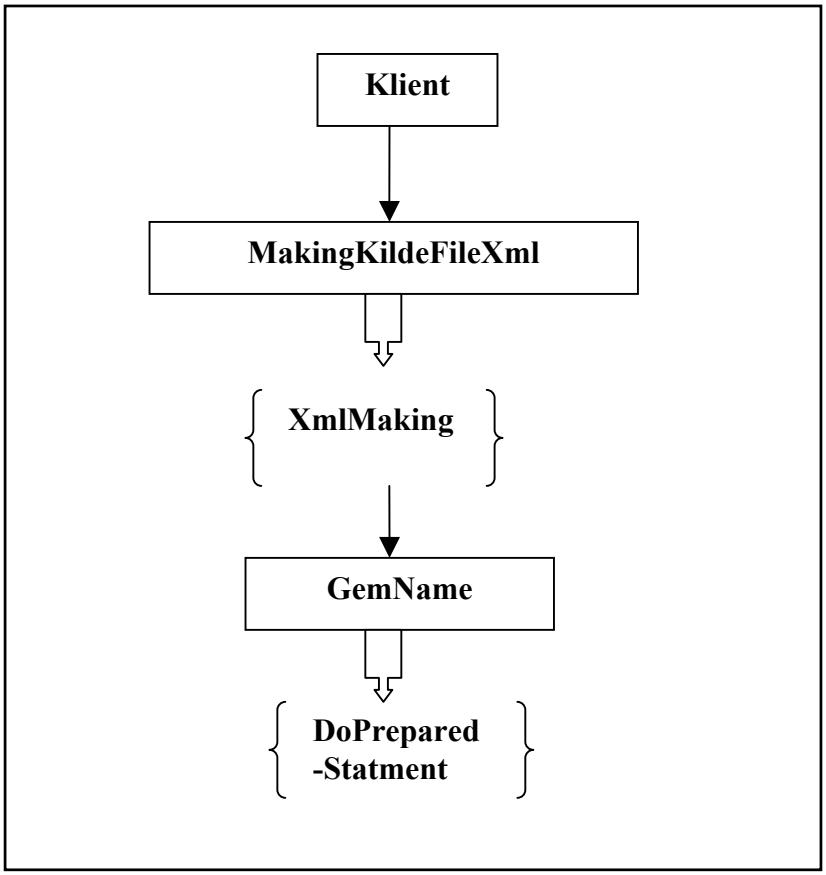
Formålet med klienten er at sende de dannede kildefiler gennem en socketforbindelse til serveren. Klienten er bygget op med en *TimerTask* der giver mulighed for at sende informationer med et bestemt tid interval . Intervallet kan anvendes for at belaste serveren ved at sende flere informationer per sekund.
Strømmen af informationer initialiseres ved at indsætte hovedlinjen af XML dokumentet, efterfulgt af tagget *harry*. Hovedlinjen læses som *start document* i Parseren, mens tagget *harry* behandles som et *start element*. Dette anvendes til at genkende afsenderen af kildefilen. Efter klienten har gjort dette, vælger den på en tilfældig måde hvilken kildefil der skal dannes, og derefter afsendes filen.
Klienten implementeres i klassen *RemindKlient3* og for at sætte klienten i gang, anvendes følgende instruktion: *Klient*.

9.2 Serveren

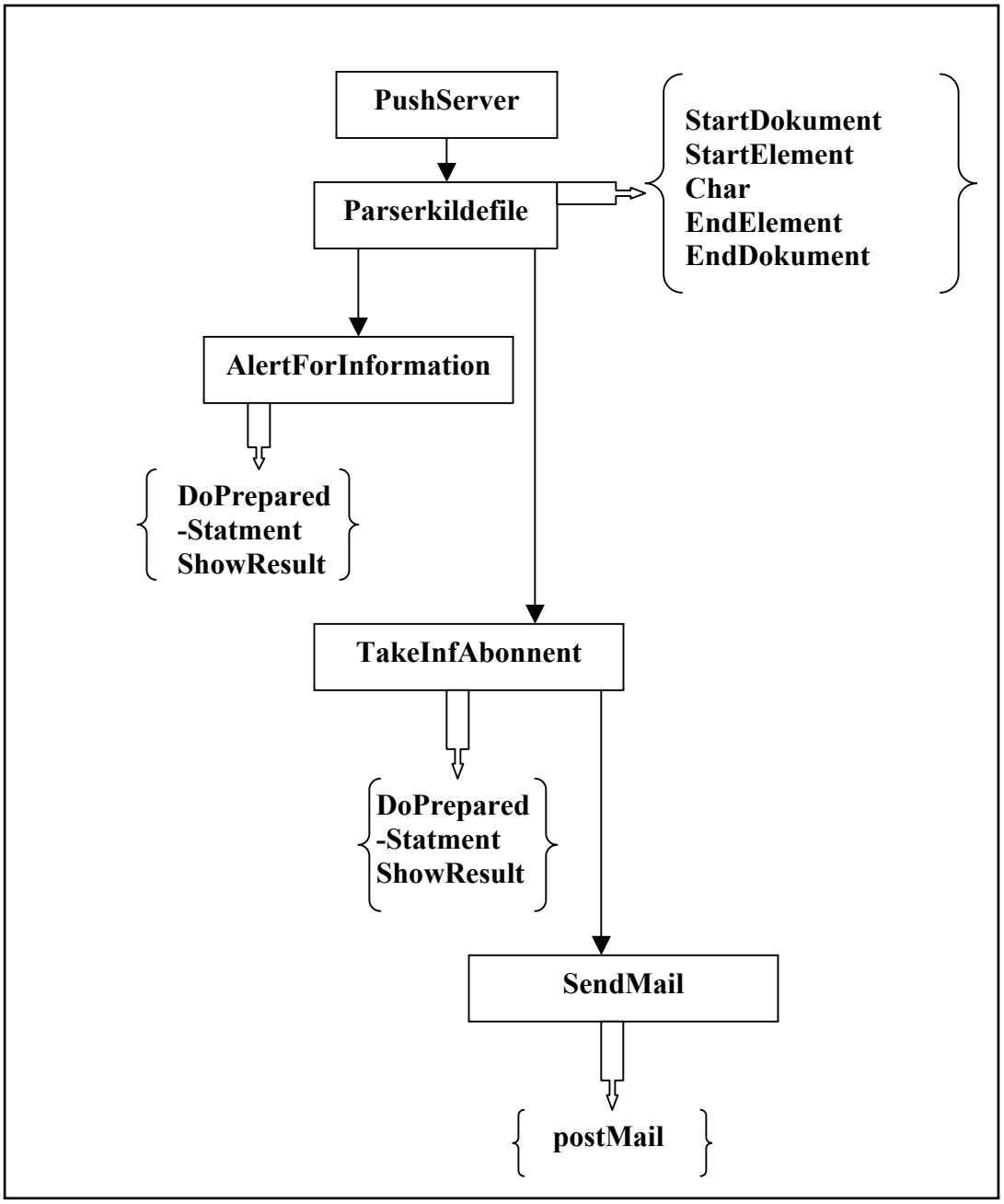
Serveren afventer og modtager datastrømmen, som klienten sender, på portnummer 8000 og datastrømmen videregives til Parseren som skal behandle den (se Parseren beskrivelse i kapitel 6).
Serveren implementeres i klassen *PushServ* og sættes i gang ved kommandoen: *PushServ*. Herefter åbnes porten og Serveren er klar til datamodtagelse.

9.3 Hierarkisk deling af klasserne

I følgende figurer vises forholdet mellem klasserne både for Klienten (Figur 9.3.1) og Serveren (Figur 9.3.2).



Figur 9.3.1 Hierarkisk træ over de forskellige klasser i Klienten



Figur 9.3.2 Hierarkisk træ over de forskellige klasser i Serveren

10. Testning

Testning er en vigtig del af implementeringen af ethvert system. Vi vil gennemgå testningen af Abonnementssystemet fra de mest basale tests til nogle lidt mere komplicerede. Testen udføres på en stationær computer, der vil have klientens rolle (vi kalder denne for Klienten), og en bærbar computer (THINKPAD i Series med ca. 500 Mhz. og 128 MB Ram), der fungerer som Serveren. Begge computere har Windows 2000 som Operativt System.
Hastigheden af systemet kan selvfølgelig påvirkes af de forskellige programmer der er installeret på Serveren og dermed er der rimelig grund til at tro at hastigheden ville blive højere på en hurtigere maskine.

Testningen vil vise funktionaliteten og ydeevnen af systemet, (funktionaliteten af *push-systemet*) da vi forøger byrden på systemet ved at pålægge dette flere transaktioner og/eller flere abonnementer/abonnenter.

10.1 Testning af funktionaliteten i Abonnementsservice

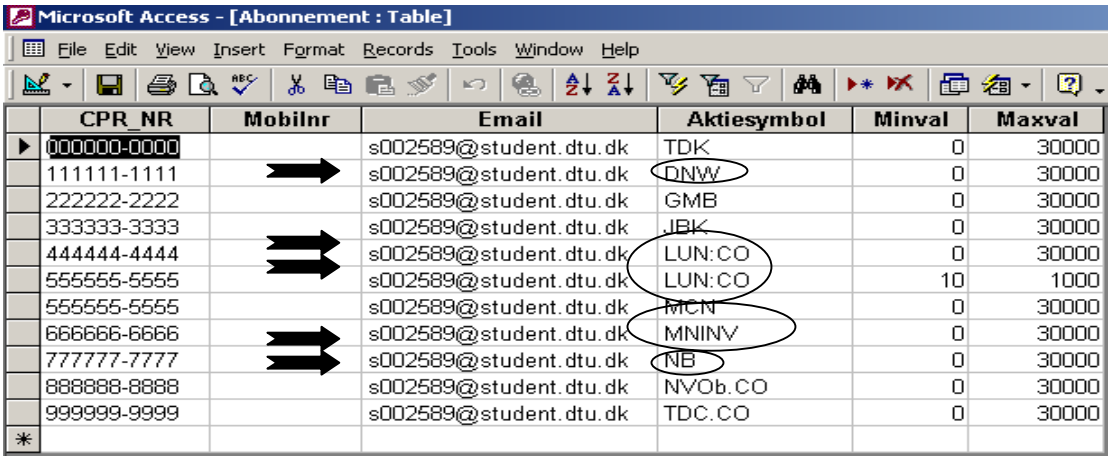
Vi vil begynde med at teste systemet med 10 brugere for at bevise at systemet kan modtage data og videresende dette, hvis tilfældet opstår.

1. test

Denne test er baseret på en database som indeholder 10 abonnementer, hvor hver, næsten alle abonnements intervalbegrænsning er mellem 0-3000. Vi vil gerne have at hver aktiekurs' værdi ligger i dette interval. Dette vil medføre at alle abonnenter får besked via e-mail, hvis de er tilmeldt de aktier der sendes besked om.
For at gøre dette sender vi en aktiefil (se figur 10.1.1), og vi vil undersøge om den sender de rigtige informationer til abonnenterne. Derfor skal vi finde ud af hvilket abonnement der skal serviceres, hvilket gøres ved at kigge på databasetabellen *Abonnent*, som vises i figur 10.1.2. I tabellen har vi omringet de aktiesymboler som er tilknyttet en interesseret abonnent. Herfra kan også ses at de aktieværdier vi har fået fra filen, er indeholdt i den angivne intervalgrænse. Vi har også sat ringe om disse aktiesymboler med deres respektive aktiekurs i figur 10.1.1, for at vise hvilken information der vil komme via e-mail. Vi har givet samme e-mailadresse til alle abonnenter, fordi man derved kan se alle modtagne beskeder på den samme e-mailprogram (Outlook Express).

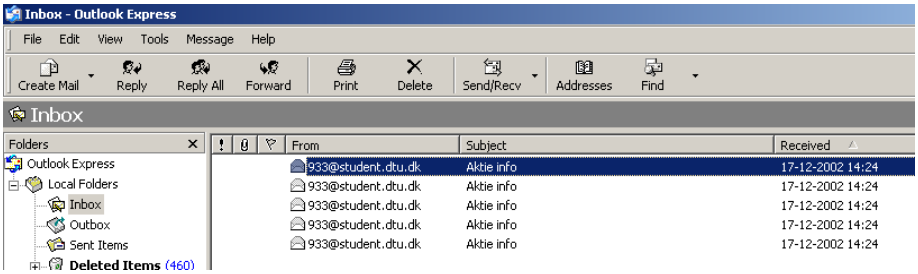
Hermed skal vi finde en e-mailbesked til de respektive abonnenter. De e-mails skal så findes i det e-mailprogram man har, som i dette tilfælde er Outlook. I figur 10.1.3 kan således ses at vi har fået 5 e-mails og deres indhold vises i de respektive figurer 10.1.4 til 10.1.8.
Forventningerne til første test er hermed opfyldt.

Figur 10.1.1 Aktiefil som bliver sendt

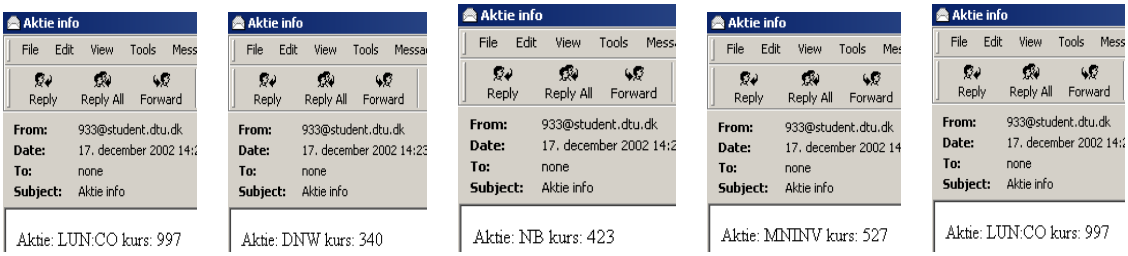


CPR_NR	Mobilnr	Email	Aktiesymbol	Minval	Maxval
000000-0000		s002589@student.dtu.dk	TDK	0	30000
111111-1111		s002589@student.dtu.dk	DNW	0	30000
222222-2222		s002589@student.dtu.dk	GMB	0	30000
333333-3333		s002589@student.dtu.dk	JBK	0	30000
444444-4444		s002589@student.dtu.dk	LUN:CO	0	30000
555555-5555		s002589@student.dtu.dk	LUN:CO	10	1000
555555-5555		s002589@student.dtu.dk	MCN	0	30000
666666-6666		s002589@student.dtu.dk	MNINV	0	30000
777777-7777		s002589@student.dtu.dk	NB	0	30000
888888-8888		s002589@student.dtu.dk	NVOb.CO	0	30000
999999-9999		s002589@student.dtu.dk	TDC.CO	0	30000

Figur 10.1.2 Abonnement tabel



Figur 10.1.3 Outlooks Express e-mails indbakke.



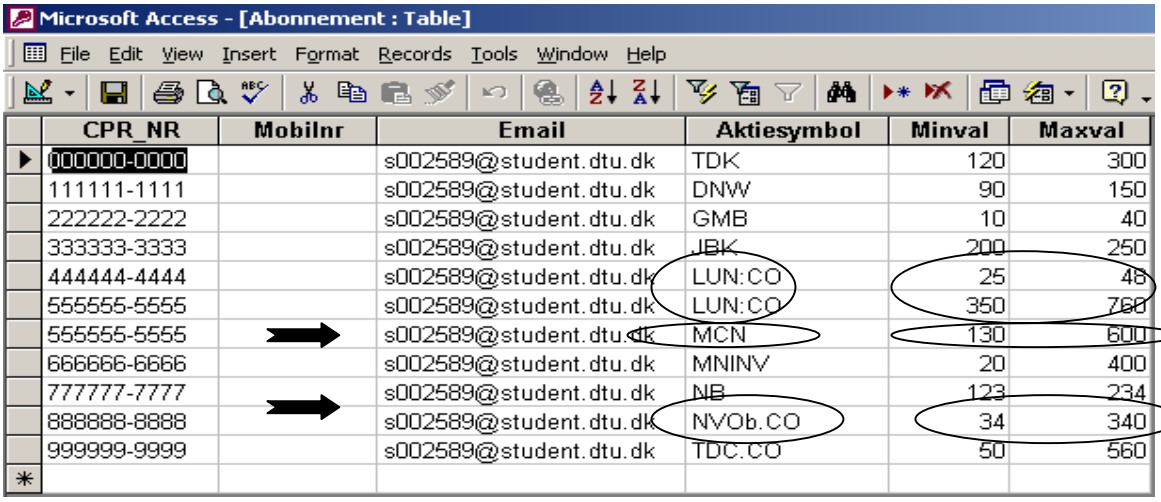
Figur 10.1.4 – 8 Beskederne

2. test

Vores formål med denne test er at bevise at intervalbegrænsningen (Minval, Maxval), som er i abonnementet, også bliver respekteret.

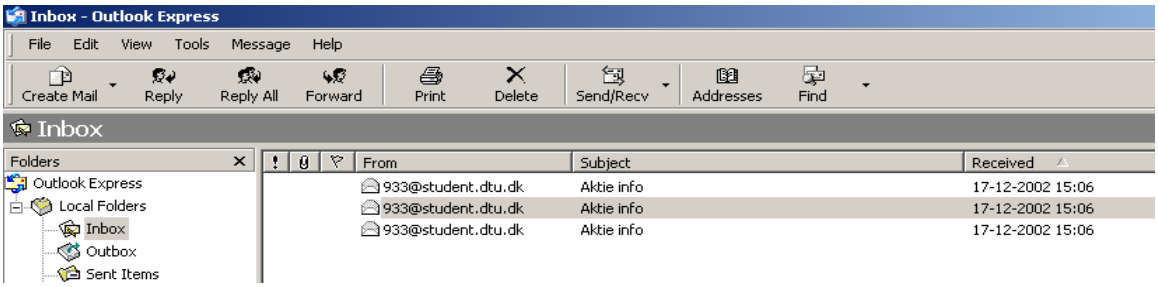
Vi sender igen en fil, se figur 10.1.9, med tilfældige værdier, og vi skal undersøge om begrænsningerne i tabellen (figur 10.1.10.) er respekteret.
Vi sætter nogle ringe og firkanter rundt om de værdier som er betydningsfulde. Firkanterne viser ”interessante” aktier, hvis værdier ikke er i intevalgrænsen. Derimod antyder ringene ”interessante” aktier hvis værdier er i intervalgrænsen.
I tabellen fra figur 10.1.10 vises med en pil de abonnenter som får tilsendt beskeder. Figur 10.1.12 viser indholdet af beskederne, og de afspejler de afsendte værdier.

Figur 10.1.9 Kildefilen

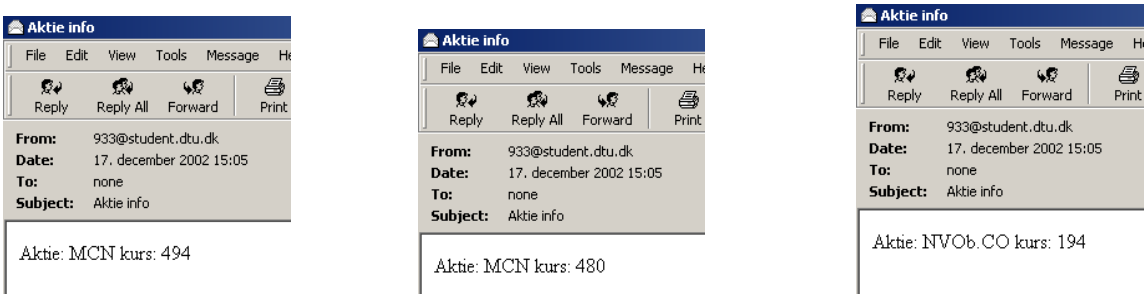


CPR_NR	Mobilnr	Email	Aktiesymbol	Minval	Maxval
000000-0000		s002589@student.dtu.dk	TDK	120	300
111111-1111		s002589@student.dtu.dk	DNW	90	150
222222-2222		s002589@student.dtu.dk	GMB	10	40
333333-3333		s002589@student.dtu.dk	JBK	200	250
444444-4444		s002589@student.dtu.dk	LUN:CO	25	48
555555-5555		s002589@student.dtu.dk	LUN:CO	350	760
555555-5555		s002589@student.dtu.dk	MCN	130	600
666666-6666		s002589@student.dtu.dk	MNINV	20	400
777777-7777		s002589@student.dtu.dk	NB	123	234
888888-8888		s002589@student.dtu.dk	NVOb.CO	34	340
999999-9999		s002589@student.dtu.dk	TDC.CO	50	560

Figur 10.1.10 Tabellen.



Figur 10.1.11 De modtagne mails.

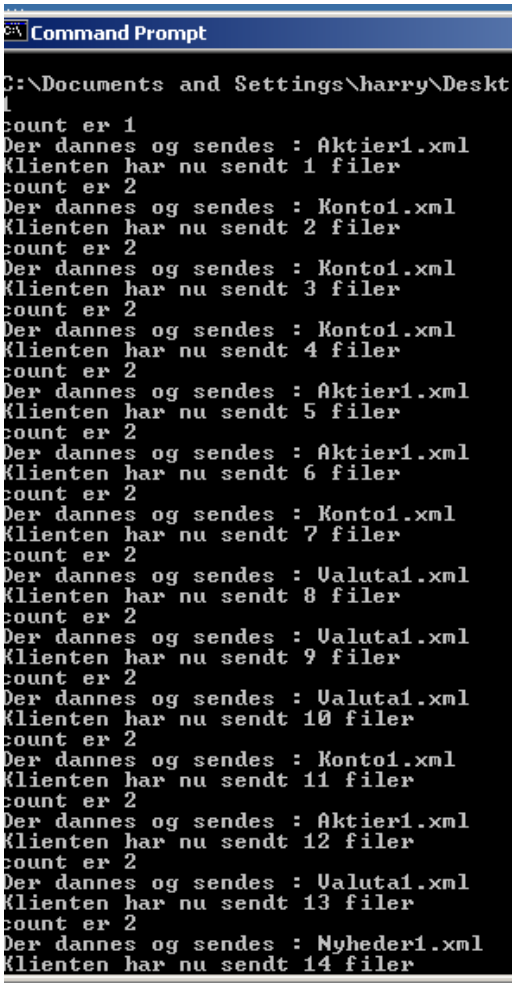


Figur 10.1.12 Beskedernes indhold.

3. test:

Formålet med denne test er at kontrollere navne, antal og ramdoms frekvens. I figur 10.1.13 ses at filerne udsendes randomæssigt og Severen modtage de respective kildefiler.

Klienten sender følgende filer:



og Serveren modtager disse:

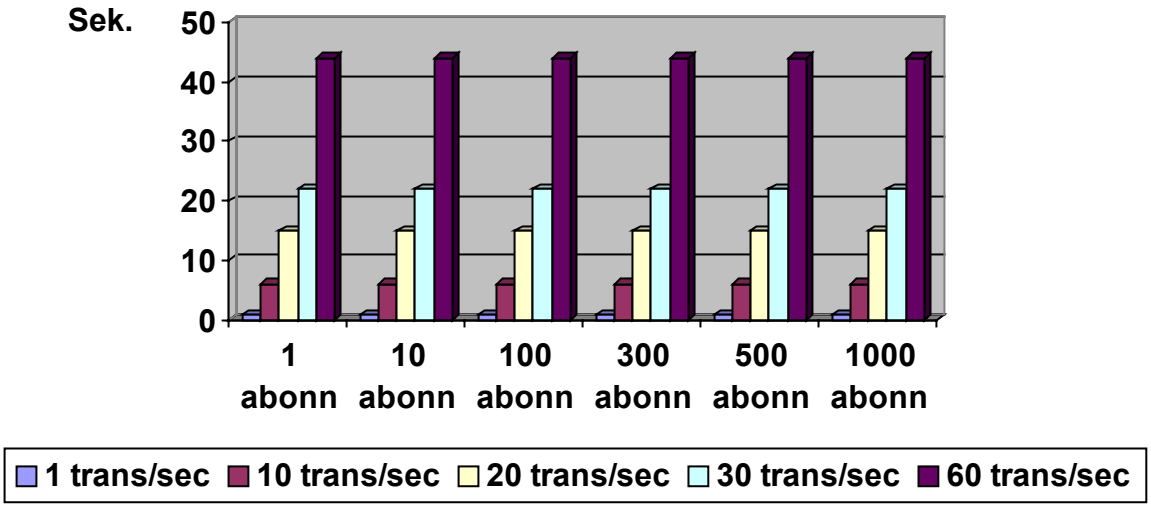
Figur 10.1.13 Udsendte og modtagne filer.

10.2 Testning af push-systemets hastighed

Denne test har som formål at finde ud af, hvor hurtigt *push-systemet* kan udføre jobbet, nemlig at modtage, udplukke og eventuelt sende data til abonnenterne. Systemet belastes med forskellige antal transaktioner og forskellige antal abonnementer.

For at lave denne test og til at forhøje nummeret af abonnenter og transaktioner, har vi fyldt databasen med forskellige antal abonnenter (og vi har lavet et program som kunne

gøre det) og vi har sendt forskellige filer med forskellige antal transaktioner. Vi har målt gennemsnittet af hastigheden for behandlingen af 10 modtagne filer ad gangen (som for eksempel indeholder 30 transaktioner per fil). Målingen af tiden er foretaget med et stopur. Vi starter uret når vi har modtaget den første fil, og vi stopper det når den sidste er behandlet.
Testen, som er repræsenteret i figur 10.2.1, viser at antallet af abonnenter ikke påvirker hastigheden af vores system fra 1 til 1000 abonnenter. Derimod påvirkes hastigheden af *push-systemet* af transaktionsantallet per sekund. Hastigheden bliver konstant ved forhøjelsen af abonnenter, hvilket vil sige at størrelsen af databasen ikke har nogen betydning, mens parsingen/udplukningen af informationer bliver langsommere, når der er flere transaktioner per sekund.

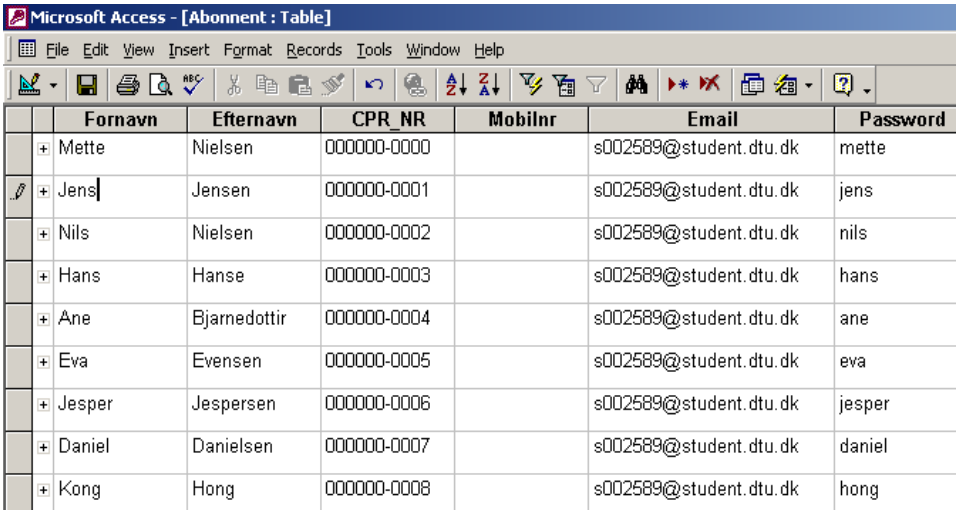


Figur 10.2.1 Grafen over testning.

10.3 Testningen af samarbejdet mellem websiden og databasen

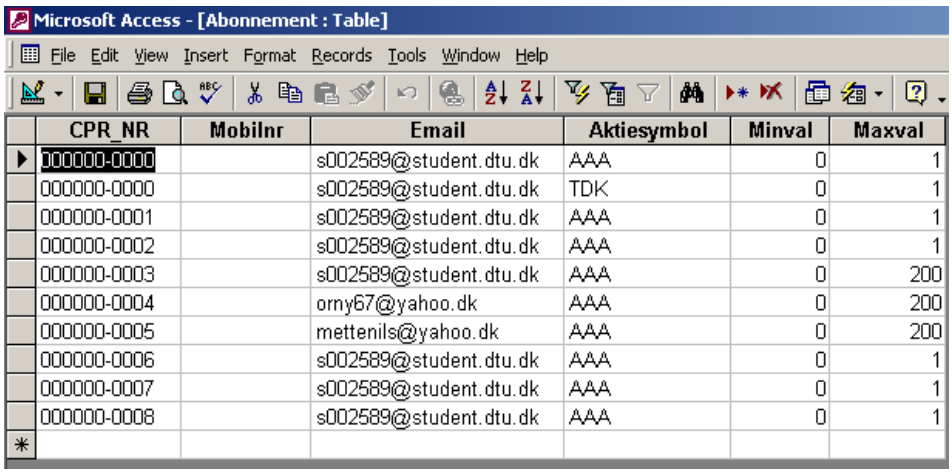
Formålet med denne test er at vise forbindelsen mellem databasen *AbService* og brugergrænsefladen, samt deres indbyrdes funktionalitet. Testen vil vise, at når en ny kunde (*Hans Viking*) tilmelder sig for eksempel aktieservice, da bliver dennes oplysninger gemt i de respektive tabeller (*Abonnent* og *Abonnement*). Samtidig bliver der sat en alert op (*j*) i tabellen *Aktier*, i den post hvor den valgte aktie forefindes. Vi viser de berørte tabeller før (se figurerne 10.3.1 -10.3.3) og efter (se figurerne 10.3.4 - 10.3.6) *Hans Vikings* tilmelding.
Testen viser også svaret fra brugergrænsefladen til *Hans Vikings* tilmelding til en aktie (Maconomy), og dette ses i Figur 10.3.7. I Figur 10.3.8 vises at databasen ikke accepterer den samme post to gange, i dette tilfælde aktien Maconomy, som *Hans Viking* allerede har tilmeldt sig. Dette er nemlig imod primærnøglens begrænsning (aktiesymbol og Cpr-nummer er en sammensat primærnøgle).

Hensigten med denne test er også at vise at alerten bliver sat ned (*n*), når en abonnent afmelder sig en service (Aktieservice), hvilket vises i Figur 10.3.10. Abonnentens oplysninger der bliver slettet fra tabellen (*Abonnement*), kan ses i Figur 10.3.9.



Fornavn	Efternavn	CPR_NR	Mobilnr	Email	Password
Mette	Nielsen	000000-0000		s002589@student.dtu.dk	mette
Jens	Jensen	000000-0001		s002589@student.dtu.dk	jens
Nils	Nielsen	000000-0002		s002589@student.dtu.dk	nils
Hans	Hanse	000000-0003		s002589@student.dtu.dk	hans
Ane	Bjarnedottir	000000-0004		s002589@student.dtu.dk	ane
Eva	Evensen	000000-0005		s002589@student.dtu.dk	eva
Jesper	Jespersen	000000-0006		s002589@student.dtu.dk	jesper
Daniel	Danielsen	000000-0007		s002589@student.dtu.dk	daniel
Kong	Hong	000000-0008		s002589@student.dtu.dk	hong

Figur 10.3.1 Tabellen *Abonnent* før tilmelding af *Hans Viking*.



CPR_NR	Mobilnr	Email	Aktiesymbol	Minval	Maxval
000000-0000		s002589@student.dtu.dk	AAA	0	1
000000-0000		s002589@student.dtu.dk	TDK	0	1
000000-0001		s002589@student.dtu.dk	AAA	0	1
000000-0002		s002589@student.dtu.dk	AAA	0	1
000000-0003		s002589@student.dtu.dk	AAA	0	200
000000-0004		orny67@yahoo.dk	AAA	0	200
000000-0005		mettenils@yahoo.dk	AAA	0	200
000000-0006		s002589@student.dtu.dk	AAA	0	1
000000-0007		s002589@student.dtu.dk	AAA	0	1
000000-0008		s002589@student.dtu.dk	AAA	0	1

Figur 10.3.2 Tabellen *Abonnement* før *Hans Viking* har tilmeldt sig en Aktie.

Microsoft Access - [Aktier : Table]

File Edit View Insert Format Records Tools Window Help

Aktienavn	Aktiesymbol	Alert	Gruppenavn	Emnenavn
American_Airlines	AAA	j	Amerikanske_aktier	Down_Jones
Bavarian_Bank	BVK	n	Europaeiske_aktier	Banker
Carlsberg_A	CARCa.CO	n	Danske_aktier	KFX
Carlsberg_B	CARCb.CO	n	Danske_aktier	KFX
Chr.Hansen	CHH	n	Danske_obligationer	Medicin_og_bioteknologi
Coloplast	COL	n	Danske_obligationer	KFX
COWI	CW	n	Danske_aktier	Finansieringsselskaber
Danske_Bank	DB.CO	n	Danske_obligationer	Banker
DHL	DHL	n	Amerikanske_aktier	Nasdaq
Danisco	DNS	n	Danske_obligationer	Foedevarer
Danware	DNW	n	Danske_aktier	Software
Genmab	GMB	n	Danske_aktier	KVX
Jyske_Bank	JBK	n	Danske_obligationer	Banker
H._Lundbeck	LUN:CO	n	Danske_obligationer	KFX
Maconomy	MCN	n	Danske_aktier	Banker
2M_Invest	MNINV	n	Danske_aktier	Software
Nordea	NB	n	Mest_efterspurgt	Banker
Novo_Nordisk_B	NVOb.CO	n	Danske_obligationer	Banker
TDC	TDC.CO	n	Danske_aktier	Telekommunikation
Topdanmark	TDK	j	Danske_aktier	Forsikring

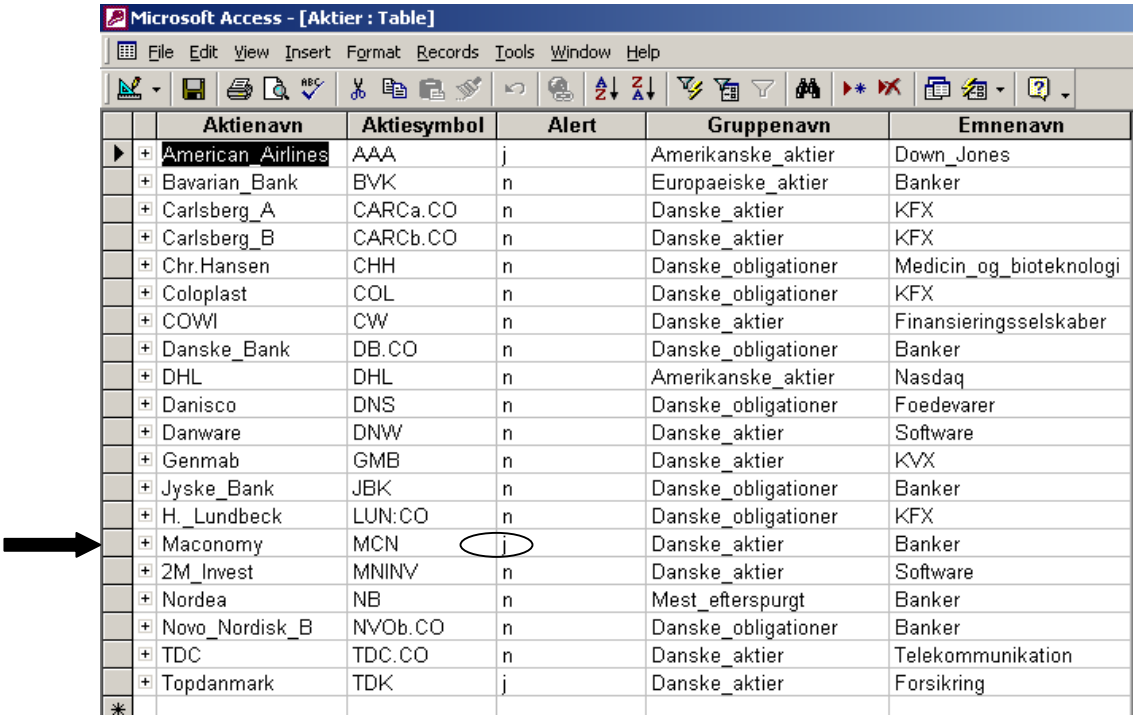
Figur 10.3.3 Tabellen *Aktier* før tilmelding af Hans Viking.

Microsoft Access - [Abonnent : Table]

File Edit View Insert Format Records Tools Window Help

Fornavn	Efternavn	CPR_NR	Mobilnr	Email	Password
Mette	Nielsen	000000-0000		s002589@student.dtu.dk	mette
Jens	Jensen	000000-0001		s002589@student.dtu.dk	jens
Nils	Nielsen	000000-0002		s002589@student.dtu.dk	nils
Hans	Hanse	000000-0003		s002589@student.dtu.dk	hans
Ane	Bjarnedottir	000000-0004		s002589@student.dtu.dk	ane
Eva	Evensen	000000-0005		s002589@student.dtu.dk	eva
Jesper	Jespersen	000000-0006		s002589@student.dtu.dk	jesper
Daniel	Danielsen	000000-0007		s002589@student.dtu.dk	daniel
Kong	Hong	000000-0008		s002589@student.dtu.dk	hong
Ornella	Ardoino	000000-0009		s002589@student.dtu.dk	harry
Hans	Viking	040103-1111	51515151	hansviking@hotmail.com	viki
Ornella	Ardoino	250167-2324		orny67@yahoo.dk	ane

Figur 10.3.4 Tabellen *Abonnent* efter tilmelding af Hans Viking.

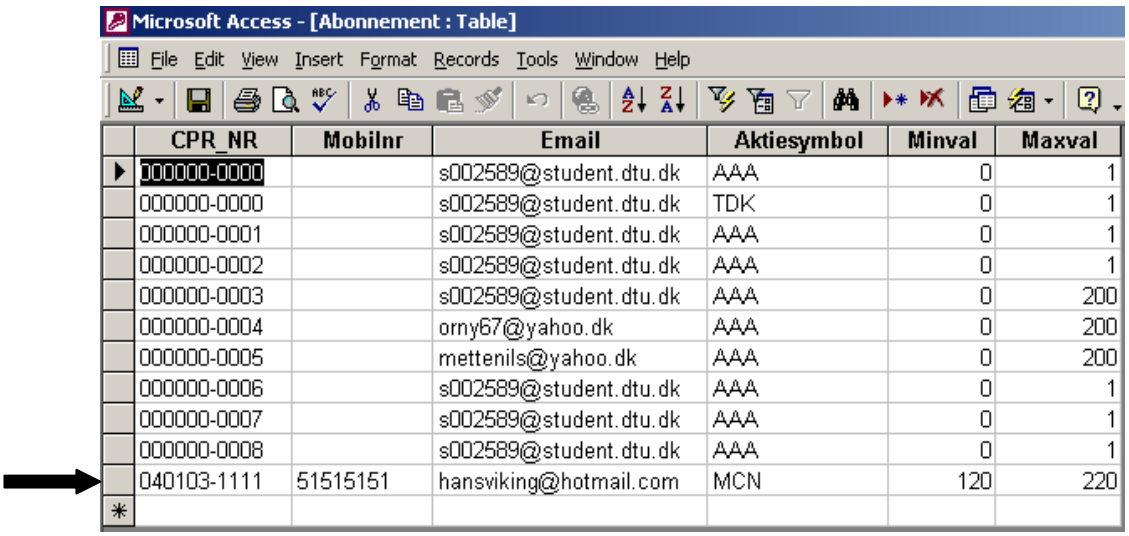


Microsoft Access - [Aktier : Table]

File Edit View Insert Format Records Tools Window Help

Aktienavn	Aktiesymbol	Alert	Gruppenavn	Emnenavn
American_Airlines	AAA	j	Amerikanske_aktier	Down_Jones
Bavarian_Bank	BVK	n	Europaeiske_aktier	Banker
Carlsberg_A	CARCa.CO	n	Danske_aktier	KFX
Carlsberg_B	CARCb.CO	n	Danske_aktier	KFX
Chr.Hansen	CHH	n	Danske_obligationer	Medicin_og_bioteknologi
Coloplast	COL	n	Danske_obligationer	KFX
COWI	CW	n	Danske_aktier	Finansieringsselskaber
Danske_Bank	DB.CO	n	Danske_obligationer	Banker
DHL	DHL	n	Amerikanske_aktier	Nasdaq
Danisco	DNS	n	Danske_obligationer	Foedevarer
Danware	DNW	n	Danske_aktier	Software
Genmab	GMB	n	Danske_aktier	KVX
Jyske_Bank	JBK	n	Danske_obligationer	Banker
H._Lundbeck	LUN:CO	n	Danske_obligationer	KFX
Maconomy	MCN	j	Danske_aktier	Banker
2M_Invest	MNINV	n	Danske_aktier	Software
Nordea	NB	n	Mest_efterspurgt	Banker
Novo_Nordisk_B	NVOb.CO	n	Danske_obligationer	Banker
TDC	TDC.CO	n	Danske_aktier	Telekommunikation
Topdanmark	TDK	j	Danske_aktier	Forsikring

Figur 10.3.5 Tabellen *Aktier* efter tilmelding af Hans Viking til Maconomy aktien.



Microsoft Access - [Abonnement : Table]

File Edit View Insert Format Records Tools Window Help

CPR_NR	Mobilnr	Email	Aktiesymbol	Minval	Maxval
000000-0000		s002589@student.dtu.dk	AAA	0	1
000000-0000		s002589@student.dtu.dk	TDK	0	1
000000-0001		s002589@student.dtu.dk	AAA	0	1
000000-0002		s002589@student.dtu.dk	AAA	0	1
000000-0003		s002589@student.dtu.dk	AAA	0	200
000000-0004		orny67@yahoo.dk	AAA	0	200
000000-0005		mettenils@yahoo.dk	AAA	0	200
000000-0006		s002589@student.dtu.dk	AAA	0	1
000000-0007		s002589@student.dtu.dk	AAA	0	1
000000-0008		s002589@student.dtu.dk	AAA	0	1
040103-1111	51515151	hansviking@hotmail.com	MCN	120	220

Figur 10.3.6 Tabellen *Abonnement* efter tilmeldingen af Hans Viking til Maconomy aktien.

Figur 10.3.7 Svar fra websiden til Hans Vikings tilmelding til aktien "Maconomy".

Figur 10.3.8 Svar til Hans Viking om tilmelding til en aktie, han allerede er tilmeldt (Maconomy).

Microsoft Access - [Abonnement : Table]

File Edit View Insert Format Records Tools Window Help

CPR_NR	Mobilnr	Email	Aktiesymbol	Minval	Maxval
000000-0000		s002589@student.dtu.dk	AAA	0	1
000000-0001		s002589@student.dtu.dk	AAA	0	1
000000-0002		s002589@student.dtu.dk	AAA	0	1
000000-0003		s002589@student.dtu.dk	AAA	0	200
000000-0004		orny67@yahoo.dk	AAA	0	200
000000-0005		mettenils@yahoo.dk	AAA	0	200
000000-0006		s002589@student.dtu.dk	AAA	0	1
000000-0007		s002589@student.dtu.dk	AAA	0	1
000000-0008		s002589@student.dtu.dk	AAA	0	1
040103-1111	51515151	hansviking@hotmail.com	MCN	120	220

Figur 10.3.9 Tabellen *Abonnement* efter afmelding af Mette Nielsen til TDK aktien.

Microsoft Access - [Aktier : Table]

File Edit View Insert Format Records Tools Window Help

Aktienavn	Aktiesymbol	Alert	Gruppenavn	Emnenavn
American_Airlines	AAA	j	Amerikanske_aktier	Down_Jones
Bavarian_Bank	BVK	n	Europaeiske_aktier	Banker
Carlsberg_A	CARCa.CO	n	Danske_aktier	KFX
Carlsberg_B	CARCb.CO	n	Danske_aktier	KFX
Chr.Hansen	CHH	n	Danske_obligationer	Medicin_og_bioteknologi
Coloplast	COL	n	Danske_obligationer	KFX
COWI	CW	n	Danske_aktier	Finansieringsselskaber
Danske_Bank	DB.CO	n	Danske_obligationer	Banker
DHL	DHL	n	Amerikanske_aktier	Nasdaq
Danisco	DNS	n	Danske_obligationer	Foedevarer
Danware	DNW	n	Danske_aktier	Software
Genmab	GMB	n	Danske_aktier	KVX
Jyske_Bank	JBK	n	Danske_obligationer	Banker
H._Lundbeck	LUN:CO	n	Danske_obligationer	KFX
Maconomy	MCN	j	Danske_aktier	Banker
2M_Invest	MNINV	n	Danske_aktier	Software
Nordea	NB	n	Mest_efterspurgt	Banker
Novo_Nordisk_B	NVOb.CO	n	Danske_obligationer	Banker
TDC	TDC.CO	n	Danske_aktier	Telekommunikation
Topdanmark	TDK	n	Danske_aktier	Forsikring

Figur 10.3.10 Tabellen *Aktier* efter afmelding af Mette Nielsen til TDK aktien.

10.4 Delkonklusion

Udfra testningerne kan konkluderes at *push-systemets* funktionalitet overholdes, med hensyn til hvad der blev påkrævet. Systemet modtager informationer fra kildefilden og derefter udsendes de relevante informationer til abonnenterne. Antallet af abonnenterne påvirker ikke *push-systemets* hastighed, derimod antallet af transaktioner.
Det ses også at samarbejdet mellem og databasen og websiden lever op til forventningerne, og at disses funktionaliteter er optimale. Dermed er de forventede resultater opnået.

11. Brugervejledning

Vi har delt dette kapitel i 3 afsnit: I det første gennemgår vi installationen af de værktøjer systemet har behov for. Andet afsnit forklarer hvordan vores program skal installeres, og til sidst redegøres for hvordan hele systemet bruges.

11.1 Installation af værktøjer

For at installere og køre vores system er der behov for at følgende er installeret i computeren: Platformen i computeren kan være en af disse; Microsoft Windows 98/NT/2000/XP.
En række programmer og arbejdsmiljøer skal installeres i computeren.
Da Databasen skal implementeres er der behov for at installere en eller flere af de nævnte værktøjer i kapitel 4 (MySQL, Microsoft Access som er en del af Microsoft Office pakken).
For at køre Java-baserede programmer skal Java-værktøjerne installeres. (J2SDKEE eller JDK v. 1.3 eller efter)
For at brugergrænsefladen kan benyttes skal der opsættes en *Personal Web Manager*.

I det følgende vil vi gennemgå måden hvorpå softwaren installeres på en Windows NT platform:

11.1.1 Installation af MySQL

Hvis man vil hellere installere ens database på en anden værktøj end MySQL, man har ikke brug for denne afsnit.
Først skal filen MySQL downloades fra http://www.mysql.com/downloads/. Herfra kan hentes den udgave man har behov for. (i vores tilfælde version Max 4.0*). Man opretter et katalog til mySQL i hardisken, f.eks. *c:\mysql* (anbefales) og filen *mysql-<version>-win.zip* kopieres til dette bibliotek. Udpak (unzip) zip filen til kataloget *c:\mysql\mysql-<version>-win* og kør filen *setup.exe* i det katalog zip filen blev unzippet til (*c:\mysql\mysql-<version>-win*).
For at igangsætte *InnoDB* typetabeller skal der i *MySQLadmin* findes *my.ini*. og tilføjes følgende: *[mysql] default-table-type=innodb.*

Start (konfiguration) af MySQL

MySQL kører som en proces på maskinen, på NT er dette en såkaldt service der oprettes hver gang maskinen startes. For at begynde at anvende MySQL man står i biblioteket *c:\mysql\bin* hvor man blot skrive *mysql*.
Hvis man nu åbner kontrolpanelet og går ind under *services* (*tjenester,* på dansk, i Windows 2000), vil man se at der er kommet en service med navnet mySQL, som er startet.

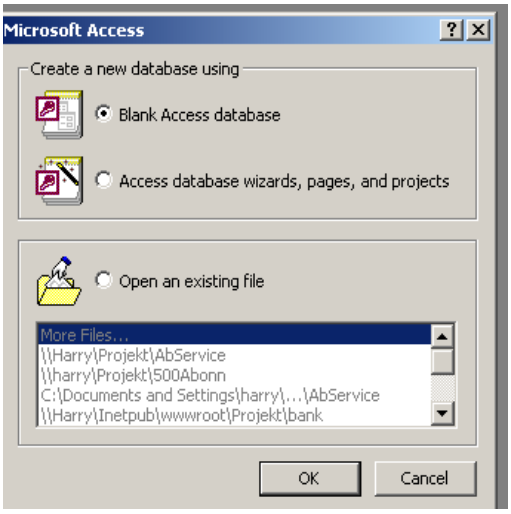
Det er påkrævet, at mySQL servicen kører (usynligt) i baggrunden, så længe man ønsker at arbejde med MySQL, i modsætning til f.eks. MS-ACCESS, som startes hver gang man bruger det.

Installation af ODBC driver til mySQL

Kopier filen *myodbc-2.50.37-nt.zip* til *c:\mysql* udpak (unzip) zip filen til kataloget *c:\mysql\ myodbc-2.50.37-nt* kør filen *setup.exe* i det katalog zip blev unzippet til (*c:\mysql\myodbc-2.50.37-nt*)
Hvis man nu åbner kontrolpanelet og går ind under ODBC, vil man se at der er kommet en ODBC driver med navnet mySQL.

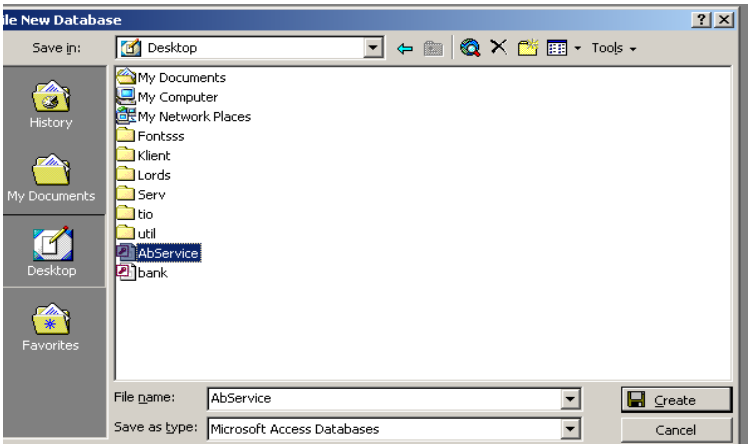
11.1.2. SQL i MS Access

Hvis man vil oprette en ny database, kan man følge de nedenstående trin.



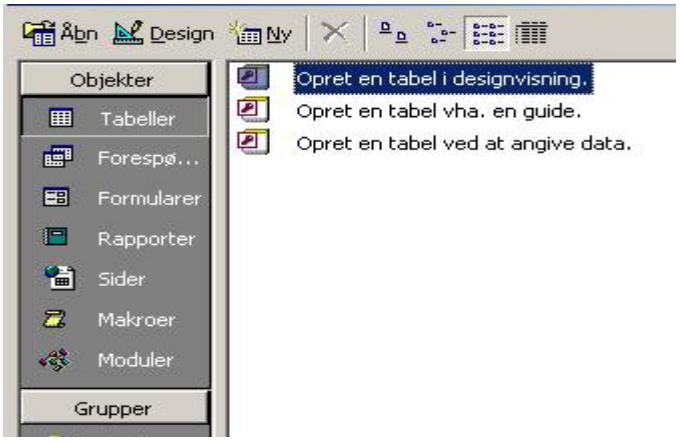
Start MS-ACCESS og opret en ny tom database.

Figur 11.1.2.1 Databasen i Access trin 1.



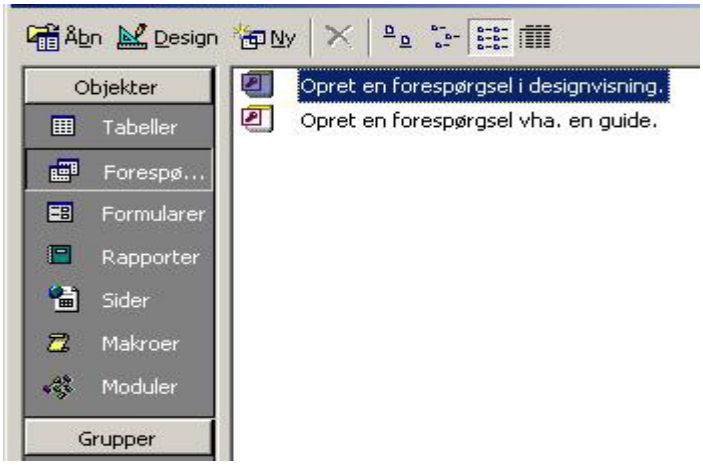
Databasen døbes for eksempel *AbService.mdb* og placer den et hensigtsmæssigt sted, for eksempel i mappen *c:\Projekt*.

Figur 11.1.2.2 Databasen i Access trin 2.



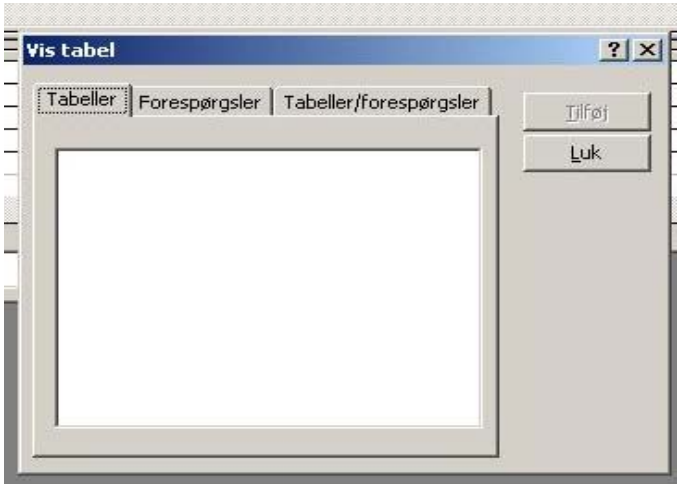
vælg nu "forespørgsler"

Figur 11.1.2.3 Database i Access trin 3.



vælg nu "Opret en forespørgsel i designvisning"

Figur 11.1.2.4 Database i Access trin 4.



Vælg ”luk”

Figur 11.1.2.5 Databasen i Access trin 5.

Og vælg SQL visning i menuen.

Figur 11.1.2.6 Databasen i Access trin 6.

Nu fås et SQL vindue

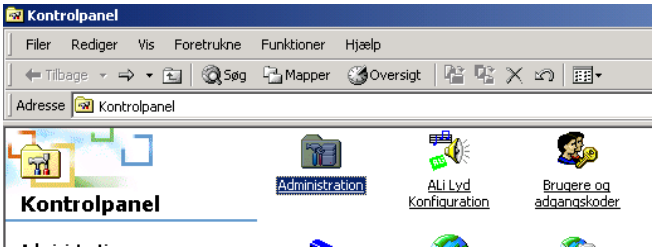
Figur 11.1.2.7 Databasen i Access trin 7.

Hvor man kan skrive SQL sætninger, som udføres ved at vælge "det røde udråbstegn !"

Figur 11.1.2.8 Database i Access trin 8.

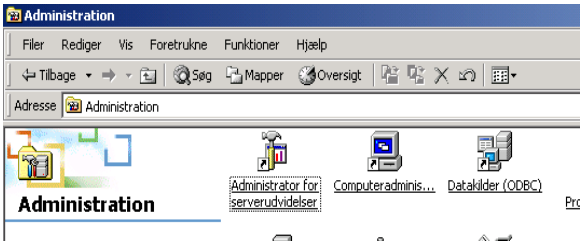
Configuration af Access driver

Databasen skal være tilgængelig for websiderne, og for at kunne se disse sider i en browser, konfigureres en Access Database Driver til databasen i ”ODBC Data Source Administrator” dialogsboks. Man finder dette via Start -> Settings -> Control Panel.



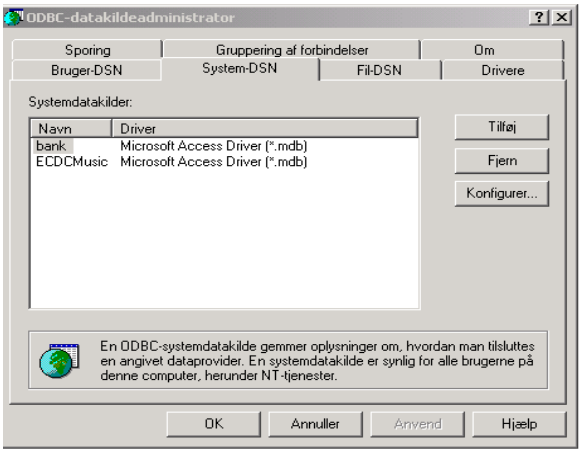
Derefter trykkes på Administrator,

Figur 11.1.2.9 Driver til databasen trin 1.



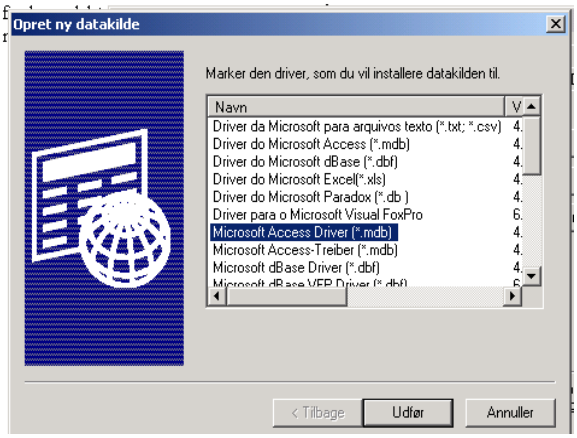
og her trykkes på *ODBC Datakilder*:

Figur 11.1.2.10 Driver til databasen trin 2.



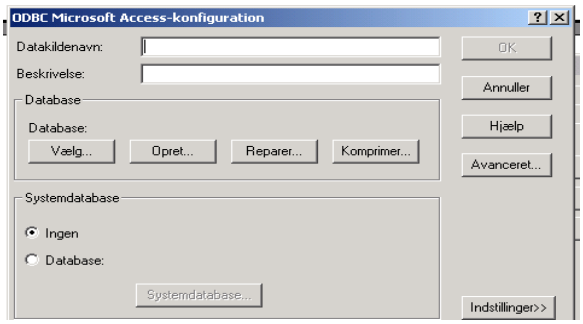
Herfra vælger man *SystemDSN's* fane og der trykkes på "Tilføj", og nedenstående boks (Figur 11.1.2.12) viser sig:

Figur 11.1.2.11 Driver til databasen trin 3.



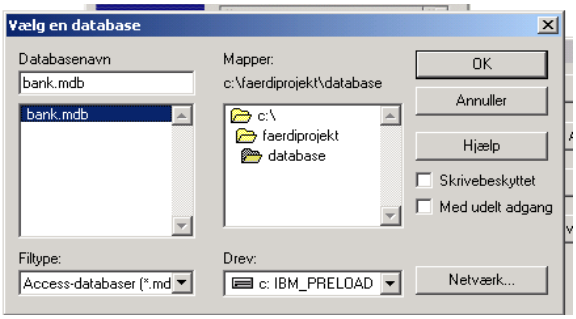
Herfra vælger man Microsoft Access Driver(*.mdb) og der trykkes på "Udfør":

Figur 11.1.2.12 Driver til databasen trin 4.



I den kommende boks kan i feltet "Datakildenavn" skrives "AbService" og eventuelt en beskrivelse af databasen i feltet "Beskrivelse". Bagefter trykkes på "vælg" for at vælge den tilgængelige database.

Figur 11.1.2.13 Driver til databasen trin 5.



Man finder databasen i dialogboksen til højre. Derefter trykker man på OK for at acceptere databasen, og OK i "ODBC Microsoft Access Konfiguration", som er den næste boks der kommer frem. Til sidst trykkes OK igen i "ODBC Datakilder Administrator" for at afslutte opsætningen af driveren.

Figur 11.1.2.14 Driver til databasen trin 6.

11.1.3 Installation af Java

JDK 1.3 eller nyere version kan downloades flere steder på www for eksempel på

- JDK 1.3 for Windows 95/98 (20 MB)

http://java.sun.com/products/jdk/1.3/download-windows.html

Efter at JDK eller J2SDKSE er blevet hentet får man denne i én pakket fil. Det er en selv-udpakkende fil, og man skal blot eksekvere den.
Programmet starter *Install Shield* (det program, der bruges ved installering af mange moderne pc-programmer), stiller nogle enkelte spørgsmål om i hvilket katalog JDK skal placeres. Pc'en skal genstartes før installeringen er helt klar.
Derefter skal man tilføje følgende *JAR* filer til c:\jdk1.3.1\jre\lib\ext\: ***xerces.jar***, **mail.jar**, **activation.jar.**
Disse filer kan downloades henholdsvis fra følgende adresser: For at hente *xerces.jar*, skal man downloade *Xerces-J-bin.1.4.4.zip* fra:
http://xml.apache.org/dist/xerces-j/.
For at hente *mail.jar*, skal man downloade *JavaMail*-pakken fra:
http://java.sun.com/products/javamail/, og for at hente ***activation.jar*** downloades pakken JavaBeans Activation Framework fra:
http://java.sun.com/products/beans/glasgow/jaf.html .

Tilføjelse til PATH
Programmerne `javac` og `java` benyttes fra en DOS-prompt. For at anvende disse java kommandoer skal kataloget, hvor disse programmer ligger, tilføjes til Systemets/PC'ens PATH[17]-miljøvariabel. Det gøres ved at tilføje en linie til filen `c:\autoexec.bat`:

- *c:\autoexec.bat* tages ind i en editor for eksempel *notepad*.

[17] Hver gang et program skal startes fra DOS-prompten, så leder *Windows* i de kataloger der står i *PATH'en*, indtil det finder filen. Hvis filen ikke findes i et katalog i *PATH'en*, da kan det ikke umiddelbart startes fra DOS-prompten [men filen kan sagtens findes i filsystemet - det er ikke alle kataloger, der er med i *PATH'en*].

- Tilføjes en linie sidst i filen `SET PATH=%PATH%;C:\JDK1.3\BIN`
- Filen gemmes og *notepad* forlades.
- Pc'en skal genstartes
- For at konstatere at PATH indeholder *JDK1.3\BIN*, kan man skrive "`set`" i DOS-prompten og finde ud af det.

Programmerne `javac` og `java` ligger i kataloget *C:\JDK1.3\BIN*, hvis man har foretaget en standard-installation. Hvis man har valgt at installere JDK i et andet katalog eller på et andet drev, så skal man naturligvis rette i ovenstående tilføjelse til `autoexec.bat`.
For at ændre PATH i Windows NT 2000, skal man gå ind i *Denne Computer*s egenskaber, og derfra vælger man "**Avanceret**" fane. Her findes "**Miljøvariabler**", og ved at trykke på denne kommer en boks frem hvor man vælger **PATH** og "**Rediger**" den ved at tilføje "**;** den nye sti"; i dette tilfælde **`C:\JDK1.3\BIN`**

11.1.4 Installation af J2EE version 1.3.1.

Denne foregår på samme måde som i JSDK, og man downloader filen fra denne adresse: http://java.sun.com/j2ee/sdk_1.2.1
Opsætningen af Classpath'en er mere kompliceret. Der skal angives et *JAVA_HOME* og *J2EE_HOME*, samt stien til Java kommandoerne. JSDK skal i forvejen være installeret. For at kunne tjekke at J2EE er installeret, angiver man denne komando: *j2ee –verbose*, og det vil komme frem at serveren er aktiv og porten er klar til modtagelse, hvilket vil sige at en forbindelse er åben.

11.1.5 Personal Web Manager opsætning

Hvis man i forvejen ikke har en webserver installeret er det nødvendig at gøre dette, for at kunne have en webside kørende.
På web-serveren (Personal Web Manager) skal der defineres et såkaldt "virtuel directory", som er en kobling imellem den fysiske placering af de html/asp dokumenter der skal vises og den URL-adresse der skrives i browseren.

Personal Web Manager findes i kontrolpanelet, mappen "Administration". Her vælges "Administration".

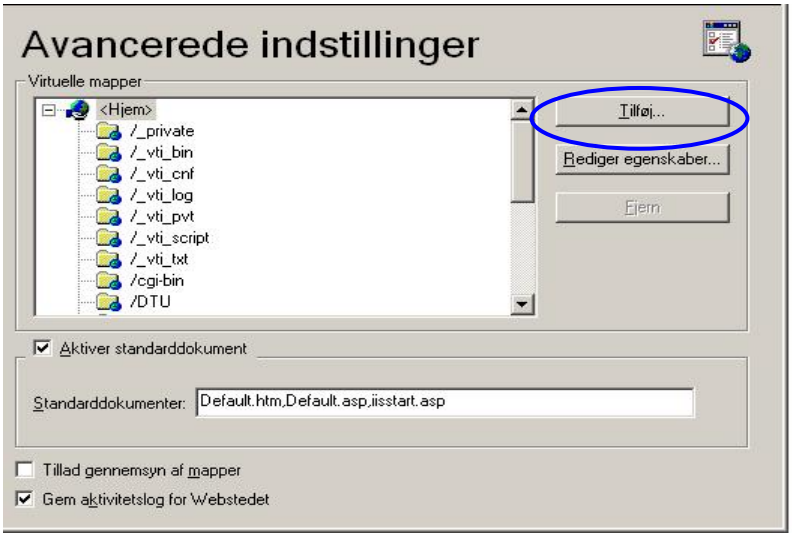
Figur 11.1.5.1 Personal Web Manager trin 1.

Vælg "Personal Web Manager" for at opsætte din personlige webserver.

Figur 11.1.5.2 Personal Web Manager trin 2.

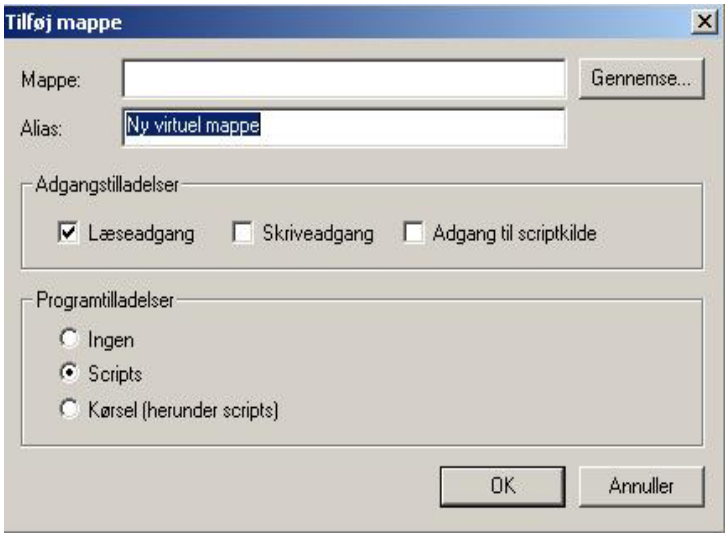
Herfra vælger man "Avanceret"

Figur 11.1.5.3 Personal Web Manager trin 3.



Vælg "Tilføj".

Figur 11.1.4.4 Personal Web Manager trin 4.



Figur 11.1.5.5 Personal Web Manager 5.

Her vælges den fysiske mappe med "Gennemse", som er den directory hvor man vil placere sin hjemmeside (default.asp), alias det logiske sted, der skrives i URL'en, for eksempel Projekts web. Herfra vil hjemmesiden blive tilgængelig for browseren ved for eksempel at indtaste følgende i adressefeltet: http://harry/Projekt/default.asp

11.2 Installering af Abonnementsservice

I Cd-rom har vi alle de nødvendige koder for at køre Abonnementssystemet. Koderne er samlet i forskellige mapper.

I mappen Klient findes koden til at sende de fiktive kildefiler, som simulerer bank-, børs- og nyheds-verdenen. Klienten har brug for en database (som findes i mappen *Bankverden*), der indeholder de forskellige atributter såsom aktiesymboler, kontonummer, nyhedsemne og valutanavne. Klienten bliver installeret og igangsættes, i den computer man vil køre på, ved at trykke *AbKlient.bat* som er inkluderet i Cd-rommen. Desuden skal der installeres en Access driver til databasen (*bank.mdb*). For installeringen af denne driver, se venligst afsnit 11.1.2.

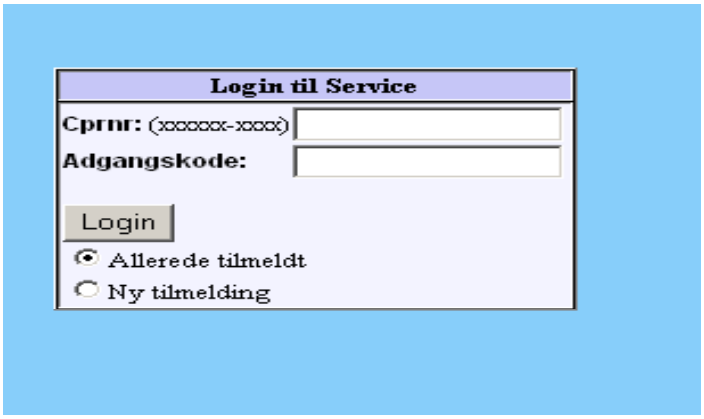
I mappen Server har vi koderne til push-systemet og webserveren, som installeres og igangsættes på samme måde som for Klienten. Nu trykkes i stedet ikonet *AbServer.bat*, som er også inkluderet i Cd-rommen, og ligeledes skal der også installeres en Access driver til databasen (*AbService.mdb*). Se igen afsnit 11.1.2.

11.3 Navigering i websiderne

I dette afsnit vil vi give et sammendrag af brugergrænsefladen i abonnementsservice

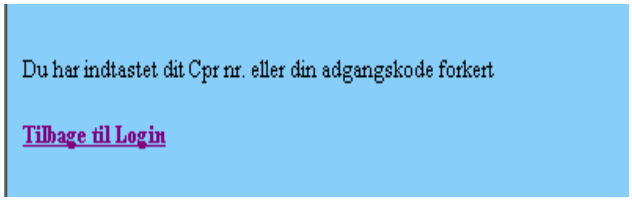
Hvis kunden vil tilmelde sig en af de tilbudte abonnementsservices, bliver vedkommende henvist (fra pengeinstituttet) til vores hovedside, og bedt om at oplyse sit cpr-nummer og adgangskode. Samme sted oplyses om vedkommende allerede er tilmeldt servicen eller ej.

Derefter præsenterer systemet kunden for en ny side, hvor der kan vælges mellem følgende forskellige services: *Aktier*, *Nyheder*, *Konti* og *Valuta*. Er kunden allerede tilmeldt, kan pågældende også her afmelde en service eller få en oversigt over sine tilmeldinger.
I det følgende vil nogle screenshots vise navigeringen i brugergrænsefladen med eksempler.
Figur 11.3.1 er forsiden af Abonnement Service, hvortil abonnenten kommer via et 'link' fra pengeinstituttet. Her skal udfyldes to felter: **Cprnr**. og **Adgangskode**.

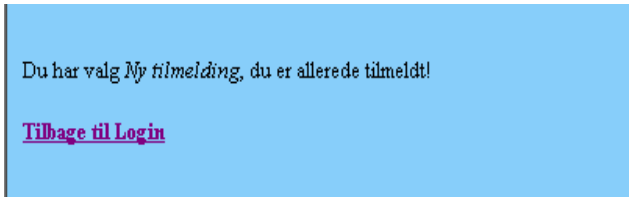


Figur 11.3.1 Login.

Adgangskoden har vedkommende i forvejen fået fra pengeinstituttet. Dette er for at tage hensyn til sikkerheden. Informationer er strengt personlig, og ved at tjekke om adgangskode tilhører cpr-nummeret er vi næsten sikre på, at personen er den som skal have de oplysninger der bliver bedt om.
Der skal også markeres om abonnenten allerede er tilmeldt eller skal tilmelde sig. Dernæst trykkes på *login* for at gå videre.
Hvis der bliver indtastet noget forkert eller hvis adgangskoden, cpr-nummer eller andre informationer ikke passer sammen i tilmeldt eller ny tilmelding, kommer der en fejlmeddelelse: Se figur 11.3.2 og 11.3.3



Figur 11.3.2 Fejlmeddelelse: *Alleredetilmeldt*.



Figur 11.3.3 Fejlmeddelelse: Ny tilmelding.

Hvis kunden glemmer at udfylde nogle af felterne, vil denne blive mindet om at udfylde de blanke felter. Et eksempel er givet nedenunder i det tilfælde man har glemt at indtaste sit Cpr-nummer, og man får følgende meddelelse i figur 11.3.4.

Figur 11.3.4 Fejlmeddelelse: Manglende Cpr-nummer.

Hvis kundens oplysninger er godkendt bliver denne budt velkommen, se figur 11.3.5.

Figur 11.3.5: Velkomstside.

Er kunden *ny tilmeldt* eller *allerede tilmeldt*, sytemet skifter til en ny side (se figur 11.3.6 og 11.3.7 henholdsvis) For konto servicen kræves at kunden allerede har konto registreret i *AbService*. Forskellen mellem de nævnte to sider er at hvis abonnenten allerede er tilmeldt, er der mulighed for at se den pågældendes oversigt eller foretage en afmelding. Vælges en af disse henvises til en oversigt, hvor abonnenten kan se sine tilmeldinger eller se en liste over de services der kan afmeldes. Se Figur 11.3.8 og 11.3.9

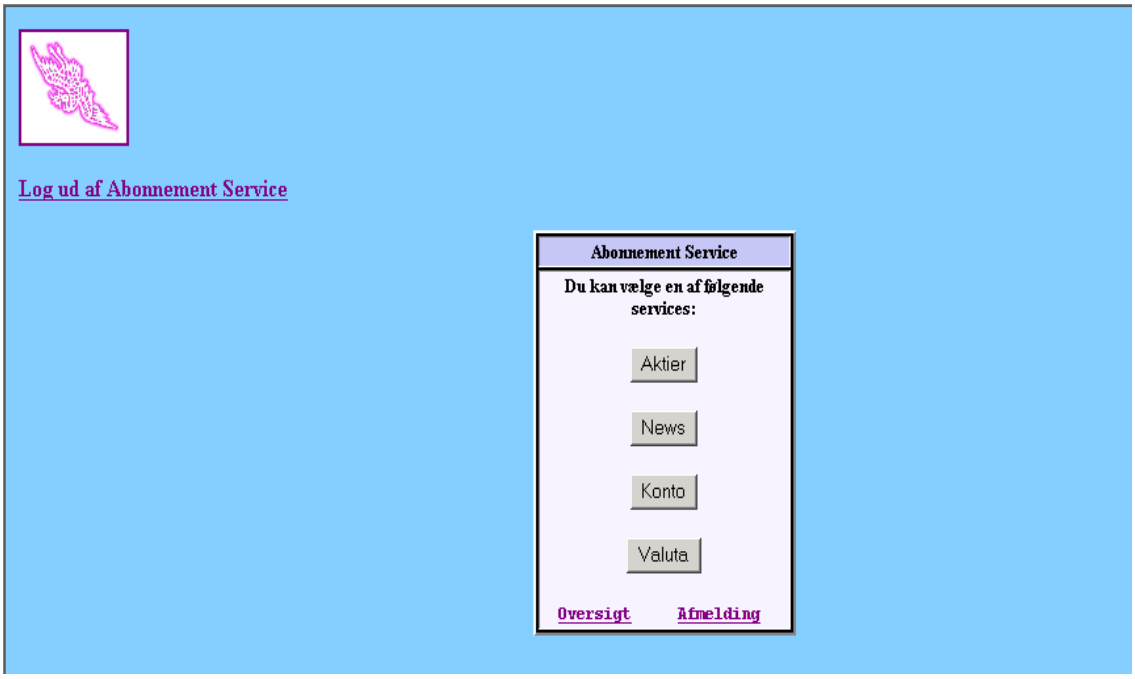
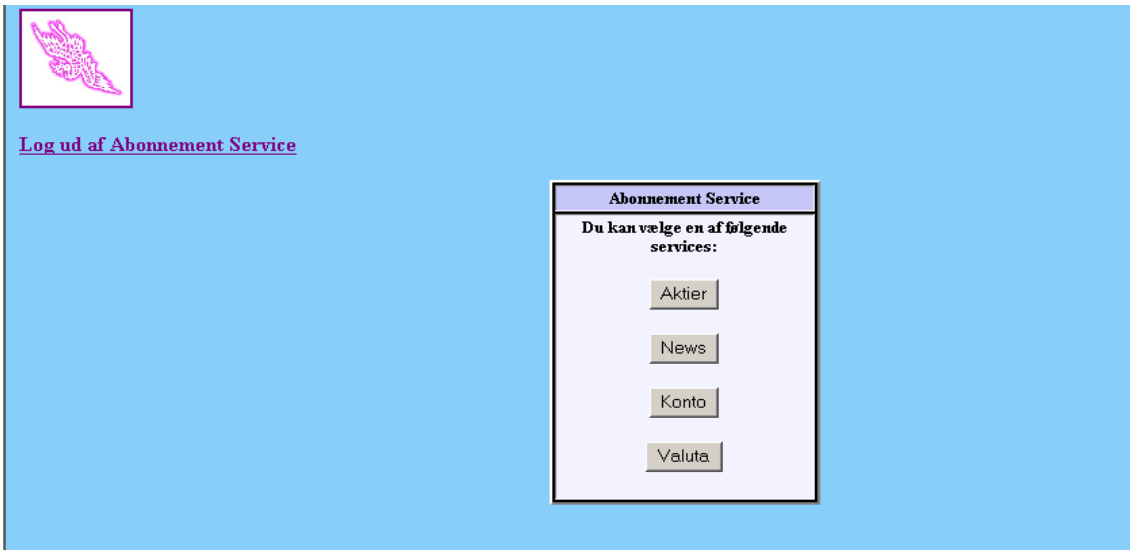


Figure11.3.6. Allerede tilmeldt



Figur 11.3.7 Ny tilmelding.

Figur 11.3.8 Oversigt over tilmeldt services.

Figur 11.3.9 Afmeldingsoversigt.

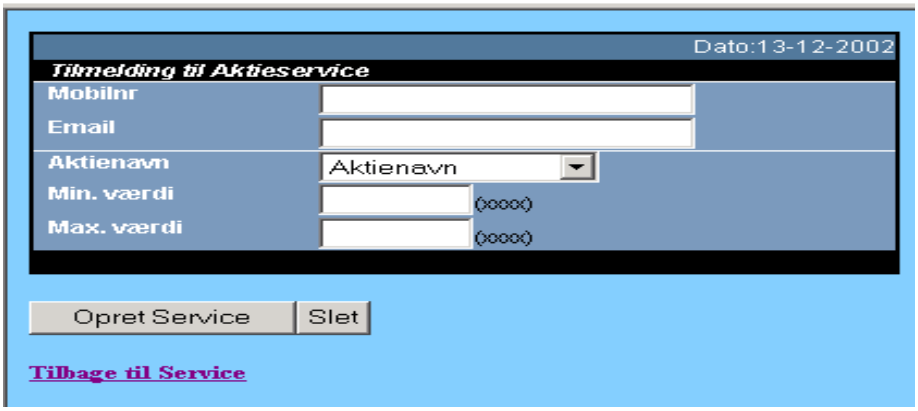
Kunden har mulighed for at tilmelde sig en af de fire services ved at vælge mellem *Aktier*, *Konto*, *News*, *Valuta* (se Figur 11.3.6 og 11.3.7).
Hvis abonnenten vælger at tilmelde sig *Aktier*, får vedkommende en ny side. Her har vi valgt, for at gøre det nemmere for kunden at finde den aktie vedkommende er interesseret i, at dele aktierne op i aktiegruppe og aktieemne. (se Figur 11.3.10).

Figur 11.3.10 Aktietilmelding.

Efter kunden har valgt enten emner eller grupper, bliver vedkommende sendt automatisk til en side hvor der skal vælges gruppe eller emne. (se Figur 11.3.11).

Figur 11.3.11: Vælg aktier via gruppenavn

Når abonnenten har valgt en bestemt aktie skal vedkommende definere sit abonnement og her er det lidt forskelligt hvis man er nytilmeldt eller ej, da man som ny tilmeldt skal give forskellige oplysninger om sig selv. Se Figurer 11.3.12 og 11.3.13



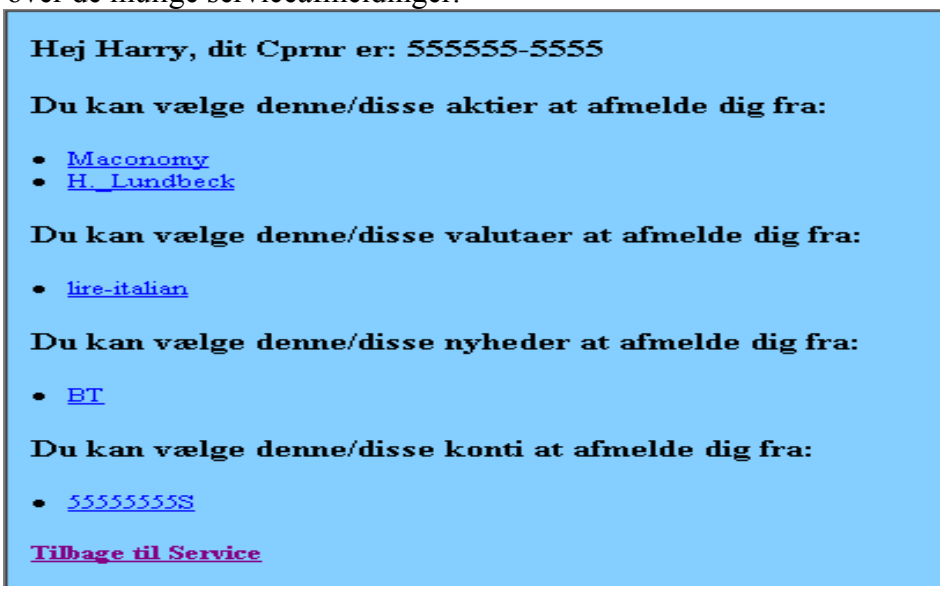
Figur 11.3.12 Aktietilmelding for *allerede tilmeldte*.

Figur 11.3.13 Aktietilmelding: Som *nytilmeldt*.

Det samme kan næsten siges for nyheder, valuta og konto. Derfor vises her kun for aktier. Efter man har indsat de forskellige oplysninger trykkes *opret service* for at tilmelde sig denne service.
Efter hver tilmelding får kunden en side, hvor de oplysninger der er givet og de valg der er truffet, vises. (se Figur 11.3.14) .

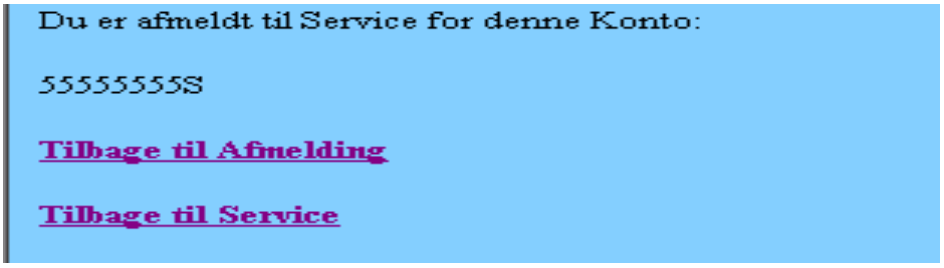
Figur 11.3.14 Profil af et aktieabonnement.

Ved at trykke på *afmelding* i den side som vises i figur 11.3.6, fås Figur 11.3.15 med oversigt over de mulige serviceafmeldinger.



Figur 11.3.15 Afmeldingsoversigt.

Når man har valgt hvilken service man vil afmelde sig fra, bekræftes dette, se Figur 11.3.16.



Figur 11.3.16 Bekræftelse på afmelding.

Vi har ikke valgt at vise alle de websider vi har lavet, men udvalgt dem af betydning, for at vise hovedprincippet i navigering.

12. Problemer undervejs

I dette kapitel vil vi komme ind på nogle af de uforudsete problemer vi stødte ind på under processen.

12.1 Implementeringen af databasen

En af de begrænsninger som vi har haft i behandlingen af data er at vi ikke har kunnet anvende de såkaldte *triggers*, som ville have lettet handlingen med forespørgler i database. Endvidere meddelte SDC at det gik imod deres sikkerhedspolitik.

Et af de største problemer vi stødte ind i, var da vi skulle oversætte vores ER diagram til SQL formulering og implementere databasen i MySQL. MySQL kunne ikke understøtte begrænsningerne (*FOREIGN KEY CONSTRAINT*) på en almindelig måde (default: MyISAM). Da vi forhørte os rundt omkring, tog det os lang tid at finde ud af at tabellerne skulle være af en bestemt type. Indtil vi fandt ud af dette fik vi anbefalet at det skulle være en anden version, det vil sige en nyere version som havde denne egenskab. Men det viste sig at den version vi skulle bruge blot skulle have tabeltypen *InnoDB* for at iværksætte ovennævnte begrænsning. Dette var imidlertid ikke nok, idet begrænsningen (CONSTRAINT) alligevel ikke virkede. Vi kom senere frem til løsningen på en empirisk måde, ved at behandle INDEX'et på den måde som er vist i afsnit 5.5.1.

12.2 Komplikationer med *Push-systemet*

Parsingen af kildefiler i XML-format har været vanskelig. Man skal først og fremmest vælge mellem de forskellige parsere i XML, og når man har forstået deres måde at arbejde på, skal man tilpasse disse til vores formål. Vi skulle vælge en hurtig parser, og derefter skulle vi overveje at genfinde en algoritme som ved hjælp af parseren kunne udplukke informationerne. Vi skulle også finde de forskellige midler til at implementere *push-systemet* såsom faciliteterne i J2EE (Javamail, Xerces, Javaxml, Javasql).

12.3 Sending af SMS beskeder

Telekommunikationsselskaber har begrænset mulighed for at sende sms som en fri service fra nettet. I juli skulle vi pga. de ændringer der opstod i afsendelse af sms'er fra Internettet, omformulere den del af vores system der behandlede sendingen af sms'er.

12.4 ASP-Problematikken

En af de problemer vi havde med implementeringen af brugergrænsefladen var, at hver gang man laver en forespørgsel til databasen, da skal man lave en forbindelse til den. Dette vil sige at man skal åbne databasen for hver enkelt forespørgsel man har. Derefter eksekverer man den, og forbindelsen skal lukkes igen.

13. Forslag til forbedring af push-systemet

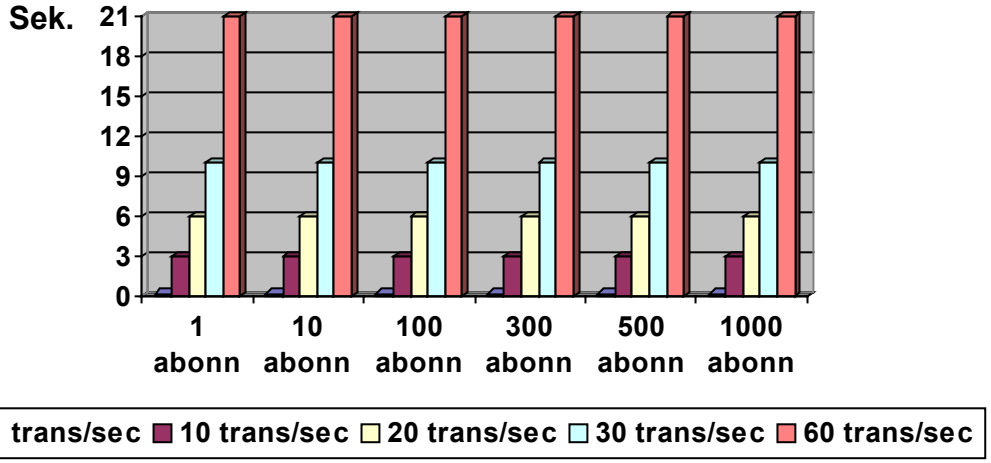
Udfra de forskellige tests af hastigheden af *push-systemet* har vi bemærket at systemets hastighed kan forøges. Problemet med det nuværende *push-system* er at parseren læser dokumentet (kildefilen) og for hver indkommende data skal Parseren hente nogle data fra databasen. Parseren skal også sende de relevante informationer og vente på at dette er færdiggjort for at gå videre med læsningen af den indkommende fil. (Dette arbejde gør at Parseren bliver langsom.)
Derfor har vi tænkt os at man kunne implementere *push-systemet* på en anden måde, som vi vil beskrive i det følgende.

13.1 Ny implementering af Parser

Når Parseren læser den indkommende kilde-fil og udplukker de informationer der skal behandles senere, kan disse gemmes i en træstruktur for hver kilde-fil. Træet kan sendes videre til andre klasser, hvor det behandles og sendes til abonnenterne.
De "andre" klasser er de samme som i det nuværende *push-system*, men de bliver eksekveret parallelt med Parseren (i det nuværende *push-system* bliver "klasserne" eksekveret, i *ParserKildeFile,* i sekvens) og dermed kan hastigheden mindst fordobles (se Figur 13.1). Dette kræver at man har flere processorer til rådighed, og det skal være en processor til hver tråd.
Den nye Parser er færdigimplementeret, men den er ikke indarbejdet i *push-systemet*, hvilket vil sige at Parseren ikke er sat sammen med de "andre klasser", da vi ikke kunne finde løsningen til at sende træer videre til de andre klasser uden at miste informationer, idet træerne bliver opdateret hvert øjeblik.

13.2 Testningen af den nye Parser

For udføre denne test har vi belastet systemet med det samme antal af både transaktioner og abonnenter som i den test (afsnit 10.2) i kapitlet Testningen.



Figur 13.1 Grafen over testning af den nye Parser.

Grafen i figur 13.1 viser *push-systemet* kan udføre sit arbejde hurtigere ved implementering, som tidligere forklaret.

Sammenligningen af graferne i figurerne 10.2.1 og 13.1 viser, at tiden for behandling af informationer er blevet halveret.

14. Konklusion

Vi har i denne rapport redegjort for den arbejdsproces vi har haft i forbindelse med opbygningen af dette abonnementssystem. Undervejs er vi kommet ind på de forskellige problemer vi skulle løse. Vi har på opfordring af SDC taget udfordringen op: at programmere et abonnementssystem der kan behandle en datastrøm fra finansverdenen og videresende dette til abonnenterne. For at kunne gøre denne har vi opbygget en database som gemmer data både fra abonnenterne og abonnementerne. Dernæst har vi implementeret en motor, det såkaldte *push-system*, som sorterer og puffer den indstrømmende data videre til abonnenterne. Ligeledes har vi lavet en brugergrænseflade for at skabe en interaktiv forbindelse mellem det system vi har skabt, databasen og den endelige modtagergruppe, abonnenterne.

Implementeringen af programmet har været vanskelig, særligt på grund af at vi ikke har fået stillet de rette redskaber til rådighed. Det har for det første ikke været muligt at arbejde ordentligt på SDC, fordi deres sikkerhedspolitik vanskeliggør at installere programmer på deres computere. Man skal have fulde brugerrettigheder til dette, og for at vores system skal kunne fungere, har det været nødvendigt at have installeret Open Source software. Hvis vores projekt skulle have foregået under optimale forudsætninger, skulle dette have været i orden i forvejen. Endvidere fik vi langt inde i projektet at vide, at kildefilerne ikke kunne blive givet fra SDC. Derfor var vi alene nødt til at implementere det hele, hvilket har betydet at vi selv skulle konstruere en klient som sendte informationerne, og det skulle laves så tæt på virkeligheden som muligt.

Det har været meget besværligt at finde ud af at implementere vores database i MySQL, da vi skulle lære om iværksættelse af tabeller af type InnoDB. Det tog lidt tid at sætte os ind i dette nye koncept, men til sidst fik vi rimelig fortrolighed med dette emne.

Parsingen af de indkommende XML kildefiler har ikke været uden besvær, det har været nødvendigt at tilegne sig kendskab til XML-parsing. Vi har skullet forstå hvordan SAX-parseren fungerede og derefter skulle vi sætte den til at arbejde således at serveren kunne forstå dataene fra klienten. Udvælgelse af parsere har også krævet mange overvejelser, da der er mange valgmuligheder. Da vi ikke har haft kendskab til XML-parsing har det krævet et ekstra arbejde og tid at forstå de forskellige slags XML-parsere og forsøge at anvende dem til vores formål. Da vi først fandt ud af at SAX-parseren var den bedste til projektet, skulle den tilpasses formålet. Hvis vi i forvejen havde haft kendskab til XML-parserne, ville vi have undgået at bruge ekstra tid på dette problem.

Endvidere har det været en udfordring at skulle koordinere så mange teknologier på én gang. Vi har før dette projekt kun arbejdet os enkeltvist med teknologierne og har altså ikke prøvet at sammensætte dem i en helhed af denne størrelse. At få alle brikker til at stemme overens kræver overblik, og det har været en ny udfordring at skulle tage op. At få databasen til at arbejde sammen med Java-programmer og ASP-teknologi har krævet en grundig bearbejdning og forståelse af de implicerede begreber. Til gengæld har

kendskabet til sprog, maskiner og modeller været en stor hjælp for at kunne konstruere *push-systemet*.

Ved testningen af programmet har vi set at formålet med vores projekt er opfyldt. Man kan abonnere på en service og modtage en besked når der sker ændringer i den pågældende service. Da vores *push-system* ikke skulle være afhængigt af brugergrænsefladen, har vi opnået at det ubesværet kan fungere uden denne. Derimod kan det ikke fungere uden vores database; til gengæld kan det samarbejde med en hvilken som helst database.
Fra nu af kan man med vores abonnementssystem forbedre måden hvorpå kunder serviceres. Abonnenterne vil løbende kunne få besked når der sker vigtige ændringer af relevans i for eksempel pengeinstitutter.

15. Litteraturliste

[1] Connolly, Thomas M. Begg, Caroline E.: *Database Systems: A Practical Approach to Design, Implementation, and Management:* Pearson Education Inc. Boston, 2000.

[2] Mathew, Neil. Stones, Richard: *Beginning Databases with MySQL.* Wrox Pres Ltd. Birmingham B27 6BH UK, 2002

[3] Atkinson, Leon; *Core MySQL*; Prentice-Hall Inc. New Jersey, 2002.

[4] Widom, Jennifer. Ullman, Jeffrey D.: *A First Course in Database Systems.* Prentice-Hall International, Inc. New Jersey, 1997.

[5] Horton,Ivor: *Beginning Java 2 JDK 1.3 Edition*; Wrox Pres Ltd. Birmingham B27 6BH UK, 2000

[6] Pohl, Ira. McDowell, Charlie: *Java by dissection: The essentials of java programming*. Addison-Wesley Longman, Inc. Santa Cruz, USA 2000.

[7] Wang, Paul S.: *JAVA with Object-Oriented Programming and World Wide Web Applications*. Brooks/Cole Publishing Company. Pacific Grove California. 1999.

[8] Nakhimovsky, Alexander. Myers, Tom. Wilcox, Mark. Zeiger, Stefan. Diamond, Jason. Griffin, John. Avedal, Karl. Holden, Mac. Tyagi, Sameer. Damme, Geert. Allamaraju, Subrahmanyam. Longshaw, Andrew. O'Connor, Daniel. Browet, Richar. Johnson, Rod. Karsiens, Tracie. Kim, Larry. Huitzen, Gordon: *Professional Java Server Programming J2EE Edition.* Wrox Pres Ltd. Birmingham B27 6BH UK. 2000.

[9] Maruyama, Horoshi. Tamura, Kent. Uramoto, Naohiko. Murata, Makoto. Clark, Andy. Nakamura, Yuichi. Neyama, Ryo. Kosaka, Kazuya. Hada, Satoshi: *XML and Java Second Edition Developing web applications*: Pearson Education Inc. Boston, 2002.

[10] Buser, David. Kauffman, John. Libre, Juan T. Francis, Brian. Sussman, David. Ullman, Chris. Duckett; John: *Beginning Active Server Pages 3.0.* Wrox Press Ltd., Birmingham B27 6BH UK, 2002

[12] Hunter, David: *Beginning XML*. Wrox Pres Ltd. Birmingham B27 6BH UK 2000.

[13] Hopcroft, John. Motwani, Rajeev. Ullman, Jeffrey D.: *Introduction to Automata Theory, Languages and Computation* Addison Wesley (Pearson Education). Stanford University, 2001.

Filename: eks2002Abonnementssystem
Directory: C:\Documents and Settings\fkc\My Documents
Template: C:\Documents and Settings\fkc\Application Data\Microsoft\Templates\Normal.dot
Title:
Subject:
Author: ornella
Keywords:
Comments:
Creation Date: 07-01-2003 13:32
Change Number: 8
Last Saved On: 14-01-2003 11:47
Last Saved By: Finn Kuno Christensen
Total Editing Time: 61 Minutes
Last Printed On: 14-01-2003 11:51
As of Last Complete Printing
Number of Pages: 108
Number of Words: 137.832 (approx.)
Number of Characters: 785.646 (approx.)