Sammendrag

I denne rapporten blir det sett nærmere på datavarehus-teknologi. Et datavarehus er en database hvor man samler data fra flere systemer og legger dem best mulig til rette for analyse. I dette ligger det at man tar vare på alle data, kobler dem til et tidspunkt og organiserer de slik at de er lette å søke i. Beslutningstagere får på denne måten tilgang til historiske data og bedre anledning til å gjennomføre avanserte analyser av dataene. I tillegg vil genereringen av rapporter og søk i datamengden kunne utføres separat fra de vanlige systemene, og vil dermed ikke påvirke responstiden til disse.

Rapporten går først inn på teorien som ligger bak et datavarehus. Her vil oppbygning og faguttrykkene bli klarlagt, før det gjennomgåes hvilke typer verktøy som eksisterer og hvodan de benyttes. Siden det fokuseres på hvordan trafikkinformasjon kan utnyttes i et datavarehus, så vil det gåes igjennom hva og hvem som kan ha interesse av slik trafikkinformasjon. Disse emneområdene jobber mot å klarlegge hvordan man bør arbeide når man starter på et datavarehus for dette formål. For å gi en bedre veiledning er prosessen der en prototype ble utviklet gjennomgått mot slutten av rapporten. Her inngår planlegging, utvikling og bruk av verktøyene.

Forord

Som en del av all den økende informasjonsstrømmen i dagens samfunn, er datavarehus blitt er verktøy for å kunne håndtere denne informasjonen slik at man kan fortelle noe om den store mengden av data. Datavarehus har blitt et verktøy som mange bedrifter har kastet seg over, uten å egentlig vite hva det innebærer.

Derfor har jeg i forbindelse med den avsluttende delen av mitt Mastergradsstudie ved Høgskolen i Agder, levert denne besvarelsen på min diplomoppgave. Vi har i løpet av det siste året jobbet med spesialisering innen ønskede retninger, og jeg fattet interessent for datavarehus etter en gjesteforelesning på skolen ved Christian S. Jensen. Ved noen samtaler sammen med Jensen utarbeidet vi en oppgavebeskrivelse som er nærmere beskrevet i innledningen.

I tillegg til ett par bøker jeg har benyttet meg av, har Internett vært en stor bidragsyter av informasjonen samlet inn her. Det finns mange gode whitepapers lagt ut for allmennheten, men også kilder som ikke er å stole på. Jeg har prøvd å sortere disse ned til det som jeg har benyttet meg av her.

Til sist ønsker å rette en takk til Christian S. Jensen som har vært min eksterne veileder, og Mikael Snaprud, min interne veileder på Høgskolen i Agder. Begge disse har vært svært behjelpelige under hele prosjektperioden. Og jeg vil takke Anders Friis-Christensen for hans hjelp over mail.

Grimstad, 26 Mars 2003

_____

Erlend Falch-Pedersen

Innholdsfortegnelse

FORORD **2**

1. INNLEDNING **5**

1.1 BAKGRUNN 5
1.2 LITTERATUR 6
1.2.1 Transformering fra GPS-posisjon til kartposisjon *6*
1.2.2 Hvordan kombinere lokasjonsinformasjon med et datavarehus. *6*
1.2.3 Effektivisering ved store datamengder. *6*
1.2.4 Hvordan legge opp datavarehuset slik at verdiene blir korrekte *6*
1.3 MÅLET FOR OPPGAVEN 7
1.4 METODE 8
1.5 RAPPORTENS STRUKTUR 9

2. DATAVAREHUS **10**

2.1 HVA ER ET DATAVAREHUS 10
2.2 OPPBYGNINGEN 12
2.2.1 Generelle begreper *12*
2.2.2 Relasjonell realisering *13*
2.2.3 Andre viktige begreper innen datavarehusteknologi *15*
2.3 BRUK AV OLAP 17
2.3.1 De forskjellige OLAP-strukturene er: *18*
2.3.2 Nøkkelfunskjoner for et OLAP-verktøy *19*

3. TRAFIKKINFORMASJON **21**

3.1 TILGJENGELIG DATA 21
3.2 ULIKE BRUKERE AV ET DATAVAREHUS 22
3.2.1 Vegvesenet *22*
3.2.2 Politiet *22*
3.2.3 Forsikringsbyrå *22*
3.2.4 Kollektivtransport *23*
3.2.5 Trafikkanter *23*
3.3 OPTIMALISERING 23
3.4 PERSONOPPLYSNINGSLOVEN 24

4. PLANLEGGING AV ET DATAVAREHUS **25**

4.1 FORANALYSE 25
4.2 TEKNISK DESIGN 26
4.3 VALG AV TEKNOLOGI 27
4.4 DATABASEN 27
4.5 ETL-RUTINENE 29
4.6 APPLIKASJONER 29

5. PROTOTYPING: ET IMPLEMENTERT DATAVAREHUS **30**

5.1 PROSJEKTPLANLEGGINGEN 30
5.1.1 Datagrunnlag *30*
5.1.2 Spesifikasjon av krav *31*
5.2 DESIGN 35
5.2.1 Tabellbeskrivelser *35*
5.3 IMPORT AV DATA 40
5.4 RESULTAT 42
5.4.1 Ytelsesammenlikning *42*
5.4.2 Spørringene *42*

6. DRØFTING **46**

6.1 PROGRAMVARE 46
6.1.1 Server *46*

6.1.2 Verktøy *46*
6.2 DESIGN 47
6.2.1 Koordinatnett *47*
6.2.2 Kjøreruter *47*
6.2.3 Utvikling av datakilde *47*
6.2.4 Segmenter av vei *47*
6.2.5 Oppløsning *48*
6.2.6 Data mining *48*
6.2.7 Sakte forandrende dimensjoner *48*

7. KONKLUSJON 49

8. ORDLISTE 50

8.1 FORKORTELSER 50
8.2 ORDFORKLARINGER 50

9. REFERANSER 51

9.1 NETTSIDER 52

10. VEDLEGG 53

10.1 JSCRIPT-KODE 53
10.1.1 Konvertering av dato og klokkeslett fra tall og over på UTC-form *53*
10.1.2 Konvertering av database og over til datavarehus *53*

FIGURLISTE

FIGUR 1.1 - "TIMEGLASSMODELL" 9
FIGUR 2.1 - ET KOMPLETT DATAVAREHUS 11
FIGUR 2.2 - EKSEMPEL PÅ RELASJON MELLOM FACT OG DIMENSJON 13
FIGUR 2.3 - STJERNESKJEMA 14
FIGUR 2.4 - SNOWFLAKING 14
FIGUR 2.5 - ROLAP 18
FIGUR 2.6 - MOLAP 18
FIGUR 2.7 - HOLAP 18
FIGUR 4.1 - DATAVAREHUS LIVSSYKLUS DIAGRAM. 25
FIGUR 4.2 – TEKNISK DESIGN 26
FIGUR 4.3 - UTNYTELSE AV PROTOTYPE FOR BESLUTNINGSTØTTE VED VALG AV PROGRAMVARE 27
FIGUR 4.4 - APPLIKASJONSUTVIKLINGEN 29
FIGUR 5.1 - DATAVAREHUSET, SLIK DET ER BYGD OPP. 35
FIGUR 5.2 - HIERARKIET TIL DDATE DIMENSJONEN 36
FIGUR 5.3 - DTIMEOFDAY 37
FIGUR 5.4 - DLOCATION 38
FIGUR 5.5 - DSPEED 38
FIGUR 5.6 - DACCELERATION 39
FIGUR 5.7 – SEAGATE ANALYSIS 42
FIGUR 5.8 – FARTSOVERSKRIDELSER 43
FIGUR 5.9 - NEDBREMSINGER 44
FIGUR 5.10 - HASTIGHET OVER FARTSGRENSE 44
FIGUR 5.11 - DEFINERING AV PUNKTREKKEFØLGE 47

TABELLISTE

TABELL 1 - DATABASE MOT DATAVAREHUS 11

1. Innledning

1.1 Bakgrunn

I et samfunn med stadig voksende informasjonsmengde får informasjonsbehandling stadig økende betydning. Det blir stadig mer kritisk at man kan få tak i akkurat de dataene man ønsker, uten å måtte gå lange omveier. Datavarehus kan hjelpe brukeren å analysere store mengder data. Vanligst er nok måten datavarehus benyttes for å analysere forretningstransaksjoner for å finne forhold som kan øke gevinsten for en butikk. Her benyttes datavarehuset som ett tilskudd til bedriftens "Business Intelligence" (BI). En annen økende metode er i forbindelse med "Customer Relations Management" (CRM), hvor datavarehuset kan samle opp og levere relevant informasjon til f.eks. en forsikringskonsulent.

I forbindelse med et samarbeid Høgskolen i Agder (HiA) har med Aalborg Universitet (AAU) hvor fra Professor Christian S. Jensen har vært på utveksling til HiA, har han vært med på å formulere denne oppgaven sammen med stipendiat. Han har fra AAU vært involvert i ett prosjekt, hvor de har samlet måledata fra biler ved trafikkovervåkning. Disse måledataene er det som har vært benyttet i prosjektet i forbindelse med prototypeutviklingen.

I rapporten beskrives hvordan man kan benytte informasjon man har tilgjengelig om trafikkdata, med hjelp av et datavarehus. Det vil igjennom rapporten klarlegges hva ett datavarehus er, og i forbindelse med trafikkdata, hva man må tenke på ved å kombinere disse. Planleggingen i forkant av et datavarehusprosjekt er viktig, og det vil belyses utover. For å øke kunnskapsverdien ytterligere, er det gjort et fullstendig forløp med utvikling av en prototype basert på de dataene som Christian S. Jensen hadde tilgjengelig fra INFATI-prosjektet.

INFATI (Intelligent Fartstilpassning) tar utgangspunkt i utvikling av trafikkinformasjon med sikte på å bidra til et bæredyktig transportsystem med særlig henblikk på å forbedre trafikksikkerheten. I under prosjektet har de testet ved å installere en bilcomputer som overvåker kjøremønsteret til 20 privatsjåfører i Aalborg. Det er dataene fra en av disse bilene som er benyttet i dette prosjektet

1.2 Litteratur

Av tidligere relevant arbeid som jeg ønsker å arbeide ut ifra, er AAU godt representert med 3 av 5 publikasjoner. AAU har en egen trafikkforskningsgruppe som har arbeidet mye med dette feltet. Den 4 rapporten er fra HiA for et år tilbake, og er også utarbeidet i samarbeid med Professor Christian S. Jensen. Den siste rapporten er ett samarbeidsprosjekt med hovedvekt i Tyskland, hvor de har sett på måter å effektivisere ROLAP over store mengder data som i dette tilfelle kan vise seg å bli svært aktuelt.

Følgende rapporter er benyttet som grunnlag.

1	Data warehousing for geografiske data	Vår 2002	HiA
2	Intelligent fartstilpassning – Adferdsendringer	2001	AAU
3	Intelligent fartstilpassning – Mapmatching	2001	AAU
4	Data warehouse design for spatio-temporal data	Høst 2001	AAU
5	Interactive ROLAP on Large Datasets	2001	IDEAS 2001, Grenoble

Jeg har i tillegg til disse fem, benyttet meg av kursmatriell i faget IKT2340 ved HiA, whitepapers, bøker og internett.

Det er spesifikke felt som denne rapporten omhandler, der det er interessant å trekke inn forskningen gjort i de fem nevnte rapportene. Disse går jeg igjennom nedenfor og forteller om hvordan situasjonen er.

1.2.1 Transformering fra GPS-posisjon til kartposisjon

Når en posisjon blir lest fra en GPS vil den alltid ha en feilmargin. Denne feilmarginen er nå på 10x10 meter. Derfor bør koordinatene ”mappes” over, slik at koordinatet kommer på den riktige veien. Dette er belyst i rapport 1 og 3.

1.2.2 Hvordan kombinere lokasjonsinformasjon med et datavarehus.

Når man benytter lokasjon i forbindelse med datavarehus, er det noen faktorer som er viktig å ta hensyn til. Det vil f.eks. være veier som hører til under flere regioner. Det er tidligere gjort arbeid i forbindelse med det ved HiA beskrevet i rapport 1. Rapport 4 fra AAU tar for seg mer omfattende hvordan et datavarehus kan se ut på grunnlag av de samme dataene som ved prototypen utviklet i dette prosjektet.

1.2.3 Effektivisering ved store datamengder.

Siden det er transportmidler her som registreres, betyr det at man kan oppnå veldig store mengder data om dette implementeres på alle biler. Rapport 5 tar for seg hvordan man kan endre datavarehuset for at spørringer skal kunne gå kjappere.

1.2.4 Hvordan legge opp datavarehuset slik at verdiene blir korrekte

Ett dilemma med det datasettet som er tilgjengelig, oppstår hvis f.eks. en bil kjører på en vei. Hvordan skal man da finne ut hvor mange biler som er på en vei over en gitt tid. Siden loggen registrere bilen på den aktuelle vei for hvert 10 sekund, vil det over 1 time kunne registreres 60 biler på veien, da det egentlig er bare 1. Dette har de funnet en løsning på i rapport 4. En annen interessant observasjon knyttet til samme problem er når man aggregerer på antall ganger hastigheten er overskredet, så vil det ved 100 km/h være halvparten så mange overskridelser som ved 50 km/h. Dette skaper et avvik proporsjonalt økende med hastigheten.

1.3 Målet for oppgaven

Oppgaven vil fokusere på hvordan posisjoneringsdata fra mobile enheter kan utnyttes i et datavarehus i ulike problemstillinger. Ut ifra det skal forskjellige designarkitekturer utforskes. Resultatet ønskes å kunne hjelpe andre på vei ved å gi ett grunnlag/startpunkt for design av datavarehus.

Som nevnt tidligere er datavarehus ett begrep som benyttes for databaser der formålet er å kunne ekstrahere viten fra store mengder informasjon. Ofte er det benyttet i bedrifter med hensikt på å hente kunnskap om kunder, slik at bedriften kan maksimere profitten eller bedre kunderelasjoner. I dette tilfellet skal datavarehuset benyttes slik at brukerene kan ekstrahere informasjon om trafikken. Det gjøres ved å aggregere forskjellige måldata, som f.eks. hastighet over en viss strekning. Her skal det sees på hvilke muligheter som kan være reelle ved bruk av slike data i et datavarehus. Aktuelle områder det kan være interessant å ekstrahere informasjon om er:

- *Hastighet*
  Hastigheten kan analyseres på flere måter. Her kan det være interessant å se på gjennomsnittshastighet, maksimumshastighet, gjennomsnittsavviket,
- *Ulykker*
  Antall ulykker, hvor de skjer, når, hvem er bl.a. det som kan gi viktig informasjon.
- *Kjøremønster*
  Kjøremønsteret kan fortelle noe om hvilke veier biler velger i forhold til hvor de skal, og hvilke veier som er ofte benyttet. Dette kan muligens være fordi bedre/kortere veier er for dårlig skiltet til at den benyttes. Det kan også fortelle noe om kollektivtrafikken er for dårlig på disse veiene
- *Trafikkflyt*
  For trafikanter kan det være veldig nyttig å vite dersom det er trafikkork lenger fremme. Det kan også fortelle om veinettet her er for dårlig

Det skal også utvikles et datavarehus på grunnlag av data fra ett prøveprosjekt gjort ved Aalborg Universitet.

Formålet med oppgaven er så å utvikle grunndesign av datavarehus, med hensyn på de forskjellige problemstillingene. Hensikten er at andre som skal ta i bruk et datavarehus for å overvåke f.eks. hastighet, kan benytte denne rapporten som veiledning for planlegging, design og utføring. Dersom det oppstår problemer i form av importering av dataene, vil det i den grad det er mulig utarbeides ett fullt fungerende datavarehus.

For at arbeidsmengden ikke skulle bli for stor, ble prototypingen redusert ned til ett sett med fact-tabell og tilhørende dimensjoner. Gjennomsnittshastighet og kjøreruter ble derfor ikke prøvet i praksis. Kork er implementert, men bare i forenklet form (hastighetssjekk). Trafikkbelastning kan heller ikke aggregeres over tid.

1.4 Metode

Oppgaven denne rapporten bygger på, har vært utarbeidet i samtale med Christian S. Jensen. Og for å ta fatt på oppgaven ut fra det, gikk jeg frem på følgende måte:

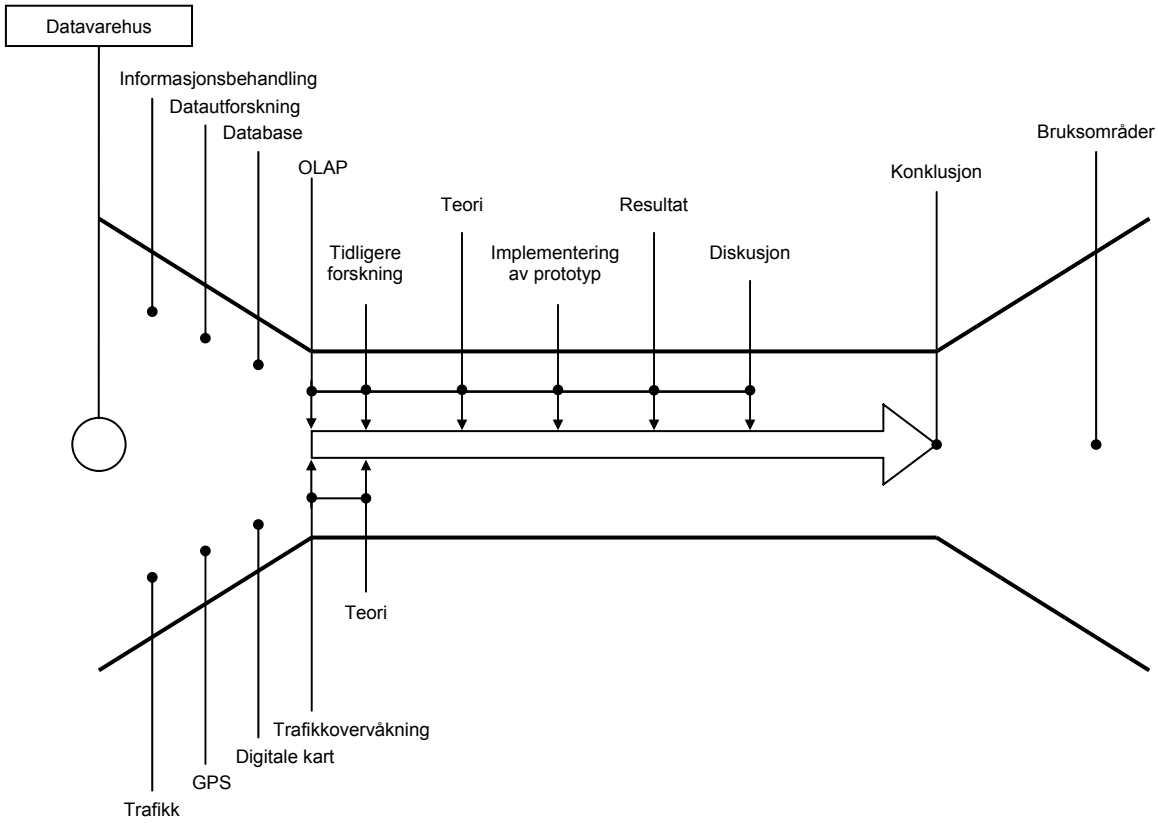
- En litteraturstudie for å gi innføring i hva datavarehus, databaseteknologi og trafikkovervåkning innebærer, for å oppnå bedre innsikt ved gjennomgang av forskning på dette området.
- Tidligere relatert arbeid ble gjennomgått og vurdert, for å kunne bygge videre og utvikle fra disse. 5 publikasjoner er valgt ut som grunnlag for denne rapporten. Et sammendrag av disse tidligere beskrevet i kapittel 1.2
- Videre litteraturstudie ble utført, for å har klarlagt angrepsprosedyren på selve prototypen.
- Ut ifra datagrunnlaget, ble det utviklet en arkitektur på prototype. Denne ble implementert ved hjelp av verktøy for å transformere verdiene i grunndataene
- Siden hovedformålet med oppgaven er å fungere som veiledning for andre med lignende prosjekter, er det viktig å legge frem materialet på en lettfattelig og interessant måte.

Siden jeg var alene på prosjektet, var en utfordring å ha et variert syn på de forskjellige måtene å løse oppgavene. For å oppnå dette ble flere kilder benyttet. Det var veiledere, bøker, internettforum og andre relevante rapporter.

1.5 Rapportens struktur

For å holde fokuset på riktig sted, benyttet jeg meg av timeglassmodellen til planleggingen av rapporten. Denne viser hvilke områder som skal omhandles i rapporten og gir en grafisk fremstilling av fremgangsmåten. Dette er vist ved Figur 1.1. Målet er at rapporten skal:

- fungere som veiledning.
- forklare hvordan jeg gikk frem for å løse problemet med utviklingen av prototypen.
- vise at jeg har en fungerende prototype.
- vise at jeg har oppnådd forståelse og innsikt i de temaene oppgaven omhandler.



Figur 1.1 - "Timeglassmodell"

Som figuren viser, vil jeg starte med å forklare det generelle rundt datavarehusteknologi og trafikkinformasjon. Jeg vil først starte på grunnivå hvor jeg går igjennom det som er viktig når man begynner på et slikt prosjekt. Selvfølgelig vil ikke denne rapporten alene være nok til å dekke alt dette. Bøker som anbefales sammen med denne rapporten vil det bli referert til. Det som er gjennomgått av tidligere litteratur jeg bygger videre på vil jeg forklare og benytte meg av i kapitel 5, hvor jeg beskriver implementasjon av datavarehuset.

2. Datavarehus

Dette kapittelet tar for seg hva et datavarehus er, og vil forklare en del begreper og metoder for å utnytte et datavarehus. Det vil også klarlegges hva verktøy man kan benytte mot et datavarehus, for å kunne hente informasjonen på en god måte.

2.1 Hva er et datavarehus

W. H. Inmon, mannen som startet denne trenden innen databaseteknologi, har skrevet en definisjon på hva et datavarehus er [INMO92]:

> *"A data warehouse is a subject oriented, integrated, time variant, nonvolatile collection of data in support of management's decision making process."*

Med det mente Inmon:

- *Subject oriented:*
  Orientert rundt temaer som er viktig for organisasjonen. Dette kan være ting som kunder, produkter, politikk, konti, osv.
- *Integrated:*
  Det er viktig at dataene integreres slik at de samme dataene ikke er representert på forskjellige måter eller med forskjellig navn. Et eksempel på dette kan være at tabellene "PåLager" og "Inne", som begge inneholder data av samme type i de operasjonelle databasene, blir overført til en enkelt tabell som heter "Lager" i datavarehuset.
- *Time variant:*
  Tidsaspektet er veldig viktig i datavarehus, og man må alltid knytte dataene opp i mot et visst tidspunkt. Det kan også være aktuelt å følge tidsavhengige trender som man ikke uten videre kan se på grunnlag av de operative data.
- *Nonvolatile:*
  Til slutt er det viktig at alle data lastes inn i datavarehuset på en gang, hvorpå de kan bli aksessert, men helst ikke forandret.
- *In support of management's decision making process:*
  Det skal være mulig for ledelsen å benytte dataene i den beslutningsprosessen som er en del av deres hverdag, dvs. datavarehuset skal danne grunnlaget for dataanalyse.

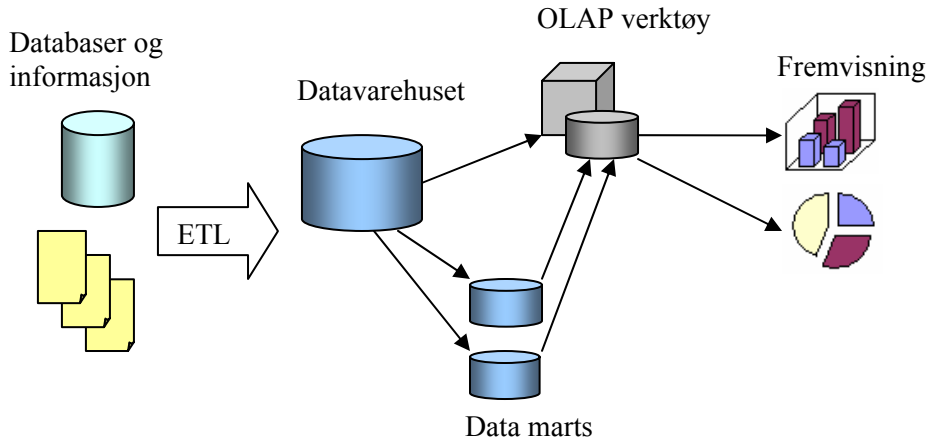
Et datavarehus er egentlig en database, men bygget opp på en spesiell måte. Derfor anbefales kunnskap om hva og hvordan en database fungerer, før man begynner på dette kapittelet. Tabell 1 viser en sammenlikning av hvordan datainnholdet benyttet i transaksjonsdatabase er mot data i et datavarehus [INMO92].

Primitive data/ operasjonelle data	Beregnede data/ datavarehus data
Detaljerte	Summerte eller på en annen måte behandlet
Korrekte på gjeldende tidspunkt	Representerer verdier over tid
Tjener saksbehandlerne	Tjener administrasjonen
Kan oppdateres	Lite behov for oppdatering
Krav til prosessering kan beregnes på forhånd	Krav til prosessering kan ikke beregnes på forhånd
Aksesseres en enhet av gangen	Aksesseres et sett av gangen
Transaksjonsdrevet	Analysedrevet
Orientert rundt applikasjoner	Orientert rundt analysen
Kontroll av oppdateringer er viktig med tanke på eierskap	Kontroll av oppdateringer er av liten betydning
Høy tilgjengelighet	Avslappet tilgjengelighet
Ikke redundante	Redundante
Statisk struktur, variabelt innhold	Fleksibel struktur
Mengden data involvert i en prosess er liten	Mengden data involvert i en prosess er stor

Tabell 1 - Database mot Datavarehus

Data i et datavarehus er tilpasset for å kunne benyttes over lang tid (8-10 år), i motsetning til en database som opererer på en kortere tidsramme (ett par måneder).

Når man snakker om et datavarehus, så tenker man ofte mer helhetlig, og ikke kun på databasen. En viktig prosess er den som kalles Extraction, Transformation, Loading (ETL). Dette er omformingen av rådataen man har, til en form i datavarehuset som er i henhold til oppbygningen. Videre er det også prosesser etter datavarehuset for å kunne utnytte informasjonen i datavarehuset til å bli kunnskap. Figur 2.1 [BMBT] er en grafisk fremstilling av sammenhengen mellom informasjonen man har, datavarehuset og hvordan man skal få utnyttet det til kunnskap.



Figur 2.1 - Et komplett datavarehus

2.2 Oppbygningen

Oppbygningen av databasen er viktig. Det er det som gjør at den kan fungere som et datavarehus og ikke bare blir en vanlig database. Derfor vil jeg nå gå igjennom hva man må vite med et datavarehus.

2.2.1 Generelle begreper

2.2.1.1 Facts

En fact er en hendelse som har skjedd. Et datavarehus er bygd opp av disse hendelsene, og lagrer informasjon om dens dimensjoner (kommer nedenfor). Klokkeslettet for når denne hendelsen fant sted kan være en dimensjon. Stedet det skjedde er en annen.

2.2.1.2 Measures

Measures er verdiene for en fact. Disse har enten en verdi i form av tall. En measure har et element til. Den som forteller hvordan man skal operere verdien. Det kan være at verdiene skal summeres eller bare telles over f.eks. en dag. Si at man har en verdien for et salg. Hvis man ønsker å finne inntekter over en måned, så vil man summere *salg.* Dermed er den andre attributten for *salg* en sum-funksjon. For at man ikke skal utføre spørringer som gir ulogiske eller i verste fall feil svar, deles measures inn i tre klasser [ØRØD]. De er:

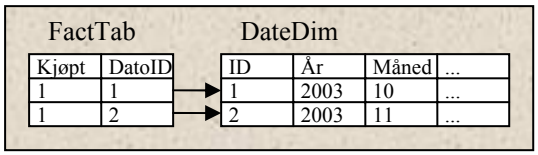
- *Additive measures*
  Disse kan kombineres på langs med alle dimensjoner. Eksempelvis, kan det være meningsfylt å se på antall ulykker på en spesifikk veg, tid, dato, sjåførens alder og type bil.
- *Semi-additive measures*
  Disse har en eller flere dimensjoner som den ikke kan kombineres med. Eksempelvis, kan man se på en vei for å finne antall biler som kjører ved ett tidspunkt, men man kan ikke direkte summere antall biler på den samme vegen over en hel dag, da hver bil vil bli telt flere ganger.
- *Non-additive measures*
  Disse kan ikke kombineres langs dimensjonene. Eksempelvis, ved måling av romtemperatur kan man ikke bruke denne langs dimensjoner uten å måle gjennomsnitt i stedet for å summere

Det bør tilstrebes å få measures additive så langt det er mulig, for å øke fleksibiliteten til datavarehuset.

Av og til kan man det virke nødvendig å benytte text i fact-tabellen. Man bør da heller forsøke å kunne generalisere innholdet, og sette det ut i en dimensjon. Hvis det ikke er mulig, har sannsynligvis ikke verdien mye relevant informasjon å tilføre datavarehuset.

2.2.1.3 Dimensjon

En dimensjon inneholder informasjon om en fact. En veldig vanlig dimensjon er en for dagen hendelsen fant sted. I dimensjonen ligger rader som inneholder informasjon om den dagen. Den inneholder gjerne flere overliggende verdier som den kan grupperes under i et hierarki. Ved en dato, setter man gjerne opp tilhørende uke, måned, kvartal og år. Her kan man også legge til egne grupperinger som er av hensikt å legge under dato.



Figur 2.2 - Eksempel på relasjon mellom fact og dimensjon

Den benyttes både for å spare plass og for å lette arbeid ved spørringer. Tenk en transaksjon i fact-tabellen ved kjøp av 1 liter melk. Da vil det lages en link videre til en tabell f.eks. kalt *Date* som angir hvilken dato, ukedag, uke, måned o.s.v. denne melken ble kjøpt på. Men i dette tilfellet legges ofte ikke uke inn i hierarkiet, men på siden i et mindre hierarki. Det skyldtes at en uke kan være over flere måneder.

Fordelene med å registrere slik i forhold til å kun registrere datoen er ganske viktige. I siste tilfellet så vil man ved en spørring med utdrag av måned 11, måtte prosessere alle rader for å sjekke om de hører til den korrekte måneden. Hvis det er et stort datavarehus dette gjelder, skjønner man at det tar mye ekstra tid. Ved bruk av dimensjoner grupperer man verdiene direkte i databasen. Det sparer plass i databasen ved å redusere antall bytes som trengs for å definere dagen hendelsen fant sted. En dato i MS SQL Server tar 8 bytes. Hvis man tenker at man skal ha databasen i 20 år (som er en lang tid for datateknologi), så trengs det kun 7300 rader i dimensjonen for å definere alle de datoene. En *smallint* tar kun 2 bytes, og kan gå mellom -32,768 to 32,767. Det holder lenge nok for å håndtere tallet som relaterer til dimensjonen. Nå er riktignok ikke oppløsningen så høy, men ved nøyaktighet på 1 sekund vil det kreves 0.6 mrd rader. Dette rekker i så fall med en vanlig *int* på 4 bytes til dette.

2.2.2 Relasjonell realisering

En relasjonsdatabase vil si at databasen inneholder flere tabeller som det settes opp relasjoner til dimensjonene. I forbindelse med et datavarehus, kalles den tabellen som inneholder fact naturlig nok en fact-tabell.

2.2.2.1 Fact-tabell

Det er fact-tabellen som blir maten i et datavarehus, og utgjør mesteparten av plassen datavarehuset tar. Det er her hendelsene ligger. I Fact-tabellen setter man opp relasjoner ut til dimensjonene ved hjelp av en foreign-key. I fact-tabellen lagres også measures, som ble gjennomgått litt tidligere. Disse har altså en verdi som benyttes i kombinasjon med dimensjonene, og kalkulerer

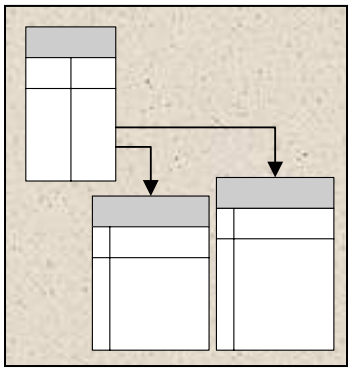
Det er disse verdiene man regne på for å finne ut mer om dataene. En rad i fact-tabellen kan typisk inneholde hver registrerte transaksjon, eller en samling av antall over ett gitt tidsrom.

2.2.2.2 Stjerneskjema/Snowflaking

Når man designer et datavarehus kan man velge mellom to forskjellige strukturer. Den vanligste er stjerneskjema, og består av en fact-tabell hvor alle dimensjoner er linket direkte ut ifra. Dette er vist ved Figur 2.3. Faktorer som er viktig ved bruk av stjerneskjema er:

+ Lette og oversiktlige
+ Relativt fleksible
+ Relativ små dimensjonstabeller
+ Bedre sammenkjøring med de fleste RDBMS og dermed bedre ytelse

- Ikke normaliserte dimensjoner
- Hierarkier blir gjemt i tabellkolonnene
-

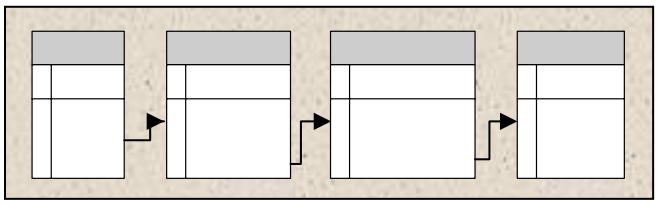


Figur 2.3 - Stjerneskjema

Ved å benyttes seg av snowflaking kobler man sammen hvert lag oppover i hierarkiet ved å linke dimensjoner. Det er vist ved Figur 2.4. De viktigste faktorene ved bruk av snowflaking er:

+ Synlige hierarkier
+ Fleksible
+ Dimensjonstabellene bruker mindre plass

- Vanskeligere å bruke p.g.a. mange koblinger
- Dårligere ytelse



Figur 2.4 - Snowflaking

2.2.3 Andre viktige begreper innen datavarehusteknologi

2.2.3.1 Granularitet

Granularitet betyr egentlig kornethet og er et ord for hvor nøyaktig datavarehuset er. Ved lav granularitet, blir registrering gjort meget nøyaktig, som f.eks. hvert sekund eller for hver transaksjon. Ved høyere granularitet kan man enten summere antall instanser over lengre tid eller registrere sjeldnere.

2.2.3.2 Redundans

Med redundans menes repetisjon, altså informasjon som er nevnt mer en ett sted. Dette kan være en fordel for å bedre responstid ved spørringer på datavarehuset. I.o.m. at dimensjonene til et datavarehus tar svært liten plass (< 5 %), er redundans ikke noe man normalt trenger å passe seg for, så lenge det holdes i bakhodet. Oppdateringstiden kan øke som følge av dette, men det er effektiviteten på spørringene som er viktig for datavarehuset, og ikke oppdateringene.

2.2.3.3 Metadata

Metadata benyttes for å vise hvordan data blir transformert fra en ekstern datakilde og over i datavarehuset. Man klassifiserer disse i to grupper. Den ene er for administrator, og beskriver datakildene, attributtene og definisjonene som kan benyttes. Den andre gruppen er men for sluttbruker, og viser hva de forskjellige dataene representerer. Eksempelvis, kan være en tabell kalt ”omsetning”. Metadataene vil da indikere for brukeren hvor i den operasjonelle databasen dataene ble hentet fra, hvordan de ble beregnet og hva de inkluderer.

Begge disse typene metadata lagres sammen, men de trenger ikke nødvendig vis ligge på samme sted som selve datavarehuset. Dette er likevel som oftest tilfellet, siden det maksimerer tilgjengeligheten til metadataene. Grunnen til at metadata er så viktig i et datavarehus, er at det er at der er behov for å kontrollere hvordan dataene transformeres når de overføres fra den operasjonelle databasen til datavarehuset. Det gjør metadataene i datavarehuset på en god måte.

2.2.3.4 Data marts

En metode for å avlaste/spre arbeidsmengden til et datavarehus på, er å benytte seg av data marts. Dette er et utdrag fra datavarehuset, som ikke inneholder alle data fra det opprinnelige datavarehuset. Det kan også ha endringer, for å kunne bedre håndtere de spørringer som er forventet at det vil få.

2.2.3.5 Data mining

Data mining er et verktøy som arbeider rundt i datavarehuset for å se om det kan snappe opp noen trender/sammenhenger som ville vært vanskelige å tenke seg på egenhånd p.g.a. tiden det ville tatt eller kompleksiteten. Dette benyttes ofte i transaksjonsdatavarehus, der man ønsker å kunne utnytte slik informasjon til å gi et konkurransefortrinn. Det er tre forskjellige grupper av data mining-verktøy:

- *Filtrering*
  Disse sortere ut det som er viktig på grunnlag av kriterier satt på forhånd av brukeren. Eksempelvis kan man be om å få ut de kjøretøy som holder en hastighet på over 90 km/t.
- *Kunstig intelligens*
  Disse "borrer" i databasen for å finne informasjon den tror kan være interessant for brukeren. Eksempelvis kanskje den finner ut at biler på store veier overskrider fartsgrensen oftere en biler på små veier ved samme fartsgrense.
- *Intelligente agenter*
  Disse verktøyene kan jobbe når brukeren ikke er til stede og finne ny informasjon ved å selv navigere gjennom databasen. Eksempelvis kan systemet som er satt til å overvåke trafikkmønstre, passe på ved økende risiko og dermed gi beskjed til person via e-services.

Disse kan benyttes i datavarehuset, eller i de dataene generert av OLAP-verktøyet.

2.3 Bruk av OLAP

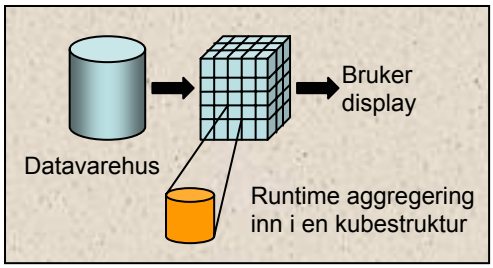
Online Analytical Processing (OLAP) har i senere tid blitt mer et jippouttrykk for mange og blir misbrukt deretter. Det er derfor viktig å vite hva det dreier seg om når man snakker om OLAP. OLAP Report [I-NP-2] foreslo derfor i 1995 å benytte FASMI som ett alternativ. Her gir ordbetydningen en bedre beskrivelse av hva egentlig man snakker om. FASMI står for Fast Analysis of Shared Multidimensional Information, og hensikten er:

- *Fast:*
  Systemet skal kunne gi en respons på ad hoc spørringer i løpet av 1 til 20 sekunder. Vanlig responstid bør være på ca. 5 sekunder. Dersom responstiden blir for lang vil brukeren sannsynlig vis bli forstyrret i tankegangen og kvaliteten på analysen vil forverres. Denne hastigheten er vanskelig å oppnå med store datamengder, spesielt når brukerens spørringer blir kompliserte.
- *Analysis:*
  Systemet skal kunne håndtere enhver form for forretningslogikk og statistisk analyse som er relevant for applikasjonen og brukeren, samtidig som det skal være brukervennlig nok for den tiltenkte brukeren.
- *Shared:*
  En del krav til sikkerhet må være implementert i systemet. Det skal være mulig at flere jobber mot samme data samtidig, og også at flere skriver til samme data uten at feilsituasjoner oppstår. På dette punktet svikter mange av dagens OLAP-produkter, da de ofte antar at applikasjonen vil være kun lesbar og ikke skrivbar.
- *Multidimensional:*
  Dette er nøkkelkravet til et OLAP-produkt; dersom man skulle bruke bare ett ord for å definere OLAP så ville det vært dette. Systemet må tilrettelegge for et multidimensjonellt syn på dataene, inkludert støtte for hierarkier og multiple hierarkier, da dette er den mest logiske måten å analysere bedrifter på.
- *Information:*
  Med informasjon menes alle de data og avledede informasjon som er nødvendig for å utføre den nødvendige analysen. Det som er relevant i denne sammenheng er hvor mye informasjon produktene kan behandle og hvor mye lagringsplass som kreves for å ta vare på dataene.

OLAP benyttes dermed som et begrep på verktøy som samler inn data til et repositori, som igjen lar brukere koble seg til for å utforske og omforme denne informasjonen til kunnskap. Måten disse dataene håndteres i dette repositoriet er gjennom kuber med de aggregerte regninger av dataene i datavarehuset. Når man har definert sine spørringer kan man få verktøyet til å vise resultatet ved f.eks. grafer eller 3-dimensjonale kuber, som gjør det lettere å oppdage den viktige informasjonen.

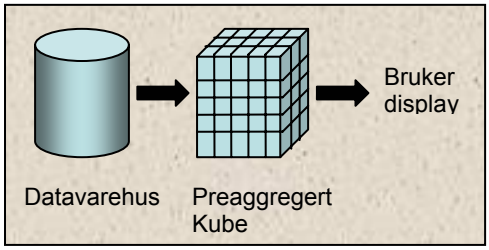
2.3.1 De forskjellige OLAP-strukturene er:

Relasjon OLAP (Figur 2.5) [NR]
Relasjons OLAP er en den enkleste, i form av at den utnytter strukturen i stjerneskjemaet (se side 14) til datavarehuset den jobber på. Alle utregninger av measures skjer run-time, så den krever mer av serveren. Disse verktøyene tilbyr på en veldig enkel måte, å få datavarehuset opp og gå siden de ikke krever mye av både bruker og server (av installering).



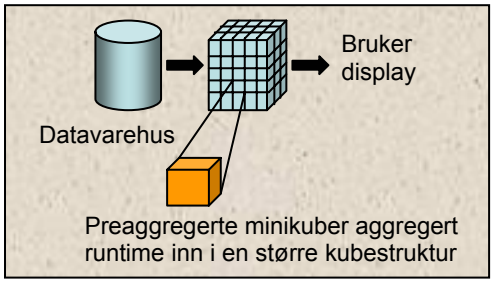
Figur 2.5 - ROLAP

Multidimensjonell OLAP (Figur 2.6) [NR]
Her lagres aggregerte data i en multidimensjonell struktur i OLAP-serveren. Dette vil bedre spørringstiden betraktelig, men kan ta mye plass grunnet dataeksplosjon. Dette er flere OLAP-verktøy som har muligheten til bl.a. finne spørringene som antas å gi best ytelsesforbedringer mens den skal holde verdiene under f.eks. 100 mb. Det er det viktig å planlegge på grunnlag av viktighet, data, størrelse, effektforbedring og hyppighet.



Figur 2.6 - MOLAP

Hybrid OLAP (Figur 2.7) [NR]
Denne metoden har blitt introdusert av Microsoft, og er en blanding av Multidimensjonal- og Relasjons OLAP. I denne lagres aggregerte data i en multidimensjonell struktur i OLAP-serveren, mens den beholder kildedataene i den eksisterende relasjonsstrukturen.



Figur 2.7 - HOLAP

2.3.2 Nøkkelfunskjoner for et OLAP-verktøy

Siden det er mange programmer på markedet som hevder å støtte OLAP, er det lurt å gjøre seg kjent med hvilke funksjoner som man med stor sannsynlighet kan få brukt for. I boken Nils Rasmussen [NR] har utarbeidet en liste over hvilke funksjoner som bør støttes [NR]:

- *Drill-down/Roll-up.*
  Denne funksjonen lar brukeren arbeide seg oppover eller nedover i hierarkiet i dimensjonene. Det kan være slik som å gå fra verdier per fylke, og ned til verdiene per by.

- *Standard grafer/kart.*
  Funksjonen gjør at det blir lettere å oppdage uregelmessigheter og å sette ting i en sammenheng, ved at tallene blir lagt frem grafisk.

- *Produsentavhengige visualiseringsverktøy.*
  De fleste analyseverktøy har utviklet sine egne metoder på å fremvise data, for å hjelpe brukeren. Disse vil selvfølgelig variere, men kan være viktig å ta med i vurderingen.

- *Avviksmerking.*
  For å gjøre brukeren oppmerksom på predefinerte advarsler, kan det ofte legges inn alarmer. Det kan være som f.eks. når produksjonen går under ett spesielt punkt, så må brukeren alarmeres slik at grunnen til dette blir funnet.

- *Kombinert visning.*
  Dette henviser til muligheten for å legge inn flere akser i en og samme graf. Det kan være slik som å få inn både prosentverdier og tallverdier på en graf.

- *Dra og slipp dimensjoner.*
  Mange verktøy har muligheten for å bytte mellom dimensjonene på en enkel måte, for å gjøre browsing i databasen mer effektiv.

- *Brukerdefinerte kalkulasjoner.*
  For å kunne gi mer handlekraft og egeneffektivisering, er muligheten for å kunne definere egne kalkulasjoner, ett viktig verkøy. Det vil også lette den administrative jobben av datavarehuset.

- *Editerbare spørringer.*
  Noen ganger er det lettere å gå inn å endre rett i spørringene, enn å måte definere i programmet. Da er dette en enkel å nyttig funksjon for de som har bedre kjennskap til datavarehuset og verktøyet.

- *Kvalitative kommentarer.*
  Muligheten for at brukeren kan legge til kommentarer ved spesielle verdier, vil i situasjoner der flere benytter systemet, gi en berikelse av verktøyet i form av kunnskapsspredning og informasjonsdeling.

- *Arbeidsbord (Dashboard).*
  Hvis verktøyet kan legge opp arbeidsbord der brukeren kommer inn, kan det gi en bedret oversikt over vital og viktig informasjon.

- *Distribuering av kuber/rapporter.*

For å kunne dele informasjonen, er dette nødvendig. Ferdige rapporter sier sjelden alt, og da kan det være aktuelt for noen å gå videre inn på informasjonen presentert, for å finne årsak til situasjonen.

- *Filtrering.*
  Filtreringsmuligheter på serversiden kan være ett godt verktøy for å fjerne overflødige spørringer fra nysgjerrige brukere, slik at man kan fokusere maskinkraften mot hver sine aktuelle områder.

- *Sortering*
  Sortering kan være nyttig for å kunne få en annen fremvisningsrekkefølge på spørringene.

3. Trafikkinformasjon

Jeg ønsker i dette kapittelet å diskutere det å samle inn trafikkinformasjon for utnytelse i et datavarehus. Det er mange forskjellige aktører som dette kan være aktuelt for, og derfor er det greit å ha belyst dette før vi fortsetter på den tekniske delen av rapporten.

3.1 Tilgjengelig data

Når man kjører en bil er det gjerne for å komme seg fra et sted til ett annet. Men hva som skjer underveis er det ikke så mange som går rundt og husker på i ettertid. Men faktum er at vi i løpet av en liten tur har foretatt drøssevis med valg. Disse valgene forteller mye om oss som sjåfører. Hastigheten man velger er nok et av de tydeligste. Men andre valg som veien man tar. Måten man planlegger kryss, når man blinker til og med hvordan vi gasser er tegn som forteller mye om en person. I dette prosjektet har det vært forsøkt å finne ut informasjon om trafikken som en enhet ved hjelp av et datavarehus, men er det flere muligheter sammen med et datavarehus som kan være av interesse?

Hvis man skal sette i gang et lignende prosjekt, må det overveies hvilke data man har, og hvilke data man trenger. En komplett database over veikoordinatene i Norge, er det uvisst om eksisterer. Dataene som er benyttet i dette prosjektet, er som nevnt fra ett tidligere prosjekt i Danmark. I det prosjektet var de nødt til å kjøre rundt på alle veier i Ålborg, og notere veinavn, fartsgrense og koordinat. Dette er tidkrevende om det skal gjøres landsdekkende, og bør nødig gjøres mer enn 1 gang. Hvis dette arbeidet ikke har vært gjort før, bør man passe på å få med seg alle tenkelige data, da det kan øke muligheten for å selge informasjonen videre.

Dimensjonene sted og tid faller ganske naturlig inn som en del av et trafikkbasert datavarehus. Tidsdimensjonen inneholder vanligvis: År, kvartal, måned, uke, dag, virk/helgdag, vanlig/feriedag, tid på døgnet (kveld, natt, o.s.v.), time, minutt og sekund. Og for stedsdimensjonen er det naturlig om mulig å ha med: Land, region, by, vei, koordinat

Hvis man titter på det som er involvert i trafikken, så er det mye informasjon som kan være interessant å ha tilgjengelig. Informasjon om sjåføren og kjøretøy er det som gjerne dukker opp først i tankene. Her kan det vært interessant å ha tilgjengelig informasjon om:

- Kjønn
- Alder
- Bilmerke
- Årsmodell
- Hestekrefter
- Biltype
- Forurensingsrate

Enda grundigere kunne det vært nyttig å vite forholdene. Da tenker jeg på vær, temperatur og veidekke. Men dette er å sette veldig høye krav. Det krever i så fall at målinger blir gjort av kjøretøyene mens de kjører.

3.2 Ulike brukere av et datavarehus

Trafikkovervåkning kan føre med seg veldig mye nyttig informasjon. Ut ifra hva som er målet med datavarehuset, vil forskjellige design være vise seg nødvendig. Derfor har jeg listet opp de forskjellige aktører som jeg vil anta kan være interessert i et slikt datavarehus. Det vil kikkes overfladisk på hva det som er mulig benytte ett datavarehus til. Hvorvidt det vil tillates av Datatilsynet er diskutert i kapitel 3.4.

3.2.1 Vegvesenet

Hvis man ser på trafikkinformasjon for vegvesenet, så er det sikkerhet og trafikkflyt som vil være viktig. Sikkerhet har med veiplanleggingen å gjøre. Det er anslått at av 100 nestenulykker, utvikler en seg til en ulykke. De steder som ligger utsatt til og skaper farlige situasjoner, vil sannsynligvis ha et høy antall nestenulykker. Men hvordan kan man oppdage en nestenulykke?

En naturlig reaksjon dersom en farlig situasjon oppstår, er at man bremser. Gjerne veldig hardt. Dette kan snappes opp ved å overvåke hastigheten på kjøretøy, og hvordan den varierer. Hvis man får mange høye verdier innenfor ett område, vil dette kunne bety at det er noe ved denne veistrekningen som skaper farlige situasjoner.

Når det snakkes om trafikkflyt så er det egentlig to underliggende faktorer. Hastigheten som holdes og mengden trafikk på veien. Dersom mengden av kjøretøy inn på veien overstiger det farten holdt klarer å få unna, vil det oppstå kork. Det dannes vanligst kø i rushtiden i byen, når ulykker skjer, ved veiarbeid eller der det reduseres antall kjørebaner. Ved å utnytte den informasjonen kan man kanskje forutse tidligere tiltak som kan bedre situasjoner som ulykker og veiarbeid. Samtidig kanskje denne informasjonen kan gi veiledning på hvordan man bør gå frem med veiplanlegging også.

3.2.2 Politiet

Politiet bruker mye resurser på å kontrollere at lovverket overholdes, for å minske ulykker der uvettig kjøring er involvert. Men det er begrenset hvilken mulighet de har til kontroll. Derfor er det mange, både blant politiet og borgere som hadde foretrukket om politiet kunne fokusert sine resurser der hvor datasystemer ikke kan gjøre jobben for dem. Trafikkovervåkning av hastighet kan gjøres på en enkel måte. Det som kreves, er kjøretøyets hastighet og hvilken fartsgrense det er der den kjører. Men det er ganske urealistisk å få tillatelse til en slik overvåkning. Mer nærliggende er det derfor å kunne se statistikken og så trekke ut trendener fra den. Fokus som er interessant da er:

- Hvor skjer det mange fartsovertredelser
- Hvor skjer de grove fartsovertredelsene
- Hvor er det tilsynelatende sammenheng mellom høy fart og ulykker

Dette kan bidra sterk til å fortelle hvor kontroller er nødvendig. Hva dette krever av datavarehuset er beskrevet på siden 32, kapittel 5.1.2.1.

3.2.3 Forsikringsbyrå

Ett forsikringsselskap er mest interessert i å vite hvor mye de må betale ut i skader, for så å kunne justere den samlede inntekten til å overstige utgiftene med ønsket margin. Men p.g.a. konkurranse prøver selskapene å justere prisene mer og mer slik at forsikringspolisen representerer det som er forventet av den gjeldene bil og sjåfør. I dette tilfellet benytter nok alle seriøse aktører seg allerede av et datavarehus, så her er det lite nytt. Men dersom man har muligheten til å følge en sjåfør mens han kjører med GPS, vil man kunne etablerer et bedre

innblikk i hvordan den sjåføren oppfører seg i trafikken. Dette kan analyseres i et datavarehus for å trekke ut trender for de forskjellige risikogruppene. Det kan være seg kjørestil, veier man ofte kjører på, tidspunkt man bruker å kjøre o.s.v.

Det har dukket opp aktører i USA der man kan oppnå rabatter på forsikringen dersom man tillater å la seg overvåke av GPS. Det er også i Danmark, nærmere bestemt Nordjylland ett prøveprosjekt på unge bilister som lar seg overvåke av GPS. Dette er ikke i samarbeid med et forsikringsselskap, men tanken er at man skal kunne dokumentere at man har en god kjørevane.

3.2.4 Kollektivtransport

I Norge i dag har man problemer med å få befolkningen til å ta mer i bruk kollektivtransport. De vanligste grunnene er tilgjengelighet, fleksibilitet og pris for at folk heller velger bil. Men de fleste er klar over at dette ikke er måten å komme seg fra A til B i fremtiden dersom ikke forurensingsproblemet blir ordnet. Siden moderne forbrenningsmotorer lider av det faktum at de trenger strøm (både elbiler og hydrogenbiler), er det sannsynlig å anta at det er flere tiår frem i tid før vi får endret dette. For å kunne lokke flere til å benytte seg av kollektiv transport kan man prøve å bedre tilbudet. Naturlig logikk sier at bedre ruter fører til økt fortjeneste, som igjen gir lavere priser. Da har kommer man inn i en god spiral.

Men hvordan man skal kunne trekke ut ruter fra et datavarehus er ikke blitt arbeidet på i dette prosjektet. Til det hadde jeg ikke tid og resurser, og er dermed er det vanskelig å si noe om hvordan det kan løses. En veldig forenklet løsning er å se på belastningen på vegnettet, men det vil ikke ta hensyn til om de som kjører der har mulighet til å ta buss på den ruten.

3.2.5 Trafikkanter

Hvis man ofte reiser med bil på steder som er utsatt for kødannelser, blir man fort veldig oppsatt på å få med seg om en kø kommer forut. Radiokanalen P4 utnytter dette godt, ved å rapportere over etheren dersom kø dannes på veiene. Men som man forstår kan de ikke passe på hele landets veinett. I tillegg vil også være en viss grad av forsinkelse på meldingene, da køene ofte blir rapportert av bilister ute i trafikken som ringer inn.

Hvis man har posisjonen på kjøretøyer i Norge, er det mulig å benytte denne informasjonen for både å se på lang og kort sikt hvor køer dannes. Men for at trafikanter skal kunne utnytte seg av dette mens de er i trafikken, må det stilles ganske høye krav til oppdatering. Her snakker vi om veldig korte tidsrammer, som ikke et datavarehus er optimalisert for.

3.3 Optimalisering

For de gruppene som ble gjennomgått, vil garantert være ønskelig å kunne optimalisere med tanke på kriterier de selv ønsker å bedre. Kriteriene kan være slik som miljø, tid, sikkerhet, slitasje, vedlikeholdskostnader o.l. Det ligger f.eks. muligheter for å benytte takster ut ifra hvor mye hver sin kjøring koster samfunnet, i stedet for bare en fast pris, slik som det er nå. Desto bedre man kjenner til en spesifikk prosess, desto større mulighet har man til å klarer å utnytte dette.

3.4 Personopplysningsloven

Mange ser seg skeptisk til at noen skal kunne ha så mye informasjon lagret om seg selv. Derfor er det noe som heter personopplysningsloven, og den lyder som følger [I-P]:

§ 1. *Lovens formål*
Formålet med denne loven er å beskytte den enkelte mot at personvernet blir krenket gjennom behandling av personopplysninger.

Loven skal bidra til at personopplysninger blir behandlet i samsvar med grunnleggende personvernhensyn, herunder behovet for personlig integritet, privatlivets fred og tilstrekkelig kvalitet på personopplysninger.

Denne loven håndheves av Datatilsynet, og de er ganske påpasselige med å gjøre jobben sin. Bl.a. er det de som har satt foten ned for såkalte smarte fotobokser der de kan måle gjennomsnittshastigheten mellom hver fotoboks. Det er fordi de mener denne dataen kan igjen benyttes til overvåkning, da den lagres over tid (selv om den er kort). Derimot kan man ved tillatelse fra den det gjelder, få dispensasjon. Til det sier loven følgende[I-P]:

§ 8. *Vilkår for å behandle personopplysninger*
Personopplysninger (jf. § 2 nr. 1) kan bare behandles dersom den registrerte har samtykket, eller det er fastsatt i lov at det er adgang til slik behandling, eller behandlingen er nødvendig for

- å oppfylle en avtale med den registerte, eller for å utføre gjøremål etter den registrertes ønske før en slik avtale inngås,
- at den behandlingsansvarlige skal kunne oppfylle en rettslig forpliktelse,
- å vareta den registrertes vitale interesser,
- å utføre en oppgave av allmenn interesse,
- å utøve offentlig myndighet, eller
- at den behandlingsansvarlige eller tredjepersoner som opplysningene utleveres til kan vareta en berettiget interesse, og hensynet til den registrertes personvern ikke overstiger denne interessen.

Spørsmålet er da: Hva vil få folk til å gå med på å la seg overvåke slik? Det går på det samme gamle. Penger. Dersom vi vet at vi kan oppnå billigere forsikring hvis vi går med på å la vårt kjøremønster kartlegges, er det meget sannsynlig at man får mange frivillige. De fleste lever jo i den oppfatning at man kjører godt, så da burde det ikke være noe problem å få en god slump rabatt. Dersom man kan benytte dette til køvarsling, så vil jeg tro de aller fleste er positive.

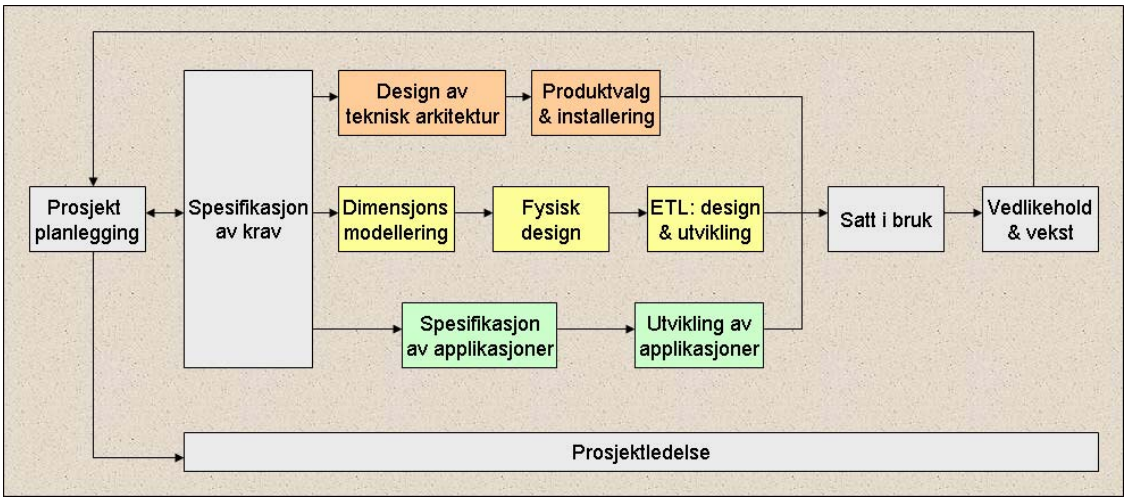
Det er litt vanskeligere å vite med kollektivtransporten. Jeg vil anta det er usannsynlig at de kjøper folk for å få de til å kjøre rundt med slikt overvåkningsutstyr. Da er loven om allmenn interesse mer nærliggende. Her kommer det an på om Datatilsynet også vurderer konkurransefortrinnet det kan gi til de forskjellige busselskapene. Vegvesenet vil jeg også anta får tillatelse hvis målet er å bedre forholdene ved de norske veiene. Og det er jo av allmenn interesse.

4. Planlegging av et Datavarehus

For å oppnå suksess med et datavarehus, er det viktig å planlegge godt fra starten av. Figur 4.1 viser en utbredt modell av livssyklusen for et datavarehus [RK]. Denne er veldig grundig gjennomgått i boken "The data warehouse lifecycle toolkit", og derfor vil ikke jeg bruke tid på den samme forklaringen her i en redusert form. I stedet vil jeg integrere det med de erfaringene jeg gjorde meg med prototypingen.

4.1 Foranalyse

På figuren kan vi se at et datavarehus alltid vil trenge videreutvikling. Det er som når man kjører bil. Man blir bare bedre til å kjøre ved å faktisk kjøre. Man må utforske og utvikle datavarehuset til stadighet, for å kunne hente og bedre dens kunnskap. Det vil etterhvert som man bruker det, dukke opp nye vinklinger som krever nye oppsett. Datavarehus er en iterativ prosess, der man arbeider i loop for stadig fremskritt.



Figur 4.1 - Datavarehus livssyklus diagram[RK]. Den oransje rekken omhandler teknologien, den gule databehandlingen og den grønne applikasjonene.

Et datavarehus er langsiktig, og vil nesten helt sikkert ikke fungere optimalt ved første forsøk. Så fort man tar det i bruk, vil det komme frem svakheter i programvare, design eller valgmulighetene for brukeren. Når man starter prosjektet, er det viktig å sette seg ned og få klarlagt selve målet. Et datavarehus tilbyr et hav av muligheter, og derfor er det store muligheter for rote seg bort på veien.

Slik som det vises på figuren så bør man starte med å kartlegge hvilke behov for datainnsamling som allerede eksisterer der det skal benyttes. Hvem som samler inn hva, og hvor mye tid som benyttes på dette. Deretter hva som de syntes mangler, eller ikke har mulighet til å finne ut av. Hvis man ikke har benyttet seg av dataanalyse tidligere, bør man sette seg ned og finne ut hva man er interessert i å vite noe om. Dette legger grunnlaget for *Spesifikasjon av krav*. Ved figuren er dette vist som *Prosjektplanlegging* og disse har en pil begge veier for å markere sammenhengen mellom de.

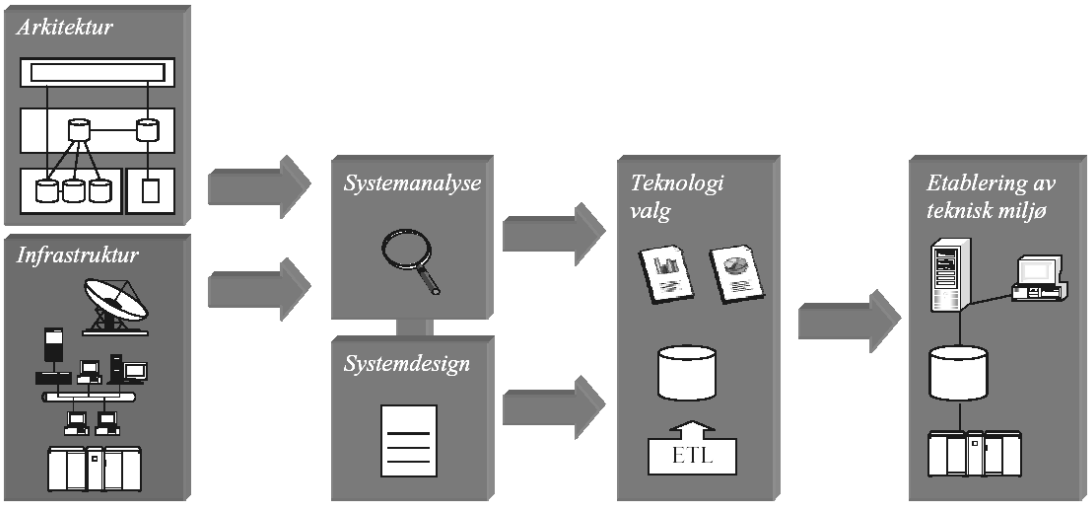
Hvis vi kobler dette mot f.eks. vegvesenet, har de sannsynligvis mange metoder de benytter for å finne ut hvor de skal sette opp en fotoboks. På en forenklet måte kan man si det er forsvarlig å utvikle et datavarehus for å finne ut dette, dersom utvikling og driftskostnadene av datavarehuset ikke overtiger kostnadene ved gammel metode. Men da er det ikke tatt høyde for sikkerhetsmargin og det ekstra datavarehuset kan føre med seg.

Viktigheten av å gjøre et grundig forarbeid for å finne ut behovet kan ikke understrekes nok. Det har den fordelen i tillegg at de som skal ta dette verktøyet i bruk, får vært med på prosessen. Hvis dette er en meget etablert arbeidsstokk vil det bedre holdningen til verktøyet, og gjøre selve innføringen enklere. Måten behovsanalysen utføres på er ganske forskjellig fra tradisjonell datadrevet behovsanalyse. Det er nødvendig at datavarehusdesigneren kan sette seg inn i nøkkelfaktorene for å drive bedriften, og klare å overføre de inn i designet.

4.2 Teknisk design

Spesifikasjonen danner grunnlaget for teknologien, databehandlingen og applikasjonene. Når man har kartlagt hvilke krav som må oppfylles, vet man hvilket data man trenger for å oppfylle disse.

Når man har fått kartlagt disse to, er det tid for å bestemme hvordan man ønsker å bygge opp det nye miljøet. Det er viktig å finne ut hva kravene fra foranalysen krever og bygger en plattform på grunnlag av det. Når det er klart kan man gå i gang med den mer praktiske delen av prosjektet. Det er vanlig å prøve å komme tidlig i gang med den tekniske biten, da et datavarehus trenger mye iterativ utvikling. Det skyldtes at etterhvert som man får forsket på og arbeidet med måldataene, så dukker det gjerne opp nye elementer som kan være nyttig. Og da kan det hende de krever en annen oppbygning. Figur 4.2 er hentet fra ”Datavarehusmetodikk” [RU] og illustrerer hvordan man benytter seg av foranalysen for å utvikle det tekniske miljøet.



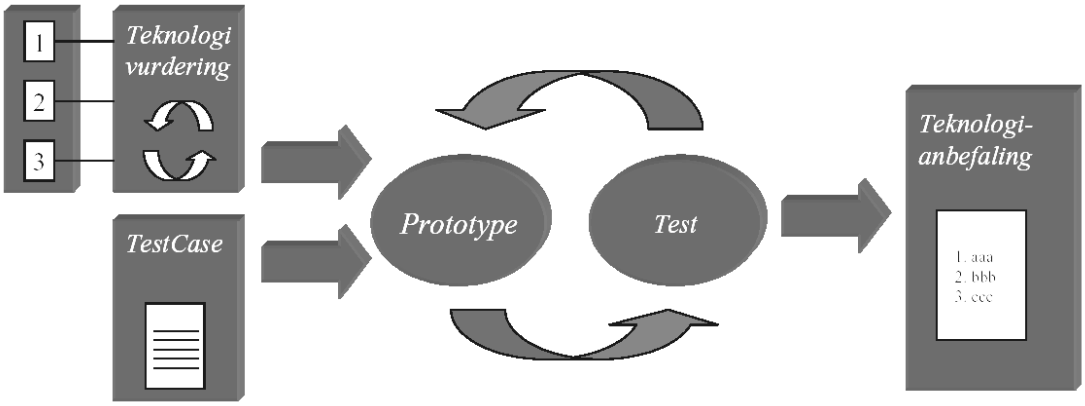
Figur 4.2 – Teknisk design

4.3 Valg av teknologi

Når man har fått på plass det nye "tekniske miljøet", bør man se på de forskjellige produktvalg. Da bør man sette opp en liste over de produsenter som støtter de kravene man har. Hvis man er usikker på hva verktøyene støtter, kan man sende ut en forespørsel til de forskjellige verktøysprodusentene om dine krav, og hvordan deres program ligger på dette. Punkter som bør tenkes igjennom er [NR]:

- Integrasjon mot bedriftens nåværende datakilder kommunikasjonsmetoder
- Nåværende og fremtidig datavolum
- Bruksområde av datavarehus
- Nøkkelfunskjonene
- Brukersikkerhet, tilgang og roller
- Implementasjon og trening.
- Data infrastruktur
- Hardware og software krav
- Spesifiserte kostnader

Før valget avgjøres, er det også viktig å sjekke ut softwaren. Det kan gjerne gjøres ved å be om referanser til kunder med likt software/hardware/design oppsett, som det man selv har planlagt. Hvis man har resurser til det, kan man også gå til anskaffelse av en prøveversjon, slik at man får en enda grundigere gjennomgang av programvaren. Som Figur 4.3 viser, bør det være et høyt nivå av prototyping/testing for å få et dekkende overblikk [RU].



Figur 4.3 - Utnytelse av prototype for beslutningstøtte ved valg av programvare

4.4 Databasen

Når man kommer over på å faktisk bygge databasen som man ønsker, kan man gå etter følgende struktur[RU]:

- *Logisk datamodell:*
  Med utgangspunkt i kravspesifikasjon, etabler en logisk datamodell. Denne bør baserer fullstendig på brukernes krav/ønsker
- *Systemanalyse:*
  Foreta en kartlegging av aktuelle kildesystemer
- *Gap-analyse:*
  Vurder og dokumenter avvik mellom krav til data og eksisterende data.

- *Logisk datamodell versjon 2:*
  Utarbeid en revidert datamodell etter gap-analysen. Her skal avvik dokumenteres.
- *Fysisk datamodell*
  Omform den logisk datamodell til en fysisk databasedesign.
- ***Opprett databasen***

Siden jeg benyttet meg av et stjerneskjema og det er ganske utbredt, går jeg igjennom noen viktige punkter som omhandler det. Men de gjelder ikke utelukkende et stjerneskjema. Stort sett gjelder også ved snowflaking.

Når man skal konstruere et stjerneskjema må man ta en del viktige valg. Disse valgene kan oppsummeres i 9 punkter:

- Man må identifisere hendelser som foregår i trafikken (f.eks. bryte fartsgrense, bremse, kjøre en strekning) og ut ifra disse lage en eller flere faktatabeller til de hendelsene.
- Hvilken granularitet skal det være på faktatabellene? Skal de inneholde individuelle bilener, eller skal det summeres?
- For hver faktatabell må det avgjøres hvilke dimensjontabeller som skal kobles til. Først må det kobles til nok dimensjoner til at en unik nøkkel for faktatabellen kan etableres, deretter kan det legges til eventuelle andre nyttige dimensjoner.
- Man må bestemme alle målbare og beregnede fakta som skal ligge i faktatabellen.
- Hvilke attributter skal dimensjontabellene ha? Beskrivelsen skal være komplett og terminologien hensiktsmessig.
- Hvilken metode skal benyttes ved forandringer i data tilhørende dimensjoner som er tilnærmet konstante (sakte forandrende dimensjoner)?
- Man må avklare en del ting vedrørende krav til fysisk lagringsplass, bl.a. aktuelle summeringer/beregninger, bruk av heterogene dimensjoner og eventuelle minidimensjoner.
- Hvor mye haster det med å gjøre informasjonen tilgjengelig. Må gårsdagens data være tilgjengelig i dag? Merk at dette ikke er det samme som granularitet. Det kan godt være nødvendig å kunne skille data fra hverandre på et daglig nivå, men ikke nødvendig å oppdatere datavarehuset oftere enn hver uke.
- Det kan være lurt å allerede nå tenke på hvor lenge data skal ligge i datavarehuset. Noen endelig avgjørelse trenger man ikke å ta på dette tidspunkt, men man bør vite hvorvidt det er snakk om 2 eller 10 år.

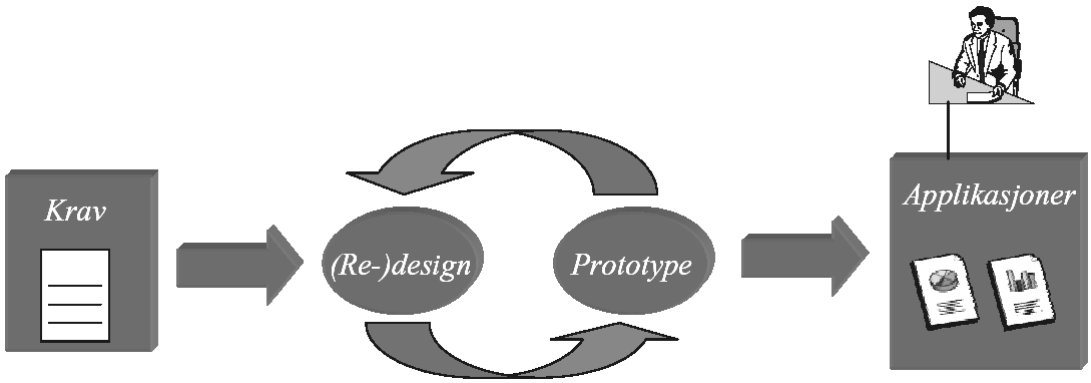
Den naturlige rekkefølgen å gjennomføre disse punktene vil være å begynne øverst og så jobbe seg nedover. Men for å kunne svare på de må man ha god kjennskap til brukernes og bedriftens behov. Derfor er det som tidligere nevnt lurt å foreta intervjuer av de forskjellige nøkkelmen. Kommunikasjonen med brukerne er meget viktig og det er derfor bare en fordel å opprette kontakt på et tidlig tidspunkt. På den måten unngår man at gapet mellom brukernes behov og systemets muligheter blir så stort at det påvirker bruken av datavarehuset. Dersom brukerne ikke benytter seg av tilbudet vil ikke IT-avdelingen få de tilbakemeldingene de trenger for å tilpasse systemet, og man er inne i en ond sirkel. Prosessen som er forklart ovenfor er utdypet ytterligere i [WHI].

4.5 ETL-rutinene

Når man skal utvikle ETL-rutinene, er en mappingmatrise godt å benytte. Bruk den med utgangspunkt i Logisk datamodell v2. Man designer mappingene fra kildesystem til datamodellen, og dokumenterer datatransformasjonene. Deretter tar man med utgangspunkt i mappingmatrisen å begynner å utvikle rutinene. Enten om man benytter seg av ETL-verktøy eller ved hjelp av egenutviklede programmer. Det er viktig å dokumentere disse godt etterpå, da slike systemer kan bli omfattende.

4.6 Applikasjoner

Når det som er gjennomgått er på plass, skal man tilbake til kravspesifikasjonen for å utarbeide brukerverktøyene. Dette bør foregå i tett samarbeid med brukerne slik at de har mulighet for å gi tilbakemeldinger. Der det er mulig, bør maler utarbeides slik at brukeren kan hjelpe seg selv. Det vil også gi gevinst i form av bedre kjennskap til verktøyet [RU].



Figur 4.4 - Applikasjonsutviklingen

5. Prototyping: Et implementert datavarehus

For å øke verdien av denne rapporten som veiledning har det blitt utviklet en prototype av et datavarehus som baserer seg på trafikkinformasjon. Dette kapittelet tar for seg hvordan jeg gikk frem for å konstruere et datavarehus ut ifra et datasett jeg hadde tilgjengelig. For å gå frem her ble livssyklusmodellen som er vist ved Figur 4.1 i forrige kapittel benyttet i den grad det gikk.

5.1 Prosjektplanleggingen

5.1.1 Datagrunnlag

Grunnlaget for denne prototypen, er loggdata fra ett prosjekt ved AAU. I det prosjektet var det 20 biler som kjørte rundt i Aalborg mens de var utstyrt med en GPS-måler. Denne noterte hastighet, posisjon, antall satellitter ved måling og tidspunkt. Denne rutinen utførte den hvert sekund, og skrev informasjonen til en logg. Denne loggen er vist ved et utdrag under.

```
ID,UNIQUE,BilNr,Driver,DATE,TIME,X,Y,MPX,MPY,SAT,HDOP,LIMIT,SPEED,VEJKODE,USERINPUT
383,11060201090254,11,127,60201,90254,552057,6302687,552058,6302687,7,1,0,38,-9,0
384,11060201090255,11,127,60201,90255,552056,6302697,552056,6302697,7,1,0,38,-9,0
```

Hver rad i denne tilsvarer en fact. Her i form av at bilen har blitt registrer med hastighet og posisjon. De forskjellige feltene bruker til:

ID	Sekvensielt nummer på raden.
UNIQUE	En unik kode bygget opp av bilNr, DATE og TIME.
BilNR	Id for hvilken bil det gjelder.
Driver	Id for hvem som er sjåfør.
DATE	Dato i form av tall.
TIME	Tid i form av tall.
X	X-posisjon mottatt fra GPS.
Y	Y-posisjon mottatt fra GPS.
MPX	X-posisjon når kjøretøy har blitt ”mappet” over på veinettet.
MPY	Y-posisjon når kjøretøy har blitt ”mappet” over på veinettet.
SAT	Antallet satellitter benyttet av GPS for å fastslå posisjon.
HDOP	Verdi for hvor troverdig målingen av posisjonen var.
LIMIT	Fartsgrense på veien som kjøretøyet befinner seg på.
SPEED	Hastighet holdt av kjøretøyet.
VEJKODE	Kode for hvilken vei kjøretøyet befinner seg på.
USERINPUT	Ikke oppgitt.

Som man kan se, er det også annen informasjon som er gitt her. Denne informasjonen har vært lagt til senere ved AAU, ved hjelp av rutiner. I tillegg er det 2 andre logger, hvor den ene inneholder tabeller for å gjøre om veikode til veinavn, og den siste har GPS-posisjonene til veinettet i Ålborg med fartsgrense. Utdrag av disse er gjengitt nedenfor.

```
Vejkode,Vejnavn
0068,ABELSVEJ
0073,ABILDGÅRDSVEJ

X-coord,Y-coord,Vejkode,kmt,Unique
55430572,632455870,7486,50,23
55430979,632457914,7486,50,23
```

Den øverste loggen er nokså selvsigende. Den nederste loggen koordinater over veiene i Aalborg med fartsgrense. Feltene betyr følgende:

X-coord	X-koordinat
Y-coord	Y-koordinat
Vejkode	Kode for aktuell vei
kmt	Fartsgrense på veien ved det punktet
Unique	Felt for pris per km

For å sikre at dataene var korrekte, ble følgende sjekket etter duplikater:

- Date og Time i logg 1
- På Vejkode i logg 2

Det ble også tilført veinavn på de veikoder som ikke eksisterte i logg 2.

De duplikater som ble funnet, slettet jeg de som var ugyldige (manuell sammenlikning)
For å forenkle prosessen med Date og Time dimensjonene, ble klokkeslett og dato gjort om til UTC-verdi (Se forkortelser).

5.1.2 Spesifikasjon av krav

Målet var at det skulle lages ett fungerende datavarehus med hensyn på den informasjonen som var tilgjengelig fra loggdataene. En god måte er da å første sette seg ned og prøve å finne ut spørringer som kan være av interesse. Det ble kommet frem til følgende:

- Fartsovertredelser
- Trafikkbelastning
- Kjøreruter
- Kraftige bremsing
- Gjennomsnittshastighet
- Trafikkork

Og så setter man opp de dimensjonene man mest sannsynligvis kan tenke seg å benytte, slik at man øker bevisstheten over hva man ønsker å oppnå:

- Tid
- Dag
- Bil
- Sjåfør
- Sted

Siden jeg nå har begynt få oversikten over hva databasen bør inneholde, har jeg bestem meg for å ta i brug en Relasjonsdatabase for ROLAP. I tillegg ønsker jeg å benytte meg av et stjerneskjema på databasen. Da kan jeg gå over på prosessen med å finne ut hva som kreves av databasen for å implementere de ønskede spørringene.

5.1.2.1 Fartsovertredelser

Det vil være forskjellige typer av fartsovertredelser som er interessant. Derfor bør det være mulig å skille mellom å se på hvor det er store overtredelser og hvor det er mange overtredelser, små som store. Man kan oppgi fartsovertredelser som en measure i fact-tabellen eller som en dimensjon. Hvis man benytter seg av en dimensjon, oppnår man en større fleksibilitet i form av utforskning av gruppene. Man kan benytte seg av forskjellige metoder for å klassifisere en overtredelse. Det kan være i form av hastighet, prosent eller en kombinasjon. Man kan f.eks. benytte seg av satsene som benyttes når politiet bestemmer et forelegg. Dette er forøvrig ikke metoden benyttet i dette datavarehuset, p.g.a. feil fra meg. Dette er forklart nærmere i tabellbeskrivelsen som kommer på side 35.

Løsning: En dimensjon som klassifiserer fartsovertredelsene i henhold til ønskede grupperinger. Faktoren benyttet regnes ut ved å finne differansen, og dele denne på 0.2. Verdien gjøres om til heltall ved en floor-funksjon.

5.1.2.2 Kraftig bremsing

Kraftig bremsing ved kjøring, tyder ofte på at det har dukket opp noe uforutsett mens man kjører. Det kan være at veien er uoversiktlig, slik at det oppstår en farlige situasjon hvor man reagerer ved å bremse kraftig. Ved å overvåke når slike oppbremsinger skjer, kan det gi en god indikator på hvor veiløsningen ikke er optimal, og tiltak som omlegging, skilting o.l. bør vurderes.

En bremsefaktor kan ganske lett lages ved å ta hastigheten ved forrige måling, og så prosessere det med nåværende fart. Da må man også passe på at verdiene kommer etter hverandre med riktig intervall, for at de skal kunne benyttes. Verdi bør oppgis i en bremsefaktor, og ikke prosent eller ren hastighet. En prosentverdi vil fungere slik at en kraftig oppbremsing i 100 km/h kan bli lik en liten brems i 50 km/h. Det gir ikke riktig forhold. Ren hastighet vil også være galt, fordi når man dobler hastighet så kvadrerer man bremselengden. Det betyr at man ved høyere hastighet bruker lengre tid på å redusere farten med like mye som ved en lavere hastighet. Formelen for dette er:

$H^2 \bullet X = B$, H = Hastighet, X = Bremsefaktor, B = Bremselengde

Dette betyr at en bremsefaktor kan mer korrekt regnes ut ved å kvadrere hastighetene før man finner differansen av de. En ulempe er hvis hastigheten beregnes ut ifra hastighetsmåleren, så vil det ved blokkering kunne gi store utslag på faktoren. Derfor bør en øvre grense settes på faktoren. Her er det heller ikke mulig å ta høyde for variering i veiforholdene. Det er avhengig av bil, dekk, veidekke og værforhold og kan være vanskelig å korrigere. Forhåpentligvis har liten innvirkning på det faktiske resultatet over hvor de farlige situasjoner oppstår, men det er det vanskelig for meg å uttale meg om på dette tidspunktet. Her må det sannsynligvis prøving og feiling til.

Det kan også være interessant å se på hvilke hastigheter biler med kraftig bremsing har. Men da vil det være mest fornuftig å se på hastigheten den hadde før nedbremsingen.

Løsning: En felt i fact-tabellen som oppgir bremsefaktoren. Denne regnes ut ved $(\text{HastighetNå})^2 - (\text{HastighetFør})^2$. En dimensjon med forskjellige grader av oppbremsing vil gi bedre mulighet for utforsking mot andre verdier som hastighet. Da bør også et felt med hastigheten før bremsing legges ved.

5.1.2.3 Gjennomsnittshastighet

For å finne gjennomsnittshastighet på en strekning, kan man summere hastigheten til alle kjøretøy på strekningen og så dele på antallet. Men vi datavarehuset kan vi ikke summere alle registreringer på strekningen og dele med antallet. Det kommer av at man logger bilene ved gitte tidsintervall. Derfor vil en bil som kjører i 50 km/h på en vei bli registrert dobbelt så mange ganger som en i 100 km/h. I dette tilfelle ville oppgitt måling av gjennomsnittshastigheten være 66 km/h. Men det vil være gjennomsnittshastigheten i forhold til tiden som benyttet på veistrekningen. Det er derimot ønskelig å vite gjennomsnittshastigheten på strekningen som er 75 km/h.

Løsning: For å løse dette kan man dele opp veiene i mindre segmenter man vet avstanden på. Disse legges inn i hierarkiet under veien. Deretter brukes antall registreringer og kjøretøy til å regne ut gjennomsnittshastigheten for disse segmentene. Tilpasning av segmentlengden er viktig. Ved for kort avstand, vil det kunne skje at en bil ikke noteres på segmentet. Derfor bør lengden være minst tiden det tar å kjøre mellom hver måling i en nokså høy hastighet. På den andre siden, hvis segmentet er for langt vil feilmargin ved avkjøring eller stans øke.

5.1.2.4 Trafikkbelastning

For å finne ut hvor mye belastningen er på aktuelle områder, bør man vite antall biler som er i det området. Hvis man ønsker å se på hvor mange biler som kjører på denne veien over ett tidsintervall, er det viktig å huske på at man ikke bare kan summere antallet. Det skyldes at hver bil blir registrer flere ganger i løpet av den tiden han bruker på veien, som igjen gjør at den bli telt mange ganger. Derfor er det i dette datavarehuset kun mulig å se på belastning av en veistrekning ved et bestemt tidspunkt.

Løsning: Telle antall kjøretøy på aktuell strekning ved ett bestem tidspunkt.

5.1.2.5 Kjøreruter

Ved å se på hvor og hvilke ruter kjøretøy velger, kan man trekke ut mye nyttig informasjon ved strekninger benyttet som f.eks.:

- Er den kollektive transporten for dårlig?
- Burde alternative ruter skiltes bedre?
- Fungere planløsningen godt/dårlig?

For å kunne si noe om dette, må en rute defineres. En stopp kan være reel dersom bilen står stille over lengre tid. Å finne ett passe intervall er viktig for kunne skille mellom veikryss, bensinstopp og en reel stopp. Men uansett vil det bli ganger der de blir tolket feil. Det mest optimale intervallet kan finnes ved forsøk, men jeg vil anta det ligger i området 7-15 minutter et sted. En metode for å bedre dette, kan være å ha rutiner for å snappe opp steder man pleier å ha lengre stans (f.eks. lengre enn 1 time). Så ved en kort stans på dette stedet, forstår systemet at dette er en også er en stans. Det vil kreve mye resurser med oppdatering av systemet. Dette er ganske omfattende og det er derfor ikke blitt fokusert på.

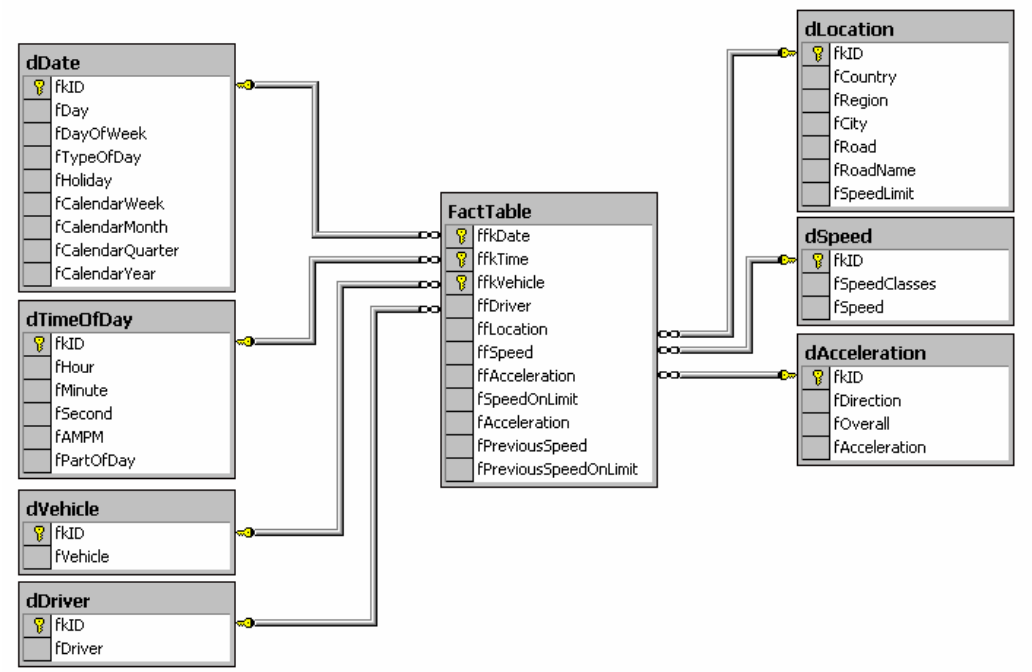
5.1.2.6 Trafikkork

For å kunne detektere hva en kork er, må man som ved en rute, igjen bestemme seg for hva som definerer en kork. En kork er ofte saktegående eller stillestående samling med biler. Å sjekke flere biler mot hverandre er en stor operasjon og krever mye resurser. I stedet kan man prøve å lage en god nok definisjon på basis av hva den enkelte bil foretar seg. Ofte hvis en kø dannes p.g.a. ulykke, vil det være køer som er så stillestående at de egentlig kommer inn under definisjonen av en stopp. Dette kan ungås ved å "svarteliste" steder, hvor det mest sannsynlig er en kork. Slike veier er store motorveier, og andre steder der det virker usannsynlig at en bil faktisk har stanset. For å bedre yttligere, kan det ved stans sjekke i hvilken grad hastigheten ble redusert ned til stans. Når man så summerer, vil de stedene hvor det faktisk har vært kork, overdøve de steder der kun en bil har stanset grunnet forskjellen i antall biler registrert i kork.

Løsning: felt i fact-tabellen som settes 1 dersom en situasjon kommer under korkdefinisjonen. Dette avgjøres ut ifra hastighet og "svartelisten".

5.2 Design

Figur 5.1 viser hvordan databasen i datavarehuset er bygd opp. Her ser vi hvordan alle dimensjonene er linket til fact-tabellen. Slik som det er bygd opp i MS SQL server, så passer relasjonen på at det ikke eksisterer en verdi i fact-tabellen som mangler noen dimensjonsverdier. I fact-tabellen er det kombinasjonen av tid, dato, og bil som utgjør primary key. Det gjør at ingen målinger kan settes inn mer enn 1 gang.



Figur 5.1 - Datavarehuset, slik det er bygd opp.

5.2.1 Tabellbeskrivelser

5.2.1.1 FactTable

Fact-tabellen inneholder referanser til dimensjonene og measures. Jeg har splittet opp hendelsestidspunkt i to dimensjoner. Det er en dimensjon for datoen , og en for tiden på døgnet hendelsen fant sted. Det er for å slippe å ha en veldig stor tabell. Jeg har ikke benyttet meg av en egen dimensjon for hastighetsoverskridelse, da det var noe jeg innså først etterpå etter hint fra Prof. Christian S. Jensen. I stedet benyttet jeg fSpeedOnLimit som regnes ut ved å trekke fartsgrensen ifra hastigheten. I etterkant oppdaget jeg også at jeg burde ha implementert hastighet som en measure, da denne i noen tilfeller er mer logisk å summere antallet på. fPreviousSpeed og fPreviousSpeedOnLimit er lagt til for å kunne utforske på sammenhengen med hastighet og nesten-ulykker (kraftig brems).

Maksimum antall rader per bil per år: 3.153.600
En rad tar 28 bytes, som dermed tar 84 Mb per bil per år.

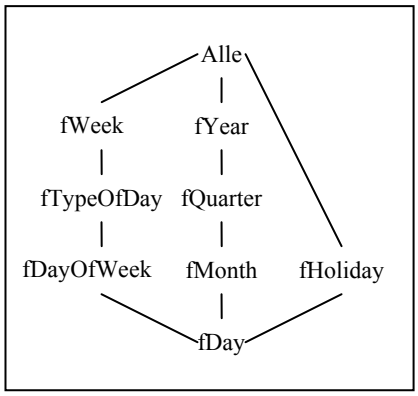
Dimensjonen inneholder:
ffkDate: Referanse til dimensjonen dDate.
ffkTime: Referanse til dimensjonen dTimeOfDay.
ffkVehicle: Referanse til dimensjonen dVehicle.
ffDriver: Referanse til dimensjonen dDriver.
ffLocation: Referanse til dimensjonen dLocation.
ffSpeed: Referanse til dimensjonen dSpeed.
ffAcceleration: Referanse til dimensjonen dAcceleration.
fSpeedOnLimit: Faktor for forholdet mellom hastighet holdt og fartsgrense
fAcceletation: faktor for akselrasjon og deselerasjon.
fPreviousSpeed: Hastighet holdt ved forrige måling.
fPreviousSpeedOnLimit: Tidligere hastighet holdt i forhold til fartsgrense

5.2.1.2 dDate

Dette er datodimensjonen i datavarehuset. Dimensjonen inneholder informasjon om dagen for hendelsen som er interessant å se på. Jeg har plassert fHoliday utenfor, fordi jeg ønsker å kunne se på ferie over år. Jeg har lagt ukedag og type dag under ukene for å kunne se trender over ukene som går.

Maksimum antall rader per år: 365
En rad tar 65339 bytes, som dermed tar 64 Kb per år

Dimensjonen inneholder:
fkID: Unik id.
fDay: Hvilken dag hendelsen inntraff.
fDayOfWeek: Hvilken ukedag hendelsen inntraff.
fTypeOfDay: Om hendelsen inntraff i en helg eller på en ukedag.
fHoliday: I dette tilfellet vurderes det om det er jul eller sommerferie.
fWeek: Hvilken uke hendelsen inntraff. Her oppgis datoen på dagen uken sluttet.
fMonth: Hvilken måned hendelsen inntraff.
fQuarter: Hvilket kvartal hendelsen inntraff.
fYear: Hvilket år hendelsen inntraff.

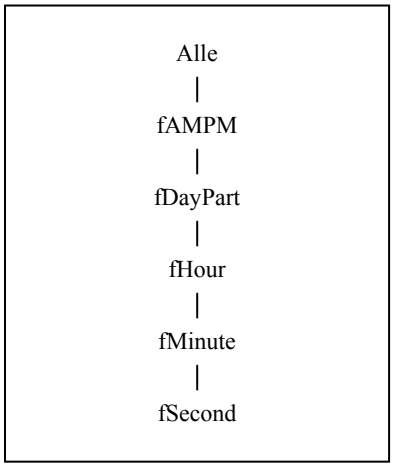


Figur 5.2 - Hierarkiet til dDate dimensjonen

5.2.1.3 dTimeOfDay

Ved første forsøk, hadde jeg redusert antall målinger fra hvert sekund til hvert 10ende sekund. Men etter å ha testet datavarehuset i praksis, fant jeg ut at jeg ville benytte meg av alle målingene som eksisterte i kildedataene. Normalt vil det neppe være interessant å se på trafikkinformasjonen for hvert sekund, men av hensyn til å kunne kikke på trafikkbelasting, var det nødvendig å implementere dimensjonen med registreringer ned på sekundet.
Maksimum antall rader: 86400
En rad tar 39 bytes, som dermed tar maksimum 3.22 Mb

Dimensjonen inneholder:
fkID: Unik id.
fHour: Hvilken time registreringen ble gjort.
fMinute: Hvilket minutt registreringen ble gjort.
fSecond: Hvilket sekund registreringen ble gjort.
fAMPM: Om registreringen ble gjort AM eller PM.
fDayPart: Hvilken tid på døgnet det ble registrert. F.eks. Afternoon



Figur 5.3 - dTimeOfDay

5.2.1.4 dVehicle

Denne dimensjonen kunne inneholdt spesifikasjoner på bilen, for videre utforskning, men siden det ikke er tilgjengelig i disse måldataene, er kun bil oppgitt.

En rad tar 32 bytes.

Dimensjonen inneholder:
fkID: Unik id.
fVehicle: Nummer på kjøretøy.

5.2.1.5 dDriver

Denne dimensjonen kunne inneholdt spesifikasjoner på sjåføren, for videre utforskning, men siden det ikke er tilgjengelig i disse måldataene, er kun sjåfør oppgitt.

En rad tar 32 bytes.

Dimensjonen inneholder:
fkID: Unik id.
fDriver: Nummer på sjåfør.

5.2.1.6 dLocation

I denne dimensjonen lagres informasjon om hvor bilen befant seg ved registrering. I det datavarehuset som er utviklet, var ikke by, region og land med i registret og derfor er alle disse registrert med bare ett navn hver. Derimot kunne jeg har utnyttet meg av koordinatene som var gitt med, men valgte å utelate disse da måldatamengden var såpass begrenset.

En rad tar 122 bytes.

Dimensjonen inneholder:
fkID: Unik id.
fCountry:
fRegion:
fCity:
fRoad
fRoadName

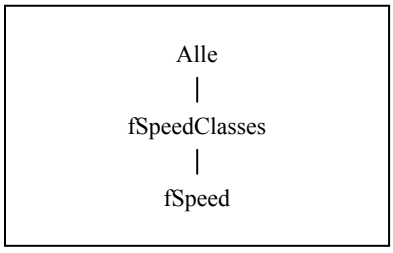
Figur 5.4 - dLocation

5.2.1.7 dSpeed

Denne dimensjonen benyttes for å aggregere med hensyn på hastigheten kjøretøyene holder. SpeedClasses deler in i en grovinndeling på 0-10, 10-50, 50-80 og over 80.

En rad tar 32 bytes.

Dimensjonen inneholder:
fkID: Unik id.
fSpeedClasses: Skiller ved følgende 10, 50, 80.
fSpeed: Hastighet i oppløsning på 10 km/h



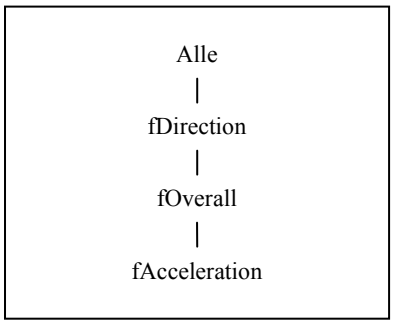
Figur 5.5 - dSpeed

5.2.1.8 dAcceleration

I denne dimensjonen er informasjon kjøretøyets hastighetsendringer. Det kan benyttes for å oppdage steder der det ofte oppstår nestenulykker. Jeg valgte meg noen grenser for fOverall verdiene, men det bør testes hva som er gode intervaller på dette ved forsøk.

En rad tar 32 bytes.

Dimensjonen inneholder:
fkID: Unik id.
fDirection: Angir om bilen akselererer eller deselererer
fOverall: Grovere oppløsning i form av minor, medium og major hastighetsendring
fAcceleration: Hastighetsforandringen i intervall på 10



Figur 5.6 - dAcceleration

5.3 Import av data

Nå begynner prosessen hvor vi skal overføre datagrunnlaget over til datavarehuset. På veien må disse dataene omformes, slik at de passer inn i datavarehuset. Vanligvis så er det dette som er ETL-prosessen (Extraction-Transformation-Loading), men i dette tilfellet er ikke det helt korrekt. ETL-prosessen er laget slik at det er datavarehuset som henter dataene fra databasene i stedet for at datakildene dytter data inn i datavarehuset. Det er mer hensiktsmessig da det kan være mange forskjellige datakilder som oppdaterer datavarehuset. Det gir mulighet for hva som legges hvor i datavarehuset når. Men siden det i dette prosjektet er kun en kildefil, så er den optimalisert for å kun kjøres en gang.

Siden dataene var levert på kommaseparerte tekstfiler, startet jeg med å importere disse til MS SQL server som en enkel flat database. Det valgte jeg fordi det er lettere å håndtere spørringer til en databasen, enn å lese ut ifra en tekstfil.

For å omforme og å sette inn dataene, benyttet jeg meg av JScript. Det skyldtes at jeg fra tidligere har god erfaring med JScript, og det har god støtte for å utnytte UTC-tid. Når man skal hente slik type dag den 13.05.2002 var, finns det funksjoner i JScript som returnerer det veldig enkelt. Koden som for å gjøre om dato og tid ligger i Vedlegget kap. 10.1.1.

Så var det tid for å begynne på konverteringen. Legger ved datagrunnlaget igjen, slik at
Det blir lettere å holde oversikten

```
ID,UNIQUE,BilNr,Driver,DATE,TIME,X,Y,MPX,MPY,SAT,HDOP,LIMIT,SPEED,VEJKODE,USERINPUT
383,11060201090254,11,127,60201,90254,552057,6302687,552058,6302687,7,1,0,38,-9,0
```

Måten jeg lot programmet jobbe seg igjennom databasen da, var å ta rekkene sekvensielt sortert ut ifra tidspunkt. For hver rekke, blir en funksjon kalt opp til hver dimensjon. Funksjonen får sendt med nødvendig data, for å sjekke om den eksisterer. Hvis ikke, blir den lagt til i den dimensjonen. I begge tilfeller returneres id til den korrekte raden. Når alle dimensjoner er sjekket, legges alle referanser og measures til i fact-tabellen.

Alle dimensjonene er lette å sjekke om de allerede inneholder gjeldende data. Disse har kun en verdi på laveste nivå som er unik for hver rad. De aller enkleste dimensjonene er dDriver og dVehicle, som man bare trenger å sette inn tallet som er oppgitt i datakilden, uten noe mer håndtering. Dimensjonene dDate, dSpeed, dAcceleration, dTimeOfDay må man endre litt på, før det kan settes inn. Dette er vist ved de respektive funksjonene som er vedlagt hvordan de håndteres. Dimensjonen dLocation er i dette tilfellet også bare lagt inn ved sjekk av ”road”. Men i realiteten så vil ikke dette være aktuelt å gjøre, siden en vei kan høre til under flere byer, fylker eller land for den saks skyld. Det vil føre til at man ikke kan finne informasjon for f.eks. E6 i Grimstad. Løsningen er å lage et ekstra nivå, hvor vei kombineres med neste overliggende. I dette tilfellet by. Grunne til at region ikke ble implementert, var at de eksisterte i en Oracle databasefil. Da jeg hadde lite tid og kunnskap til å sette meg inn i Oracle, fikk jeg ikke til den importeringen.

Først ble det tidligere nevnte programmet for å generere UTC-verdier ut ifra dato og klokkeslett laget og prosessert. Men jeg støtte på ett nokså underlig problem her. Når jeg skulle kjøre denne på det største datasettet (den med alle gyldige målinger), gikk programmet i lås når den skulle oppdatere feltet UTC for den gjeldende raden. Og etter flere timers frustrasjon med prøving og feiling, fungerte det plutselig. Hva som forårsaket dette forblir uvisst desverre, men det virket som om det hadde noe med at datasettet var på så mange rader (70.000 stk).

Når programmet var ferdig skrevet, ble det kjørt på serveren via kommandolinje med cscript.exe. Jeg importerte først en miniversjon med kun målinger tatt hvert 10ende sekund, men siden databasen ikke ble spesielt stor, laget jeg like gjerne datavarehuset med alle gyldige data.

Da databasen var på plass i Microsoft SQL Server, benyttet jeg ”DB Converter” for å importere databasen over i MySQL server. DB Converter er et lite gratisprogram som kan konvertere Access databaser over til MySQL, Oracle og PostgreSQL. Dette fungerte som det skulle unntatt når det skulle importere dataene fra Fact-tabellen. Da gikk hele programmet i stå. I stedet laget jeg en løkke som hentet en og en rad, og la den inn i MySQL databasen. Dermed hadde jeg to identiske databaser som jeg kunne teste ytelsen på og sammenlikne.

5.4 Resultat

5.4.1 Ytelsesammenlikning

For å kunne teste ytelse ved forskjellige databaseservere, installerte jeg Microsoft SQL Server og MySQL Server. I disse var det lagt inn to identiske datavarehus. Mot disse benyttet jeg Seagate Analysis (SA) for å kjøre spørringene. Det er et OLAP-verktøy som er veldig enkelt å benytte, og har de viktigste funksjonene. Desverre var det et problem mellom server og klient med MySQL databasen. Når en spørring ble kjørt, frøs serveren i ca 2 minutter før den kom seg videre på vanlig måte. Dette hendte ikke med MS SQL serveren. Derfor fikk jeg desverre ikke tatt noen sammenlikning mellom disse databasene.

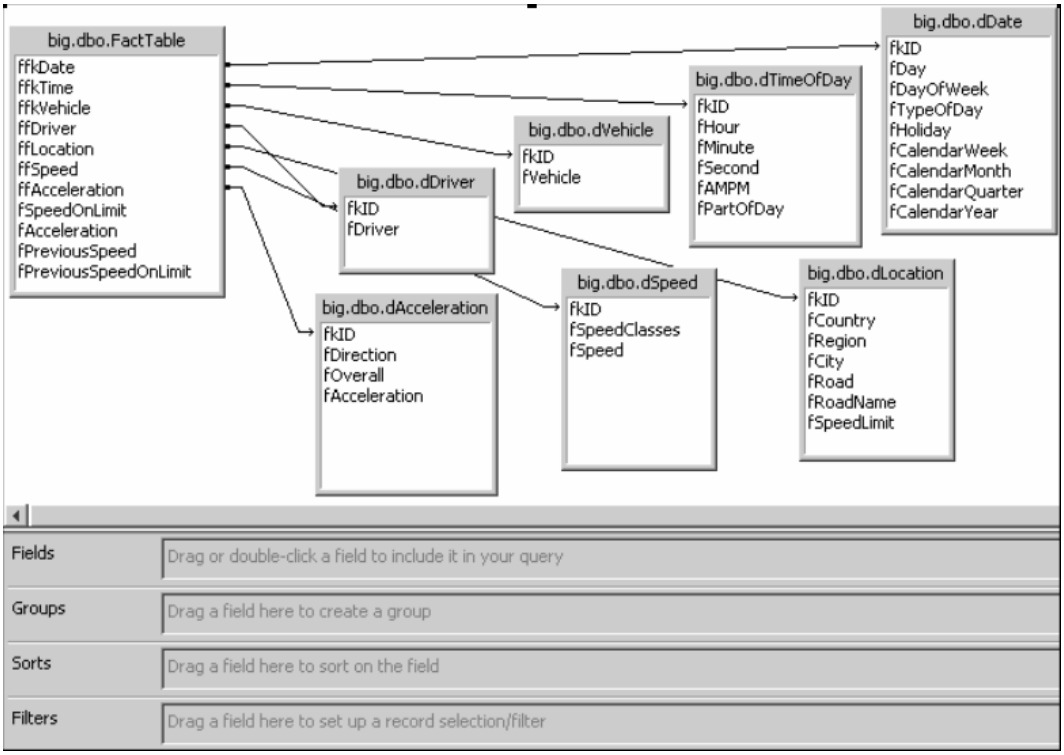
5.4.2 Spørringene

De fire påfølgende spørringene var tenkt å kunne utforske i dette datavarehuset:

- Fartsovertredelser
- Kraftige bremsing
- Trafikkbelastning
- Trafikkork

Jeg benyttet meg av SA her også. Den benytter ROLAP, og er veldig enkel å benytte. Men jeg fikk erfare at det var ett par ting med verktøyet som ikke fungerte helt tilfredsstillende. Bland annet var det at man ikke kunne sortere rekkefølgen på verdiene til dimensjonen når de er satt inn i kuben. Derfor er det ofte en ganske ulogisk rekkefølge som er presenter i bildene fra kubeanalysene.

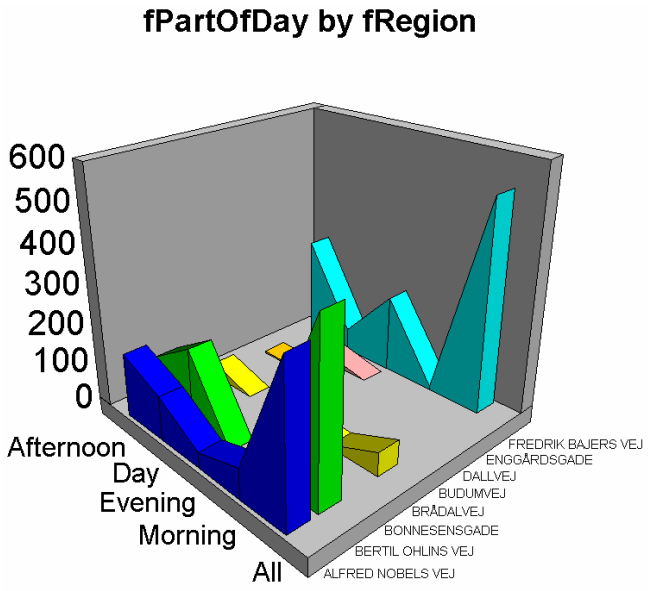
Error! Reference source not found. viser hvordan man i SA setter opp relasjonene i datavarehuset. Deretter drar man de feltene man ønsker å benytte i aggregeringen ned i rutene under.



Figur 5.7 – Seagate Analysis

Fartsovertredelser

Når man skal utforske trendene ved fartsoverskridelser, er det feltet fSpeedOnLimit man benytter seg av. Som beskrevet i Kapittel 5.1.2.1, er dette en rate på hvor stor overtredelsen er. I dette forsøket ønsket jeg å finne ut i hvilken gate politiet bør stå, og når på døgnet.



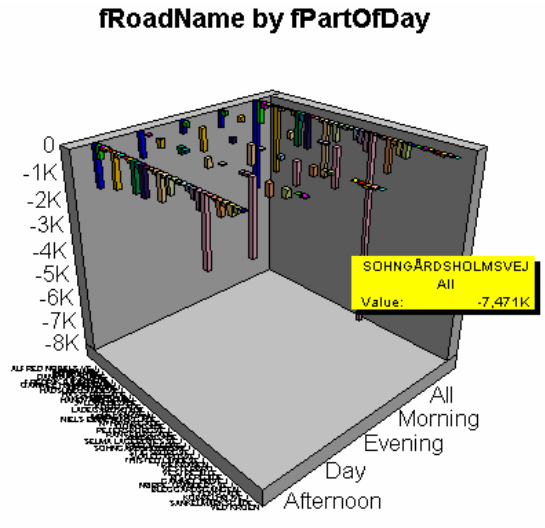
Figur 5.8 – Fartsoverskridelser

For å få til dette måtte jeg sette *field* til SpeedOnLimit. Ved å summere denne verdien, vil det være det stedet som har overvekt av høye overtredelser men også små komme tydelig frem. Dimensjonene som ble benyttet var fPartOfDay og fRoadName. Ett utdrag av resultatet er vist ved Figur 5.8. Grunnen til at overskriften skriver fRegion skyldes at jeg satt opp hele hierarkiet nedover fra region. Dermed blir den satt inn automatisk.

I grafen kan vi se at det på morgenen er mange fartsoverskridelser i Fredrik Bajers Vej, mens politiet mot dagen bør trekke til Bertil Ohlins Vej.

5.4.2.1 Kraftig bremsing

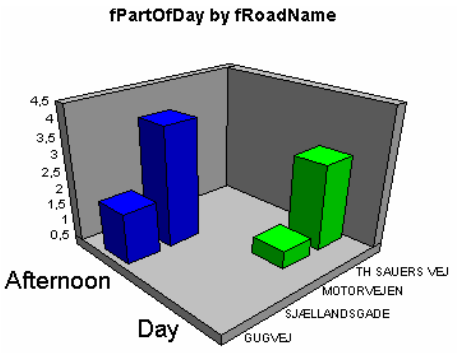
For å finne når og hvor det skjedde kraftige nedbremsinger, må man aggregere på fAccelration. Men siden denne regner ut både akselerasjon og deselerasjon, må man passe på å tilføre en begrensing i form av filtrering. Hvis man bare summere over hele linjen vil akselrasjonene jevne ut deselerasjonene, og vi får gale verdier. Dette løses ved å filtrere bort de radene hvor fAcceleration er over eller lik 0.



Figur 5.9 - Nedbremsinger

Figur 5.9 viser resultatet av dette. Men her kommer også to andre svakheter ved SA frem. Det første er at man ikke kan velge hvilken retning verdiene skal gå. Dermed bli grafen "på hodet", og det kan være vanskelig å få oversikt. En annen svakhet er muligheten for hvordan forklaringene på rekkene skal vises. Disse er låst i bunnen, og ved mange rekker slik som her blir det bare grøt. Den gule lappen dukker opp når man peker på en søyle, og gir litt mer informasjon. Vi kan ut i fra diagrammet se at det er på ettermiddagen de fleste oppbremsinger skjer. Dette skyldes kanskje at man vil hjem fra jobben og kjører litt for fort.

En mulighet til, er å kombinere å se på sammenhengen mellom hastighet og kraftige nedbremsinger. Hvis det er steder som utpeker seg for at mange har høy hastighet, og så må plutselig bremse kraftig så kanskje tiltak som "Pass farten!"-skilt er på plass. Måten dette gjøres på er å aggregere på PreviousSpeedOnLimit, og benytte dimensjonen dAccelerasjon for å klassifisere graden av nedbremsing. Figur 5.10 viser at det er på ettermiddagen på Sjællandsgade dette hender. Men her ser vi også at det mangler en del datagrunnlag siden det er så få som oppfyller disse kriteriene.



Figur 5.10 - Hastighet over fartsgrense, etterfulgt av medium eller kraftigere nedbremsning

5.4.2.2 Trafikkbelastning

For å finne trafikkbelastningen, så kan man i dette tilfellet bare kikke ved såkalt snapshot. Det vil si at man teller biler ved et bestemt tidpunkt. Det gjøres i filteret ved å sette dDate.fDato = ”ønsket dato” og dTimeOfDay.fSekund = ”ønsket tidspunkt”. Dette fører til at hver bil maks kan bli telt 1 gang.

5.4.2.3 Trafikkork

Som nevnt i kapittel 5.1.2.6, så vil den eneste formen for sjekk etter kork, gå på hastigheten. Dette gir er ganske ukorrekt bilde, og anbefales ikke. Men måten det gjøres på er ved å aggregere på de verdiene som har en hastighet mellom 0 og 10 km/t.

6. Drøfting

6.1 Programvare

6.1.1 Server

Det datavarehuset jeg implementerte, hadde jeg på både Microsoft SQL Server og MySQL. Planen med dette var å kunne kjøre de samme spørringene på begge, slik at jeg kunne se hvilken som ga best performance i "real-life". Desverre gikk ikke dette da en ukjent feil rammet MySQL. Når analyseverktøyet sendte en forespørsel, frøs maskinen i 2 minutter før den forsatte vanlig. Dette skjedde uavhengig av størrelse på datasettet den skulle hente. Men foruten om det bar ikke MySQL på store problemer. Selv om jeg aldri har benyttet meg av denne databasen før, så gikk det ganske smertefritt å få serveren opp å gå. Men den krever litt mer innsats fra brukeren for å lære seg.

Microsoft SQL Server har ett veldig greit grafisk grensesnitt, og gjør det lett for en som ikke har benyttet databasen før å komme i gang. Jeg syntes det er bra å kunne få et skjema over databasen og sammenhengene på en god grafisk måte, og der har SQL Server en liten fordel. Men det forekom noen besynderlige feil med denne også. Det hendte noen ganger at det kodede programmet ikke fikk tilgang til databasen, eller bare over kort tid. Dette ble svært frustrerende en stund da jeg måtte sjekke opp og ned etter feil i koden. Og plutselig uten at noe spesielt var gjort, så fungerte det som det skulle igjen. Dette var ikke helt tilfredsstillende.

6.1.2 Verktøy

Jeg benyttet meg av Seagate Analysis (SA) i hovedsak for å teste ut datavarehuset. SA har et godt grafisk grensesnitt. SA er ikke lenger tilgjengelig på nett siden de har blitt kjøpt opp av Crystal Reports. Den versjonen jeg hadde var fra tidligere i høst og var en trial-versjon. Siden OLAP-verktøyene kan variere mye fra produsent til produsent, så vurderte jeg det dit hen at det ikke var noen hensikt å gå dypt ned i verktøyet fra Microsoft. SA benytter seg av ROLAP, og dermed lå egentlig alt til rette for å kunne kjøre tester av ytelse på de to forskjellige serverene, hadde det ikke vært for den tidligere beskrevet feil som rammet MySQL.

SA hadde ett par svakheter som jeg merket meg. Dette gjaldt:

- Kun count og sum på measures.
- Ikke mulighet for å distribuere en rapport uten at mottakeren har kobling til database.
- Ingen måte å skrive inn kommentarer på verdier.
- Overvåkning (varsel) av verdier ikke mulig
- Kløvete fast oppsett i kuber

Ellers arbeidet programmet ganske rask på generell basis. De fleste spørringene ble besvart på under 10 sekunder, noe som er tilfredsstillende i forhold til kriteriene til FASMI-standarden.

Prototypen som er laget i dette datavarehuset, er ikke laget for at noen skal ta i bruk det. Det er ment å fungere som en praktisk gjennomgang av hvordan man bygger et datavarehus. Derfor er det ingen konkrete mål som skal være tatt hensyn til og prøvd oppnådd. Men likevel var det mål som ble satt i innledningen av prototypingen som ikke har blitt implementert tilstrekkelig eller i det hele tatt. Denne avgrensingen er blitt beskrevet i innledningen, side 7. I tillegg vil jeg gå igjennom det jeg syntes kan være nyttig å arbeide videre med.

6.2 Design

Designen på datavarehuset er ikke helt optimalt som jeg erfarte etterhvert. Men det er likevel fullt mulig å hente den informasjon som var ønskelig.

6.2.1 Koordinatnett

I dette datavarehuset benyttet jeg meg ikke av koordinatene som var tilgjengelig i de opprinnelige dataene. I senere tid ser jeg at dette nok kunne vært benyttet for å oppnå bedre oppløsning på hvor hendelsene fant sted. Det kunne ganske lett gjøres ved å implementere en dimensjon der man grupperer koordinatene inn i kvadrater.

6.2.2 Kjøreruter

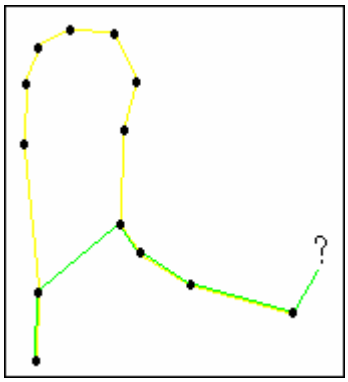
Som nevnt i kapittel 5.1.2.5 side 33, Er det ikke mulig å se på hvor og hvilke ruter kjøretøy velger. Dette har jeg liten formening om hvordan kan gjøres siden dette blir ganske omfattende.

6.2.3 Utvikling av datakilde

Siden datakilden har lagret posisjonene til hvor bilen er registret, vil det være mulig å videreutvikle dataene den inneholder. Jeg har vurdert følgene som interessante:

6.2.4 Segmenter av vei

Hvis man ønsker å ha bedre oppløsning en bare veg slik som jeg har benyttet med av, så er det 2 måter det kan løses på. Enten kan man benytte seg av de koordinatene som eksisterer i datakilden, eller man kan generere segmenter av veien. Å benytte seg av koordinatsystemet er relativ enkelt. Man kan bare lage en dimensjon som grupperer koordinatene i kvadrater. Men dette har også den ulempen at hvis 2 veier går tett nok inntil hverandre, så vil de telle inn på hverandre.



Figur 6.1 - Definering av punktrekkefølge

Derfor hadde en oppdeling av veien i segmenter vært å foretrekke. Men dette byr på noen problemer igjen. Det må lages en rutine som klarer å finne ut hvilken rute mellom punktene som er korrekt. Hvis man bare går etter den nærmeste kan det oppstå situasjoner som vist ved Figur 6.1. Det må bedømmes hvorvidt man skal benytte seg av tidsintervall mellom hver måling,avviksmåling ut ifra forrige retning eller avstandsmåling. Den sistnevnte vil gjøre arbeidet med å finne rekkefølge lettest, da man kan finne den neste ut ifra avstand. Men man må også huske på at dersom flere finnes innen den riktige avstanden, må en mer avansert rutine for å definere ruten kalles opp. Ulempen er dersom samme avstand må benyttes for by og bygd, så vil det bli veldig mange punkter etterhvert.

6.2.5 Oppløsning
I datakilden blir tilhørende vei for kjøretøy sløyfet dersom 2 eller flere veier er i nærheten (innenfor 10 meter) av hverandre, slik at usikkerheten for hvilken vei den tilhører blir for stor. Disse dataene kan utbedres ved å lage en rutine for å finne ut hvorvidt hvilken vei kjøretøyet fortsetter på, og ut ifra det bedømme tilhørende vei. Ved kryss kan man gå ut ifra at kjøretøyet holder seg på den gamle veien inntil f.eks. 3 sekunder før den er registrert på den nye veien. Det skyldtes at der er sjelden man blir stanset på vei ut av et kryss.

6.2.6 Data mining
Databasen er ikke testet sammen med noen form for data mining-verktøy. Dette kunne vært interessant for å se om det er noe som kunne trekkes ut av informasjonen som ikke er vært utforsket før.

6.2.7 Sakte forandrende dimensjoner
Noen av dimensjonene i et stjerneskjema inneholder data som for det meste er konstant og sjelden forandrer seg. Det kan f.eks. være informasjonen på sjåførene som sivilstatus, alder o.l. forandrer kun seg en gang iblant. Slike dimensjoner kalles gjerne ”sakte forandrende dimensjoner” [KIMB96]. Det er ikke tatt høyde for dette i prototypen. Når slike forandringer skjer er det viktig å vite hvordan man skal oppdatere datavarehuset i forhold til dette. Det er tre forskjellige måter å oppdatere datavarehuset på, og valget man gjør påvirker muligheten til å analysere de historiske dataene senere.

Den første og enkleste måten å gjøre denne oppdateringen på er å skrive over de gamle dataene i dimensjonstabellen. Denne metoden forhindrer at man kan avgrense data til før og etter forandringen. Noen ganger er ikke dette så farlig, men ofte vil det være nyttig å ha muligheten til å analysere forholdene med forandringen som et skillemerke. Dersom dette er tilfelle må man oppdatere datavarehuset på en annen måte.

For å bevare de historiske dataene kan man lage en ny post i dimensjonstabellen der den nye verdien er representert. Deretter benytter man det nye innslaget til fremtidige koblinger mot faktatabellen. For å kunne gjøre dette må man generere en ny nøkkelverdi til posten i dimensjonstabellen. I stedet for å benytte f.eks. personnummeret som nøkkel i en sjåfør-dimensjon kan man benytte personnummeret pluss et versjonsnummer. Antall siffer i versjonsnummeret avgjør hvor mange forskjellig «kopier» det kan være av posten. Håndteringen av denne prosessen gjøres av overføringssystemet, noe som krever at metadataene holder orden på hvilke nøkkelverdier som allerede er benyttet.

Som nevnt vil denne metoden føre til en splittelse i de historiske dataene. Dette vil gjøre det mulig å se på data før forandringen med den gamle verdien og data etterpå med den nye. Men noen ganger kan det være ønskelig med en mer fleksibel måte å analysere data før og etter forandringen. Siden metoden nevnt ovenfor splitter historien vil man f.eks. ikke ha muligheten til å benytte den nye verdien til en attributt på gamle data eller omvendt. For å få til dette må man benytte en tredje måte å registrere forandringen på. Man legger til et nytt felt i dimensjonstabellen som representerer «nåværende verdi». Samtidig forandrer man feltet med den gamle verdien til et felt som representerer «gammel verdi» eller «original verdi». I tillegg kan man også legge til et felt som forteller når forandringen fant sted. Denne metoden «husker» bare den originale verdien og den nåværende verdien, alle mellomliggende verdier går tapt (alternativt huskes den forrige og nåværende verdien). Dersom en mer detaljert historie er nødvendig må man benytte metode nummer to. For å kunne løse alle aktuelle problemstillinger hadde det vært mulig å kombinere metode to og tre, men resultatet av en slik kombinering vil ikke være særlig godt egnet for sluttbrukerne p.g.a. den kompleksiteten det vil innebære.

7. Konklusjon

Målet med denne rapporten er å kunne hjelpe andre som ønsker å ta i bruk et datavarehus for å kunne finne informasjon i trafikkdata. Derfor er det lagt større vekt på å få med teorien som ligger til grunn i et datavarehus. Det har vært både utfordrende og interessant å være alene om et slikt prosjekt, og personlig har jeg lært veldig mye om meg selv. Og det var nettopp derfor ville jeg ønsket å ha denne oppgaven på egenhånd.

Ettersom oppgaven skulle fungere som veiledning, ble jeg nødt til å prøve å trekke ut erfaringen jeg syntes var viktig. Dette merket jeg ikke var så lett da den eneste erfaring jeg hadde fra tidligere, var et miniprosjekt i halvåret før denne oppgaven. Men det var en utfordring som jeg håper jeg har klart på en god måte.

Prototypen er utviklet for å kunne utnytte de måldataene jeg hadde tilgjengelig, men det er forsatt muligheter for å utvide denne. Jeg valgte å legge noen begrensninger på utviklingen undervei, fordi jeg anså hensikten med prototypingen allikevel ble oppnådd.

Når jeg ser på teknologien datavarehus, så har jeg stor tro på den utvikling videre. Det har vært en del negativ omtale om slike prosjekter, men hvis man er grundig, og spesielt med forarbeidet, så er det liten grunn til at det ikke skal kunne lykkes. Selv er jeg er særdeles fornøyd med å vite at dette er den teknologien jeg vil utvikle meg innen.

8. Ordliste

8.1 Forkortelser

BI	Business Intelligence	Informasjon som firmaet innehar som kan heve bedriftens konkurransefortrinn
ETL	Extraction, Transformation, Loading	Fellesord for det å transformere en database over til et datavarehus
HOLAP	Hybrid OLAP	Introdusert av Microsoft, og er en blanding av MOLAP og ROLAP
MOLAP	Multidimensjonal OLAP	Verdiene i databasen preaggregeres og lagres på disk, for å kunne effektivisere kalkulering senere. Har ulempe at datamengden øker betraktelig
OLAP	Online Analytical Processing	Ett program/prosess som benyttes for å kunne trekke ut informasjon fra store datamengder.
ROLAP	Relational OLAP	Ved å benytte en relasjonsdatabase kan man mer effektiv arbeide seg rundt og utforske datasettet.
UTC	Coordinated Universal Time (ja, det er riktig rekkefølge)	UTC er en måte å oppgi tid på i form av millisekunder siden 00:00, 1.1.1970.
GPS	Global Positioning System	Rundt jorden ligger det 24 satellitter som benyttes for å beregne ens posisjon på jorden i lengde og breddegrad.

8.2 Ordforklaringer

Onto/Non-onto	Hvis deler av hierarkiet i dimensjonene er non-onto, så betyr det at det er forskjellige høyder på hierakiene.
Strict/Non-strict	Hvis et barn har flere foreldre i et hierarki, så er dimensjonen non-strict.
Aggregere	Å aggregere vil si å slå sammen flere facts over en dimensjon. Det kan f.eks. være å summere salg per uke per tid, for å hvordan salget varierer.
Normalisering	Det finnes flere typer normalisering, men hovedprinsippet er at man lager relasjoner mellom foreldre og barnetabeller for redusere repeterende grupper.
Covering/Non-covering	Når en rute i hierarkiet hopper over et trinn, så sier man at dimensjonen er non-covering.
Iterativ utvikling	Ved prosess: Gå tilbake i kjeden for å videreutvikle produktet
Fact-tabell	I en relasjonsdatabase er det den som binder seg til dimensjonene.

9. Referanser

Alle internettadresser ble sjekket og verifisert den 23.05.2003.

[BMBT] M. Body, M. Miquel, Y. Bédard, A. Tchounikine
”Handling Evolutions in Multidimensional Structures”
http://sirs.scg.ulaval.ca/yvanbedard/enseigne/SCG66124/316.pdf

[CSJ] C. S. Jensen
”Data Warehousing”, Forelesningsslides
http://fag.grm.hia.no/ikt2340/themes/dataWarehousing/notes/ikt2340.pdf

[DR-1] D. Røed
”En introduksjon til datavarehusarkitektur”
http://www.datavarehus.net/WP%20En%20introduksjon%20til%20datavarehus arkitektur.rtf
November 1999

[DR-2] D. Røed
”Estimering av datavarehus pilotprosjekter”
http://www.datavarehus.net/WP%20Estimering%20av%20et%20datavarehus% 20pilotprosjekt.pdf
Oktober 1999

[DR-3] D. Røed
”Hvordan oppnå suksess med datavarehus-løsninger?”
http://www.datavarehus.net/WP%20suksessfaktorer.rtf
Oktober 1999

[EKU] J. Eliasen, C. Kjær, H. Urban
”Data Warehouse Design for Spatio-Temporal Data”
Aalborg University, January 2002

[HTA] K. F. Hansen, T. H. Thomsen, R. Aaholm
”Strukturering af Informationer til Analyseformål”
http://it.civil.auc.dk/it/education/reports/km_dw_mii_2001/process _mii_2001.pdf

[JFPPST] C. S. Jensen, A. Friis-Christensen, T. B. Pedersen, D. Pfoser, S. Saltenis, N Tryfona
”Location-Based Services - A Database Perspective”
http://www.cs.auc.dk/~tbp/Teaching/DAT5E01/csjlbs.pdf

[JJ-1] J. Juhl
“Intelligent fartstilpassning – Adferdsendringer”
AAU, 2001

[JJ-2] J. Juhl
“Intelligent fartstilpassning – Mapmatching”
AAU, 2001

[MJ] M. Jürgens
”Index Structures for Data Warehouses”
Springer, July 2002

[MS] N. Matthew, R. Stones
”Beginning Database With MySQL”
Wrox Press, February 2002

[NR] N. Rasmussen
”Financial business intelligence”
Wiley, 2002

[PJ-1] T. B. Pedersen, C. S. Jensen
”Multidimensional Databases”

http://www.cs.auc.dk/~tbp/Teaching/DAT5E01/mddatabasesPJ.pdf

[PJ-2] T. B. Pedersen, C. S. Jensen
"Multidimensional Database Technology"
Desember 2001
http://fag.grm.hia.no/ikt2340/themes/dataWarehousing/notes/dw.pdf

[RK] R. Kimball
"The Data Warehouse Lifcycle Toolkit"
Wiley, 1998

[RMFBR] F. Ramsak, V. Markl, R. Fenk, R. Bayer, T. Ruf
"Interactive ROLAP on Large Datasets"
IDEAS 2001, Grenoble

[RU] D. Røed, T. Unholt
"Datavarehusmetodikk"
http://www.datavarehus.net/datavarehus.net%20datavarehusmetodikk.pdf
April 2001

[WHI] W. H. Inmon:
"Building The Data Warehouse"
Wiley, 1992

[WY] W. Yu
"Database Basic Concepts", Forelesningsslides
http://www.cs.uit.no/studier/kurs/d212/info/2003v/slides/DBconcepts.pdf

[ØRØD] Ø. B. Jakobsen, W. R. Lund, Ø. Thorsen, D. K. Walle
"Data warehousing for geografiske data"
HiA, Vår 2002

9.1 Nettsider

[I-NP-1] N. Pendse
"How not to buy an OLAP product"
http://www.olapreport.com/How_not_to_buy.htm

[I-NP-2] N. Pendse
"What is OLAP"
http://www.olapreport.com/fasmi.htm

[I-P] Personopplysningsloven
http://www.lovdata.no/all/tl-20000414-031-001.html
olapreport.com/fasmi.htm

10. Vedlegg

10.1 Jscript-kode

10.1.1 Konvertering av dato og klokkeslett fra tall og over på UTC-form

```
//Åpner en kobling til databasen gjennom ODBC
var db = WScript.CreateObject("ADODB.Connection");
db.Open("bigSQL");

//Her hentes alle rader i databasen for prosessering
var rs = db.Execute("SELECT * FROM Bil ORDER BY ID");

//For-løkken går igjennom allle radene, en for en
for(;!rs.EOF; rs.MoveNext()){
	var id = rs("ID").value;

//Fordi datoer skrevet som ett tall, må disse endres til en string med lik lengde
//Derfor må datoer som 61203 gjøres om til en string med 061203. Samme gjelder for
klokkeslett
	var temp = rs("Dato").value;
	if(temp<100000){temp1 = "0" + temp.toString();}
	else{var temp1 = temp.toString();}

	var temp2 = rs("Tid").value;
	if(temp2<100000){temp3 = "0" + temp2.toString();
	}else{var temp3 = temp2.toString();}

	//Ekstrahere Dato fra string
	var dag = temp1.substr(0,2);
	var maaned = temp1.substr(2,2);
	var aar = temp1.substr(4,2);

	//Ekstrahere Klokkeslett fra string
	var time = temp3.substr(0,2);
	var minutt = temp3.substr(2,2);
	var sekund = temp3.substr(4,2);

// Her konverteres dato og tid til UTC-form, og legges inn i det nye feltet i
databasen
	var temp4 = Date.UTC((20 + aar), maaned, dag, time, minutt, sekund);
	db.Execute("UPDATE Bil SET fUTC = "+temp4+" WHERE ID = "+id);
}
rs.Close();
db.Close();
```

10.1.2 Konvertering av database og over til datavarehus

```
//************** Hovedrutine start ********************************

// Kobler seg til MS SQL Server databasen kalt raadataSQL
var db = WScript.CreateObject("ADODB.Connection");
db.Open("bigSQL");

var PreviousSpeed = -1;
var PreviousSOL = 0;
var LastTime = 0;

// Henter er recordset av verdier tabellen tBil
var rs = db.Execute("SELECT * FROM tBil INNER JOIN tVeiTilGate ON tVeiTilGate.Vejkode
= tBil.fVeikode ORDER BY fUTC");

// Dette er en løkke som går igjennom alle rekker i tabellen tbil
for(;!rs.EOF; rs.MoveNext()) {

	//Sjekker om denne målingen og forrige kom rett etter hverandre med 1 sekund
mellomrom
	if (LastTime != (rs("fUTC").value-1000)){PreviousSpeed = -1;}
```

```
        //Sjekker om forrige måling er gyldig for å bruke, hvis ja, regnes ut
bremsefaktor
        if (PreviousSpeed != (-1)){
                var Accel = (rs("fHastighet").value)*(rs("fHastighet").value) -
(PreviousSpeed*PreviousSpeed);
        }else {Accel = 0;}
        var AccRatio = Math.floor(Accel/50);

        //Regner ut faktor for forhold mellom fart og fartsgrense
        if (rs("fFartsgrense").value == 0){SOL = 0;
        }else{var SOL = Math.floor((rs("fHastighet").value -
rs("fFartsgrense").value)*0.2);}

// Her kalles en funskjoner for hver dimensjon. Nødvendige verdier blir sendt med for
å sjekke etter
// om den allerede har vært lagt inn, eller om en ny rad må legges til i dimensjonen.
Funskjonen
// returnere id til korrekt rad i dimensjonen.
        var Dat = getDateID(rs("fUTC").value);
        var Tim = getTimeOfDay(rs("fUTC").value);
        var Veh = getVehicle(rs("fBilNR").value);
        var Dri = getDriver(rs("fDriver").value);
        var Loc = getLocation(rs("fVeikode").value, rs("fFartsgrense").value,
rs("Vejnavn").value);
        var Spe = getSpeed(rs("fHastighet").value);
        var Acc = getAcceleration(AccRatio);

// Setter inn en ny rad i fact-tabellen, med de korrekte referansene til dimensjonene
        db.Execute("INSERT INTO FactTable(ffkDate, ffkTime, ffkVehicle, ffDriver,
ffLocation, ffSpeed, ffAcceleration, fSpeedOnLimit, fAcceleration, fPreviousSpeed,
fPreviousSpeedOnLimit) VALUES "+

"("+Dat+","+Tim+","+Veh+","+Dri+","+Loc+","+Spe+","+Acc+","+SOL+","+AccRatio+","+Previ
ousSpeed+","+PreviousSOL+")");

// Lagrer nåværende verdier for sammenlikning med neste verdi
        PreviousSOL = SOL;
        PreviousSpeed = rs("fHastighet").value;
        LastTime = rs("fUTC").value
}
rs.Close();
db.Close();

//************** Hovedrutine slutt *********************************


//************** Her kommer funskjonene ****************************

// Dette er bare en funksjon for å kunne skrive til skjerm enkelt
function wr(s){WScript.StdOut.Write(s+"\n");}


//********* Dimensjonsfunksjoner ********************
// Disse sjekker om aktuell verdi finns i dimensjonen. Hvis den gjør
// det, returneres den id'en. Hvis ikke legges den nye verdien til i
// dimensjonen, og returnerer id'en til den

//Sjekker om dato eksisterer i dimensjonen dDate
function getDateID(UTCTime) {
        var d = new Date(UTCTime);
        var Dato = d.toDateString();
        var rs = db.Execute("SELECT fkID FROM dDate WHERE fDay = '" + Dato + "'");
        if (rs.EOF) {
                var Day = Math.floor(UTCTime/(60*60*24));
                var dayName = new
Array("Sunday","Monday","Tuesday","Wednesday","Thursday","Friday","Saturday");
                var monthName = new
Array("January","February","March","April","May","June","Juli","August","September","O
ctober","November","December");

                var d = new Date(UTCTime);

                var Dato = d.toDateString();
```

```
                var ChristmasStart = new Date(d.getYear(), 11, 23);
                var ChristmasEnd = new Date(d.getYear(), 0, 1);
                var Holiday = "Regular";
                if (d.getWeek>28 && d.getWeek<32){Holiday = "Summer";}
                if (d>=ChristmasStart && d<=ChristmasEnd){Holiday = "Jul";}

                var DayOfWeek = dayName[d.getDay()];

                var TypeOfDay = ((d.getDay()+1)%7 < 2)?"WE":"WD";

                var wd = new Date(UTCTime + ((6-d.getDay())*24*60*60*1000));
                var CalWeek = "WE "+wd.toDateString();

                var CalMonth = monthName[d.getMonth()];

                var CalQuarter = "Q" + (Math.floor(d.getMonth()/4)+1);

                var CalYear = d.getFullYear();

                //Sette de nye dataene inn i dimensjonen, og returnerer id til fact-
tabellen
                db.Execute("INSERT INTO
dDate(fDay,fDayOfWeek,fTypeOfDay,fHoliday,fCalendarWeek,fCalendarMonth,fCalendarQuarte
r,fCalendarYear) VALUES "+

"('"+Dato+"','"+DayOfWeek+"','"+TypeOfDay+"','"+Holiday+"','"+CalWeek+"','"+CalMonth+"
','"+CalQuarter+"','"+CalYear+"')");
                var rr = db.Execute("SELECT IDENT_CURRENT('dDate') AS fkID");
                ID = rr("fkID").value;
                rr.Close();
        }else{
                ID = rs("fkID").value;
        }
        rs.Close();
        return ID;
}

//Sjekker om tidspunkt eksisterer i dimensjonen dTimeOfDay
function getTimeOfDay(UTCTime) {
        var UTCDate = new Date(UTCTime);

        if(UTCDate.getHours() < 10){var hour = "0" + UTCDate.getHours();}
        else{var hour = UTCDate.getHours();}

        if(UTCDate.getMinutes() < 10){var minute = hour + ":0" +
UTCDate.getMinutes();}
        else{var minute = hour + ":" + UTCDate.getMinutes();}

        if(UTCDate.getSeconds() < 10){var second = minute + ":0" +
UTCDate.getSeconds();}
        else{var second = minute + ":" + UTCDate.getSeconds();}

        var rs = db.Execute("SELECT fkID FROM dTimeOfDay WHERE fSecond =
'"+second+"'");
        if (rs.EOF) {

                var AMPM = (hour<12)?"AM":"PM";
                var daypart = "";

                if (hour>=6 && hour<9) daypart = "Morning";
                if (hour>=9 && hour<12) daypart = "Day";
                if (hour>=12 && hour<18) daypart = "Afternoon";
                if (hour>=18 && hour<23) daypart = "Evening";
                if (hour>=23 || hour<6) daypart = "Night";

                //Sette de nye dataene inn i dimensjonen, og returnerer id til fact-
tabellen
                db.Execute("INSERT INTO
dTimeOfDay(fHour,fMinute,fSecond,fAMPM,fPartOfDay) VALUES
("+hour+",'"+minute+"','"+second+"','"+AMPM+"','"+daypart+"')");
                var rr = db.Execute("SELECT IDENT_CURRENT('dTimeOfDay') AS fkID");
                ID = rr("fkID").value;
                rr.Close();
        }else{
                ID = rs("fkID").value;
```

```
	}
	rs.Close();
	return ID;
}

// Sjekker om bilnummer eksisterer i dimensjonen dVehicle
function getVehicle(Vehicle){
	var rs = db.Execute("SELECT fkID FROM dVehicle WHERE fVehicle = "+Vehicle);

	if (rs.EOF){

		//Sette de nye dataene inn i dimensjonen, og returnerer id til fact-
tabellen
		db.Execute("INSERT INTO dVehicle(fVehicle) VALUES ('"+Vehicle+"')");
		var rr = db.Execute("SELECT IDENT_CURRENT('dVehicle') AS fkID");
		ID = rr("fkID").value;
		rr.Close();
	}else{
		ID = rs("fkID").value;
	}
	rs.Close();
	return ID;
}

// Sjekker om sjåfør eksisterer i dimensjonen dDriver
function getDriver(Driver){
	var rs = db.Execute("SELECT fkID FROM dDriver WHERE fDriver = "+Driver);

	if (rs.EOF){

		//Sette de nye dataene inn i dimensjonen, og returnerer id til fact-
tabellen
		db.Execute("INSERT INTO dDriver(fDriver) VALUES ('"+Driver+"')");
		var rr = db.Execute("SELECT IDENT_CURRENT('dDriver') AS fkID");
		ID = rr("fkID").value;
		rr.Close();
	}else{
		ID = rs("fkID").value;
	}
	rs.Close();
	return ID;
}

// Sjekker om nedbremsingfaktor er registrert i dimensjonen dDeceleration
function getAcceleration(AccRatio){
	var AccRatio2 = Math.floor(AccRatio/10);
	var temp = AccRatio2*10 +" to "+ (AccRatio2+1)*10;
	var rs = db.Execute("SELECT fkID FROM dAcceleration WHERE fAcceleration =
'"+temp+"'");
	if (rs.EOF){

		if (AccRatio2 > 0){
			Direction = "Accelerating";
		}else if (AccRatio2 < 0){
			Direction = "Decelerating";
		}else{Direction = "Unknown";}

		var AccRating = new Array("Minor","Medium","Major");
		var AccRatio3 = Math.abs(Math.floor(AccRatio/30));
		if (AccRatio3 >= 3){AccRatio3 = 2;}
		var Overall = AccRating[AccRatio3];

		//Sette de nye dataene inn i dimensjonen, og returnerer id til fact-
tabellen
		db.Execute("INSERT INTO
dAcceleration(fDirection,fOverall,fAcceleration) VALUES
('"+Direction+"','"+Overall+"','"+temp+"')");
		var rr = db.Execute("SELECT IDENT_CURRENT('dAcceleration') AS fkID");
		ID = rr("fkID").value;
		rr.Close();
	}else{
		ID = rs("fkID").value;
	}
	rs.Close();
	return ID;
```

```
}

// Sjekker om hastighet er registrert i dimensjonen dSpeed
function getSpeed(Speed){
	var SpeedInterval = Math.floor(Speed / 10);

	var SI = (SpeedInterval*10) + " to " + (SpeedInterval*10 + 10) + " Kmh";

	if (SpeedInterval < 1){SpeedClass = "Below 10";
	}else if (SpeedInterval >= 1 && SpeedInterval < 5){SpeedClass = "10 to 40";
	}else if (SpeedInterval >= 5 && SpeedInterval < 8){SpeedClass = "50 to 80";
	}else{SpeedClass = "Above 80";}

	var rs = db.Execute("SELECT fkID FROM dSpeed WHERE fSpeed = '"+SI+"'");
	if (rs.EOF){

		//Sette de nye dataene inn i dimensjonen, og returnerer id til fact-
tabellen
		db.Execute("INSERT INTO dSpeed(fSpeedClasses,fSpeed) VALUES
('"+SpeedClass+"','"+SI+"')");
		var rr = db.Execute("SELECT IDENT_CURRENT('dSpeed') AS fkID");
		ID = rr("fkID").value;
		rr.Close();
	}else{
		ID = rs("fkID").value;
	}
	rs.Close();
	return ID;
}

// Sjekker om lokasjon er registrert i dimsensjonen dLocation
function getLocation(Road, SpeedLimit, Veinavn){
	var rs = db.Execute("SELECT fkID FROM dLocation WHERE fRoad = "+Road+" AND
fSpeedLimit = "+SpeedLimit);

	if (rs.EOF){

		//Sette de nye dataene inn i dimensjonen, og returnerer id til fact-
tabellen
		db.Execute("INSERT INTO
dLocation(fCountry,fRegion,fCity,fRoad,fRoadName,fSpeedLimit) VALUES
('Denmark','Aalborg Amt','Aalborg','"+Road+"','"+Veinavn+"','"+SpeedLimit+"')");
		var rr = db.Execute("SELECT IDENT_CURRENT('dLocation') AS fkID");
		ID = rr("fkID").value;
		rr.Close();
	}else{
		ID = rs("fkID").value;
	}

	rs.Close();
	return ID;
}
```