Datavetenskap

Eva Pettersson

Johan Westerdahl

Insamling, lagring och bearbetning av data för beslutsstöd i stora organisationer

- En övergripande studie av datalager

Examensarbete, C-nivå

2005:06

Insamling, lagring och bearbetning av data för beslutsstöd i stora organisationer

- En övergripande studie av datalager

Eva Pettersson

Johan Westerdahl

Denna rapport är skriven som en del av det arbete som krävs för att erhålla en kandidatexamen i datavetenskap. Allt material i denna rapport, vilket inte är mitt eget, har blivit tydligt identifierat och inget material är inkluderat som tidigare använts för erhållande av annan examen.

Eva Pettersson

Johan Westerdahl

Godkänd, 1 juni 2005

Handledare: Thijs Holleboom

Examinator: Donald F. Ross

Sammanfattning

Redan på 1960-talet kom tankarna på att samla in data från transaktionssystem för att användas i beslutsstödjande syfte. För att möjliggöra beslutsstöd i en organisation måste ett datalager byggas och hänsyn tas till många olika tekniker för bland annat datainsamling, datalagring och rapporteringsmöjligheter. Syftet med uppsatsen är att beskriva vad ett datalager är ur ett tekniskt perspektiv, beskriva hur data lagras samt hur ett datalager kan effektiviseras för snabb dataåtkomst och möjliggöra skalbar lagring av data. Ett datalager är en ämnesindelad databas med historisk, detaljerad och integrerad data, som inte ändras efter inladdning, med syftet att utgöra ett stöd för ledningsbeslut. Vi beskriver en arkitektur för datalager där den detaljerade datan är normaliserad enligt relationsmodellen. Utifrån den datan lagras summerad eller på annat sätt aggregerad data för avdelningsspecifika behov i vad som kallas data marts. Data i en data mart är inte normaliserad utan lagras enligt dimensionsmodellen som frångår normaliseringsprinciperna för att optimera prestanda. Företag har ofta behov av att göra många olika jämförelser av data från ett relativt litet antal tabeller vilket dimensionsmodellen är anpassad för. Kombinationen av historisk och detaljerad data leder till enormt stora datavolymer vars lagringsbehov i praktiken ofta ökar exponentiellt. För att minska databasens svarstider krävs utökad indexering och partitionering men även aggregerad redundant data används i stor utsträckning. En av våra slutsatser är att det är viktigt att ha en pragmatisk inställning till designfrågorna, att i första hand lösa organisationernas problem och i andra hand följa teoretiska designprinciper för att möjliggöra flexibel användning i framtiden.

Retrieval, Storage and Processing of Data for Decision Support in Large Organisations

- A Comprehensive Study of Data Warehousing

Abstract

In the 1960's the thought of collecting transactional data for decision support arose. To enable decision support in an organisation a data warehouse must be built taking into consideration a variety of techniques for data retrieval, storage and reporting. Our purpose with this dissertation is to describe a data warehouse from a technical perspective, how data is stored, how a data warehouse can be streamlined for fast data retrieval and how to enable scalable data storage. A data warehouse is a subject-oriented, integrated, non volatile, time-variant data store supportive of managerial decisions. We describe an architecture for a data warehouse where the detailed data is normalised according to the relational model. The warehouse data is summarized or aggregated in a fashion designed to accommodate departmental needs and stored in what is called a data mart. Data in a data mart is not normalised but is stored according to the dimensional model which abandons the principles of normalisation in favour of optimized performance. Organisations often need to compare data from multiple tables for which the dimensional model is well suited. The combination of historical and detailed data leads to enormous volume requirements which, in practice, often increases the need for storage exponentially. This demands heavy use of indexing and partitioning of the database. To decrease response time aggregated redundant data is used to a great extent. One of our conclusions is the importance of a pragmatic approach to data warehouse design. The main priority must be to solve the organisation's problems. Following theoretical design principles is of secondary concern.

Tack!

Vi vill rikta ett stort tack till vår handledare, Thijs Holleboom, som har hjälpt oss att fokusera på vår uppgift.

Vi vill också tacka Christer Ingemarsson på SAP, Daniel Ringquist på SAS Institute, Björn Thodenius på Handelshögskolan i Stockholm samt respondenten som har valt att vara anonym. Ni har gett ett stort bidrag till vår uppsats och varit generösa med er tid. Vi är mycket tacksamma!

Innehållsförteckning

Figurförteckning

1 Inledning

Från det att datorn började användas för att effektivisera och rationalisera företagens rutiner och processer kom under 1960-talet tankarna på att utnyttja den data som genererades i systemen för att skapa rapporter och andra sammanställningar som kunde användas i beslutsstödjande syfte (Thodenius, 2005). Historiskt sett har inte systemen levt upp till de förväntningar som ställts på dem. Beslut som högre chefer fattar är inte repetitiva, alltid föränderliga och för stunden.

De applikationer som används för analyser av företagsinformation kallas beslutstödssystem och målet med dem är att hjälpa ledningen upptäcka trender, lokalisera problem och fatta intelligenta beslut (Date, 2004). Den grundläggande idén med de här systemen är att samla operativ data och reducera den till en form så att den kan användas till att analysera företagets beteende och modifiera det beteendet på ett intelligent sätt. Den största gruppen användare av beslutsstödssystem återfinns inte bland företagsledningen utan hos ekonomiavdelningar och andra enheter som mer arbetar med analys och ekonomisk rapportering (Thodenius, 2005). Ett datalager är en typ av databas som används av beslutsstödssystem och de har vuxit fram av två skäl (Date, 2004):

- behovet av en enhetlig datakälla med integrerad och tvättad data för beslutsstödsapplikationer
- behovet av att uppfylla ovanstående behov utan att negativt påverka de operativa systemen

Ett datalager kan stödja verksamhetsstyrningen genom att (Söderström, 1997):

- data kan väljas och analyseras i ett brett verksamhetsperspektiv och ett långt tidsperspektiv
- beslutsfattaren kan själv vid varje tillfälle välja data fritt och få den presenterad på det sätt som önskas
- data från olika verksamhetsgrenar och extern data kan kombineras
- datakvaliteten är känd
- inom hela organisationen kan aktuell data anpassad till varje verksamhet vara lokalt tillgänglig

Datalager byggdes ursprungligen av flera skäl (Inmon et al., 1999). Ett av de ursprungliga målen var att få en konsoliderad bild av data i annars osammanhängande och totalt oberoende operativa system. Datalager möjliggjorde att integrerad data blev tillgänglig för organisationen för att fatta bättre strategiska beslut. Ett annat mål var att avlasta analysfunktionen av data från de prestandakänsliga och svarstidskänsliga operativa systemen. Istället för att försöka hantera ett enda system för de olika behov som operativa användare och analytiker har, är det betydligt effektivare att dela upp systemen i två kategorier, operativa system och system för analyser, som var för sig är optimerade för de olika användarnas speciella behov (Inmon et al., 1999).

Idag har företagen kommit längre än de ursprungliga målen och nu vill de samla mer data och analysera den mer utförligt för att öka sin totala effektivitet (Inmon et al., 1999). I detaljhandeln vill företag spåra och analysera varje interaktion med en kund, hålla reda på alla produkter och vara säker på att rätt produkter finns i rätt affär och marknadsförs till rätt personer vid rätt tidpunkt. I finansvärlden vill företag bygga upp en enhetlig bild av sina kunder, förstå vilka banktjänster de använder och vilka de kan vara intresserade av. Det här är bara två exempel på användningsområden som visar att företagen efterfrågar mer från de moderna datalagren än de har gjort tidigare.

För att uppfylla de krav som företagen ställer på datatillgång måste datalagret innehålla både detaljerad och historisk data vilket har gett en explosionsartad ökning i datavolym (Inmon, 2002). Dessutom upptäcker företagen ofta att de vill samla data inom fler områden vilket bidrar till att kraftigt öka volymen. Frågan om hur datavolymerna skall hanteras är en av de allra viktigaste för ett datalager, databaserna kan i praktiken växa så mycket som 50 % per år vilket ställer stora krav på skalbarhet på både hårdvara och databashanterare.

1.1 Problemformulering

För att möjliggöra beslutsstöd i en organisation måste datalagret byggas med hänsyn till många olika tekniker för bland annat datainsamling, datalagring och rapporteringsmöjligheter. Det är stora datamängder som skall hanteras, konsolideras och optimeras för snabb åtkomst. Dessutom behöver datalagret vara anpassat för fler och förändrade behov och byggas på en skalbar hårdvaruplattform och databashanterare för att överleva en organisations ständiga förändringar och växande krav.

Det är ett väldigt komplext och omfattande arbete att få alla delar att fungera tillsammans och att bygga ett datalager kan ta lång tid eftersom många avvägningar behöver göras och

många designbeslut fattas under vägen. Något som ytterligare komplicerar bygget är att kraven och behoven ofta inte är kända från början utan upptäcks allteftersom datalagret skapas.

Målet med uppsatsen är att undersöka minst en befintlig applikation med avseende på hur affärskritisk data kan samlas in och analyseras i stora organisationer, vilka tekniker som används samt att relatera tekniken till befintlig teori.

1.2 Syfte

Syftet med uppsatsen är att beskriva vad ett datalager är ur ett tekniskt perspektiv, hur data lagras samt hur ett datalager kan effektiviseras för snabb dataåtkomst och möjliggöra skalbar lagring av data. Det ger oss följande frågor att besvara:

- Vad är ett datalager?
- Vilken data skall sparas i ett datalager?
- Hur kan data lagras för att möjliggöra beslutsstöd med rimliga svarstider?
- Vilka prestandaaspekter behöver beaktas?

1.3 Avgränsning

Vi har valt att undersöka datalager utifrån ett brett perspektiv. För att sätta datalager i ett sammanhang beskriver vi en övergripande arkitektur anpassad för flexibilitet men vi undersöker inte i detalj hur applikationer för beslutsstöd fungerar. Vi har fokuserat på tekniker för datalagring och prestandaoptimeringar. Vi har valt att undersöka en befintlig applikation för att se hur väl teorierna stämmer med en produkt på marknaden.

1.4 Metod

I det här avsnittet redogörs för hur vi har gått tillväga i vår undersökning med litteraturstudier och intervjuer.

1.4.1 Kvalitativ och explorativ undersökning

Vi har valt att göra en kvalitativ och explorativ undersökning eftersom vi översiktligt vill beskriva vad datalager är, vilka syften de har, hur de används och vilka problem eller utmaningar de medför. Kvalitativt inriktad forskning innebär att datainsamlingen fokuserar på mjuka data, t.ex. kvalitativa intervjuer och tolkande analyser (Patel & Davidson, 2003). Det

främsta syftet med en explorativ undersökning är att inhämta så mycket kunskap som möjligt om ett bestämt problemområde, att försöka belysa ett problemområde allsidigt. Undersökningen syftar ofta till att nå kunskap som kan ligga till grund för vidare studier, därför är idérikedom och kreativitet viktiga inslag. Oftast används flera olika tekniker för att samla information (Patel & Davidson, 2003). Kvalitativ egenskap innebär att egenskapen kan identifieras, det går att avgöra om den finns eller ej men inte gradera den. I kvalitativ forskningsmetodik brukar beskrivningar av egenskaper, samband m.m. ske enbart i ord. Datainsamlingsmetoderna är oftast intervjuer och observationer på plats (Wallén, 1996). En undersökning av kvalitativ karaktär fokuserar på tingens underliggande mening och innebörd. Helhetsförståelsen och sammanhanget är viktigare än delarna. En kvalitativ undersökning genererar konceptuella beskrivningar av verkligheten i form av text och modeller vilka syftar till att upptäcka, lyfta upp och belysa de samband som vuxit fram. Den kvalitativa analysen är i sitt förhållningssätt begränsad till undersökarens förmåga att förstå och tolka det kvalitativa datamaterialet (Christensen et al. 2001).

1.4.2 Genomförande

Vi började arbetet med en inledande litteraturstudie för att orientera oss i ämnet. Då litteraturstudien inte gav oss mycket stöd för hur vi skulle gå vidare kontaktade vi sex leverantörer av datalagerprodukter för att få en inblick i hur befintliga system på marknaden fungerar. Vi fick svar från SAP och SAS Institute och vi har besökt dem för att genomföra intervjuer.

Inför besöken tog vi fram en lista med intervjufrågor baserat på det vi hittat i litteraturen, men vi ställde även andra frågor som vi trodde kunde vara relevanta. Vi ställde inte frågorna i någon speciell ordning utan ville i så stor utsträckning som möjligt låta de vi intervjuade själva berätta om sin syn på datalager och dess användning utan att bli påverkade av oss. Däremot såg vi till att få svar på det vi undrade. Vid besöken ställde vi frågor om datalager i allmänhet och hur de används men även specifika frågor om företagens produkter.

Vi har dessutom närvarat vid två användarseminarier hos SAP för att få en bättre inblick i hur deras produkt fungerar och används av deras kunder. Vid ett användarseminarium fick vi kontakt med Företag A som relativt nyligen har börjat använda SAP:s produkt.

Efter besöken hade vi betydligt mer att utgå ifrån och vår förståelse för vad ett datalager är och vad de används till hade ökat avsevärt. Vi genomförde då en mycket större litteraturstudie där vi samlade in all teori vi kunde hitta och få tag på som verkade relevant för vår

undersökning. Därefter utformade vi en intervjuguide med detaljerade frågor om de specifika områden som vi kommit att förstå var viktiga. Vi tog fram en uppsättning frågor för leverantörer och en för användare.

Eftersom Företag A använder SAP:s produkt idag besökte vi dem för att få ett användarperspektiv på ämnet och se hur deras datalager används i verksamheten. Först genomförde vi en intervju på samma sätt som med SAP och SAS Institute där respondenten fritt berättade om och demonstrerade användningen av datalagret. Därefter övergick vi till att använda intervjuguiden för att se till att få svar på alla frågor. För att få ytterligare ett användarperspektiv genomförde vi en intervju med Björn Thodenius som är forskare vid Handelshögskolan i Stockholm och har skrivit en avhandling om användning av ledningsinformationssystem (beslutstödssystem). Utifrån avhandlingen sammanställde vi frågor om hur verksamhetsledare använder sig av datalager för beslutsstöd.

Slutligen genomförde vi telefonintervjuer med SAP och SAS Institute där vi ställde våra frågor utifrån intervjuguiden. Efter intervjuerna jämfördes den insamlade teorin och empirin i en analys för att söka svar på våra undersökningsfrågor.

1.5 Begreppsdefinitioner

Datalager (Data Warehouse): Ett datalager är en ämnesindelad databas med historisk, detaljerad och integrerad data, som inte ändras efter inladdning, med syftet att utgöra ett stöd för ledningsbeslut (Inmon, 2002). Ett datalager innehåller i huvudsak normaliserad data och är anpassad för flexibilitet och förändrade krav.

Data mart: En data mart innehåller denormaliserad och summerad eller på annat sätt aggregerad data för avdelningsspecifika behov och är prestandaoptimerad för en viss typ av frågor och rapporter (Inmon, 2002). Data marts hämtar oftast sin data från ett datalager och de baseras på dimensionsdatabaser där dimensionerna kan modelleras som sidorna på en multidimensionell kub. På grund av datastrukturen kallas data marts ibland för kuber eller hyperkuber. Vi har inte hittat någon bra svensk benämning för en data mart och använder därför det engelska begreppet i uppsatsen.

Operativa system: Företag har ofta en mängd olika system för att stödja den operativa verksamheten. Det är ofta applikationer med databaser som hanterar transaktioner och kan till exempel vara system för ekonomisk administration, lagerhantering, produktionsstyrning och orderhantering. För till exempel en bank kan det vara system för att hantera finansiella transaktioner mellan konton. I uppsatsen kallar vi traditionella transaktionssystem med

transaktionsdatabaser för operativa system eftersom det är så de benämns i de här sammanhangen.

2 Teori

Det finns två principer för hur datalager skall byggas. Vi har valt att beskriva datalager utifrån W. H. Inmons arkitektur (2002) som skiljer sig från Ralph Kimballs (Kimball et al., 1998) synsätt. Vi har utgått från Inmons definition av datalager i avsnitt 1.5. Enligt Kimball är en data mart en logisk delmängd av ett datalager som är den datakälla som frågor kan ställas mot i en organisation. Ett datalager är en union av alla integrerade data marts, och ingenting annat. Kimball anser att ett datalager inte kan organiseras runt en entitet-relationsmodell eftersom både prestanda och förståelse då går förlorat (Kimball et al., 1998).

2.1 Vad är ett datalager?

I det här delkapitlet beskrivs Inmons arkitektur för ett datalager, vilken resten av uppsatsen baseras på. Arkitekturen är tydligt beskriven i hans böcker och ger en bra utgångspunkt för vidare resonemang.

2.1.1 Bakgrund

Ett datalager kräver en arkitektur och en metod som utgår från en helhet för att sedan arbeta ner mot detaljerna (Inmon, 2002). Detaljerna är viktiga, men de är bara viktiga när de ses i ett större sammanhang. Användaren av ett datalager är först och främst verksamhetsanalytiker eller beslutsfattare som i första hand är fokuserad på hur verksamhetens affärer går. Analytikerns uppgift är att definiera och ta fram information som skall användas som beslutsunderlag för ledningen. Det är oftast inte ett repetitivt arbete utan mer utforskande och heuristiskt där analytikern prövar sig fram och försöker hitta intressanta samband. Inmon säger att analytikerns inställning till design av ett datalager kan beskrivas med ”ge mig det jag säger att jag vill ha, så skall jag tala om vad jag verkligen vill ha”. Analytikern behöver utforska möjligheterna med systemen för att bättre kunna bilda sig en uppfattning. Eftersom analytikern arbetar med att utforska är det omöjligt att på förhand definiera hur systemet skall fungera, dessutom kan det vara svårt att standardisera systemen då olika företag har olika förutsättningar och behov. För att bygga ett datalager bör man använda en iterativ utvecklingsmetod som tar hänsyn till analytikerns behov av att utforska.

Tillsammans med de första diskbaserade lagringsmedierna för direktåtkomst av filer kom även de första databashanterarna (Inmon, 2002). Syftet med en databashanterare är att göra det enkelt för programmeraren att lagra och komma åt data på diskarna. I mitten på 70-talet möjliggjorde online transaction processing (OLTP) ännu snabbare tillgång till data och skapade helt nya möjligheter för företag att behandla data. Nu kunde datorerna utnyttjas för uppgifter som tidigare inte var möjliga, bland annat bankomatsystem och produktionsövervakningssystem.

När PC:n och 4GL-teknologin kom skapades nya möjligheter att bearbeta data och det gick att göra mer än att bara behandla transaktioner (Inmon, 2002). Management information systems (MIS), som de kallades förr, började implementeras för att underlätta ledningens beslutsfattande. Idag kallas de applikationerna för decision support system (DSS). Tidigare användes de till att fatta detaljerade operativa beslut baserat på transaktioner. I den här uppsatsen används det svenska begreppet beslutstödssystem för den här typen av applikationer.

En enda databas kunde inte möjliggöra både OLTP och analytisk databehandling samtidigt (Inmon, 2002). För att möjliggöra analyser användes små program för att extrahera data ur OLTP-systemen. De här programmen blev populära av i huvudsak två anledningar, prestanda och kontroll. Prestandan förbättrades eftersom analyserna inte behövde stjäla resurser från de prestandakänsliga transaktionssystemen när databehandlingarna behövde göras på stora mängder data samtidigt. När datan hade kopierats kunde slutanvändaren själv bestämma vad som skulle göras med den. Det här ledde till att data började extraheras i mycket stor skala ute på företagen. Ett ”spindelnät” av dataextraheringar och analyser började ta form. Först extraherades data, och sen extraherades data ur extraheringarna och så vidare. Enligt Inmon var det inte ovanligt att ett stort företag gjorde upp till 45 000 extraheringar per dag. Det här mönstret av okontrollerade extraheringar blev så vanligt i organisationerna att det fick ett eget namn ”the naturally evolving architecture”, ju större organisationen var desto svårare var det att hantera extraheringarna. Den ökade kontrollen var främst möjligheten att förändra och uppdatera den kopierade datamängden efter behov utan att behöva ta hänsyn till någon annan.

Enligt Inmon får ”the naturally evolving architecture” tre betydande konsekvenser: problem med datans trovärdighet (se avsnitt 2.1.1.1), produktivitetsproblem (se avsnitt 2.1.1.2) samt en oförmåga att skapa information ur data (se avsnitt 2.1.1.3). Eftersom en organisation ofta har flera olika transaktionssystem är det inte säkert att de olika transaktionerna har en gemensam ursprunglig källa. Om det inte finns en tydlig struktur och kunskap om hur dataextraheringarna påverkar varandra går det inte att vara säker på att

likvärdiga siffror jämförs. De här problemen visar sig till exempel när två avdelningschefer rapporterar till en gemensam chef, en av dem påstår att produktionen har ökar med 10 procent medan den andra säger att produktionen minskat med 15 procent. Spårbarheten för den data som de två rapporterna bygger på är mycket dålig och det kan i praktiken vara omöjligt att veta vilken som är mest rätt. Ledningen får då fatta beslut som inte enbart är baserade på fakta. Enligt Inmon är de här problemen både vanliga och förutsägbara.

2.1.1.1 Problem med trovärdighet

Inmon (2002) ger ett antal exempel på problem som kan uppstå när extraheringarna sker okontrollerat. För det första finns det ingen tidstämpel på datan så att det går att veta att det som jämförs är från samma tidpunkt, självklart skiljer sig data som extraherats på en söndag förmiddag ena veckan från data som extraherats nästföljande onsdag eftermiddag. För det andra så kan de olika analyserna vara baserade på olika algoritmer som extraherar data baserat på olika grunder. Till exempel kan en avdelning välja att analysera gamla kunder medan en annan avdelning analyserar alla stora kunder, att det ger olika resultat är knappast förvånande. För det tredje så görs extraheringarna i flera nivåer vilket förvärrar felen av de första två anledningarna vid varje tillfälle. Enligt Inmon är det inte ovanligt att företag genomför extraheringar i åtta eller nio nivåer innan en rapport presenteras som beslutsunderlag för ledningen. För det fjärde så är det lätt att ta in extern data vid en analys men oftast sparas inte information om varifrån den kommer, vilket leder till att ingen skillnad görs på intern och extern data. Extern data blir generell data som skulle kunna ha kommit från vilken källa som helst. För det femte så baseras inte nödvändigtvis analyserna på data från en gemensam källa. Två analyser kan vara baserade på två helt olika källor som inte är synkroniserade. I slutändan leder ”the naturally evolving architecture” till stora problem med trovärdighet som måste hanteras på något sätt.

2.1.1.2 Problem med produktivitet

En annan typ av problem beror på att i ”the naturally evolving architecture” så finns det många olika typer av system som är skapade för olika syften för olika delar av verksamheten och dessutom bygger många av dem på tekniker och arkitekturer som det kanske är svårt att hitta kompetens för idag (Inmon, 2002). De gamla systemen har samlat på sig data genom åren och när en viss rapport skall tas fram är den kanske beroende av att relevant data är åtkomlig i de gamla systemen. En annan faktor som komplicerar insamlandet av data är att de olika systemen använder olika semantik, dataformat och datastrukturer. Data som i två olika

system kallas för samma sak, t. ex. saldo, behöver inte betyda samma sak utan kan ha helt olika betydelse. Samtidigt kan data som betyder samma sak i olika system ha helt olika beteckningar.

För att kunna ta fram en rapport på koncernnivå är det nödvändigt att lokalisera och analysera datan för att säkerställa att ”äpplen” inte jämförs med ”apelsiner” innan data kan sammanställas ur de olika systemen. Det är ett arbete som kräver mycket tid och programmerarresurser, speciellt om dokumentationen är undermålig och det inte finns någon gemensam datamodell.

Inmon (2002) ger exempel där det i en koncern kan ta upp till ett år enbart för att lokalisera all data, sedan kan det ta upp till två år för att samla in och sammanställa den. Det stora problemet i det här sammanhanget är att om organisationen inte på förhand känner till och kan ta hänsyn till framtida krav på rapporter krävs det en nästan lika stor arbetsinsats nästa gång en rapport skall ta fram.

Inmon drar slutsatsen att möjligheten att ta fram nya rapporter är en mycket viktig fråga för organisationer som låtit floran av gamla och nya system växa enligt ”the naturally evolving architecture”. Informationen blir dyr att ta fram och det tar lång tid.

2.1.1.3 Från data till information

I ”the naturally evolving architecture” finns ett annat problem som Inmon (2002) exemplifierar med en bank som vill ta fram en rapport som beskriver hur kontoaktiviteten i år skiljer sig från aktiviteten de senaste fem åren. Det är mycket svårt att extrahera data ur de befintliga systemen eftersom de ofta varken är gjorda för att integrera med varandra eller för att exportera data. Det är i praktiken nästan omöjligt att regelbundet extrahera data ur dem. En bank kan t. ex. ha olika applikationer för att hantera lån, sparkonton och fonder. Det kan i praktiken vara en mycket komplicerad uppgift att lösa. De operativa systemen har sällan tillräckligt med historisk data och har dessutom olika mycket historisk data beroende på hur långt tillbaka i tiden användarna normalt behöver gå för att sköta den dagliga driften. En applikation för att hantera lönekonton kanske har data som stäcker sig ett år tillbaka i tiden och var aldrig anpassat för att göra analyser flera år tillbaka i tiden.

2.1.2 Arkitektur

För att lösa problemen med ”the naturally evolving architecture” och skapa mer robusta och framtidssäkra system som underlättar analyser krävs en arkitekturförändring. Inmon (2002) föreslår en arkitektur där det i huvudsak finns två typer av data, primitiv och

aggregerad. Primitiv data är operativ data medan aggregerad data används för analyser. En naturlig förlängning blir enligt Inmon en strukturerad miljö där data separeras i olika nivåer av aggregationer och beräknad data.

2.1.2.1 Olika typer av data

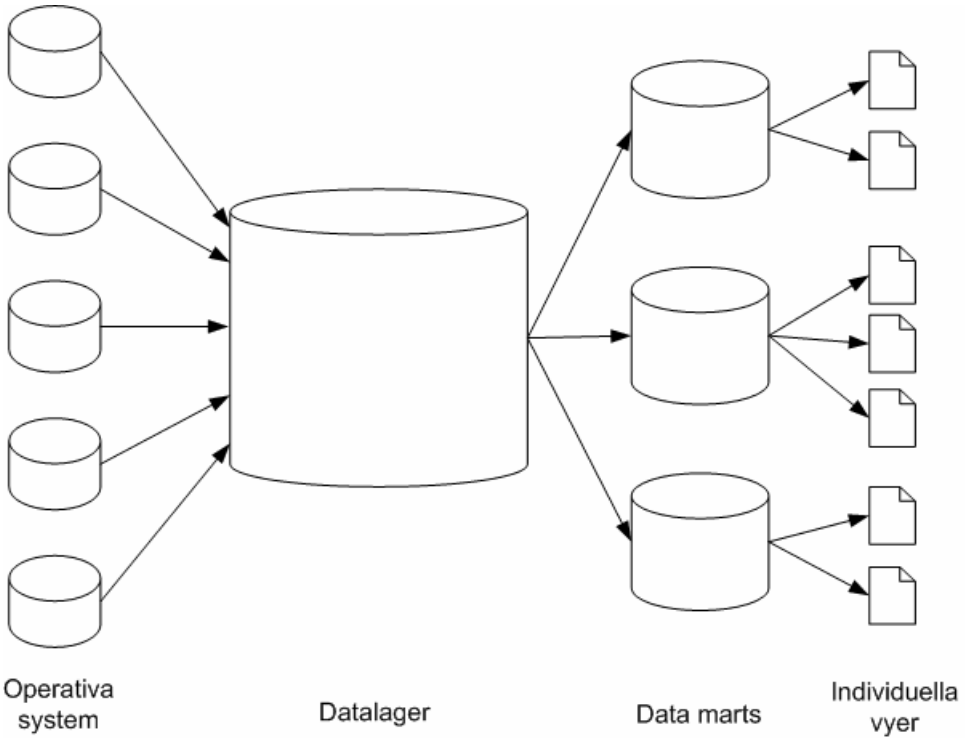
Primitiv data är detaljerad data för den operativa verksamheten. Aggregerad data är summeringar av, eller är på något sätt beräknad eller härledd utifrån, primitiv data som skall tillfredställa informationsbehoven som verksamhetens ledning har (Inmon, 2002). Istället för att sammanställa data ur databasen varje gång en viss fråga ställs kan värdet på vanligt efterfrågad data beräknas i förväg, t.ex. summan av månadens försäljning.

- Primitiv data kan uppdateras. Aggregerad data kan beräknas igen men inte uppdateras direkt.
- Primitiv data visar ett nuvarande tillstånd, t.ex. hur mycket pengar som finns i konto X just nu. Aggregerad data är ofta historisk data.
- Primitiv data behandlas med repetitiva procedurer eller funktioner medan aggregerad data ofta behandlas heuristiskt.
- Primitiv data används ofta av tjänstemän för att övervaka verksamheten medan aggregerad data används som beslutsstöd för ledningen.
- Primitiv data behandlas en post eller i små enheter åt gången medan aggregerad data behandlas i stora mängder.
- Åtkomsten av primitiv data i transaktionssystem är tidskritisk medan tillgängligheten är mindre betydande för aggregerad data.

Historiskt sett har det gjorts försök att ha en enda databas som skall uppfylla alla krav och tillfredställa alla behov. Inmon (2002) anser dock att primitiv data och aggregerad data är så olika, då de används för helt olika ändamål, att de definitivt skall skiljas åt.

2.1.2.2 Strukturerade nivåer

I Inmons strukturerade miljö finns det fyra nivåer av data: operativa nivån, datalagernivån, data mart-nivån och den individuella nivån, se Figur 1. Den här uppdelningen i nivåer av data är grunden för en mycket större arkitektur som Inmon kallar ”the corporate information factory”.



Figur 1: Strukturerade datanivåer (fritt efter Inmon, 2002)

Den operativa nivån innehåller applikationsspecifik primitiv data och är främst till för tidskritiska transaktionssystem (Inmon, 2002). Det finns många fördelar när stora volymer data flyttas över från de operativa systemen till ett datalager. Ofta finns det en stor andel historisk data i de operativa systemen som inte behövs för tranaktionshanteringen och när datan flyttas blir de operativa systemen lättare att underhålla, omstrukturera, övervaka och indexera. Inmon säger att en stor del av det underhåll som görs på databaserna i de operativa systemen beror på förändringar som behöver göras för att stödja analytisk databehandling. När historisk data flyttas över i ett datalager så blir inte bara underhållet lättare, de operativa systemen belastas inte heller med analytiska beräkningar på den typen av data utan blir mer fokuserade på transaktionshantering.

Datalagernivån integrerar data som hör ihop och kompletterar varandra på ett strukturerat enhetligt sätt (Inmon, 2002). På den här nivån finns primitiv atomär data samt data som är minimalt härledd, den är så detaljerad den behöver vara vägt mot lagrings- och prestandabegränsningar. En mycket viktig uppgift för den här nivån är att se till att det finns en tidstämpel på all data samt att spara de historiska värdena med sofistikerade ”snapshots” (se avsnitt 2.1.4.1) så det går att se hur uppgifter har ändrats och uppdaterats med tiden. När

en post ändras skapas egentligen en ny post medan den gamla finns kvar med information om när i tiden den var giltig. På så vis finns alla historiskt giltiga värden för en post kvar, jämfört med de operativa systemen där det bara finns ett värde för varje post. På den operativa nivån har data, beroende på tillämpning, en tidshorisont på en till några månader medan datalagernivån innehåller betydligt mer historisk data och har en tidshorisont på ca fem till tio år. På den här nivån går det inte uppdatera eller ändra data direkt men det är möjligt att göra om de beräkningar som ligger till grund för aggregerad data. En uppdatering av ett datalager innebär att nya aktuella poster läggs till, inte att inaktuell data i en post byts ut mot aktuell data. Det gör att databasen kan optimeras för läsning då den t.ex. inte behöver hantera transaktioner.

Till skillnad från applikationsspecifik data är data i ett datalager insamlad för ett övergripande syfte och delas in i ämnesområden baserat på en gemensam datamodell för hela företaget (Inmon, 2002). Ett ämnesområde representerar något som är viktigt för företaget, t.ex. ordrar, kunder eller inköp. Varje ämnesområde kan bestå av tio, hundra eller ännu fler fysiska tabeller som alla är relaterade till varandra med en gemensam nyckel. Inmon betonar att det är viktigt att datamodellen gäller övergripande för både operativa system och datalagret och att den bara representerar primär data. Den gemensamma modellen används som utgångspunkt när datamodeller sedan designas som gäller specifikt för operativa system respektive datalager och är anpassade för de väldigt olika användningsområdena. I datamodellen för datalagret har primär data som är för detaljerad och enbart används i de operativa systemen tagits bort. Sedan utökas nyckelstrukturen med ett attribut som representerar tid och därefter läggs de attribut och tabeller till som innehåller härledd data. Grundinställningen är att tabellerna i ett datalager skall vara normaliserade men att de principerna kan frångås om det finns uppenbara prestandaskäl, t. ex. om det på förhand går att veta hur data kommer att efterfrågas.

Den tredje nivån kallar Inmon (2002) för data mart-nivån men den kallas också för avdelnings-, OLAP- eller multidimensionella nivån. Här finns nästan bara härledd data som ofta är denormaliserad och summerad med hög aggregationsgrad. Den är anpassad efter de användarkrav som finns på den specifika avdelningen och hämtar data från datalagret som är nödvändig för deras behov. En typisk avdelningsindelning kan innehålla avdelningar för marknadsföring, produktutveckling och tillverkning. Som exempel kan kundaktivitet redovisas summerat per kund per månad.

På individuella nivån hämtar enskilda användare specifik data från avdelningsnivån för att göra utforskande heuristiska analyser och försöka hitta intressanta samband som kanske inte

varit kända tidigare. Oftast handlar det om små datavolymer som sparas tillfälligt och i regel arbetar användaren med Excel eller ett webbgränssnitt på en vanlig PC.

2.1.2.3 Operational Data Store

En operational data store (ODS) är en databas med integrerad data indelad i ämnesområden och innehåller aktuell, eller nästan aktuell, data från de operativa systemen (Date, 2004). En ODS innehåller inte historisk data utan uppdateras som en operativ databas. De blir oftast inte speciellt stora men uppdateras väldigt ofta eller kontinuerligt. De används ibland som en tillfällig lagringsplats för att fysiskt strukturera om operativ data. Om datalagret laddas från flera operativa system så kan en ODS användas för att integrera data från de olika källorna. En ODS kan också användas för att skapa operativa rapporter och vara ett operativt beslutsstöd. Data i en ODS är i huvudsak detaljerad och normaliserad. I avsnitt 2.1.5 ges ytterligare ett exempel på vad en ODS kan användas till.

2.1.2.4 Ett datalagers utveckling

Ett helt datalager byggs inte på en gång utan växer sakta fram och utvecklas eftersom behov uppstår (Inmon, 2002). Det börjar med att några tabeller skapas och fylls inom ett ämnesområde och några användare börjar upptäcka vad ett datalager och dataanalyser kan användas till. Efterhand fylls ämnesområdet ut med fler tabeller och fler ämnesområden kommer till, fler användare upptäcker möjligheterna med att göra analyser på data med ett historiskt perspektiv. Vid ungefär den här tidpunkten börjar användarna upptäcka den riktiga nyttan med ett datalager och flyttar dessutom över stora delar av den historiska datan från de operativa systemen där den egentligen inte hör hemma. Nu börjar det bli så mycket data att hantera att prestanda börjar påverkas och vissa användare blir missnöjda med svarstider och tillgängligheten till sitt datalager. Då börjar det komma mindre avdelningsspecifika databaser, data marts, som bara innehåller den data som är relevant för den egna avdelningen. Dessutom kanske datan aggregeras till en lämplig nivå anpassat efter de behov som finns. Fler data marts skapas och avlastar det centrala datalagret och därmed kan fler användare ta del av analysmöjligheterna. På grund av att ämnesområdena utökas och fler ämnesområden skapas samtidigt som detaljerad historisk data samlas in växer datavolymerna i praktiken ofta exponentiellt i ett datalager.

2.1.3 Data Extraction, Transformation and Load

För att uppdatera ett datalager används oftast programvara som kallas för ETL vilket är en förkortning för Extraction, Transformation och Load (Inmon, 2002). Det är verktyg för att

automatisera inläsning från operativa system, konvertering och integrering av inputdata samt skrivning av outputdata till ett datalager.

2.1.3.1 Integrering av inputdata

En av de viktigaste fördelarna med en strukturerad arkitektur är att hela organisationen får en gemensam syn på data, en gemensam datamodell, med ett datalager (Inmon, 2002). För att det skall vara möjligt att ha gemensam data krävs, förutom format- och datastrukturkonverteringar, att data integreras vid inläsning. Inmon säger till och med att det inte är någon idé att föra över data från de operativa systemen om den inte integreras i datalagret. För att visa vad som görs vid integreringen ger han ett exempel med ett försäkringsbolag. Försäkringsbolaget har olika applikationer för liv-, bil- hus- och sjukförsäkringar och i datalagret har de valt att ha ett ämnesområde som de kallar kund. Kunden J Jones finns med i de operativa systemen med olika uppgifter beroende på vilken typ av försäkring det gäller. Vid inläsningen integreras data om J Jones så att de olika kategorierna av data sammanställs till en gemensam post i ämnesområdet kund. Därmed finns all data om J Jones samlad på ett ställe.

2.1.3.2 Komplexitet vid extrahering och konvertering

Att läsa data från ett system och flytta det till ett datalager kan verka ganska enkelt, men det växer snabbt till en komplex och stor uppgift (Inmon, 2002). Att läsa data från ett transaktionssystem kan innebära att inte bara läsa från en annan typ av DBMS, det kan ofta vara ett helt annat operativsystem på en annorlunda hårdvara som inte använder samma typ av datarepresentation. Att välja ut och läsa data från ett applikationsspecifikt system kan vara väldigt komplext. För att veta om en post skall läsas eller inte kan flera koordinerade uppslagningar krävas till en mängd andra poster som ligger i andra tabeller, dessutom kanske det krävs tillhörande logik för att få fram rätt värden.

Väldigt sällan läses data från de operativa systemen utan att nyckelstrukturen behöver ändras för att passa i ett datalager (Inmon, 2002). Data behöver formateras om för att passa det gemensamma formatet som används i ett datalager och ibland behöver algoritmer eller, i komplexa fall, artificiell intelligens användas för att säkerställa att det som sparas är korrekt. I gamla system finns det ofta komplicerade datatyper som sällan beskrivits med lämplig dokumentation och därför kan det krävas mycket tid till att förstå och tolka dem. När data kopieras och konverteras så byter attributen ofta namn vilket kräver noggrann dokumentation. Det kan vara så att för en given input skall flera olika resultat härledas samtidigt beroende på

olika nivåer av summering (Inmon, 2002). Eftersom ett datalager skall spegla historisk data krävs det ofta att datan märks med en tidstämpel.

För att påverka det operativa systemet så lite som möjligt är det mycket viktigt att avgöra exakt vad som behöver extraheras och inte läsa data som inte har uppdaterats (Inmon, 2002). Vid dataaggregeringen kan det spela stor roll i vilken ordning olika källor läses för att få rätt resultat vilket kräver koordinering. När mycket stora datamängder skall kopieras kan det krävas att data t.ex. läses och kopieras parallellt. Dessutom kan det spela roll om själva konverteringen utförs vid de operativa systemen eller om rådata överförs till datalagret för att utföra konverteringen där.

Det finns många fler faktorer att ta hänsyn till, det här är bara ett urval (Inmon, 2002). Förr behövde program specialskrivas för att läsa, konvertera och kopiera data till ett datalager. Det var ett väldigt enformigt och repetitivt arbete och det dröjde inte länge innan det kom applikationer för att automatisera arbetet.

2.1.4 Datahantering

Normalt är inte databaser anpassade för att återspegla hur enskilda värden har utvecklats över tid, oftast sparas bara ett aktuellt värde för en viss post. I ett datalager sparas alla giltiga värden som har varit aktuella för en post vid en viss tidpunkt. För att hantera de stora datavolymer som historisk data medför krävs tekniker för att göra datalagret hanterbart och en strategi för hur detaljerad data som skall sparas behövs.

2.1.4.1 Händelser och snapshots

För att möjliggöra ett historiskt perspektiv på innehållet i ett datalager och ge det en tidsdimension används en struktur som kallas för snapshot (Inmon, 2002). Snapshots är resultatet av en händelse. Det finns många olika typer av händelser men de kan grovt delas in i affärshändelser och tidsrelaterade händelser. Affärshändelser är diskreta aktiviteter som företaget vill hålla reda på och spara information om. En kund till en teleoperatör har kanske ringt ett telefonsamtal och en bankkund har överfört pengar mellan två konton. Det kan även vara en intern affärshändelse i form av en varuleverans. Affärshändelser sker hela tiden men mer eller mindre slumpmässigt. Tidsrelaterade händelser inträffar ofta regelbundet vid en specifik och på förhand bestämd tidpunkt, t.ex. vid slutet på dagen, veckan, månaden eller någon annan tidpunkt som är viktig för företaget. Om det finns en snapshot för alla händelser som inträffar i de operativa systemen inom ett visst ämnesområde är all historisk aktivitet inom det specifika ämnesområdet sparad.

Snapshots består av fyra grundläggande delar, en tidsenhet, en identifierande nyckel, primär data och sekundär data, varav endast sekundär data är valfri (Inmon, 2002).

En tidsenhet, t.ex. år, månad, dag eller timme, representerar oftast tidpunkten för händelsen (Inmon, 2002). Ibland kan den representera den tidpunkt då snapshoten togs, dvs. när data extraherades från datakällan. I vissa fall kan det finnas anledning att göra skillnad på de olika tidpunkterna. En nyckel är oftast sammansatt av flera attribut och är till för att identifiera posten och den primära datan. Primär data relaterar till postens nyckel och är det som företaget egentligen vill samla data om. Sekundär data är valfri och är till för att spara data som inte direkt hör till den inträffade händelsen men som kan vara intressant att spara för analyser i efterhand, det kan vara omständigheter som råkar vara på ett visst sätt. Det finns inget krav på att sekundär data har en direkt relation till primär data i form av främmandenycklar eller liknande. Det finns en implicit relation till den primära datan då de finns i samma snapshot.

I en snapshot kan nyckeln identifiera försäljningen av en produkt och tidsattributet representerar tidpunkten när försäljningen genomfördes (Inmon, 2002). Den primära datan beskriver t.ex. vilken produkt som sålts, till vilket pris, vilka villkor som gällde, var den såldes och vilka som var representanter för köpare och säljare. Den sekundära datan kan representera hur många exemplar av den aktuella produkten och eventuellt liknande produkter som fanns i lager vid försäljningstillfället. Vad företagen väljer att ta med blir en avvägning av det utrymme som krävs mot förväntade eller troliga analysbehov i framtiden.

2.1.4.2 Aggregerad data och profile records

Om datavolymerna är relativt små och det finns ett stort behov av att ha tillgång till alla transaktionsdetaljer kan datalagret spara alla aktiviteter och detaljer om dem i form av snapshots (Inmon, 2002). Om volymerna blir ohanterliga eller behovet av detaljer inte är så stort eller åtminstone minskar med tiden kan en så kallad profile record skapas. De representerar i motsats till en snapshot inte en enskild aktivitet utan är ett aggregat av flera aktiviteter. Precis som en snapshot skapas profile records av händelser men består av värden som summerar, eller utför andra beräkningar på, värden för en mängd associerade aktiviteter. Exempelvis kan en bank summera en kunds kontoaktivitet månadsvis om den är äldre än ett visst antal dagar medan ett telefonbolag kan summera de viktigaste detaljerna för en kunds telefonsamtal. Det är också vanligt att skapa flera profile records utifrån samma data i olika aggregationsnivåer, t.ex. kan ett företag aggregera försäljningsdata per kund, region och land. Det finns oändligt med användningsområden och sätt att aggregera data. Värden från operativ

data kan summeras innan den sparas i ett datalager. Data kan analyseras för att hitta högsta, lägsta, medelvärden och/eller andra viktiga värden. Baserat på olika kriterier kan en speciell delmängd väljas ut och värden från den kan analyseras och beräknas, t.ex. kan försäljningsvärden summeras för kunder inom en viss kategori som handlade mellan jul och nyår.

Enligt Inmon (2002) går det att spara oerhört stora volymer data om profile records används rätt, han säger att det i praktiken är normalt att de minskar datavolymerna mellan 100 och 1000 gånger. Naturligtvis går detaljerna förlorade vid aggregeringen men det är inte säkert att det finns ett affärsbehov för alla detaljer eller att det ekonomiskt går att försvara den extra kostnad det innebär att spara dem. Det är inte all data som aggregeras bort utan bara detaljer som företaget förutser inte är nödvändig i framtiden. Problemet med val av data som skall aggregeras och lämplig aggregationsnivå är att det inte går att veta vilka behov som användarna har eller kommer att ha när datalagret byggs.

För att försäkra sig om att viktig data även är tillgänglig i framtiden föreslår Inmon (2002) två olika tillvägagångssätt. Det första alternativet är att skapa sina profile records iterativt, genom att stegvis aggregera detaljerad data är det lättare att upptäcka när för mycket detaljer går förlorade. Det andra alternativet är att använda billigare lagringsmedium för att arkivera detaljerad data som troligtvis inte kommer behövas eller sällan användas. Då går det alltid få tillbaka och analysera detaljerad data i efterhand även om det tar längre tid och blir mer komplicerat. Förutom att spara plats kan redundanta profile records skapas för data som ofta efterfrågas av användare vilket sparar datorresurser eftersom samma värden inte behöver beräknas flera gånger (Inmon, 2002).

2.1.4.3 Val av lämplig detaljnivå för data i ett datalager

En mycket viktig fråga för ett datalager är hur detaljerad data som skall sparas (Inmon, 2002). Hög detaljrikedom påverkar både prestanda, eftersom många fler poster måste gås igenom, och diskutrymme. Det krävs att en noggrann avvägning görs baserat på vilken typ av frågor det finns behov av att ställa mot databasen. Till exempel är det kanske inte relevant att för varje kund i ett telefonbolag spara detaljerad information om varje enskilt telefonsamtal utan det kanske räcker med en summering av de viktigaste uppgifterna per månad. Om det ändå bara är den eller ännu högre nivåer av aggregation som är av intresse är det bara slöseri med resurser att låta databasen göra samma aggregationer varje gång.

Flexibiliteten vid framtida dataanalyser är direkt beroende av detaljnivån. Men det är inte lätt, det kan till och med vara omöjligt, att förutse alla behov användarna kommer att ha.

Inmon (2002) föreslår att de första delarna i ett datalager byggs i många väldigt små och snabba iterationer och att utvecklarna lyssnar nog på användarna. Det är först när användarna ser vad de kan göra som de kan säga vad de verkligen behöver. Inmon betonar att det är viktigt att komma igång med utvecklingen; det är ett misstag att försöka ta reda på mer än 50% av kraven innan utvecklingen börjar. Om 50% av designen är riktig efter den första iterationen är det en framgång.

När ett datalager byggs måste en uppskattning göras över hur detaljerad data som behövs och hur stora volymer det medför (Inmon, 2002). En första uppskattning kan säga mycket om i vilken storleksordning företagen samlar på sig data och hur noga de behöver överväga vilka detaljer som skall vara med. Om 10 000 poster behövs spelar det ingen roll om alla detaljer tas med. Om däremot 10 miljoner poster sparas behöver antagligen vissa detaljer aggregeras bort. Blir det så mycket som 10 miljarder poster behöver inte bara den mest detaljerade datan aggregeras bort, dessutom behöver en stor del av posterna flyttas över till ett alternativt lagringsmedium.

Inmon (2002) föreslår att det görs en första analys av vilka tabeller som behövs och hur stora det är troligt att de blir. Det räcker att veta i vilken storleksordning tabellerna blir, oftast är det en eller två tabeller som blir riktigt stora och sedan många mindre tabeller som hör ihop med dem. Det behövs en grov uppskattning för varje tabell om hur stor varje post blir, det är kanske inte lätt men det bör göras försök att uppskatta den minsta troliga respektive största troliga storleken. Därefter föreslås att en uppskattning av det största respektive minsta antalet poster inom ett respektive fem år görs. För en kundtabell kan t.ex. marknadens storlek uppskattas och den egna organisationen kan jämföras med konkurrenter för att få en bild av vilka marknadsandelar som är möjliga att ta. När det största eller minsta antalet poster multipliceras med den största eller minsta poststorleken och hänsyn tas till indexstorleken vet utvecklarna vilket storleksintervall de behöver ta hänsyn till när de designar datalagret. Inmon varnar för att utrymmesbehovet ofta underskattas och att volymerna ökar fortare än vad som förutsetts, men det är samtidigt slöseri med tid att försöka bedöma storleken med stor noggrannhet.

2.1.4.4 Hantering av data som används sällan

Medan datavolymerna växer med tiden blir samtidigt stora mängder data gammal och mindre använd (Inmon, 2002). Data som inte används ofta kallas för vilande eller inaktiv och det behövs någon form av mjukvara som övervakar hur ofta viss data efterfrågas för att avgöra när den inte är lika viktig längre. Om det skall vara möjligt att ställa detaljerade frågor

i framtiden är det lämpligt att flytta ut inaktiv data på billigare lagringsmedium, t.ex. lagringssystem med billigare hårddiskar eller magnetband. Eftersom detaljerna blir mindre viktiga med tiden finns det två olika strategier att välja mellan för hur detaljerad data skall hanteras. Vid en bestämd tidpunkt, t.ex. efter 24 månader, kan data flyttas till ett alternativt lagringsmedium. En annan möjlighet är att detaljerad data aggregeras i olika detaljnivåer; när den blir en månad gammal kan en första aggregation göras medan detaljerna flyttas över till det billigare lagringsmediet. Därefter görs flera aggregationer vid olika tidpunkter så att data i det primära lagringsmediet blir mindre detaljerad med tiden. Om billigare lagringsalternativ inte används för inaktiv data begränsas detaljnivån av hur mycket diskutrymme det är ekonomiskt försvarbart att köpa. En annan fördel med att flytta inaktiv data till ett alternativt lagringsutrymme är att frågor mot aktiv data blir effektivare eftersom mindre datavolymer skall hanteras.

Ett datalager bör logiskt sett hantera de olika lagringsalternativen som en enda logisk volym så att inte användarna behöver bry sig om var olika data finns lagrade (Inmon, 2002). Även här krävs mjukvara för att hålla reda på var efterfrågad data finns och automatiskt göra inaktiv detaljerad data tillgänglig vid behov. Eftersom frågor mot de billigare lagringsmedierna tar betydligt längre tid att utföra är det viktigt att se till att det verkligen är sällan och endast i specialfall data efterfrågas därifrån. Det är inte säkert att det bara är med tiden som viss data blir inaktiv, det kan finnas andra skäl till att flytta data om sannolikheten för att den skall efterfrågas minskar, men det kan samtidigt bli aktuellt att flytta tillbaka data till det primära lagringsmediet om förutsättningarna förändras. Det enda sättet att bedöma vilken detaljnivå som behövs och vad som är aktiv eller inaktiv data är att låta användarna se och använda datan och iterativt anpassa detaljnivåerna (Inmon, 2002). I regel bör företagen börja med en hög detaljnivå och därefter aggregera data efter behov.

2.1.5 Behov av att använda datalagrets data i de operativa systemen

Det finns många användningsområden för data från ett datalager i de operativa systemen men det finns ett antal begränsningar som gör att det inte är lämpligt eller möjligt för de operativa systemen att direkt ställa frågor mot ett datalager (Inmon, 2002). Det går inte att räkna med att få ett snabbt svar, det kan ta flera timmar. De operativa systemen är inte anpassade för att ta emot svar på många MB eller GB. Dessutom måste svaret konverteras till ett format som kan hanteras av de applikationerna.

En lösning är att indirekt använda data ur ett datalager för att regelbundet uppdatera profile records för speciella ändamål till en ODS (Inmon, 2002). Speciell programvara analyserar

datalagret regelbundet för att spara aggregerad data i en ODS för att tillgodose specifika behov som användare eller applikationer har. ODS:ens roll blir att som mellanhand erbjuda de operativa systemen aggregerad data och bör därför uppdateras i samma takt som motsvarande data förändras i datalagret. Inmon (2002) ger ett exempel med en bank som har behov av att snabbt kunna göra en kreditbedömning på kunder som kommer in på kontoret och vill låna pengar. Efter att ha lämnat några grundläggande uppgifter vill kunden snabbt ha besked om lånet beviljas eller inte. Om lånet gäller ett tillräckligt stort belopp och kundens bakgrund inte självklart avgör om lånet skall beviljas eller inte behövs en ganska grundläggande bedömning för att banken skall våga låna ut pengar. I datalagret finns information om kundens tidigare förmåga att betala tillbaka, tillgångar, inkomst, utgifter och andra uppgifter som kan vara avgörande. Även om kreditkontrollen skulle ske i bakgrunden medan kunden betjänas tar det mer tid än bara några minuter innan bearbetningen är klar och det går inte att säkert säga när svaret kommer. Lösningen blir, för att betjäna kunderna så snabbt som möjligt, att låta ett analysprogram regelbundet scanna datalagret och utifrån givna kriterier aggregera relevant data och bedöma alla kundernas kreditvärdighet. Programmet körs regelbundet när belastningen på systemet är låg och sparar resultatet i en separat databas som är lätt åtkomlig för de operativa systemen. Därmed kan banktjänstemännen snabbt få en relevant sammanställning av viktig information och kan ofta ge kunden ett besked direkt.

2.2 Multidimensionella databaser

Ett datalager innehåller i huvudsak normaliserad data som är anpassad för flexibilitet och förändrade krav. Däremot är data i en data mart inte normaliserad utan lagras enligt dimensionsmodellen som frångår normaliseringsprinciperna för att optimera prestanda. I det här delkapitlet jämförs dimensionsmodellen med traditionella teoretiska databasprinciper och argument för eller emot de olika synsätten presenteras.

2.2.1 Relationsmodellen och relationsdatabaser

En databas är en samling relaterad data som representerar någon aspekt av verkligheten, den är designad, byggd och fylld med data för ett specifikt syfte (Elmasri & Navathe, 2003). Data i en databas är beständig, dvs. den försvinner inte när programmet som använder den avslutas (Padron-McCarthy, 2003). Ett databashanteringssystem är en samling program som används för att skapa och underhålla databaser (Elmasri & Navathe, 2003). Ett databassystem består av databashanteraren tillsammans med själva databasen. Relationsdatabaser bygger på relationsmodellen som först introducerades av E.F Codd, 1970. Modellen fick stor

uppmärksamhet tack vare sin enkelhet och matematiska grund. Relationsmodellen är en formell teori som främst är baserad på mängdlära och predikatlogik. Relationsmodellen representerar databasen som en mängd relationer.

I relationsmodellen representerar varje rad i tabellen ett faktum som vanligtvis motsvarar en entitet eller relation i verkligheten (Elmasri & Navathe, 2003). Tabell- och kolumnnamnen används för att hjälpa till att tolka betydelsen av värden i varje rad. I den formella relationsmodellterminologin kallas en rad för en *tupel*, en kolumnrubrik kallas för *attribut* och tabellen kallas för en *relation* (för en utförligare beskrivning av skillnaden mellan en relation och en tabell hänvisar vi till bl.a. Date, 2004). Datatypen som beskriver de typer av värden som kan förekomma i varje kolumn är representerad av en *domän* av möjliga värden. En domän D är en mängd atomära värden; med atomär menas att varje värde i domänen är odelbart. En domän har ett namn, en datatyp och ett format. En relations schema R, betecknat genom $R(A_1, A_2, \ldots, A_n)$, består av ett relationsnamn R och en lista med attribut $A_1, A_2, \ldots, A_n$. Varje attribut A är ett namn på det som varje domän D representerar i relationsschemat R. En relations schema används för att beskriva en relation; R kallas för namnet på den relationen. En databas ses som en mängd relationer $DB(R_1, R_2, \ldots, R_n)$. Antalet tupler i en relation kallas relationens kardinalitet och antalet attribut i en relation kallas relationens grad.

Relationsdatabaser baseras på en princip som kallas ”the information principle” (Date, 2004). Principen säger att all information i databasen är representerat på ett och endast ett sätt, nämligen som explicita värden i kolumnpositioner i rader i tabeller. Den här metoden för representation är den enda tillgängliga metoden i ett relationssystem. Det finns inga pekare som kopplar ihop tabeller. De enda kopplingar som finns mellan tabeller är genom att samma värde finns i samma kolumn i de olika tabellerna.

De värden som används för att unikt identifiera en tupel och koppla samman relationer kallas för nycklar. Ett eller flera attribut som unikt identifierar en tupel kallas för relationens primärnyckel. Det attribut i en relation som refererar till en annan relation kallar för främmandenyckel.

För att undvika redundans i databasen och för att få en så bra semantik som möjligt strävar databasdesignern efter att normalisera sin databas så långt det är möjligt. Grundregeln för hur tabeller (relationer) skall se ut är att varje tabell skall beskriva en typ av sak, varje rad (tupel) i tabellen skall innehålla data om en enda sådan sak, och data som finns lagrad för varje sak skall finnas på en enda rad (Padron-McCarthy, 2003).

Det finns olika grader av normaliseringsfomer för att uppnå så låg redundans som möjligt. De vanligaste formerna är första normalformen (1NF), andra normalformen (2NF), tredje

normalformen (3NF) och Boyce-Codds normalform (BCNF). Det finns även en fjärde (4NF) och femte (5NF) normalform men den praktiska nyttan av dem kan ifrågasättas eftersom de begränsningar som de är baserade på är svåra att förstå eller upptäcka av databasdesigners och användare (Elmasri & Navathe, 2003). Nedan listas Dates (2004) definitioner av de vanligaste normalformerna för att ge en inblick i vad normalisering är och en förståelse för 3NF som det ofta hänvisas till senare i uppsatsen:

- En relation är i första normalformen (1NF) omm (om och endast om) varje tupel innehåller exakt ett värde för varje attribut.
- En relation är i andra normalformen (2NF) omm relationen är i 1NF och varje attribut som inte ingår i primärnyckeln är fullständigt funktionellt beroende av den.
- En relation är i tredje normalformen (3NF) omm relationen är i 2NF och varje attribut som inte ingår i primärnyckeln är icketransitivt beroende av den.
- En relation är i Boyce-Codd normalform (BCNF) omm varje determinant är en kandidatnyckel, dvs. skulle kunna vara primärnyckel. Ett eller flera fält i en tabell är en determinant om ett och samma värde på detta alltid entydigt bestämmer värdet på ett annat fält. Till exempel om fält A=1 och fält B=2 i en tupel så är fält A en determinant (för B) och för alla tupler där A=1 måste även B=2.

För att grafiskt illustrera en relationsdatabas används en semantisk modell som kallas entitet/relationsmodellen (E/R-modellen), där entiteter, attribut och relationer ritas ut (Date, 2004). Entiteter representerar ett verkligt objekt eller koncept, t.ex. en anställd eller ett projekt, som är beskrivet i databasen (Elmasri & Navathe, 2003). Ett attribut representerar en egenskap av intresse som ytterligare beskriver entiteten, t.ex. att den anställde har namn och lön. En relation mellan två eller flera entiteter representerar en association, t.ex. en arbetar-i-relation mellan en anställd och ett projekt. För ytterligare detaljer och fler exempel hänvisas till ovanstående författare.

Join är en relationsoperation för att koppla samman tabeller med ett gemensamt attribut. När två tabeller joinas skapas en ny tabell med alla tupler från de två tabellerna där det gemensamma attributet har samma värde (Date, 2004). Ett mycket vanligt och viktigt specialfall av join är sammanslagningen av två tabeller som har en inbördes relation, det vill säga att det gemensamma attributet är primärnyckel i den ena tabellen och främmandenyckel i den andra. Det finns olika varianter av join, se Date (2004) för en utförligare förklaring.

Med begreppet pre-join menas att join-operationen är utförd på förhand, vilket kan göras antingen logiskt eller fysiskt. Ett viktigt argument för pre-join är att det minskar I/O-belastningen (se avsnitt 0). Om en pre-join utförs logiskt innebär det att den påverkar tabellernas utformning, det eller de efterfrågade attributen som finns i den andra tabellen läggs helt enkelt till redundant i den första. Om operationen utförs fysiskt påverkas inte det logiska schemat utan datan placeras optimalt på disk för att minska antalet läsningar. För argument mot konventionell pre-join se avsnitt 2.2.2.

Vi kommer i fortsättningen att i huvudsak använda oss av de mer informella begreppen tabeller, rader eller poster och kolumner eller attribut eftersom databaser som används i datalager oftast inte uppfyller alla de formella krav som ställs på relationsdatabaser.

2.2.2 Databaser för beslutsstöd

Date (2004) skriver att så vitt han vet finns det ingen databashanterare på marknaden som stödjer relationsmodellen i sin helhet. Det betyder inte att någon del av modellen inte är viktig, det är precis tvärtom. Syftet med relationsmodellen är att ge en grund att bygga system på som är så praktiskt användbara som möjligt. Den dystra sanningen, skriver Date, är att ingen tillverkare riktigt har antagit utmaningen att implementera teorin i sin helhet.

Beslutsstödsfrågor kräver ofta åtkomst till många typer av fakta (Date, 2004). I en helt normaliserad databas skulle sådana frågor involvera många joins. Tyvärr, skriver Date, har konventionell join-implementeringsteknologi aldrig klarat att hålla jämn takt med de hela tiden växande kraven på beslutsstödsfrågor. Date hänvisar till McGoveran som i början av 1980-talet observerade att en join av tre tabeller av till och med måttlig storlek lätt kunde ta flera timmar. Joins av fyra till sex tabeller ansågs allmänt alltför dyrt. Idag är joins av sex till tio väldigt stora tabeller vanligt och teknologin fungerar vanligtvis bra. Men det är fortfarande lätt, och vanligt, att skapa frågor som utför en join av fler tabeller än tekniken kan hantera. Frågor som utför en join på mer än tolv tabeller kan bli för komplexa att utföra men kravet på sådana frågor är vanliga.

De som designar datalager väljer då att denormalisera databasen genom att pre-joina vissa tabeller, men strävan efter ökad prestanda skall bara påverka den fysiska designen och inte den logiska. Enligt Date (2004) skall databasdesign alltid utföras i två steg; logisk och sen fysisk:

- Den logiska designen skall göras först. I det stadiet skall fokus ligga på ”relational correctness”: Tabeller måste representera riktiga relationer och genom det garanteras att relationsoperationer fungerar som de skall och inte skapar förvånande

reultat. Domäner och relationer specificeras och utifrån det kan normaliseringen börja.

- Därefter skall den fysiska designen härledas från den logiska. I det stadiet skall fokus ligga på effektiv lagring och prestanda. I princip kan vilken fysisk lagring som helst förekomma så länge det finns en informationsbevarande transformation, uttryckt i relationsalgebra, mellan det logiska och fysiska schemat. Relationssystem kräver bara att databasen uppfattas av användarna som tabeller. På den fysiska nivån kan system lagra datan på vilket sätt som helst, t.ex. genom att använda sekventiella filer, indexering, hashning eller pekare, förutsatt att det kan mappa den lagrade representationen till tabeller på den logiska nivån.

Date (2004) anser att ändringar skall göras i det fysiska schemat och inte i det logiska. Om två tabeller väldigt ofta skall joinas kan det göras en pre-join på den fysiska nivån för att minska I/O- och joinkostnader. Men det logiska schemat måste fortsätta vara oförändrat om fysisk dataoberoende skall uppnås. Om de två tabellerna vid ett senare tillfälle skulle sluta efterfrågas i en join och istället efterfrågas individuellt kan det fysiska schemat ändras igen så att tabellerna åter blir fysiskt separerade och återigen görs det utan att påverka den logiska nivån.

2.2.3 Dimensionsmodellen och dimensionsdatabaser

Relationsmodellen används oftast inte på data mart-nivån i ett datalager. Istället används en modell som kallas dimensionsmodellen.

Dimensionsmodellen gäller enbart för data marts och inte för datalager (Inmon, 2002). I motsats till ett datalager är en data mart i stor utsträckning format efter kraven som ställs på den. För att bygga en data mart måste det finnas mycket kunskap om de bearbetningskrav som omger den. När de kraven väl är kända kan en data mart formas till en optimal starschemastruktur (se avsnitt 2.2.3.6). Ett datalager är fundamentalt annorlunda eftersom det tillhandahåller sina tjänster till en mycket stor grupp användare och användningsområden och är därför inte optimerat för en enskild uppsättning krav. Datalager är formade runt företagets krav på information och inte på en speciell avdelnings krav. Därför skulle det vara ett misstag att skapa ett starschema för datalager eftersom det skulle resultera i ett datalager optimerat för en del av företaget på bekostnad av andra delar.

2.2.3.1 Dimensioner och kuber

Dimensionsmodellering är en logisk designteknik som försöker presentera data i ett intuitivt ramverk som möjliggör hög prestanda vid databasfrågor (Kimball et al., 1998). Varje dimensionsmodell är sammansatt av en tabell med en sammansatt nyckel, kallad faktatabell, och en mängd mindre tabeller som kallas för dimensionstabeller. Varje dimensionstabell har en enkel primärnyckel som exakt motsvarar en av komponenterna i faktatabellens sammansatta nyckel. Den här karaktäristiska stjärnliknande strukturen kallas ofta för starschema. Det är den enda teknik som i praktiken är användbar för att leverera data till slutanvändare av ett datalager (Kimball et al., 1998).

Dimensionsmodellen drar fördel av de naturliga relationerna som finns i en datamängd för att ladda data i multidimensionella, så kallade, kuber (Elmasri & Navathe, 2003). För data som passar dimensionsformatet kan prestandan för frågor bli betydligt bättre än om relationsmodellen används. En kub skulle t.ex. kunna innehålla dimensionerna produkt, region och ett tidsintervall. Om kuben innehåller fler än tre dimensioner kallas den ibland för en hyperkub, men vi kallar dem för enbart kuber oavsett antal dimensioner eftersom det är så de oftast benämns i de här sammanhangen. Data kan efterfrågas direkt i vilka kombinationer som helst utan att komplexa databasfrågor behöver skapas. Det finns verktyg för att grafiskt titta på data utifrån de kombinationer användaren har valt.

Den fundamentala idén med dimensionsmodellering är att nästan all typ av företagsdata kan representeras som en typ av kub av data, där kubens celler innehåller mätvärden och kanterna av kuben definierar de naturliga dimensionerna av datan (Kimball et al., 1998). Dimensionsmodeller som används i affärsvärlden består i praktiken av 4-15 dimensioner.

2.2.3.2 Dimensionsmodellens bakgrund

Många som har försökt leverera data till slutanvändare har sett det som omöjligt att presentera enormt komplexa E/R-scheman för slutanvändarna och har börjat om med en enklare design (Kimball et al., 1998). Den här enklare designen påminner om dimensioner och det är anmärkningsvärt att all den enklare designen ser nästan likadan ut. Kimball et al. (1998) säger att det antagligen är rätt att säga att det inte var en enda person som uppfann det dimensionella angreppssättet. Det är en oemotståndlig kraft i designen av databaser som alltid blir resultatet när designers sätter förståelse och prestanda som högsta mål.

När Kimball pratar om E/R-modellen och E/R-modellering kan det tolkas som att han både menar E/R-modeller som är semantiska och grafiska beskrivningar över relationer och

relationsdatabaser baserade på relationsmodellen (se bl.a. Date (2004) och stycke 2.2.1 i den här uppsatsen).

Kimball och hans medförfattare (Kimball et al., 1998) har lång erfarenhet av att designa och använda databaser för beslutsstöd. Det är utifrån den erfarenheten de har dragit slutsatsen att dimensionsmodellsangreppssättet är det bästa sättet att modellera beslutsstödsdata. Dimensionsmodellering ger det bästa resultatet, både för att det är enkelt att använda och ger hög prestanda.

2.2.3.3 Varför dimensionsmodellen skall användas

E/R-modellering är en logisk designteknik som eliminerar dataredundans och som används för att belysa de mikroskopiska relationer som finns mellan dataelement (Kimball et al., 1998). Det är enormt fördelaktigt för transaktionshantering eftersom det gör transaktioner väldigt enkla, fördefinierade och förutsägbara. Framgången för transaktionshanteringen i relationsdatabaser beror till stor del på disciplinen vid E/R-modellering.

I ivern att göra transaktionshanteringen effektiv förlorades siktet på de ursprungliga målen med relationsdatabaser (Kimball et al., 1998). Transaktionsorienterade databaser som det inte går att ställas frågor mot har skapats. Stora affärssystem för företagsadministration har ofta tusentals entiteter, vilket blir oöverskådligt för en normal användare, var och en av dem blir ofta en fysisk tabell när databasen implementeras. Kimball anser att:

- Slutanvändare kan inte förstå, komma ihåg eller navigera i en E/R-modell. Det finns inget användargränssnitt som tar en E/R-modell och gör den användbar för slutanvändaren.
- Mjukvara kan inte på något lämpligt sätt ställa frågor mot en E/R-modell. Kostnadsbaserade optimerare som försöker göra det gör notoriskt fel val, med katastrofala konsekvenser för prestandan.
- Användandet av E/R-modellering motverkar syftet med ett datalager, nämligen intuitiv och effektiv dataåtkomst.

Dimensionsmodellen har flera viktiga fördelar för datalager som E/R-modellen saknar (Kimball et al., 1998). Det gör att användargränssnitt kan göras mer förståeliga med dimensionsmodellen och bearbetningen kan effektiviseras. Istället för att använda en kostnadsbaserad optimerare kan en databasmotor göra bestämda antaganden. Genom att först begränsa dimensionstabellerna och sen ”attackera” alla faktatabeller samtidigt med den kartesiska produkten av de dimensionsnycklar som motsvarar användarens fråga (se avsnitt

2.2.3.6). Med det här angreppssättet är det möjligt att utvärdera ett godtyckligt antal join till en faktatabell i en enda genomgång av faktatabellens index.

En annan fördel med dimensionsmodellen är att starschemat (se avsnitt 2.2.3.6) står emot oväntade förändringar i användarbeteende (Kimball et al., 1998). Alla dimensioner är likvärdiga och den logiska designen kan göras nästan oberoende av förväntade frågemönster.

En tredje fördel är att den är flexibel och kan ta emot nya oväntade dataelement och går att anpassa efter nya designbeslut (Kimball et al., 1998). Alla existerande tabeller kan förändras utan att data behöver laddas om. Inget fråge- eller rapporteringsverktyg behöver omprogrammeras för att anpassas till förändringen. Gamla applikationer kan fortsätta köra utan att skapa olika resultat.

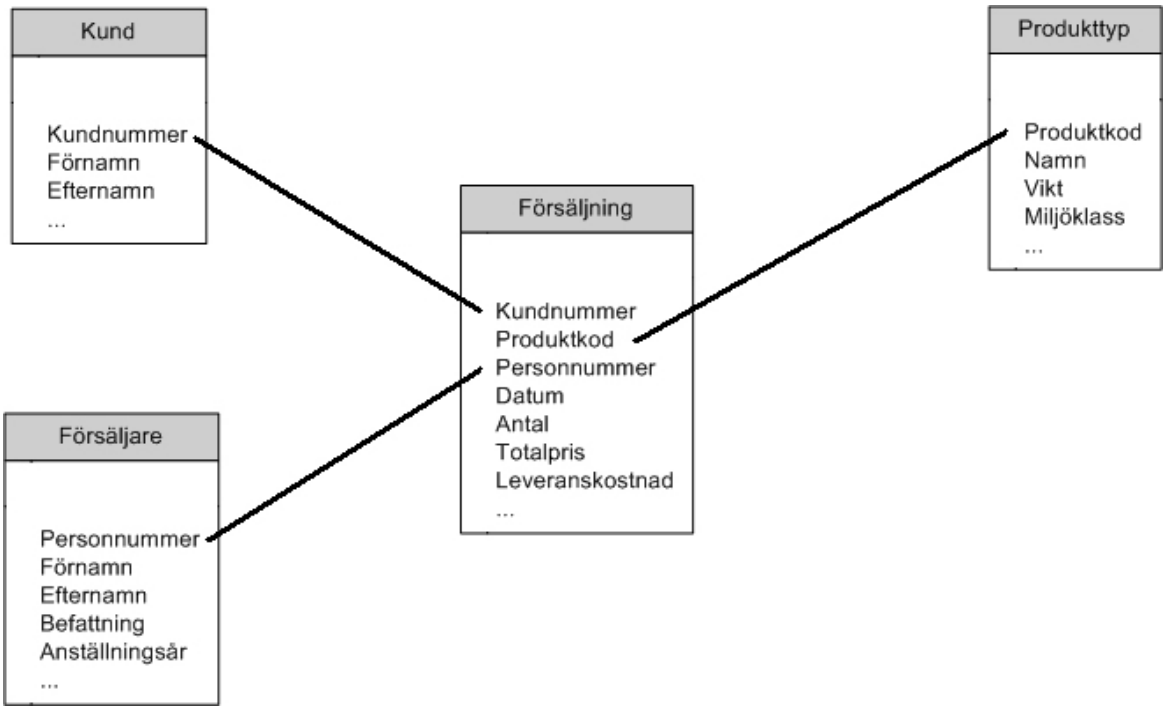
2.2.3.4 Faktatabeller

En entitet representerar vanligtvis ett identifierbart affärsobjekt, som t.ex. Kund, Produkt eller Order (Imhoff et al., 2003). I relationsmodellen påminner strukturen av entiteter, deras relationer och attribut om den verkliga strukturen i organisationen. Så är inte fallet i dimensionsmodellen. Dimensionsmodellen består av två klasser av objekt: dimensioner och fakta. En faktatabell innehåller mått och främmandenycklar till dimensioner. Ett mått är vanligtvis ett numeriskt värde som representerar omfattningen av en händelse som har ägt rum i organisationen, t.ex. antal sålda enheter eller försäljningsvärde. Främmandenycklarna till dimensioner associerade med måtten definierar måttens kontext genom att referera till rätt dimensionsrader.

En faktatabell uttrycker alltid ett många till många förhållande eftersom den har en flerdelad primärnyckel bestående av två eller flera främmandenycklar (Kimball et al., 1998). De mest användbara faktatabellerna består också av en eller flera numeriska fakta som förekommer för kombinationen av nycklar som definierar varje post. Det mest användbara faktat i en faktatabell är numeriskt och additivt. Additivitet är centralt eftersom datalagerapplikationer nästan aldrig får tillbaka en enda faktatabellpost. Snarare hämtas hundratals, tusentals eller till och med miljontals poster på en gång och det enda lämpliga att göra med så många poster är att lägga ihop dem.

Fakta är vanligtvis något som inte är känt i förväg. Oftast numeriska värden men det kan även vara textuella värden (Kimball et al., 1998). Faktatabellen kallas som den gör eftersom det är olika kombinationer värden från den tabellen som det är av intresse att få som svar (Berild, 1997). Om ett företag till exempel är intresserade av att analysera försäljningen under de senaste åren har de uppgifter om det lagrade i en tabell, se Figur 2. Varje försäljning är

dokumenterad med vilken produkt (Produkt) som såldes till vem (Kund), av vem (Försäljare) och när (attributet ”Datum” i faktatabellen). De tre första attributen utgör tillsammans tabellens primärnyckel. Utöver det innehåller faktatabellen i det här exemplet även hur många enheter som såldes (Antal), totalpris och leveranskostnad. Kund och produkt har var för sig en uppsättning attribut utöver Kundnummer och Produktkod som finns med i faktatabellen, dels som uppgifter av allmänt intresse och dels för att användas i mer nyanserade frågeställningar mot försäljningstabellen (faktatabellen) De här attributen lagras i separata tabeller som kallas dimensionstabeller.



Figur 2: Faktatabellen med tillhörande dimensionstabeller (fritt efter Berild, 1997)

2.2.3.5 Dimensionstabeller

Dimensionstabeller innehåller oftast beskrivande textuell information (Kimball et al., 1998). Attributen i dimensioner är vanligtvis textfält och beskriver en karaktäristik av en verklig eller konkret sak. De mest påtagliga attributen är beskrivning av produkter. Beskrivningar, som t.ex. smak, storlek, färg, mäts inte utan är kända på förhand.

En dimension är en samling attribut (Imhoff et al., 2003). Termen attribut som den används i dimensionsmodellen refererar endast till kolumner i en tabell medan attribut i relationstabellen är en mer allmän term. I relationsmodellen skulle både dimensionsattribut och mått betraktas som attribut.

Det är vanligt att ha dimensioner som representerar affärsobjekt, t.ex. en order, men det är inte nödvändigtvis så (Imhoff et al., 2003). Till exempel innehåller en dimensionsmodell kanske inte dimensioner för transaktionsentiteter som en order. Istället kan det finnas flera dimensioner som innehåller attribut som beskriver olika aspekter av en order, som betalningsvillkor och leveranssätt. Främmandenycklarna i faktatabellen Order skulle då tillhandahålla associationen mellan dimensionerna så att det genom de relationerna ges en komplett bild av ordern.

De textuella attributen som beskriver saker är organiserade inom dimensionerna, t.ex. produktdimension, kunddimension, tidsdimension, affärsdimension (Kimball et al., 1998). En dimension är en samling, ofta textuella, attribut som har en stark relation till varandra. I en återförsäljardatabas skulle produktdimensionen med affärsdimensionen kunna kombineras och skapa en enhetlig produkt-affär-dimension. Om det finns 1000 produkter och 100 affärer och det inte finns något meningsfullt samband mellan produkt och affär och varje produkt såldes i alla affärer skulle den kombinerade produkt-affär-dimensionen bli den kartesiska produkten av de två ursprungliga dimensionerna och det skulle finnas 100 000 produkt-affärer. Den här dimensionen skulle inte användas eftersom den skulle ge prestandaproblem och inga fördelar för användargränssnittet. Om den kombinerade dimensionen istället bara innehöll 2000 produkt-affärer skulle de indikera att produkterna såldes i specifika affärer och då skulle den kombinerade dimensionen vara väldigt användbar. Den skulle vara väldigt kompakt och avslöja mycket intressant information om förhållandet mellan produkter och affärer. Designern kan använda intuition för att bestämma om dimensioner skall kombineras. För abstrakta dimensioner kan de behöva utföra operationer för att se om kombinationen av attribut är oberoende eller inte.

Dimensionstabellerna är beskrivande till sin natur och inte additiva (Inmon et al. 1997). De beskriver eller kategoriserar data i den relaterade faktatabellen. För att optimera tabellerna för frågor genomförs på förhand join-operationer för att inte behöva genomföra dem för varje fråga som ställs. De här pre-joinade främmandenyckelrelationerna kopplar ihop faktatabellen med dimensionstabellerna. Nyckeln i faktatabellen är en sammansatt nyckel som består av alla dimensionstabellers nycklar. Resultatet av det är att det kommer att finnas en rad i faktatabellen för varje unik nyckelkombination av faktatabellens nycklar och alla nycklar i alla dimensionstabeller. Därför bör dimensionstabellernas nycklar inte ha högre detaljnivå än vad som är nödvändigt för att uppfylla det syfte som starschemat var skapat för.

2.2.3.6 Starschema och star-join

Vanligtvis resulterar dimensionsmodelleringen i vad som kallas för ett starschema. Det har fått sitt namn efter utseendet som påminner om en stjärna eftersom det illustreras med faktatabellen i mitten omgiven av dimensionstabellerna (Date, 2004). Ett starschema finns illustrerat i Figur 2 och Figur 3.

Inmon (2002), anser att nackdelen med datamodeller är att de visar inbördes relationer mellan tabeller i en platt struktur och det är därför lätt att entiteterna uppfattas som likvärdiga. I verkligheten är entiteter allt annat än likvärdiga, en del entiteter kräver speciell hantering. Skillnaden i kardinalitet mellan faktatabellen och dimensionstabellerna kan vara mycket stor. Om faktatabellen innehåller orderinformation och dimensionstabellerna innehåller kund- och produktinformation kommer ordertabellen mest sannolikt vara mycket större än dimensionstabellerna. Faktatabellen innehåller både data som unikt identifierar ordern och data om själva ordern. Dessutom innehåller faktatabellen nycklar för att identifiera vilka kunder och produkter som hör till ordern. Av prestandaskäl kan faktatabellen redundant innehålla icke-nyckeldataattribut från dimensionstabellerna om det är data som ofta efterfrågas tillsammans med ordern.

En fördel med att skapa starscheman är att optimera data för beslutsstödsbearbetning (Inmon, 2002). Genom att utföra en pre-join och skapa selektiv redundans förenklas och optimeras datan enormt för åtkomst och analys, vilket är precis vad som behövs för en data mart. Utanför data mart-miljön, där data ofta uppdateras, skulle ett starschema mest troligt vara ohanterligt att bygga och underhålla. Men eftersom ett data mart laddas batchvis, innehåller historisk data och hanterar stora mängder data är en starschemastruktur idealisk för bearbetningen som görs i en data mart.

Frågor mot ett starschema innebär vanligtvis att alla nycklar av intresse först letas upp i dimensionstabellerna, sedan beräknas den kartesiska produkten (alla möjliga kombinationer) av dem (Date, 2003). Resultatet används för att komma åt relevanta poster i faktatabellen. Istället för att utföra det här arbetet med två olika frågor kan det slås ihop till en enda fråga, vilket kallas för en star-join. Normalt undviker frågeoptimeraren att beräkna den kartesiska produkten men i dimensionsdatabaser är det nästan alltid effektivare att först beräkna den kartesiska produkten på de mycket mindre dimensionstabellerna för att sedan använda resultat för indexbaserade uppslagningar mot faktatabellen.

2.2.3.7 Hierarkier

Relaterade entiteter i ett datalager uttrycker ofta en förälder-barn-relation (Imhoff et al., 2003). I den typen av relation kan en förälder ha flera barn medan ett barn bara kan ha en förälder. Till exempel kan en person tillhöra en avdelning och avdelningen kan bestå av flera personer. I en förälder-barn-relation behöver inte föräldern och barnet ha något gemensamt utöver relation mellan dem. En hierarki är en speciell typ av en förälder-barn-relation där ett barn representerar en mer detaljerad nivå än en förälder. I de flesta hierarkier i affärsrapportering har varje barn bara en förälder. Det är önskvärt när rapporter skall tas fram eftersom varje barn då bara räknas en gång i en aggregation. Om ett barn har flera föräldrar förekommer samma data under varje förälder och det gör förfrågningar och rapportframställningar mer komplicerat. Hierarkier implementeras med hjälp av olika typer träd- och nyckelstrukturer (för en utförlig beskrivning se Imhoff et al., 2003).

2.2.3.8 Slice and dice

Slice and dice innebär att informationen kan ses ur olika perspektiv genom att den kan grupperas på ett visst sätt och analyseras för att sedan grupperas på ett annat sätt och analyseras på nytt (Inmon, 2002). Slice and dice innebär att projection utförs på dimensionerna, dvs. istället för att hämta ut rader från en dimension väljs de attribut som är av intresse att analysera ut ur en eller flera dimensioner (Elmasri & Navathe, 2003).

2.2.3.9 Drill down och Roll up

För att kunna utföra slice and dice är det nödvändigt att kunna borra sig ner, drill down, i datan till mer detaljerade nivåer (Inmon, 2002). Med drill down menas att gå från en aggregationsnivå och bryta ner den stegvis till mindre aggregerade nivåer. På det sättet går det att komma åt mer detaljerad information om de delar som är av intresse medan det samtidigt går att välja att bara se summerad data om det räcker för att uppfylla användarens syfte. Till exempel kan en användare titta på organisationens försäljningssiffror för olika regioner. Användaren kan sedan välja ut en del av regionen att titta närmare på för att sedan gå ner till enskilda städer för att i detalj analysera hur försäljningen har sett ut där. Kanske går det också att gå vidare ner till varje enskild försäljare i staden för att se vem som har sålt vad. På det här sättet kan t.ex. någon på ledningsnivån spåra speciella försäljningsresultat och i detalj analysera var det har gått bra eller dåligt för att kunna ta beslut om åtgärder.

Hur långt ner det är möjligt att gå i en hierarki och hur detaljerad data som går att komma åt beror på vilken data som finns i datalagret (Inmon, 2002). Ju mer detaljerad data som finns

desto mer exakta och detaljerade analyser kan användaren göra. När användare gör en drill down går de från den individuella nivån där den mest summerade datan finns, vidare till data mart-nivån, till datalagret och slutligen till den arkiverade datan där flest detaljer finns. Genom alla nivåer är datan relaterad till varandra genom en nyckelstruktur som gör drill down-analysen enkel och naturlig.

Begreppet drill up används bland annat Kimball och Ross (2002), medan t.ex. Elmasri och Navathe (2003) använder begreppet roll up för samma sak. Det innebär att data summeras med stigande generalisering, t.ex. genom att användaren går från att titta på försäljningen veckovis till kvartalsvis till årsvis.

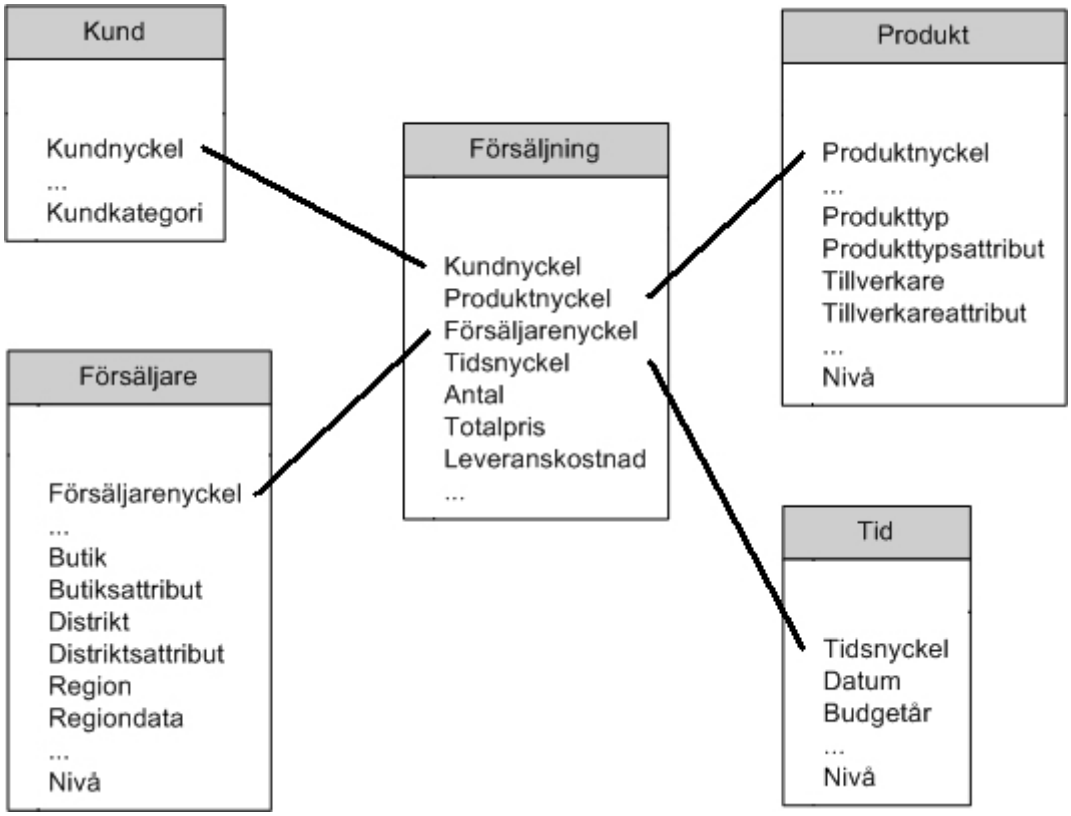
Drill down och drill up är i praktiken inget annat än att lägga till eller ta bort attribut från en dimension i en rapport (Kimball & Ross, 2002). Genom att lägga till attribut av intresse görs en drill down till nästa detaljeringsnivå. Första gången en användare gör en drill down i en rapport läggs t.ex. ett produktkategoriattribut till som en rubrik i rapporten. Nästa gång läggs en underkategori till och tredje gången läggs märkesattributet till. Till slut kommer användaren ner till den lägsta nivån i produkthierarkin där attributet för detaljerad produktbeskrivning finns. Det här korta exemplet är dock, enligt Kimball och Ross, väldigt begränsat och ofta inte var användarna vill. Riktig drill down blandar hierarkiska och icke-hierarkiska attribut från alla tillgängliga dimensioner. Det finns nästan alltid mer än en väldefinierad hierarki i en given dimension. Till exempel kan marknadsavdelningen och ekonomiavdelningen på ett företag ha olika och inkompatibel syn på produkthierarkin och då finns båda avdelningarnas attribut med i produktdimensionen. Användare måste kunna gå över alla hierarkier och välja orelaterade attribut som inte är en del av hierarkin.

2.2.3.10 Ett större exempel

Följande exempel med figurer är en modifiering av ett exempel från Berild (1997). För enkelhetens skull förutsätter vi att kuben laddas med data batchvis en gång varje natt. Kuben laddas med data från datalagret som i sin tur får datan från ett transaktionssystem.

Ett starschema med Försäljning som faktatabell och Kund, Produkttyp och Försäljare som dimensionstabeller skulle kunna visa relationer i en vanlig relationsdatabas, se Figur 2. Skulle försäljarna och produkterna vara indelade i en intern struktur, där försäljarstrukturen skulle bestå av fyra hierarkiska nivåer (Region, Distrikt, Butik, Försäljare) och produktstrukturen av tre hierarkiska nivåer (Tillverkare, Produktkategori, Produkttyp) skulle det ställa krav på en tabell per nivå, om varje nivå kräver en beskrivning, för att uppfylla 3NF. Den enkla

grundstrukturen som vi hade tidigare blir nu en komplex struktur med ytterligare fem tabeller vilket framförallt gör det svårare att formulera frågor mot den nya strukturen.



Figur 3: Starschema med flera nivåer och en extra dimension (fritt efter Berild, 1997)

Istället för att utöka strukturen med de fem nya tabellerna kan ett alternativ vara att föra in samtliga generaliseringsnivåer i den lägsta, mest detaljerade, nivåns tabell. I Försäljartabellen tillkommer då en rad för alla butiker, distrikt och regioner med deras attributsuppgifter, förutom alla enskilda försäljare, se Figur 3. De nya kategorierna kan ses som mer generaliserade ”försäljare”. Försäljare ”Dalarna” representerar alla försäljare i alla butiker i distriktet Dalarna. Nu räcker inte Personnummer som nyckel eftersom det attributet inte finns för ”försäljare” på nivå 2 och uppåt. Istället används en neutral Försäljarenyckel som primärnyckel. På samma sätt görs med Produktstrukturen. När alla nivåers samtliga attribut finns integrerade på en och samma plats kan frågor ställas på en valfri delmängd av samtliga attribut vilket Berild (1997) anser är en mycket flexibel förutsättning.

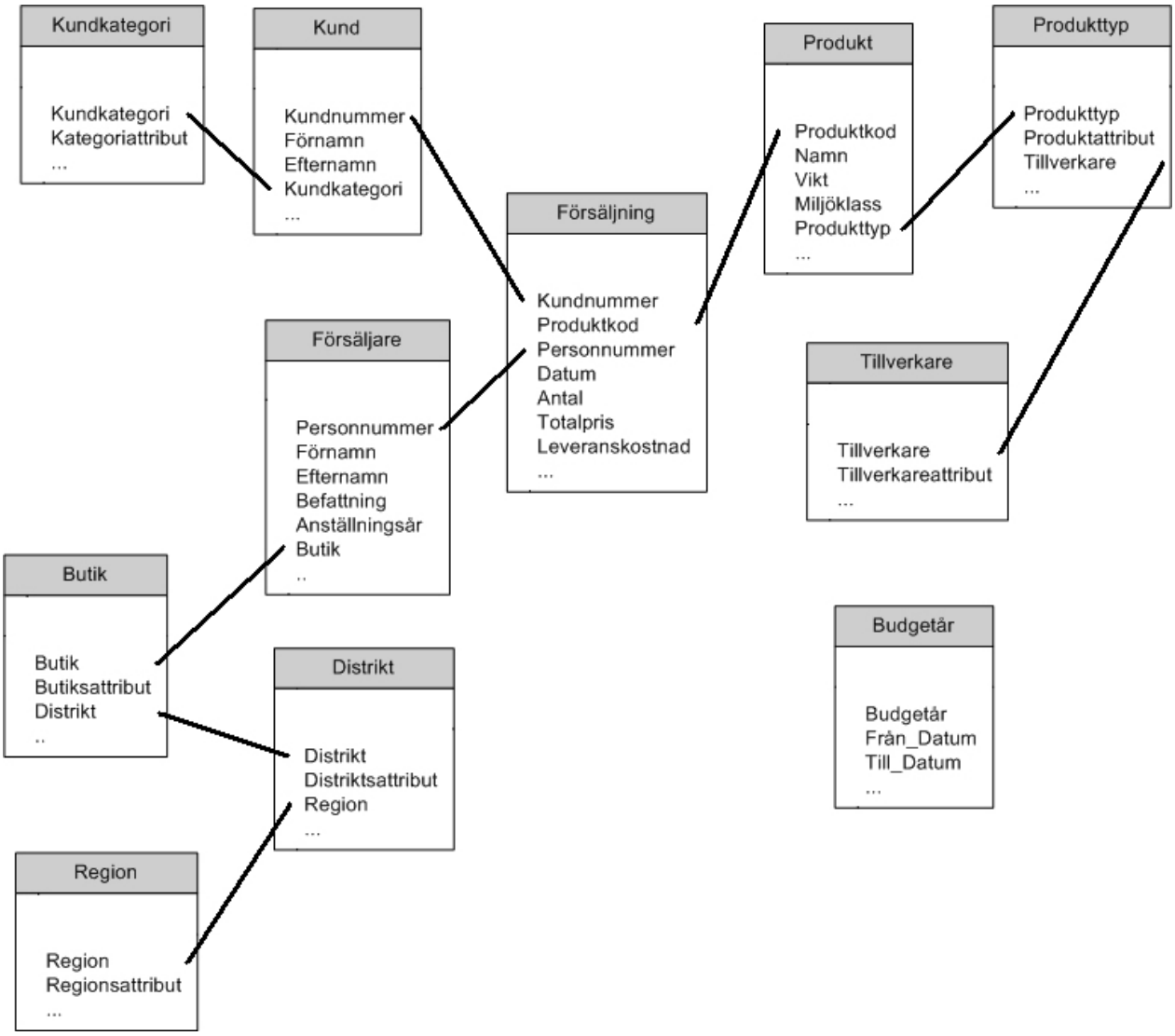
För att utökningen av dimensionstabellernas betydelse skall vara till nytta bör också faktatabellen ändras för att svara mot de nya förutsättningarna. Faktatabellen skall innehålla

försäljningsuppgifter både för enskilda och generaliserade försäljare. För t.ex. försäljare ”Dalarna” skall det finnas beräknat samtliga kombinationer av övriga primärnyckelkomponenter. Dit hör i en tupel ”Dalarnas försäljning under datum X till kund Y av produkttyp Z” och i en annan tupel ”Region Norrs försäljning under datum X till kund Y av produkttyp Z” osv.

Eftersom dimensionstabellerna nu måste kunna innehålla attribut från samtliga nivåbeskrivningar blir tabellerna både denormaliserade och mycket stora. Ett extra attribut i form av en nivåangivelse måste läggas till dels för att indikera vilken uppsättning attribut som är aktuella för en aktuell tupel, (t.ex. Butiksdata, Distriktsdata) men framförallt för formuleringen av entydiga frågor.

Faktatabellen innehåller nu både de tidigare detaljerade uppgifterna och summeringar över alla dimensionsnivåkombinationer. När en enskild försäljare säljer en produkt till en viss kund sparas all information om försäljningen i transaktionssystemet. När data nästa dag laddas in i datakuben från datalagret laddas inte bara informationen kopplat till den enskilda försäljaren utan även summerad data för varje butik, distrikt och region. Om t.ex. 5 olika försäljare har sålt 100 stycken produkt X och de alla tillhör butik A som tillhör distrikt B som tillhör region C kommer faktatabellen att innehålla en tupel för varje försäljare med 100 som Antal på Produkt X, medan butik A, distrikt B och region C har varsin tupel med 500 som Antal på Produkt X. Förutsatt att produkt X inte har sålts av någon annan försäljare i samma region samma dag.

Antag att även Kund har en intern struktur med ”Stamkund” och ”Strökund” som två möjliga värden för Kundkategori samt att dimensionen Tid läggs till med Budgetår som högsta nivå. Om följande fråga ställs (med pseudokod): ”Ge mig Leveranskostnad för region ”Norr” (Försäljare), kundkategori ”Stamkund” (Kund), tillverkare ”AB Spis och Kyl” (Produkt), budgetår ”1995” (Tid)” ges ett mycket summerat värde till svar. Med t.ex. 5 regioner, 5 kundkategorier, 20 tillverkare och 8 budgetår finns det i faktatabellen 5 * 5 * 20 * 8 = 4000 summavärden av samma dignitet, att jämföras med att kanske behöva summera värden från många miljoner tupler med primitiv data. Summeringarna för alla nivåer med aggregerade värden bör göras i samband med laddningen till datakuben. Om det hade varit tillåtet att uppdatera värden i datakuben skulle väldigt många aggregerade värden behöva uppdateras för varje uppdatering på lägre nivåer. För att illustrera hur tabellstrukturen skulle kunna se ut i en normaliserad databas som kan besvara den här frågan ritade vi Figur 4.



Figur 4: Normaliserade tabeller istället för starschemat i Figur 3 (egen figur)

Som det framgår av exemplet ovan används inte samma designprinciper som vid vanlig datamodellering (Berild, 1997). Det kan vara effektivt att denormalisera tabellerna för att snabba upp frågor och göra datamodellen mer överblickbar. Behovet av joins begränsas genom att varje dimension endast är en tabell. Summerad data kan snabbt tas fram, ofta utan beräkningsbehov. Ställs andra frågor än rent nivåbaserade måste dock beräkningar utföras på vanligt sätt, t.ex. ”För alla distrikt som…och alla tillverkare som…”.

Bland nackdelarna med starscheman nämner Berild (1997):

- Dimensionstabellerna blir stora
- En artificiell nivåindikator måste alltid anges

- Både primitiv och aggregerad data finns i samma jättelika faktatabell istället för att de normaliseras till mindre tabeller
- Statisk hierarki-struktur

Date (2004) säger att på grund av brist på designdisciplin är det inte säkert att tabellernas innehåll i ett starschema kan tolkas enhetligt. Han säger att det bland annat inträffar när en tabell samtidigt innehåller data med olika grad av aggregation. För olika poster har attributen olika innebörd. Om tabellen skall innehålla tidsangivelser som t.ex. år, månad och dag så måste exempelvis dag tillåta null-värden för poster som summerar värden för en månad. Det är precis vad som händer i det här exemplet och därför behövs ett attribut för att indikera vilken aggregationsnivå posten representerar. Det leder enligt Date (2004) till att tabellerna blir mindre effektiva att använda och svårare att förstå och underhålla.

2.2.3.11 Snowflakeschema

Nivåindikatorn i starschemat är ett bekymmer på grund av den stelbenthet den bygger in i strukturen (Berild, 1997). Samtidigt är den nödvändig, t.ex. för att korrekt specificera villkorsangivelser vid frågor för att inte blanda ihop olika nivåer. Ett sätt att undvika nivåindikationer är att ta ett steg tillbaka mot normalisering, men inte fullt ut. Nivåindikatorerna tas bort från dimensionstabellerna och istället skapas en ny sub-dimensionstabell för varje aggregerad nivå med samtliga relevanta attribut för den nivån samt nycklar till nivåer med mer aggregation. Därigenom kan frågor ställas lika flexibelt som förut samtidigt som nivåindikatorn ersatts med en explicit dimensionstabell per nivå. Faktatabellen delas upp på ett motsvarande sätt. Den här typen av schema kallas för snowflakeschema.

Bland fördelarna med att använda snowflake kan följande nämnas (Berild, 1997):

- Varje uppgift upplevs ligga på rätt plats
- Risken för felaktigt ställda frågor minskar
- Dimensionstabellerna kan joinas för att vid behov nå uppgifter på den lägsta nivån
- Strukturella ändringar blir enklare
- Utsökningar av summerad data går snabbare över mindre tabeller

Bland nackdelarna kan nämnas (Berild, 1997):

- Ett mycket fragmenterat schema
- Enhetlig namngivning av faktatabell-attribut är ett krav
- Inkonsistensrisker mellan tabeller

- Tuff databasadministratörsuppgift
- Tungt/svårt att besvara vissa typer av frågor

2.2.3.12 Varför snowflakeschema inte bör användas

Dimensionsmodellen bryter mot traditionella modelleringsregler eftersom fokus ligger på att leverera affärsnytta genom enkel användning och hög prestanda och inte på effektiv transaktionsbearbetning (Kimball & Ross, 2002). Argument för att normalisera tabellerna är ofta att spara utrymme och förenkla underhållet av tabellerna. Att skapa ett snowflakeschema är ett sätt att normalisera dimensionstabellerna. Redundanta attribut bryts ut ur de flata, denormaliserade dimensionstabellerna och placeras i normaliserade dimensionstabeller. Om starschemat var ett snowflakeschema fullt ut skulle det vara ett fullständigt E/R-diagram i 3NF.

Kimball & Ross (2002) avråder från att använda snowflakescheman och hänvisar till enkelheten att använda tabellerna och prestanda som de två primära drivkrafterna för designen. De anser att:

- Mängden snowflaketabeller ger en mycket komplex presentation som försvårar för användaren
- Flera tabeller och joins ger lägre prestanda när frågor skall bearbetas
- Snowflake gör att användandet av bitmapindex försvåras. Bitmapindex är väldigt anändbara när attribut med låg kardinalitet skall indexeras. De ökar prestandan på frågor mycket. Användandet av snowflake skulle ofrånkomligen minska möjligheterna till prestandaoptimeringar

Dimensionstabellerna skall fysiskt fortsätta vara flata tabeller (Kimball & Ross, 2002). Diskutrymme som sparas genom normalisering av dimensionstabellerna är, enligt Kimball och Ross, ofta mindre än en procent av det totala diskutrymmet som behövs. Den tid som läggs ner på att försöka spara diskutrymme är därför bortkastad. De säger dock att det finns tillfällen när snowflake kan användas men de manar till att använda den designen konservativt och bara när det är helt uppenbart att det behövs.

Angående snowflake konstaterar Date (2004) enbart att de även går under namnet ”blizzard”, med för honom uppenbar innebörd.

2.2.4 Kritik mot vanliga designprinciper för multidimensionella databaser

Date (2004) tar kortfattat upp några vanliga praktiska designtekniker som han anser är dåliga. Han konstaterar att många av designsvårigheterna med beslutsstödssystem beror på att goda designprinciper inte disciplinerat följs utan att designbeslut fattas ad hoc baserat på intuition. Dessutom förvärras problemen av begränsningar i SQL, vi hänvisar till Date (2004) för en beskrivning.

2.2.4.1 Denormalisering

Att utföra pre-join och att använda olika typer av härledd data för att minska I/O-belastningen kan vara acceptabelt på den fysiska nivån men inte om det märks på den logiska nivån (Date, 2004). Date är direkt kritisk mot Kimball som anser att dimensionstabellerna inte bör normaliseras.

2.2.4.2 Starscheman

Dimensionsscheman är ofta ett försök att ta genvägar istället för att disciplinerat följa riktiga designtekniker (Date, 2004). Det görs ingen skillnad mellan logisk och fysisk design med det angreppssätt som används för starscheman. Det går inte att vinna mycket på att ta de här genvägarna, ofta blir både prestanda och flexibilitet lidande när databasen växer. Att försöka lösa de problemen när de uppstår genom att förändra den fysiska designen kan få omfattande konsekvenser. Starscheman är egentligen fysiska scheman som är exponerade mot applikationer som också behöver förändras.

2.2.4.3 Hantering av Null-värden

Designers försöker ofta spara utrymme genom att tillåta null-värden (Date, 2004). Det kan fungera då vissa databashanterare t.ex. fysiskt representerar ett variabelt långt attribut som en sträng med en tom sträng vid null. Det är ett misstag som generellt bör undvikas, databasen både går och bör designas så att null-värden undviks helt. Det ger scheman som oftast kräver mindre lagringsutrymme och ger mindre I/O-belastning.

2.2.4.4 Tabeller med aggregerad data

Om en logisk design inte genomförs, vilket är vanligt, av tabellerna med aggregerad data så riskeras okontrollerad redundans och problem med inkonsistent data (Date, 2004). Användarna kan bli förvirrade över vad den aggregerade datan verkligen betyder och hur de skall ställa frågor mot den. Alla sådana tabeller bör designas som om de utgjorde en egen separat databas. Då går det att undvika problem med cykliska uppdateringar genom att se till

att data aggregeras från den lägsta detaljnivån (atomär normaliserad data) och uppåt på ett kontrollerat sätt.

2.2.5 Online Analytical Processing

Online Analytical Processing (OLAP) definieras som ”en kategori av mjukvaruteknologi som möjliggör för analytiker och företagsledare att få insyn i data genom snabb, konsistent, interaktiv access till en bred variation av möjliga vyer av information som har transformerats från rådata till att reflektera de olika dimensionerna av en organisation så som de ses av användarna” (Inmon et al., 1997). Mer koncist kan sägas att OLAP är en mängd funktionalitet som försöker underlätta multidimensionell analys, vilket innebär att manipulera data som har aggregerats till olika kategorier eller dimensioner. Syftet med multidimensionell analys är att hjälpa användaren sammanställa företagsinformation genom jämförbara, individualiserade vyer, och genom analys av historisk och beräknad data.

OLAP är vanligtvis implementerat på avdelningsnivån eller individuella nivån (Inmon et al. 1997). Avdelningsnivån kallas även för OLAP-nivån eller data mart. Data i OLAP-nivån kommer ursprungligen från den organisatoriska nivån i ett datalager (Inmon et al. 1997). Den detaljerade, historiska datan är det centrala i ett datalager och är en perfekt grund för datan i OLAP-nivån. En av OLAP-miljöns karaktäristik är flexibilitet. Vissa avdelningar har extremt dynamiska krav på information och OLAP-miljön kan agera och svara på de frekvent och ofta radikalt förändrade kraven utan att kräva en förändring av det underliggande datalagret.

OLAP beskrivs av Kimball et al. (1998) som aktiviteten att fråga efter och presentera textuell och numerisk data från en data mart. OLAP-tillverkarnas tekniker är inte baserade på relationsmodellen utan är nästan alltid baserad på en multidimensionell datakub. OLAP-databaser kallas också för muldidimensionella databaser.

Date (2004) definierar OLAP som ”den interaktiva processen för att skapa, underhålla, analysera och rapportera data”. Han säger också att OLAP vanligtvis definieras med tillägget att datan upplevs och hanteras som om den var lagrad i en multidimensionell vektor.

2.2.5.1 Multidimensional OLAP

Multidimensional OLAP (MOLAP) är det traditionella sättet att utföra OLAP-analyser på (Lee, 2004). I MOLAP är data lagrad i en multidimensionell kub. Datan är inte lagrad i en relationsdatabas utan i ett proprietärt format. En fördel med MOLAP är utmärkt prestanda, MOLAP-kuber är byggda för snabb dataåtkomst och är optimala för slicing och dicing. En annan fördel är att komplexa beräkningar kan utföras. Alla beräkningar har förgenererats när

kuben skapats, därför är det inte bara möjligt att utföra komplexa beräkningar utan de returnerar också snabbt. En nackdel är att bara en begränsad mängd data kan hanteras, eftersom alla beräkningar görs när kuben byggs är det inte möjligt att inkludera stora datamängder i kuben. Det betyder inte att data i kuben inte kan härledas från en stor mängd data. Det är möjligt, men i så fall kan bara summerad data inkluderas i kuben. En annan nackdel är att det ofta krävs extra investeringar eftersom kubteknologi ofta är proprietär och inte redan finns i organisationen. Därför är det sannolikt att ny programvara behöver köpas in och att personalen behöver utbildas om MOLAP-teknologi skall tas i bruk.

MOLAP är en mängd användargränssnitt, applikationer och proprietär databasteknik som har ett stark dimensionellt utseende (Kimball et al., 1998).

2.2.5.2 Relational OLAP

Relational OLAP (ROLAP) manipulerar data som är lagrad i relationsdatabaser för att ge intryck av att det finns traditionell OLAP slicing och dicing-funktionalitet (Lee, 2004). En fördel med ROLAP är att stora mängder data kan hanteras. ROLAP har ingen begränsning på datamängden, den enda begränsningen är hur mycket data de underliggande relationsdatabaserna kan hantera. En annan fördel är att ROLAP effektivt kan använda sig av och utnyttja funktionalitet som finns i relationsdatabaser. ROLAP-teknologin kan erbjuda den funktionaliteten eftersom den ligger ovanpå en relationsdatabas. En nackdel med ROLAP är att prestandan kan vara låg. Eftersom varje ROLAP-rapport i grund och botten är en eller flera SQL-frågor till relationsdatabasen kan det ta lång att bearbeta frågorna om det är mycket data som skall bearbetas. En annan nackdel är att ROLAP är begränsat av funktionaliteten i SQL. Eftersom ROLAP-teknologin mest förlitar sig på att skapa SQL-uttryck för att ställa frågor mot databasen och de inte passar för alla behov (t.ex. är det svårt att utföra komplexa beräkningar med SQL) är ROLAP begränsat till vad SQL kan göra.

ROLAP är en mängd användargränssnitt och applikationer som ger en relationsdatabas en dimensionskänsla (Kimball et al., 1998).

2.2.5.3 Val av OLAP-teknik

I sin mest allmänna form är OLAP ett medel att underlätta multidimensionell analys (Inmon et al. 1997). En metod för att utföra multidimensionell analys är att använda datadesigntekniker som starscheman för att simulera en multidimensionell struktur i en relationsdatabas. En annan är att använda ett slutanvändarverktyg som kan komma åt data i en relationsdatabas och erbjuda multidimensionella analysmöjligheter i det verktyget. En tredje

metod är att använda en multidimensionell databas för att direkt spara datan multidimensionellt.

Hur OLAP bäst skall implementeras debatteras mycket mellan olika typer av tillverkare (Devlin, 1997). Tillverkare av OLAP-servrar och multidimensionella databaser menar att relationsdatabaser inte är kapabla att effektivt möta användarnas krav på analyser och erbjuda mjukvara som är optimerad för de kraven. Tillverkare av relationsdatabaser och beslutstödsverktyg som jobbar mot sådana databaser pekar på starscheman, parallell bearbetning och förbättrade indexeringstekniker, som alla tillåter relationsdatabaser att effektivt hantera datan medan slutanvändarverktyg tillhandahåller den multidimensionella vyn.

2.3 Prestandaaspekter och optimeringar

För att datalagret skall kunna besvara användarnas frågor i rimlig tid även när datavolymerna kraftigt ökar krävs en mängd tekniker för att optimera prestandan. I det här delkapitlet redogörs för några av de viktigaste teknikerna. Datavolymerna växer i praktiken mer än vad som kan förutses av mängden affärsdata (Winter Corporation, 2003). Den verkliga lagringskapaciteten som krävs för att lagra affärsdata i ett datalager kan vara fem till sex gånger större än enbart affärsdatan i sig. Det extra utrymmet utgörs av index och aggregat och därför är det viktigt att de teknikerna utnyttjas effektivt.

2.3.1 Indexering

Det är inte bara denormalisering som kan användas för att begränsa mängden arbete som behöver utföras för att ställa frågor mot datalagret (Inmon et al., 1999). Indexeringstekniken är ett annat sätt att uppnå samma resultat. Genom att indexera data i ett datalager på ett intelligent sätt kan både prestandan och skalbarheten öka.

Det finns två grundläggande indexeringstekniker som en datalagerdesigner bör känna till; binärträdindexering (B-träd) och bitmapindexering (Imhoff et al., 2003). B-träd indexering används i stor utsträckning i OLTP-databasdesign och är det bästa valet där frågevägarna är kända och kontrollerade. Bitmapindex är bäst i ad hoc frågemiljöer och är det bästa valet för dimensionella data marts.

Rätt sorts index kan radikalt minska I/O-belastningen (Date, 2004). De första SQL-produkterna på marknaden använde bara B-trädsindex men numera finns en mängd olika index, optimerat för olika ändamål, speciellt i samband med beslutsstödsdatabaser, t ex:

- B-träd är effektivt för att komma åt en sammanhängande delmängd av poster om inte delmängden inte är för stor. B-träd är effektiva för uppdateringar.
- Bitmap-indexens viktigaste fördel är att flera relationsoperationer (t.ex. join, union och lika med-begränsningar) kan utföras direkt inom indexen med booleska operationer på bit-vektorerna. Det är inte nödvändigt med någon dataåtkomst förrän resultatet av operationerna är klart. Uppdateringar är däremot relativt ineffektiva med Bitmap-index.

Datalagrets underliggande teknologi skall inte bara kunna stödja skapandet och laddningen av nya index utan de måste också gå att komma åt på ett effektivt sätt (Inmon, 2002). Det finns flera tekniker för effektiv indexåtkomst, t ex:

- Använda index i flera nivåer
- Lagra alla delar av ett index i primärminnet
- Komprimera indexposter i de fall datan som indexeras tillåter det

2.3.1.1 Binärträdindex

Binärträd (B-träd) index använder en rekursiv trädstruktur för att lagra indexeringsvärdet och pekare till andra indexeringsnoder och, om det är en lövnod, en pekare till dataraden (Imhoff et al., 2003). Strukturen kallas för ett binärträd eftersom ett binärt val görs vid varje nod som besöks: Är värdet på indexnoden mindre än eller större än värdet som eftersöks? Testet används för att välja en av de två vägar som tillhandahålls av pekarna till nästa noder. Till sist leder vägen till en lövnod som matchar eller inte matchar det värde som eftersöks. Om det inte matchar har det eftersökta värdet inte hittats och om det matchar använder databasen lövnodspekaren för att hämta dataraden.

2.3.1.1.1 Olika typer av B-trädindex

Det finns två vanliga typer av B-träd index; enkla index och sammansatta index (Imhoff et al., 2003). Ett enkelt index är ett index baserat på en enda kolumn. Ett sammansatt index är ett index baserat på två eller fler kolumner. När ett sammansatt index designas är ordningen på kolumnerna mycket viktig. Positionen av kolumner i indexdefinitionen har samma betydelse som positionerna för bokstäverna har i en alfabetisk lista. Den första kolumnen är den mest signifikanta och krävs för att användningen av indexet skall vara effektiv. Om den första kolumnen inte används som ett predikat i frågan är indexet inte användbart. Det är samma

situation som när man försöker hitta ett namn i telefonkatalogen och bara känner till andra, tredje och fjärde bokstaven i namnet. Det betyder inte att sammansatta index skall undvikas (Imhoff et al., 2003). B-träd index kan inte användas i kombination med varandra. Om en tabell har ett enkelt index på datum och ett enkelt index på kund och en specifik kund efterfrågas för ett specifikt datum skulle databasen behöva välja vilket index som skall användas, båda kan inte väljas. Valet skulle utföras av databasoptimeraren och som resultat skulle den titta efter alla rader för det specifika datumet och scanna efter kunder eller titta på alla rader för kunder och scanna efter datum. Om istället ett sammansatt index bestående av både kund och datum hade använts skulle det indexet användas för att hitta rätt rader direkt. Det sammansatta indexet skulle utföra sökningen bättre än något av de två enkla indexen.

2.3.1.1.2 Fördelar med B-trädindex

B-trädindex fungerar bäst i en kontrollerad miljö där åtkomsten till tabeller, för uppdateringar och frågor, går att förutse (Imhoff et al., 2003). Det är möjligt i ett datalager eftersom både uppdateringar och frågor är kontrollerade av utvecklingsteamet. Noggrann design av indexen ger optimal prestanda med minimal overhead.

B-trädindex kräver lite underhåll. Databastillverkare har, enligt Imhoff et al. (2003), gjort mycket för att optimera sina indexstrukturer och algoritmer för att träden hela tiden skall vara balanserade. Frekvent uppdatering av tabeller ger ingen markant försämring av indexprestandan men det är en god idé att bygga om indexen med jämna mellanrum som en del av det normala underhållet.

2.3.1.1.3 Nackdelar med B-trädindex

Eftersom B-trädindex inte går att använda i kombination måste det skapas sammansatta index för att stödja alla förväntade frågor mot tabellerna (Imhoff et al., 2003). Det innebär inte att ett enkelt index behöver skapas för frågor efter enskilda attribut om det samtidigt ingår i ett sammansatt index.

En annan nackdel är betydelsen av ordningen på kolumnerna i ett sammansatt index (Imhoff et al., 2003). Ett val behöver då göras mellan att skapa flera sammansatt index eller acceptera att vissa frågor kräver sekventiell scanning efter att de har tagit så stor hjälp de kan från de existerande indexen. Vilket val som bör göras beror på vilka index som finns och på datans natur. Om de existerande indexen resulterar i en scanning av ett inte alltför stort antal rader för en viss fråga är det antagligen inte värt att skapa fler index för att slippa den scanningen. Ju fler indexstrukturer som skapas desto långsammare blir uppdateringsprocessen.

B-trädindex kan bli stora (Imhoff et al., 2003). Förutom kolumnerna som utgör indexet består en indexrad också av mellan 16 och 24 byte pekare och intern data som används av databassystemet. Det behövs också 40% av storleken som overhead för att täcka icke-lövnoder och tomma eller outnyttjade utrymmen.

2.3.1.2 Bitmapindex

Bitmapsindex är det bästa indexet att använda i dimensionella data marts, men de används nästan aldrig i OLTP-databaser (Imhoff et al., 2003). Bitmapindex används bäst i miljöer vars primära syfte är att stödja ad hoc frågor. Indexen kräver dock mycket underhåll och hanterar inte uppdateringar speciellt bra. Bitmapindexering kan avsevärt förbättra I/O och ge lagringsutrymmesfördelar för domäner med låg kardinalitet (Elmasri & Navathe, 2003). Dramatiska förbättringar kan uppnås för jämförelser, aggregering och join-prestanda.

2.3.1.2.1 Bitmapstrukturen

En bitmapindexstruktur består av en serie bitvektorer (Imhoff et al., 2003). Det finns en vektor för varje unikt värde som är giltigt för ett attribut i en tabell. Varje vektor innehåller en bit för varje rad i tabellen. Imhoff et al. (2003) ger ett exempel med en tabell som innehåller information om bilar som kan ha färgerna röd, vit eller silver. Det finns då tre vektorer för de tre färgerna. Vektorn för röd kommer att innehålla 0 om färgvärdet i den raden inte är röd och 1 om färgen i raden är röd. Om alla röda bilar skulle efterfrågas skulle databasen leta igenom vektorn för färgen röd för att hitta alla bitar som är satta till 1. Sökningen går ganska snabbt men skiljer inte så mycket från ett B-trädindex, den kan till och med gå långsammare. Fördelen med ett bitmapindex är att det kan användas i kombination med andra bitmapindex. Till exempel skulle ett bitmapindex kunna finnas för biltyp. I exemplet finns två typer av bilar; sedan och kupé. Om alla bilar som är kupé och inte vita skulle efterfrågas kan databasen lösa frågan genom att använda ett bitmapindex och booleska operationer. Datan behöver inte läsas innan rätt poster har hittats. Först går databasen genom vektorn för färgen vit och genomför en NOT-operation och tar det resultatet och genomför en AND-operation med vektorn för kupé. Resultatet blir en vektor som identifierar alla rader som innehåller röda och silver kupéer. Booleska operationer på bitvektorer är mycket snabba operationer för vilken dator som helst. Ett databassystem kan utföra det här urvalet mycket snabbare än om ett B-trädindex hade skapats för biltyp och färg. Eftersom B-trädindexet måste hämta värden för varje jämförelse tar det längre tid än om jämförelseoperationerna kan göras direkt i indexet, vilket är fallet med bitmapindex.

Bitmapindex förekommer ofta i datalagersammanhang (Berild, 1997). Det är en indexeringsteknik som erbjuder mycket snabba utsökningar i de fall man har ett eller flera attribut med en mycket begränsad kardinalitet. Berild (1997) ger exempel på en faktatabell med bl.a. attributen År, Månad och Region, se Figur 5.

År	Månad	Region
1995	Januari	Syd
1997	Juni	Sydväst
1995	Mars	Nord
1996	September	Nordost
…	…	…

Figur 5: Faktatabell som skall indexeras (Berild, 1997)

I stället för att etablera en, två eller tre indextabeller för dimensionerna mot faktatabellen kompletteras faktatabellen med en kolumn för varje möjligt värde (Berild, 1997). Varje kolumn upptar en bit där 1 anger att värdet gäller för raden medan 0 anger att så inte är fallet, se Figur 6.

1995	1996	1997	Jan	Feb	Mar	Apr	Maj	Jun	Jul	Aug	Sep	Okt	Nov	Dec	Syd	Sydväst	Väst	Ost	Nordost	Nord
1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0
1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0

Figur 6: Exempel på bitmapindex (Berild 1997)

2.3.1.2.2 Storlek på bitmapindex

Längden på en bitmapvektor är direkt relaterad till storleken på tabellen (Imhoff et al., 2003). Vektorn behöver en bit för att representera en post och för en tabell med 8 miljoner poster skulle alltså bitvektorn behöva 1 Mb för att lagra alla bitar. Om kolumnen som skall indexeras har väldigt hög kardinalitet med 1000 möjliga värden skulle storleken på indexet, med 1000 vektorer, vara 1 Gb (8 miljoner rader * 1000 kolumner).

I realiteten tar inte vektorerna upp så mycket plats, eftersom ett bitmapindex med index på flera tabeller i så fall skulle kunna bli flera gånger större än själva tabellen (Imhoff et al.,

2003). Med 1000 möjliga värden innehåller en vektor som representerar ett värde mycket fler 0:or än 1:or, därför kan datakomprimeringstekniker användas för att markant minska storleken på vektorerna. I praktiken kräver ett bitmapindex på en tabell med 1 miljon rader och en kolumn med 30 000 olika värden bara 4Mb för att lagra indexet. Ett jämförbart B-trädindex behöver 20 Mb eller mer, beroende på storleken på kolumnen och den overhead som skapas av databassystemet. Komprimeringen ger också stor effekt på hastigheten av indexen. Eftersom de komprimerade indexen är små kan utvärderingen av dem ske helt i minnet, till och med på väldigt stora tabeller.

2.3.1.2.3 Underhåll av bitmapindex

Den största nackdelen med bitmapindex är att de kräver konstant underhåll för att fortsätta vara kompakta och effektiva (Imhoff et al., 2003). När ett kolumnvärde ändras måste databasen uppdatera två vektorer. För det gamla värdet måste biten ändras från 1 till 0. Först måste vektorsegmentet för biten lokaliseras och dekomprimeras innan det kan ändras och slutligen komprimeras igen. Det finns då risk för att segmentets storlek har ändrats så att systemet måste placera segmentet någon annanstans på disken och skapa en länk tillbaka till resten av vektorn. Samma process genomförs för det nya värdet, om det inte har någon vektor skapas den och kommer då att innehålla en bit för varje rad i tabellen.

Alla ändringar och skapandet av nya vektorer fragmenterar kraftigt bitmapvektorerna (Imhoff et al., 2003). När vektorerna delas upp i mindre och mindre segment minskar komprimeringseffektiviteten. Storleksökningen kan vara enorm, index kan växa till 10 till 20 gånger den normala storleken efter att 5 % av en tabell har uppdaterats. Dessutom måste databasen lägga ihop segmenten som är utspridda över olika delar av disken för att undersöka en vektor. Både storleksökningen och fragmenteringen bidrar till att göra indexen långsammare. Lösningen på problemet är att bygga om indexen efter varje dataladdning. Det är, som tur är, inget stort problem i ett datalager eftersom de flesta databassystem kan utföra den operationen snabbt.

2.3.1.2.4 När bitmapindex bör användas

Imhoff et al. (2003) förespråkar omfattande användning av bitmapindex i data marts, varje faktatabells främmandenyckel och vissa av dimensionsattributen bör indexeras på det här sättet. De skriver också att B-trädindex och bitmapindex skall undvikas att användas i kombination i en data mart. Anledningen till det är att förhindra databasoptimeraren från att göra ett val. Om enbart bitmapindex används kan frågor utföras på ett enhetligt och

förutsägbart sätt. Om B-trädindex också används kommer situationer att uppstå där optimeraren gör ett felaktigt val och frågorna tar lång tid att köra.

2.3.2 Partitionering av data

Partitionering kan underlätta hanteringen och tillgängligheten för en given tabell (Date, 2004). Partitionering innebär att en tabell delas upp horisontellt i två eller flera separata delar baserat på värdet av utvalda attribut. En partitioneringsfunktion tar attributvärden som argument och returnerar ett partitionsnummer eller en adress. Varje partition kan tilldelas hårdvara, som t.ex. en CPU eller en disk, baserat på hur den används. Ett system som stödjer partitionering bör göra det så att varken applikationer eller användare behöver känna till att en tabell är partitionerad. Logiskt oberoende gör att en administratör kan skapa och slå ihop partitioner baserat på prestandabehov. Det är optimerarens ansvar att hålla reda på vilka partitioner som fysiskt behöver läsas för att presentera ett svar på en given fråga.

Ett datalager behöver partitioneras på ett lämpligt sätt, annars blir de enorma datavolymerna nästan ohanterliga (Inmon, 2002). Partitioneringen kan t.ex. göras efter datum, affärsområde, geografiskt område eller organisatorisk enhet. Data separeras i olika fysiska enheter som kan hanteras oberoende av varandra, men ett viktigt krav är att varje given mängd data endast tillhör en partition.

2.3.3 I/O-relaterade prestandaaspekter

Ett datalager behöver minimera antalet I/O-operationer för att inte begränsas av att diskarna relativt sett är långsamma. Om läsningarna kan minimeras ökar datalagrets totala prestanda.

2.3.3.1 Fysisk databasdesign

Det är viktigt att väga in prestandaaspekter när en detaljerad datamodell tas fram och försöka gruppera data i de olika tabellerna baserat efter hur ofta den förändras (Inmon, 2002). Val av detaljrikedom, dvs. hur detaljerad data som skall sparas, och val av lämplig datapartitionering är mycket viktig ur prestandahänseende.

Eftersom data normalt inte uppdateras i datalagret kan det vid den fysiska datadesignen tas hänsyn till hur många I/O-förfrågningar som krävs för att läsa in data (Inmon, 2002). Denormalisering av tabeller med data som har hög sannolikhet att efterfrågas samtidigt i join-operationer kan minska I/O-operationer om tabellernas data placeras tillsammans på disk. I speciella fall kan det dessutom löna sig att placera data sekventiellt i så kallade ”arrayer” för att ytterligare, fysiskt, föra samman data som hör ihop. Enligt Inmon är det då helt nödvändigt

att i förväg känna till att läsningarna sker sekventiellt och att data läggs till, och eventuellt uppdateras, på ett förutsägbart sätt. Eftersom data i ett datalager alltid är relevant ur något tidsperspektiv och det är väldigt vanligt med data med en tidsenhet är det ofta naturligt att skapa arrayer baserat på en lämplig tidsenhet, t.ex. per månad.

Inmon (2002) anser också att det går att minska antalet I/O-operationer genom att lägga till redundant data i vissa, annars normaliserade, tabeller medan andra tabeller bryts upp i delar baserat på hur ofta olika attribut efterfrågas. Det första fallet exemplifierar han med en textuell beskrivning av en produkt som finns i en huvudtabell för produkter tillsammans med ett unikt identifierande produktnummer och relaterade produkttabeller med produktnumret som främmandenyckel. Tabellerna är normaliserade och beskrivningen är lätt att uppdatera och referensintegriteten är lätt att upprätthålla. Om frågor ofta ställs mot de andra produkttabellerna men ofta enbart behöver beskrivningen ur huvudtabellen blir det många uppslagningar i ”onödan” varje gång som beskrivningen används jämfört med om beskrivningen redundant hade funnits i de andra produkttabellerna också. Eftersom uppdateringar inte är ett stort problem i ett datalager så kan bättre prestanda uppnås med den här typen av frågor på bekostnad av extra utrymme. Det andra fallet exemplifierar Inmon med en bankkontotabell där det är stor sannolikhetsskillnad mellan hur ofta olika attribut efterfrågas. Attributet med saldot används betydligt oftare än attributet med data om när kontot öppnades. Inmon föreslår en uppdelning i två tabeller baserat på hur ofta de olika attributen används för att populär data skall sparas mer kompakt och mer samlat för att minska antalet läsningar från disk.

2.3.3.2 Aggregerad data

I vissa fall kan aggregerad data spara I/O-operationer och beräkningskraft om den läggs till i den fysiska datadesignen (Inmon, 2002). Om vissa beräkningar görs regelbundet, t.ex. för att sammanställa hur mycket lön som betalats ut under ett år och vad den aktuella skatten blev, så kan kolumner läggas till i tabellen för att spara de aktuella värdena första gången de beräknas. Fördelen är att om de uppgifterna behöver användas flera gånger har de redan beräknats och är lättillgängliga. För komplicerade beräkningar som utförs regelbundet och kräver tillgång till stora mängder data kan datalagret avlastas betydligt. Ett annat sätt att utnyttja aggregerad data är att, om det i förväg är känt att användarna regelbundet är intresserade av specifik information, utföra beräkningarna medan primitiv data laddas in till ett datalager. Eftersom all data ändå skall behandlas vid inläsningen krävs det relativt lite extra datorkraft för att utföra de här beräkningarna. Den här metoden kallas ”creative index” eller ”creative profile” och går

ut på att nyckeltal av intresse beräknas av vid ett tillfälle och sparas undan för effektiv åtkomst vid senare tillfällen. Exempel på användningsområden kan vara att hålla reda på transaktionerna med det största eller minsta värdet, genomsnittligt transaktionsvärde för den aktuella inläsningen, de mest aktiva eller inaktiva kunderna, eller kunder som var aktiva utan att handla, t.ex. i en webbshop.

2.3.4 Kontrollerad redundans

Kontrollerad redundans är en viktig teknik för att minska I/O-belastningen (Date, 2004). Kontrollerad betyder att den hanteras av databashanteraren och är dold för användaren. Om redundansen hanteras ordentligt på den fysiska nivån är den osynlig på den logiska nivån och påverkar inte den logiska modellens riktighet. Aggregerad data är speciellt viktigt i databaser för beslutsstödssystem och innebär att vissa beräkningar endast behöver utföras en gång. Det finns främst två typer av aggregerad data, beräknade kolumner och summeringstabeller (Date, 2004):

- Beräknade kolumner har ett attribut per rad som innehåller ett aggregerat värde baserat på andra attribut i raden *eller* aggregerat från flera rader i samma eller någon annan tabell. Eftersom det innebär att värdet för en rad beror på andra raders värden så kräver uppdatering av en rad att många andra rader behöver uppdateras. Det kan kraftigt påverka prestandan vid laddning och uppdatering av databasen.
- Summeringstabeller innehåller härledda (summeringar, medeltal, antal och på annat sätt beräknade) värden i andra tabeller och är ofta olika aggregat av samma atomära data.

Aggregerad data bör, enligt Date (2004), implementeras med hjälp av systemkontrollerade ”triggers”, inte med procedurer skapade av användaren, för att undvika problem med inkonsistent data. För att redundansen verkligen skall vara kontrollerad krävs det att den är helt dold för användaren.

2.3.5 Skalbarhet

En plattform som möjliggör hög prestanda är nödvändig i en datalagermiljö (Inmon et al., 1999). Det är plattformen som begränsar hur mycket processorkraft och I/O-bandbredd som finns tillgänglig, hur många användare som kan hanteras, hur mycket data som kan lagras och hur lätt miljön kan anpassas till växande och ändrade användarkrav. Eftersom datalager kan ses som stora samlingar data från många olika operativa system är datavolymerna som

behandlas betydligt större än volymerna som behandlas i de operativa systemen (Inmon et al., 1999). Men det är inte bara att de ofta är ovanligt stora som gör dem speciella. Det är det faktum att, oavsett hur stora de är när de först utvecklas, de oundvikligen kommer att växa sig mycket större. De måste stödja fler användare, mer data, mer komplexa frågor och mer funktionalitet. Det är därför datalager måste ses som konstant växande applikationer i ständig förändring.

Ett datalager måste byggas med skalbarhet i åtanke för att möta kraven på kommande prestandakrav (Inmon et al., 1999). En skalbar plattform måste kunna växa i antalet slutanvändare, datamängd, komplexitet och funktionalitet och samtidigt behålla sin existerande prestandanivå. Inmon et al. nämner att de från erfarenhet har sett att datalagerplattformar som inte är byggda med skalbarhet i åtanke kollapsar ”under sin egen vikt” inom ungefär 18 månader. Slutsatsen de drar från det är att plattformen måste vara skalbar för att möjliggöra hög prestanda i datalagermiljön som kan bevaras över tiden.

2.3.5.1 Parallellism

För att stödja skalbarhet måste nyare teknologi än datalagerplattformar tidigare varit byggda på användas (Inmon et al., 1999). Det finns två huvudsakliga fördelar med att använda parallellism. För det första kan prestandanivåer uppnås som är långt bortom vad som kunde uppnås med traditionella teknologier tidigare.

Det finns två sätt att mäta prestandaökningen: speed-up och scale-up. Med speed-up menas att den enskilda uppgiften utförs snabbare. Det påverkar responstiden och kan mätas genom att dela den tidigare responstiden för samma uppgift med speed-up responstiden. Med scale-up menas att även om den enskilda uppgiften inte nödvändigtvis går att utföra fortare så kan fler användare använda systemet samtidigt utan att responstiden påverkas. Scale-up är relaterat till genomströmningen i systemet och kan mätas genom antal frågor färdiga per minut eller timme. Den andra stora fördelen med parallell teknologi är ett överlägset pris per prestandaförhållande. Det kostar inte bara dubbelt så mycket att köpa en dubbelt så snabb processor men däremot kan två likadana processorer köpas till det dubbla priset mot en samtidigt som den totala processorkraften teoretiskt fördubblas. Förbättrat pris per prestanda genom parallell datorkraft har dock en kostnad: ökad operativ komplexitet.

Det finns tre huvudsakliga typer av parallellism (Inmon et al., 1999):

- Funktionell parallellism
- Dataparallellism
- Pipelineparallellism

2.3.5.1.1 Funktionell parallellism

Funktionell parallellism är den vanligaste av de tre typerna (Inmon et al., 1999). Funktionell parallellism drar fördel av flera processorer genom att låta varje processor arbeta på olika uppgifter. Förfrågningar kommer in till en schemaläggare och skickas vidare till första lediga processor. Varje processor kan vanligtvis hantera alla typer av förfrågningar. Det finns en variation på funktionell parallellism som kallas partitionerad funktionell parallellism som innebär att varje processor är dedikerad till att utföra en viss typ av uppgift där schemaläggaren skickar förfrågan till rätt processor för uppgiften. I datalager används den senare tekniken ofta för att utföra parallella frågor. Då är vissa processorer dedikerade till att läsa från diskarna, några processorer används till att utföra join på tabeller medan andra utför sorteringsoperationer.

Målet med fuktionell parallellism är att uppnå scale-up (Inmon et al., 1999). Eftersom varje förfrågan behandlas av en processor tar det teoretiskt sett lika lång tid att behandla en förfrågan som tidigare, men genom att använda flera processorer samtidigt kan fler förfrågningar bearbetas på samma tid. På så sätt kan fler användare stödjas utan att prestandan sjunker, förutsatt att det inte finns några andra flaskhalsar i systemet som t.ex. begränsad I/O.

2.3.5.1.2 Dataparallellism

Dataparallelism innebär att data delas upp i mindre, mer hanterbara bitar och fördelas till flera olika processorer istället för att en processor utför en förfrågan (Inmon et al., 1999). Huvudmålet med dataparallellism är speed-up. En processor behöver bara utföra beräkningar på en del av datan och hela beräkningen kommer att utföras på kortare tid. Om det t.ex. är 5 processorer som arbetar parallellt kommer arbetet, teoretiskt sett, ta en femtedel av tiden det skulle ha tagit för en enda processor.

2.3.5.1.3 Pipelineparallellism

Med pipelineparallellism bearbetas förfrågningar och data i olika steg och varje steg utförs av olika processorer. Den första processorn utför det första steget i databehandlingen och sedan skickas datan vidare till nästa processor som tar över, så fortsätter det tills behandlingen är klar. Med pipelineparallellism utförs inte frågor fortare men det kan dramatiskt bidra till scale-up och att förbättra systemets genomströmning. Det sker eftersom processorerna jobbar på allteftersom och då levereras resultat till användaren med kortare tidsintervaller, varje fråga

kräver fortfarande lika lång behandlingstid men på samma tid genomförs fler frågor istället för bara en.

2.3.5.2 Hårdvaruarkitekturer

Idag finns ett flertal olika hårdvaruarkitekturer som utnyttjar parallellism och har därmed bra skalbarhet (Inmon et al., 1999). Det är skalbarheten som gör dem speciellt bra lämpade för datalager. Det är dock inte så enkelt att det bara går att välja en lösning och vara säker på att den kommer att möta prestandakraven. Varje arkitektur har både styrkor och svagheter. I en enprocessorarkitektur är det processorn som är flaskhalsen. Mängden data att behandla och antalet användare att ta hand om ökar men den tillgängliga processorkraften kan inte öka, hårdvaran tillåter inte det. Prestandan minskar linjärt med ökad arbetsbelastning.

Nedan tar vi upp fyra olika skalbara hårdvaruarkitekturer som alla adresserar skalbarhetsbegränsningarna i enprocessorarkitekturen, men till priset av ökad komplexitet. Multiprocessorsystem har tre huvudfördelar (Silberschatz, Galvin & Gagne, 2003):

1. Ökad genomströmning: Genom att öka antalet processorer kan mer arbete utföras på kortare tid, men ökningen är inte proportionerlig mot antalet processorer som läggs till. När flera processorer samarbetar på en uppgift skapas en del overhead för att få alla delar att fungera rätt. Processorerna utnyttjas inte heller maximalt eftersom de måste vänta på att få tillgång till de gemensamma resurser som används av andra processorer. Därför minskar den förväntade nyttan av extra processorer.

2. Skalfördelar: Multiprocessorsystem kan vara billigare än flera enprocessorsystem eftersom de kan dela kringutrustning, lagringsenheter och strömkällor. Om flera program arbetar med samma datamängd är det billigare att lagra den datan på en disk som alla processorer delar än att ha flera datorer med lokala diskar och flera kopior av samma data.

3. Ökad tillförlitlighet: Om funktioner kan distribueras mellan flera processorer och en processor slutar fungera, slutar inte hela systemet fungera utan går bara långsammare. Om tio processorer används och en går sönder måste de övriga nio dela på den tionde processorns arbete. Men istället för att hela systemet slutar fungerar går det bara 10% saktare. Det här sättet att fortsätta ge service proportionellt mot nivån av fortsatt fungerande hårdvara kallas ”graceful degradation”. System som är designade för det kallas också feltoleranta.

2.3.5.2.1 Symmetriskt multiprocessorsystem

Istället för att koppla en processor till systembussen går det med ett symmetriskt multiprocessorsystem (SMP) att koppla ihop flera processorer som delar på samma buss och minne (Inmon et al., 1999). Eftersom alla hårdvarukomponenter delas, kallas SMP också för ”shared-everything hardware architectures”. Ett SMP system kan bestå av två till 32 eller fler processorer (TechEncyklopedia, 2005). Om en processor går sönder går hela systemet ner. Det kan lösas genom att använda kluster av två eller fler SMP system för högre tillgänglighet. Om ett SMP system slutar fungera fortsätter de andra att arbeta.

En av processorerna startar systemet och laddar operativsystemet som startar upp de övriga processorerna (TechEncyklopedia, 2005). Det finns bara en instans av operativsystemet och en instans av applikationen i minnet. Operativsystemet använder processorerna som en pool av processorkraft där alla processorer kör samtidigt. Antingen bearbetar de data eller väntar på att göra någonting. Processorerna tilldelas nästa tillgängliga uppgift eller tråd som kan köras samtidigt. SMP snabbar upp alla processer som kan överlappas. Processorerna kommunicerar inte direkt med varandra under arbetet, de utväxlar data genom att skriva och läsa i det gemensamma minnet (Computer Swedens språkwebb, 2005).

Det är svårt att konstruera symmetriska system med fler än cirka 30 processorer, eftersom det blir hög belastning på den gemensamma minnesbussen. Däremot är symmetriska system relativt enkla att programmera jämfört med alternativet, massivt parallella system. Men eftersom processorerna delar på resurser är de också mindre effektiva. Termen symmetrisk syftar på att processorerna är identiska och har tillgång till samma data. I ett SMP system är alla processorer jämlikar, det finns inget master-slav förhållande mellan dem (Silberschatz, Galvin & Gagne, 2003). Fördelarna med en sådan arkitektur är tydlig, enligt Inmon et al. (1999). Man kan börja med ett litet antal processorer och lägga till fler allteftersom arbetsbelastningen ökar. Dessutom behöver inte applikationsutvecklarna lära sig någon ny programmeringsparadigm eftersom operativsystemet hanterar tilldelningen av processer till de olika processorerna transparent och eftersom alla processorer delar på samma minne. SMPs kan därför programmeras i stort sett på samma sätt som enprocessorsystem. En annan fördel är att kommunikationen mellan processorerna är enkel efter som de delar samma minne. Alla processorer kan enkelt, snabbt och effektivt kommunicera genom att skriva till och läsa från det delade minnet.

Flaskhalsen i ett enprocessorsystem är, som vi nämnt tidigare, processorn. Det kommer vi ifrån genom att använda SMP men vi får istället en ny flaskhals, nämligen bussen eftersom vi kopplar flera processorer till samma buss (Inmon et al., 1999). Eftersom bussen har en fast

bandbredd kommer arbetsbelastningen som skapas av processorerna att vid en viss punkt mätta den bandbredden och att lägga till fler processorer kommer då inte att öka prestandan utan tvärtom, ofta kan prestandan minska istället. Processorerna ägnar all sin tid åt att tävla om bussen istället för att utföra användbart arbete. Högre bandbredd kan medföra att fler processorer kan användas, men det kan snabbt bli en dyr lösning.

2.3.5.2.2 Kluster

Skillnaden mellan kluster och parallella system är att kluster består av två eller flera individuella system som är sammankopplade (Silberschatz, Galvin & Gagne, 2003). Om problemet med SMP är att bussen är flaskhalsen, är den naturliga lösningen att erbjuda flera bussar (Inmon et al., 1999). Processorerna och minnet delas upp mellan bussarna, den här kombinationen av en mängd processorer, en minnespool och en buss kallas för en nod. En mängd noder kallas för ett kluster. Alla noder är sammankopplade i ett speciellt höghastighetsnätverk för att de skall kunna kommunicera effektivt och koordinera med varandra. I ett kluster med t.ex. tre noder behöver varje buss bara ta hand om en tredjedel av det totala antalet processorer i systemet och behöver också bara ta hand om minnesförfrågningar som hör till den tredjedelen av minnet. Därför behövs inte en dyrare och snabbare buss.

Definitionen av en nod är identisk med definitionen av en SMP (Inmon et al., 1999). Klusterlösningen på prestandaproblemen säger att om en SMP blir för lite är det bara att ta in ytterligare en SMP. Ett kluster är egentligen inget annat än flera SMP system som är sammankopplade med ett snabbt nätverk som kallas interconnect.

För ett datalager är den största fördelen med kluster att de ger ökad skalbarhet (Inmon et al., 1999). Om ett datalager byggs med SMP och man ser till medeltiden för en datalagerplattforms tillväxt, kommer man mest troligt att nå gränsen för genomströmning inom 18 till 24 månader. Men genom att använda kluster kan man ta in ytterligare en SMP och koppla ihop dem med en interconnect och uppskattningsvis fördubbla den tillgängliga genomströmningen. En annan fördel med att använda kluster är att feltoleransen ökar (Inmon et al., 1999). Eftersom varje nod i ett kluster i grunden är en komplett dator finns det ett element av inbyggd redundans i systemet. Om en nod slutar fungera kan de andra noderna fortsätta arbeta.

Det finns förstås även nackdelar med kluster (Inmon et al., 1999). Varje nod har sin egen lokala minnespool som den kan se direkt men en programmerare kan inte skriva kod som direkt kan skriva till eller läsa från de andra nodernas minnespooler. Dessa separata

minnespooler lägger till komplexitet till systemet. I en SMP kan processorerna kommunicera genom att skriva till och läsa från ett delat minnesområde. Om en nod har 5 processorer men behöver en sjätte för att utföra en speciell uppgift behöver de ta hjälp av en processor från en annan nod. För att processorer i olika noder skall kunna kommunicera måste de explicit skicka meddelanden till varandra över interconnecten och det här sättet att kommunicera är mycket långsammare än att kommunicera med minnet. För att få bra prestanda på en klusterplattform måste applikationen konstrueras så att så lite kommunikation som möjligt sker över interconnecten.

Flaskhalsen i ett klustrat system är interconnecten (Inmon et al., 1999). Den har en fast bandbredd och kan bara stödja en viss mängd kommunikation. Det är viktigt att ha så lite kommunikation som möjligt över interconnecten men ju fler noder som läggs till i klustret desto fortare kommer bandbredden att utnyttjas maximalt. I ett klustrat system finns en kritisk gräns hur många noder som kan finnas utan att prestandan påverkas negativt.

2.3.5.2.3 Massivt parallellt datorsystem

I ett massivt parallellt datorsystem (MPP) har varje processor ett eget operativsystem och eget minne (Computer Swedens språkwebb, 2005). Massiv i detta sammanhang betyder kraftfull, i jämförelse med symmetriska multiprocessorsystem som har begränsad kapacitet.

I ett MPP system delas varje problem upp i separata delar som bearbetas var för sig samtidigt (TechEncyklopedia, 2005). Varje delsystem kommunicerar med de andra över en höghasighetsinterconnect. För att använda MPP effektivt måste problemet eller uppgiften gå att dela upp i delar som kan lösas parallellt.

För många av de största datalagren krävs mer processorkraft och mer I/O bandbredd än vad det går att få ut av de största SMP:erna eller klustrade arkitekturerna (Inmon et al., 1999). För de här datalagerplattformarna krävs ett massivt parallellt datorsystem som egentligen inte är något annat än ett väldigt stort kluster.

Den största fördelen med MPP är möjligheten att uppnå nivåer av skalbarhet som överträffar till och med den största traditionella SMP och/eller klustrade lösningen (Inmon et al., 1999). Eftersom MPP kan konfigureras med ett väldigt stort antal processorer och hantera mycket stora mängder I/O är MPP ofta den arkitektur som väljs för de största datalagerplattformarna. Inmon et al. nämner att flera av världens största finansiella institut, återförsäljare, flygbolag och telefonbolag använder MPP arkitekturen för att få den prestanda de behöver för sina stora datalager som ofta består av flera års information och har tabeller med hundratals miljoner, ibland till och med miljarder, rader.

Nackdelen med MPP arkitekturen är att väntetiden när meddelanden skickas kan vara 100 till 1000 gånger långsammare än väntetiden för SMP meddelanden och det kan vara en stor utmaning att hantera en maskin med många noder (Inmon et al., 1999). Dessutom är prestandan i en MPP maskin väldigt känslig för mängden data som flyttas och kommunikationen över interconnecten när många noder används. Genom att göra interconnecten skalbar kan man med MPP komma förbi flaskhalsen som finns i en vanlig klustrad arkitektur (Inmon et al., 1999). När mer processorkraft tillkommer, tillkommer även mer interconnect-bandbredd. Den verkliga fördelen med MPP är just skalbara interconnects, enligt Inmon et al., och att det teoretiskt sett inte finns några ingående komponenter som kan bli flaskhalsar när systemet skalas upp. Eftersom varje koppling bara är punkt till punkt mellan två noder kan varje koppling byggas ganska billigt, så det här är ett kostnadseffektivt sätt att koppla samman ett väldigt stort antal processorer. Kostnaden med en skalbar interconnect är att längden på kommunikationsvägen mellan två noder blir längre ju fler noder som läggs till vilket ökar väntetiderna (Inmon et al., 1999).

2.3.5.2.4 Nonuniform Memory Access

SMP har begränsningar eftersom systembussen blir en flaskhals och därför skapades lösningar som kluster och MPP som inte delar minne (Inmon et al., 1999). Priset för det får betalas av applikationsutvecklarna som måste hantera flera olika minnesområden och hålla reda på vilket minnesområde som innehåller vilken data och hantera utbytet av data. Det sker genom att explicit skicka meddelanden mellan noder och det är upp till programmeraren att hantera hela flödet av meddelanden som skickas. Istället för att välja mellan en enkel programmeringsmodell med låg skalbarhet eller en mer komplex programmeringsmodell med högre skalbarhet skapades Nonuniform Memory Access (NUMA) som har separata minnen och bussar för att undvika flaskhalsar och inbyggda mekanismer i hårdvaran som transparent hanterar alla minnesområden som om de vore del av en logisk, angränsande minnespool, och på så sätt minska bördan för programmerarna.

Skillnaden mellan NUMA och klustrade system är, enligt Inmon et al. (1999) att NUMA introducerar koncepten lokalt minne (minne som fysiskt finns på den egna noden) och fjärrminne (minne som fysiskt finns på andra noder). Det konceptet finns dock bara på hårdvarunivån, utvecklaren hanterar allt minne som lokalt minne. Allt minne kan kommas åt men det går mycket fortare att komma åt det lokala minnet än det som finns i andra noder.

Fördelen med NUMA-arkitekturen är att den erbjuder större skalbarhet än SMP men fortfarande med samma programmeringsmodell som SMP (Inmon et al., 1999). Teoretiskt sett

finns det inga direkta nackdelar med NUMA, men förhållandet mellan väntetiden på minnesåtkomsten för minne på andra noder och väntetiden till det lokala minnet måste vara så nära 1:1 som möjligt. För att illusionen av att använda ett enhetligt lokalt minne skall vara trovärdig kan inte väntetiderna vara för hög.

De största fördelarna med att använda någon av de beskrivna skalbara plattformarna (SMP, kluster, MMP, NUMA) är att de möjliggör byggandet av datalagerplattformar med prestandanivåer som kan fortsätta att möta slutanvändarnas krav i framtiden (Inmon et al., 1999). Om datalagermiljön fördubblas i storlek, och hårdvaruarkitekturen inte är skalbar, kommer prestandan snabbt att sjunka. Men med en skalbar hårdvaruplattform kan extra beräkningskapacitet läggas till för att leva upp till prestandakraven.

2.3.5.3 Skalbara databashanterare

En skalbar högprestandalösning är endast användbar för datalager om databaserna kan utnyttja hårdvarans skalbarhet (Inmon et al., 1999). DBMS:erna måste kunna använda flera processorer när datan skall bearbetas. För traditionella relationsdatabaser betyder det att de måste kunna behandla frågor parallellt för att datalagerplattformen skall uppnå skalbarhet.

De tre typerna av parallellism som förklarats tidigare (se avsnitt 2.3.5.1) kan utnyttjas tillsammans för att vara säker på att den skalbara hårdvarulösningen används fullt ut och för att få snabba svarstider även vid stora frågor (Inmon et al., 1999). Med ett stort antal samtidiga användare kan funktionell parallellism användas för att tilldela olika användares frågor till olika uppsättningar processorer. Olika uppsättningar processorer kan arbeta på olika frågor samtidigt. Teoretiskt sett kan antalet processorer som en fråga tilldelas bestämmas i runtime. Vid hög arbetsbelastning kan en fråga tilldelas till fåtal processorer medan den kan tilldelas till fler processorer när arbetsbelastningen är lägre. Ett annat alternativ är att prioritera frågorna och ge frågor med hög prioritet tillgång till flera processorer medan lågprioriterade frågor får tillgång till ett färre antal. Men, skriver Inmon et al. (1999), databassystemen är väldigt begränsade i sina möjligheter att automatiskt och dynamiskt tilldela olika antal processorer till en fråga baserat på arbetsbelastning eller prioritet. I de flesta system bestäms antalet processorer som används efter hur tabellerna ligger på disken eller efter vilka inställningar datalageradministratören har gjort.

När frågor har tilldelats ett antal processorer används konceptet partitionering och pipelining för att både snabba upp svarstiden på frågan och för att öka hela systemets genomströmning av frågor (Inmon et al., 1999). Istället för att en processor behandlar en fråga kan flera processorer användas och arbetet delas upp mellan dem. Olika processorer kan

utföra olika delar av arbetet, t.ex. scan, join och sort, och samtidigt kan varje steg i arbetet delas upp mellan flera processorer. Inmon et al. ger ett exempel för att illustrera det med två tabeller där scan, join och sort skall utföras. Varje steg av arbetet tar fyra minuter att utföra i ett enprocessorsystem och hela arbetet tar 12 minuter. Om istället 12 processorer används så kan 4 processorer användas till scanningen av de två tabellerna, 4 till joinprocessen och de sista 4 till sorteringen. I scanningsfasen kan två av processorerna dela på arbetet med att scanna den första tabellen och de andra två får då dela på att scanna den andra tabellen. Nu tar den delen av arbetet en minut per processor istället för fyra minuter som det tog en ensam processor. På det här sättet kan tiden det tar att scanna tabellerna reduceras till en fjärdedel av tiden jämfört med ett enprocessorsystem. Samma uppdelning och tidsbesparing gäller för operationerna join och sort. Det förekommer dock en viss overheadkostnad med det här sättet att arbeta, t.ex. vid partitioneringen av arbetsmängden för varje steg och vid ihopslagningen av de sorterade delmängderna till en mängd innan de returneras till användaren (Inmon et al., 1999). Men om frågan är tillräckligt stor är den extra overheadtiden försumbar. En fråga som med en processor tog 12 minuter går nu att genomföra på 3 minuter. Men med tanke på att 12 processorer har använts istället för 1 så är en 4 gånger snabbare svarstid inte så imponerande. Den stora vinsten kommer när partitionering och pipelining används tillsammans. När processorerna som hanterat scanningen är färdiga med frågan lämnas resultatet över till join-processorerna och de kan då börja scanna tabellerna för nästa fråga (Inmon et al., 1999). Samma sak gäller för alla processorer, när de är klara med sin exekvering fortsätter de med nästa jobb i kön. Efter 12 minuter har nu 12 frågor färdigställts och parallella frågor har nu 12 gånger högre output än sekventiell bearbetning. Varje fråga går 4 gånger snabbare att exekvera och genomströmningen har ökat med en faktor 12. Ovanstående siffrorna är bara teoretiska exempel tagna direkt från exemplet och ingen hänsyn har tagits till hårdvaruoverhead.

Den beskrivna användningen av parallella frågor gäller direkt för SMP hårdvaruarkitektur (Inmon et al., 1999). Den arkitekturen kallas ”shared-everything hardware architecture” eftersom alla hårdvarukomponenter fysiskt delas av alla processorer. Det gör det enklare att dela arbetsbelastningen av en parallell fråga mellan flera processorer eftersom varje processor direkt kan komma åt datan på vilken disk som helst. Samma sak gäller för ”shared-disk hardware architecture” som används i vissa MPP-, kluster- och NUMA-plattformar. De lösningarna har fysiskt separerade minnespooler men alla diskar delas lika mellan alla noder. De flesta kluster- och MPP-implementationer är dock ”shared-nothing hardware architecture” och processorer i en nod kan då inte direkt komma åt data på en disk som är kopplad till en

annan nod. Beskrivningen av parallella frågor gäller då inte direkt för de lösningarna. Själva konceptet med parallella frågor ändras inte med den senare arkitekturtypen, dvs. funktionell parallellism, partitionering och pipelining gäller fortfarande, men det är nya prestandafrågor att ta hänsyn till eftersom det blir mer overheadkostnader genom meddelanden som skickas på interconnecten när data som ligger på en annan nod skall kommas åt.

2.3.5.4 Databasarkitekturer

Shared-disk database architecture och shared-nothing database architecture skall inte blandas ihop med shared-disk *hardware* architecture och shared-nothing *hardware* architecture (Inmon et al., 1999). Nu handlar det om arkitekturen för databasservrarna och inte hårdvaran. Både shared-disk och shared-nothing *database* architecture kan köras på shared-nothing *hardware* architecture. Shared-disk database architecture innebär att på ett konceptuellt plan tillåts vilken processor som helst att efterfråga data som finns varsomhelst i databasen. Om efterfrågad data finns på en annan nods disk skickas meddelanden automatiskt i bakgrunden för att skicka datan till processorn för lokal bearbetning. Med den här arkitekturen känner inte DBMS:en till att en viss uppsättning diskar tillhör en specifik nod utan ser alla diskar som logiskt delade även om de inte är det fysiskt. Meddelandehanteringen i bakgrunden ansvarar för att tillåta databasen att behandla alla diskar som åtkomliga för alla processorer. Den teoretiska fördelen med den här databasarkitekturen är ökad flexibilitet i användandet av systemets resurser (Inmon et al., 1999). DBMS:en kan dynamiskt välja hur många processorer och hur många noder som skall involveras i en fråga, men även välja vilka processorer och noder som skall användas. Det förbättrar möjligheten att genomföra lastbalansering. Den teoretiska nackdelen är att flexibiliteten medför extra overhead vid kommunikationen mellan noderna när data skickas som meddelanden över interconnecten.

Shared-nothing database architecture fokuserar på att minimera meddelandeoverheaden genom att dela upp varje tabell med hjälp av någon partitioneringsnyckel som administratören väljer, t.ex. efternamn, och tilldela varje partition till en nod (Inmon et al., 1999). Då ”äger” varje nod en del av tabellen och om en processor behöver data som finns på en annan nod skickar inte databasen datan till processorn för lokal bearbetning utan istället skickas frågan till den andra noden, som exekverar den och returnerar svaret. Den teoretiska fördelen är att kommunikationsoverheaden mellan noderna minskas och interconnectbandbredden kan användas mer effektivt eftersom bara frågan och svaret skickas istället för all data (Inmon et al., 1999). En teoretisk nackdel är minskad flexibilitet eftersom en fråga som involverar en speciell databaspartition bara kan bearbetas av processorerna i den nod som äger partitionen.

En annan nackdel består av två biprodukter av strategin att partitionera databastabellerna och framhäver hur kritiskt det är att välja rätt partitioneringsnyckel. Den första biprodukten är att om en fråga inte använder tabellens partitioneringsnyckel måste den skickas till alla noder som hanterar en del av den tabellen, istället för att bara skicka frågan till de noder som hanterar datan som efterfrågas, som är fallet om partitioneringsnyckeln används. Om t.ex. nyckeln är första bokstaven i efternamnet och frågan är att hitta en kund som heter Svensson så behöver bara de noder som hanterar S involveras i frågan. Om frågan istället gäller ett personnummer kan inte DBMS:en på någon snabbt sätt avgöra på vilken nod det personnumret finns och frågan måste skickas till alla noder som kan ha den datan. Nackdelen med det är att overheaden ökar när frågan måste skickas till flera noder. Den andra biprodukten är att svarstiden för frågor är väldigt känslig för snedfördelning av data som uppstår när en dålig partitioneringsnyckel har valts. Det kan leda till att en partition har oproportionerligt mycket data. Problem uppstår t.ex. när en tabellscanning skall genomföras eftersom det kräver att varje nod söker igenom sin egen partition och frågan inte blir färdig innan alla noder har scannat sina partitioner av tabellen. Om en nod har oproportionerligt mycket data kommer den noden att dra ner hastigheten för frågan eftersom det inte är möjligt för andra noder att hjälpa till att avlasta en del av scanningsarbetet.

2.3.5.5 Slutanvändarverktyg

För att datalagermiljön skall vara verkligt skalbar måste även slutanvändarverktygen hantera skalbarhetsfrågor (Inmon et al., 1999). Frågeoptimerare är inte perfekta och det finns en lukrativ marknad för begåvade applikationsutvecklare som vet hur datalagerfrågor kan prestandaoptimeras. Dessutom har olika databaser olika funktionalitet för att utnyttja parallella frågor vilket kräver olika optimeringstekniker. Tillräcklig prestanda kan ofta uppnås genom att helt enkelt skicka generella SQL-frågor till en av databaserna men bättre prestanda kan uppnås om verktygen är anpassade för den specifika databasfunktionaliteten som finns i datalagerplattformen och kan skräddarsy frågan därefter.

2.3.5.6 Sammanfattningsvis

Det är inte bara hårdvaran och databasservern som måste vara skalbar (Inmon et al., 1999). Om någon komponent eller del av processen inte är skalbar kommer datalagerplattformen till slut att stöta på flaskhalsar när bandbreddgränsen för den minst skalbara komponenten eller processen nås. Därför är det nödvändigt att varje komponent analyseras för att säkerställa att

den är tillräckligt skalbar för att möta framtida krav. Tillräckligt skalbar för att göra det möjligt att bibehålla de prestandanivåer som krävs när datalagret växer snabbt och oförutsett.

3 Empiri

I det här kapitlet beskriver vi hur en befintlig produkt fungerar utifrån intervjuer med tillverkaren och redovisar en intervju med en användare av den produkten. För att få en bredare förståelse för hur datalager används och de krav som ställs på dem så har vi intervjuat ytterligare en leverantör av datalager för att få deras synpunkter på ämnet samt en forskare som har intervjuat ett stort antal storföretag om deras användning av beslutsstödssystem.

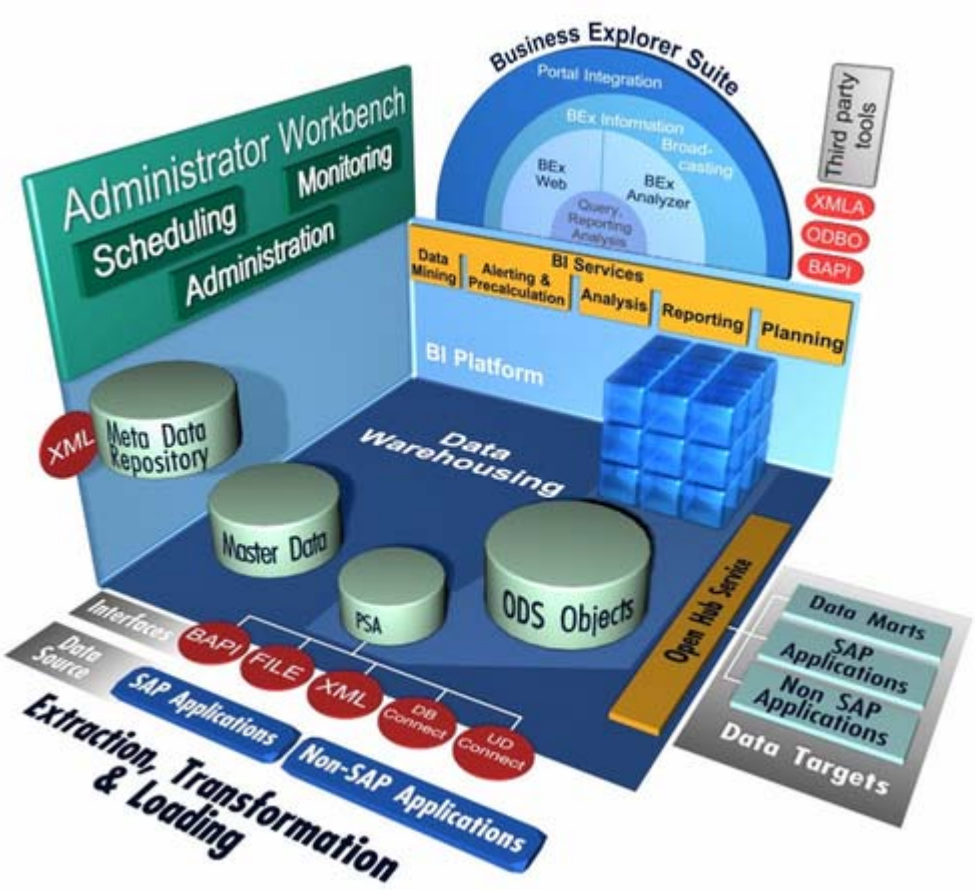
3.1 Beskrivning av en befintlig produkt för datalager

För att undersöka hur en befintlig lösning för datalager fungerar har vi kontaktat SAP som är en av världens största leverantörer av mjukvara, framförallt affärssystem (system för att bland annat hantera företagsadministration). SAP har dessutom programvara för datalager och beslutsstöd. Företaget grundades 1972 och har 12 miljoner användare och 91 500 installationer världen över. De har mer än 32 000 anställda i drygt 50 länder. Huvudkontoret ligger i Walldorf, Tyskland. 2004 hade företaget en omsättning på 7,5 miljarder euro.

Vi börjar med att beskriva SAP:s produkt för datalager och därefter redogör vi för våra intervjuer med Christer Ingemarsson, Nordic Business Development Manager, Business Intelligence & Analytics på SAP. Vi redovisar inte varje intervju för sig utan tar upp ett urval av de viktigaste aspekterna som har diskuterats vid våra besök på SAP:s användarseminarier och telefonintervjuer med Christer Ingemarsson. Vid vilken intervju olika aspekter behandlats ser vi inte som relevant för uppsatsen.

3.1.1 SAP - Business Information Warehouse

SAP:s programvara för att bygga och underhålla ett datalager heter Business Information Warehouse (BW) och innehåller bland annat verktyg för ETL, att göra analyser, att skapa och dela med sig av rapporter samt exportera data till externa system, se Figur 7. I slutet av 2003 fanns det mer än 6000 installationer av BW på marknaden.



Figur 7: SAP Business Intelligence architecture (SAP)

3.1.1.1 Grundläggande komponenter i BW

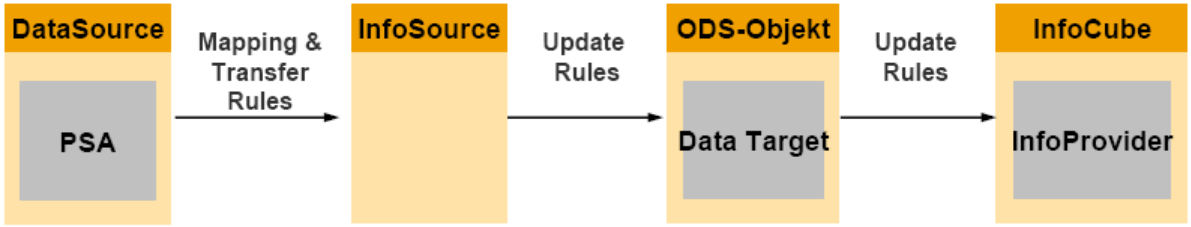
I det här avsnittet beskrivs ett antal grundläggande och viktiga komponenter som tillsammans utgör BW:s arkitektur.

Masterdata: Data som sällan ändras kallas för masterdata, det kan exempelvis vara kundadresser eller en produktspecifikation. Det är ofta beskrivande data i textform och läggs i dimensionerna i ett starschema, se Figur 10 och Figur 11.

InfoObject: BW baseras på en grundläggande byggsten som kallas för InfoObject. InfoObject innehåller data om t.ex. kunder och ordrar. De innehåller dessutom metadata som beskriver dess innehåll, t.ex. källsystem och dess ålder. Det finns i huvudsak tre typer av metadata:

- Teknisk metadata som beskriver tekniska egenskaper, t.ex. datatyper och attributstorlek
- Metadata om användarrättigheter

- Metadata om verksamhetsdefinitioner för att skapa en enhetlig förståelse av begrepp som är viktiga för verksamheten, t.ex. olika nyckeltal



Figur 8: SAP BW Information modell (SAP, 2004)

DataSource: DataSource-objektet gör det möjligt att ansluta olika typer av källsystem och andra datakällor till BW. Det innehåller definitioner av källdata som läses in i tabellform. En DataSource har något som kallas för extraherare kopplad till sig. Det finns extraherare som är fördefinierade för att läsa från specifika datakällor och leverera data i ett bestämt format, men det finns också generella extraherare för att läsa från en mängd olika typer av datakällor, exempelvis flatfiler och XML. När stora datavolymer skall hanteras finns det funktionalitet för vad som kallas deltaöverföring, då kopieras endast poster från källsystemen som ändrats sedan förra läsningen.

Persistent Staging Area (PSA): När data läses in i BW lagras det fysiskt i PSA-objekt som i praktiken är i tabellform. Här lagras den efterfrågade datan i oförändrad form enligt vad som är definierat i dess DataSource.

InfoSource: InfoObject som har en logisk relation ur ett verksamhetsperspektiv grupperas till en InfoSource. En InfoSource, och dess ingående InfoObject, kan fyllas med data från både de egna källsystemen och externa källor. De består av både transaktionsdata och masterdata.

ODS-objekt: Både datalagernivån och ODS:en består av objekt som kallas för ODS-objekt (namnet kan vara förvirrande och i framtida versioner kallas de istället för data store object). De består av en mängd tabeller med normaliserad data från en eller flera InfoSourses. Data från ett ODS-objekt kan laddas in i andra ODS-objekt eller i InfoCubes. Innehållet i ett ODS-objekt kan analyseras med ett analysverktyg som heter SAP BW Business Explorer (BEx).

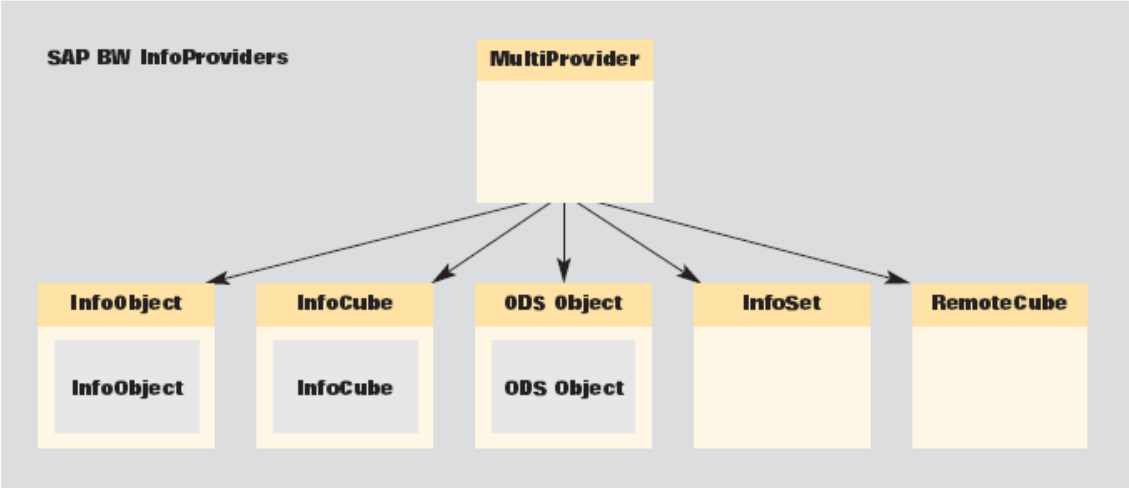
InfoCubes: InfoCubes innehåller och organiserar data i en multidimensionell struktur. Även InfoCubes är åtkomliga med SAP BW Business Explorer och används för OLAP.

Det går att kombinera ovanstående komponenter på andra sätt än det som illustreras i Figur 8, t.ex. kan data laddas direkt in i en InfoCube och en InfoSource kan kopplas till flera DataSources. Vid analyser är det möjligt att göra en drill down från ett objekt till ett annat, t.ex. från en InfoCube till ett ODS-objekt.

En DataSource kopplas till en InfoSource med de datatransformeringsregler som finns i BW. Reglerna mappar fälten i en DataSource mot InfoObjekten som en InfoSource består av, se Figur 9. Därefter hanterar BW:s uppdateringsregler dataflödet från InfoSources till ODS-objekt och InfoCubes.

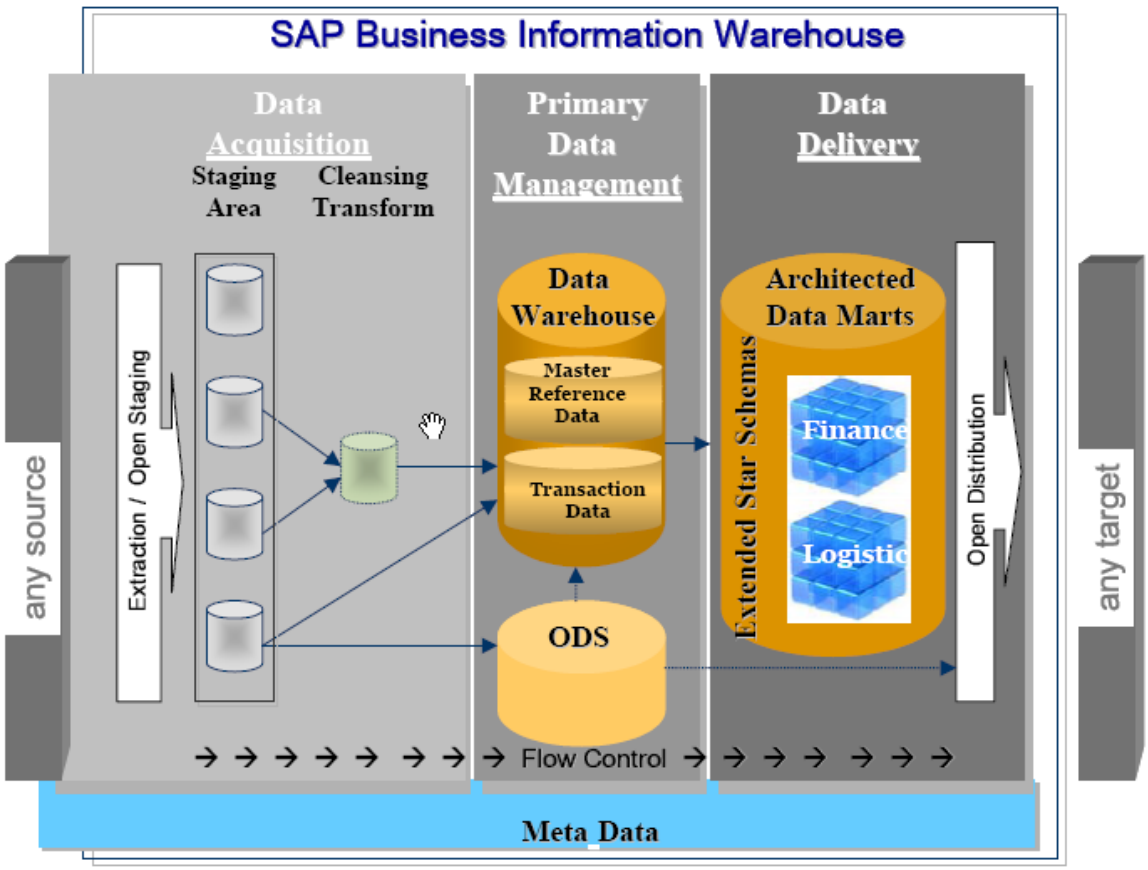
Ofta behöver metadata läggas till datan som lagras i PSA-objekt. Metadatan läggs till både när InfoObjekt skapas och när de grupperas för att skapa en InfoSource.

I BW kallas objekt som kan analyseras för InfoProviders, se Figur 9. Det finns två typer av InfoProviders, en innehåller fysisk data medan den andra inte gör det. InfoObjekt, InfoCubes och ODS-objekt innehåller data fysiskt. MultiProviders används för att kombinera data från flera InfoProviders och kan användas för analyser och rapporter.



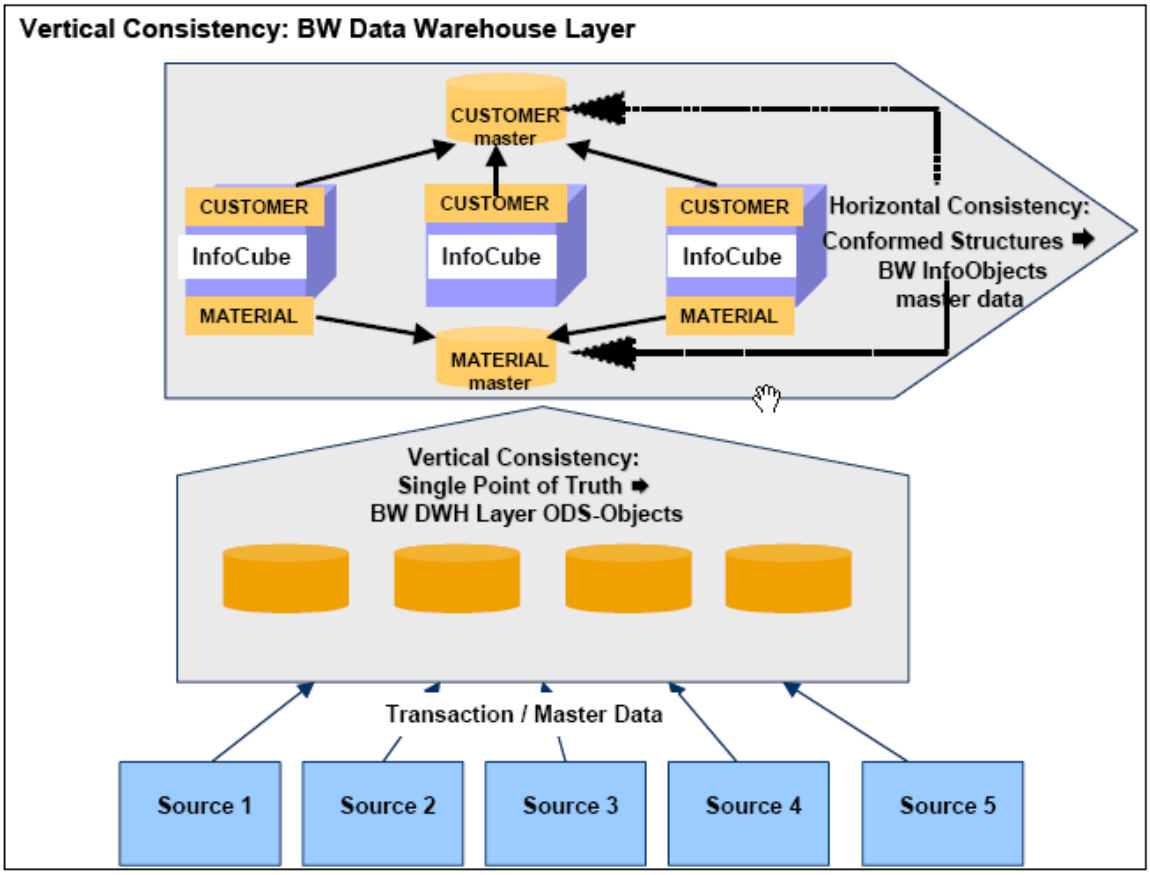
Figur 9: SAP BW InfoProviders (SAP, 2002)

SAP använder samma lagerindelade arkitektur som Inmon (2002) beskriver och den implementeras med ovanstående objekt och strukturer, se Figur 10.



Figur 10: SAP BW lagerindelad arkitektur (SAP, 2002)

Till "staging area" kommer tupler från källsystemen. I datalagret sparas normaliserad, integrerad och detaljerad data med en tidsstämpel. I data mart-nivån finns integrerad, aggregerad och applikationsspecifik data. I ODS:en sparas integrerad, detaljerad och applikationsspecifik data nära realtid.



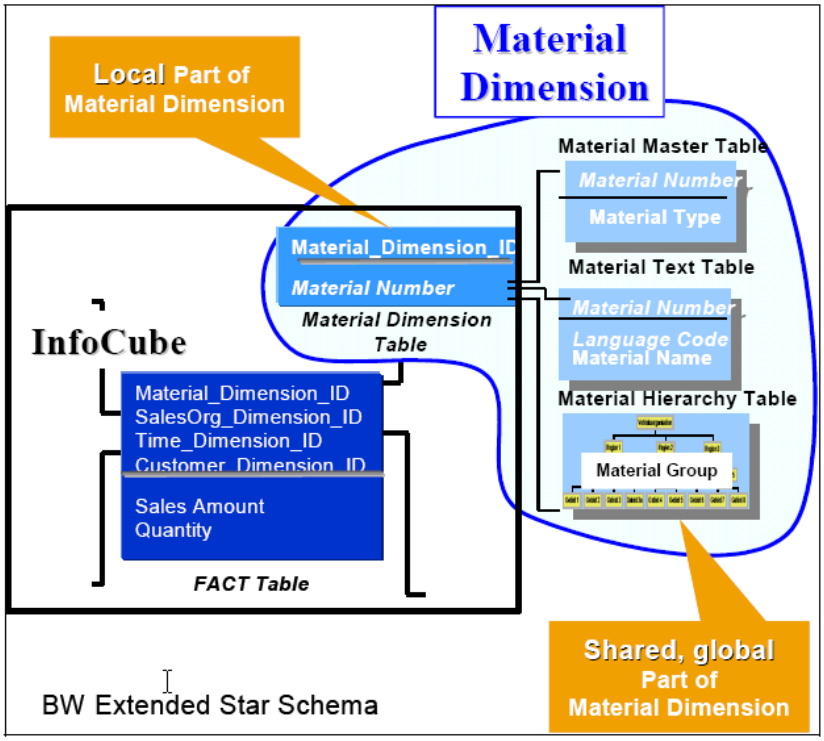
Figur 11: SAP BW Datakonsistens (SAP, 2002)

För att se till att användarna ser data med samma värde bevarar BW datakonsistensen dels genom att integrera datan i datalagret, men också genom att kuberna delar på masterdata, vilket illustreras i Figur 11.

3.1.1.2 BW Architected Data Mart Layer

Data marts baserade på InfoCubes möjliggör flexibel generering av frågor och navigering till integrerad, detaljerad eller aggregerad data. För att jämförelser skall kunna göras kan en egendefinierad mängd med historisk data finnas.

InfoCubes är multidimensionella datastrukturer (kallas även starscheman) som organiserar data på ett sätt så att nyckeltal (eller fakta eller mått) är associerade med egenskaper och deras attribut som bildar de så kallade dimensionerna (BW InfoCube-dimensioner och masterdata). För att öka kapaciteten i ett InfoCube-schema delas dimensionerna upp i lokala och globala operativa delar, se Figur 12. De är globala eftersom dimensionsdelarna bara sparas en gång och efterfrågas av flera olika InfoCubes.



Figur 12: Utökat starschema i BW (SAP, 2003)

På det här sättet stödjer InfoCubes konceptet med ”conformed dimensions” som kan ses som ett sorts lim mellan InfoCubes. Conformed dimensions stödjer i huvudsak horisontell konsistens i data mart-lagret (se Figur 11). Det är en del av BW-konceptet och hjälper till att förhindra förekomsten av inkonsistent data. Termen som används i BW för conformed dimensions är masterdata. Genom att använda surrogatnycklar i InfoCuben och på grund av konceptet med master data, är det möjligt att samtidigt mappa flera vyer till nyckeltal i en InfoCube.

Frågan om hur datalagret skall organiseras skapar ofta debatt, åsikterna varierar från 3NF till helt denormaliserade starscheman. SAP vill inte bestämt ta ställning i frågan men föreslår att ett pragmatiskt angreppssätt används. Eftersom datalagernivån utgör grunden för data marts bör data sparas så att det lätt kan överföras till InfoCubes och conformed dimensions (masterdata). Join-operationer på normaliserade datalager-objekt på väg till data marts bör undvikas.

3.1.1.3 Multidimensionella modeller och aggregat

Multidimensionella modeller erbjuder de datavyer som behövs för att utföra OLAP-analyser som samtidigt tittar på flera dimensioner av data, som t.ex. tid, plats och produkt.

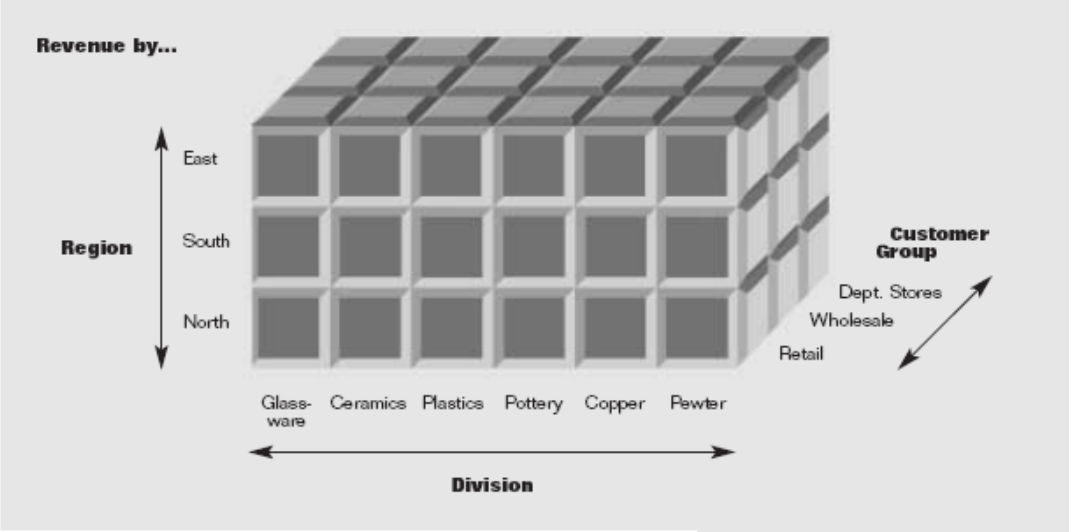
Med multidimensionella analyser kan komplexa affärsfrågor besvaras som t.ex.; ”Vilka intäkter har vi, per region och kontor i varje region, year-to-date, och hur är de i jämförelse med samma period förra året?” eller ”Vilka produkter i avdelning A står tillsammans för 80% av den totala lönsamheten för den avdelningen, baserad på försäljning och direkta och indirekta associerade kostnader för de tre senaste åren?”. OLAP-motorn i BW exekverar de databasfrågor som formuleras utifrån de här affärsfrågorna.

I BW utgörs de multidimensionella modellerna av InfoCubes. En InfoCube innehåller två typer av data som används vid analyser:

- Nyckeltal, som försäljningsinkomster, fasta kostnader, sålda kvantiteter eller antal anställda
- Egenskaper, som produkt, kundtyp, fiskalår, period eller region. Egenskaper används för att skapa utvärderingsgrupper för analys.

De underliggande InfoObjecten som en InfoCube består av är kategoriserade efter de här två typerna av data. Med det menas att ett givet InfoObject representerar antingen ett nyckeltal eller en egenskap. En tredje typ av InfoObject, attribut, innehåller metadata som beskriver andra InfoObject.

De två typerna av databärande InfoObject kombineras för att utföra analyser över flera dimensioner. I Figur 13 är inkomst för produkter som glasföremål (glassware) och keramik (ceramics) exempel på nyckeltal. Regioner (North, South och East) och kundgrupper (Retail, Wholesale, Dept. Stores) är exempel på egenskaper. En analytiker som använder den här modellen kan till exempel undersöka ”inkomsten för lergods som sålts i detaljhandel i region norr.”



Figur 13: En multidimensionell modell (SAP, 2002)

En InfoCube är representerad i en RDBMS som en mängd relationstabeller arrangerade enligt starscheman där egenskaper är summerade i dimensionstabeller.

3.1.1.4 Starschema i BW

I BW används ett utökat starschema, extended starscheme, som bygger ut ett vanligt starschema genom att lagra master data om attribut, hierarkier och text i separata tabeller som delas mellan InfoCuber. Det här minskar dataredundans eftersom masterdata bara lagras en gång och sen används av olika InfoCuber. På det här sättet kan hierarkier lätt byggas för att återspegla affärsstrukturen och förändringar i data kan hanteras direkt och konsekvent. En annan fördel med ett utökat starschema är att ändringar i InfoCuben automatiskt även gäller för motsvarande aggregat. Den här viktiga funktionen ser till att InfoCubes och aggregat är synkroniserade trots att affärsprocesser ändras ofta.

Eftersom aggregat är förberäknade summeringar av data förbättrar de svarstiden för frågor genom att svaret finns färdigt innan frågan ställs. Ett aggregat lagrar en summering av en InfoCubes datamängd. Det betyder att den kortar ner InfoCubens ursprungliga faktatabell. Den summerade tabellen möjliggör snabb åtkomst till en InfoCubes data, vilket förbättrar prestandan för frågor. Aggregat hålls uppdaterade av systemet, när en underliggande InfoCube ändras modifieras aggregatet automatiskt för att reflektera den nya datan. I BW skapas förslag för att skapa optimala aggregat och systemadministratören kan välja de aggregaten skall skapas. InfoCubes lagrade i BW är baserade på relationsdatalagring och

används för ROLAP. De associerade aggregaten kallas ROLAP-aggregat. Om BW körs på en Microsoft SQL Server kan företag också bygga aggregat som använder multidimensionell datalagring för att skapa MOLAP-aggregat. I det fallet är data lagrad i en multidimensionell struktur på databasservern och åtkomsten till den sker genom OLAP processorn i BW. Flera variabler påverkar valet mellan de två typerna av aggregat: mängden data i InfoCuben, datans detaljnivå, krav på detaljerade analyser och antal standardfrågor. Det finns ingen generell OLAP-arkitektur som erbjuder den mest effektiva lösningen för varje organisation.

BW tillhandahåller ett integrerat angreppssätt för datalagring och överföring av data mellan nivåerna. Det finns två underliggande principer: minska antalet gånger data måste överförs och tillåta en given mängd data att användas på en mängd olika sätt. Det förbättrar dataprecision, minskar latens och förbättrar systemets hanterbarhet. Det möjliggör också datatransparens och förbättrar möjligheten att utföra drill down.

3.1.1.5 OLAP

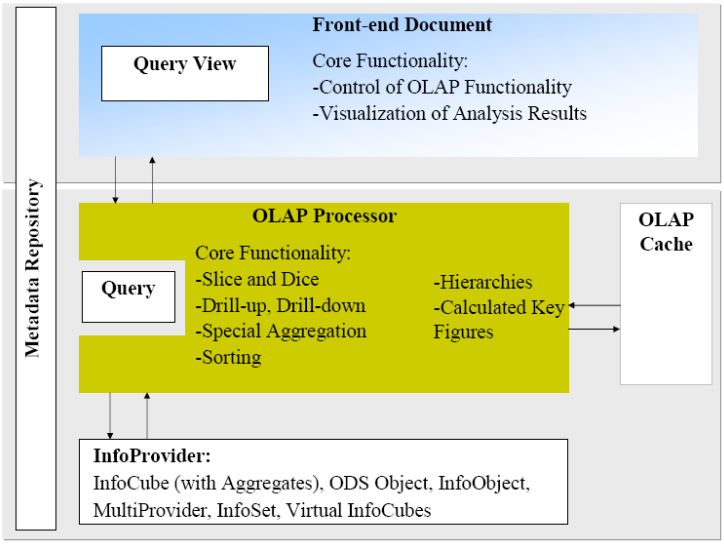
SAP Business Information Warehouse använder OLAP-teknologi för att analysera data i datalagret. Enligt SAP är det OLAP som gör att BW kan ses som ett beslutsstödssystem. OLAP låter beslutsfattare snabbt och interaktivt analysera den multidimensionellt modellerade datan som är lämplig ur affärssynpunkt.

3.1.1.5.1 OLAP-processorn

OLAP-processorn är en del av BW-servern och ligger mellan slutanvändaren och databasen. Den gör den multidimensionellt formaterade datan tillgänglig för både BW:s användargränsnitt (front-end) och för tredjepartsapplikationer. OLAP-processorn är optimerad för att göra analyser och ta fram rapporter från väldigt stora datamängder. Slutanvändare kan ad hoc efterfråga individuella vyer av affärsrelevant data.

Frågor är grunden för all analys i BW. För att formellt definiera hur multidimensionell data efterfrågas bestäms strukturen, filtret som påverkar strukturen och navigeringsutrymmet av frågan. BW har en mängd analys- och navigeringsfunktioner för formatering och utvärdering av ett företags data. De funktionerna gör det möjligt för användarna att formulera individuella förfrågningar med utgångspunkt ur de multidimensionellt modellerade datamängderna, InfoProviders. Därför kan användarna se och utvärdera datan från olika perspektiv under runtime. All funktionalitet för att hämta, bearbeta och formatera data finns i OLAP-processorn.

Se Figur 14 för OLAP processorns status och uppgifter i databehandlingsprocessen när en multidimensionell analys utförs.



Figur 14: OLAP-processorn (SAP, 2004)

Några av de OLAP-funktioner och tjänster som är implementerade i BW är:

Funktioner	Användningsområden
• Navigering	• Drilldown till egenskaper. Ta bort element (dice) • Utöka (drill down) och gömma (drill up) hierarkinoder • Utbyte av drill down element (swap)
• Filter	• Begränsa (slice) egenskaper till val
• Aggregation	• Standardaggregation • Undantagssggregation • Lokala aggregat eller lokala beräkningar
• Structuring	• Hierarkisk tilldelning av egenskapsvärden med drill down för mer än ett element

3.1.1.5.2 Aggregat

För att förbättra prestandan när frågor exekveras går det i BW att spara en InfoCubes datamängd i en databas i komprimerad form redundant och beständigt i ett aggregat i databasen. Aggregat gör det möjligt att snabbt komma åt data i InfoCubes för rapportering.

BW har stöd för två olika fysiska lagringskoncept:

- ROLAP som lagrar multidimensionell data i en relationsdatabas
- MOLAP som är ett aggregat av en InfoCube, vars data är fysiskt lagrat i en MOLAP store

Det är inte möjligt att använda både MOLAP- och ROLAP-aggregat samtidigt för en InfoCube, men en MultiProvider kan hämta data från flera InfoCuber som använder ROLAP- eller MOLAP-aggregat. Om MOLAP- eller ROLAP-aggregat skall användas för en InfoCube beror bland annat på den databas som används eftersom olika databaser stödjer olika funktioner.

Aggregat som skall vara tillgängliga för rapporter måste aktiveras i förväg och fyllas med data. Vid navigering är de olika resultaten konsistenta. Slutanvändare märker inte om aggregat används eftersom OLAP processorn finner de optimala aggregaten för en förfrågan om de behövs. Ett aggregat byggs utifrån egenskaper och navigeringsattribut av en InfoCube. Det kan innehålla tidsberoende komponeneter (attribut eller hierarkier). De kan grupperas efter alla värden av en egenskap eller navigeringsattribut, eller efter noder i en speciell nivå av en egenskaphierarki. Aggregat kan också filtreras utifrån individuella värden.

3.1.1.5.3 ROLAP-aggregat

Ett ROLAP-aggregat är ett aggregat av en InfoCube som fysiskt är sparad i en relationsdatabas. Det går att definiera ett godtyckligt antal aggregat för en InfoCube efter behov. För att skapa ett aggregat kan systemet generera och automatiskt optimera förslag baserade på BEx-frågor som är skapade för den valda InfoCuben, eller baserad på data samlad i BW Statistics. Därefter kan de föreslagna aggregaten redigeras individuellt.

Om ny data laddas in i den underliggande InfoCuben är de inte tillgängliga direkt med ett aggregat i rapporter. För att aggregatet med den nya datan skall vara tillgänglig måste den laddas in i aggregattabellerna vid en vald tidpunkt. När det är gjort är den nya datan tillgänglig för roll up i rapporter. Om hierarkier av eller attribut på egenskaper ändras efter att masterdatan har laddats kan användaren manuellt starta anpassningen av aggregaten som har påverkats av förändringen. BW har också funktionalitet för att ge en översikt av statusen på alla befintliga aggregat.

3.1.1.5.4 MOLAP-aggregat

Ett MOLAP-aggregat är ett aggregat av en InfoCube, vars data är fysiskt lagrad i en MOLAP store. MOLAP store i BW använder Microsoft Analysis Services och representerar

därför en plattformspecifik metod för att optimera frågeprestanda. MOLAP i BW är bara tillgänglig för Microsoft SQL Server. De InfoCubes som MOLAP aggregationer skapas för har vissa begränsningar. De kan bara innehålla ett visst antal egenskaper, navigeringsattribut och hierarkier, dessutom kan de inte använda vissa BW objekt.

Strukturen av MOLAP-aggregat för en InfoCube består av två nivåer till skillnad mot den direkta härledningen av många ROLAP-aggregat. I det första steget härleds exakt en MOLAP-kub, MOLAP-aggregatet. Från det kan så många MOLAP-aggregat som behövs härledas.

3.1.2 Redogörelse för intervjuerna med Christer Ingemarsson, SAP

I BW används en variant av starschemat, som SAP kallar extended starschema (se Figur 12). Ingemarsson säger att det utökade starschemat ger större flexibilitet jämfört med om masterdata ligger i dimensionstabeller. Om attribut istället ligger utanför kan de uppdateras på ett ställe och kuberna har automatiskt tillgång till rätt data. Det är en dynamisk lösning men den kräver däremot en extra läsning eftersom en join behöver utföras för varje tabell med masterdata. När gemensam masterdata används till kuberna minskas underhållet men av prestandaskäl är det bättre att inte använda det utökade startschemat. BW stödjer båda alternativen och kunderna kan själva välja mellan prestanda och flexibilitet.

Det finns ingen fysiskt lagrad kub i BW utan OLAP processen kopplar samman dem i runtime. Fysiskt är datan lagrad i relationstabeller och det finns inte heller några fysiskt lagrade summeringar i BW. Aggregat är mindre versioner av kuben för att snabba upp åtkomsten av datan. En nackdel med att använda många aggregat är att det påverkar omladdningsprestandan och att de måste indexeras om.

Angående Dates kritik mot dimensionsmodellen säger Ingemarsson att SAP inte riktigt har relationsdatabassynsättet utan de utgår mer från kundernas behov. De har inte ett teoretiskt helhetsperspektiv enligt Dates teorier, utan försöker se det pragmatiskt och lösa praktiska problem åt sina kunder. BW stöds på relationsdatabaser i botten men har egen funktionalitet för att hantera dimensioner. Hur prestandan ökar när dimensioner används istället för relationer är enligt Ingemarsson beroende av hur stora dimensionsmodellerna blir. SAP har rekommendationer för hur stora dimensionerna bör vara, men Ingemarsson hade inga siffror att ge oss vid tidpunkten för intervjun.

Ingemarsson anser att det är viktigt att ha en sammanhängande logisk datamodell innan kuber och data marts designas och säger att det är något som SAP blir allt bättre på. BW har stöd för en datalagerarkitektur med normaliserade strukturer. Ett datalager byggs först för att

sedan utökas med kuber som modelleras utifrån datalagret. Kuber ger en viss vy av datan men de är inte dynamiska. Ingemarsson säger att utvecklingen går mer mot att betona den logiska modellen, det är viktigare än kuberna eftersom kuberna inte ger någon samsyn. Arkitekturen i BW ligger närmare Inmons än Kimballs, dvs. SAP anser att det finns stora fördelar med att ha ett datalager som data marts byggs utifrån, men Ingemarsson säger att det inte finns några tekniska begränsningar för att bygga datalagret som en enda stor kub eller för att bygga datalagret enligt Kimballs synsätt.

SAP betonar vikten av att ha ett datalager för möjligheten att lägga till nyckeltal i framtida rapporter och för användningen av historiska data. Med enbart kuber är kundens historiska data låst, men med ett datalager går det att modellera om kuberna hursomhelst utifrån datalagret. Det behövs ett flexibelt datalager som stödjer många användningsområden eftersom det skall vara lätt att skapa nya kuber. Användare kan ställa frågor både direkt mot datalagret eller mot en kombination av källor via multiproviders.

Ingemarsson säger att data i ett datalager kan vara hur detaljerad som helst och trenden går mot att mer detaljerad data lagras. Men detaljnivån kan behöva begränsas eftersom det fort blir stora datavolymer som påverkar prestandan.

Det är InfoObjecten som möjliggör ett historiskt perspektiv på innehållet i datalagret och de kan jämföras med Inmons koncept snapshot. All data som laddas in till datalagret laddas via InfoObject som schemaläggs. Varje sådan laddning får ett id i kuberna eller ODS-objekten som den laddas till. Historisk data tas bort genom att InfoObject tas bort.

Enligt Ingemarsson går det att köra BW i samma databashanterare som operativa system använder men det är lämpligt att datalagret körs separerat. Det finns inga tekniska begränsningar med att göra så men enda fördelen som Ingemarsson ser är kostnadsbesparingen, det är billigare att ha ett system. Två olika databaser bör användas för att minska belastningen på systemen.

Ingemarsson säger att det är väldigt flexibelt och enkelt att lägga till externa datakällor för att användas vid analyser. Det är möjligt att koppla till i princip vilka källor som helst och den externa datan kan vara i princip vad som helst, det går bland annat att använda ett XML gränssnitt för externa källor.

Enligt Ingemarsson finns det inte så mycket tekniska begränsningar för hur komplicerade SQL-frågor som kan ställas mot databasen, det är i så fall prestandan som påverkas och sätter begränsningarna. Han säger att begränsningarna antagligen speglar en dålig datamodell snarare än att det är begränsningar i verktyget. Om det inte går att ställa de frågor som användarna behöver har något gjorts fel vid modelleringen.

BW är beroende av den underliggande hårdvaruarkitekturen och den databashanterare som används för att lösningen skall vara skalbar eftersom det inte finns någon inbyggd databashanterare. Däremot finns det funktionalitet i BW för att möjliggöra skalbarhet. BW är optimerat för flera olika databashanterare och SAP försöker utnyttja deras funktionalitet så effektivt de kan. SAP rekommenderar inte någon speciell hårdvaruarkitektur för BW utan försöker vara oberoende. Däremot försöker de utnyttja den arkitektur som används så effektivt som möjligt, t.ex. kan klustring och grid-teknik användas för att förbättra prestandan.

De underliggande databashanterarnas funktionalitet för lagring och partitionering utnyttjas för prestandaoptimeringar av BW, men det finns även en logisk partitionering i BW där kuber kan delas upp, t.ex. efter region, och en multivy kan skapas ovanpå som joinar kuberna. För indexering utnyttjas databashanterarnas funktionalitet.

3.2 Användning av befintlig produkt

Vi har intervjuat ett företag som använder SAP:s BW för att se hur att datalager kan användas i praktiken. Företaget har ca 1500 anställda, har produktion i Sverige och USA och finns representerat i ca 30 länder. Respondenten på företaget är ansvarig för BW inom företaget och skapar bland annat kuber och rapportmallar. Företaget vill vara anonymt i vår uppsats och därför hänvisar vi till företaget som Företag A och den vi intervjuat för Respondenten.

3.2.1 Redogörelse för intervjun med Företag A

Företaget började implementera SAP Business Warehouse (BW) i organisationen efter årsskiftet 2003/2004. De tittade på andra verktyg men valde BW för att de sedan tidigare i stor utsträckning använder SAPs affärssystem R/3 och såg det som mycket vunnet att de redan har färdiga datakällor. Det är lätt att integrera då det finns många färdiga gränssnitt och Respondenten tycker att de fick mycket på köpet på så sätt.

I dagsläget är det två inom företaget som arbetar med implementeringen av BW och en till skall precis anställas. De har hjälp av en konsult en dag i veckan men tanken är att de skall bli helt oberoende av konsulter. Företaget kör BW på databashanteraren DB2 från IBM som de valde eftersom det företag som sköter driften är starka på den miljön.

I dagsläget har företaget 60 användare av BW men när säljarna skall börja använda det blir det ca 300 användare totalt, men inte samtidiga användare eftersom de finns utspridda över hela världen. Fördelningen är ca 160 användare i USA och resten över hela världen. Just nu är det mest controllers som använder systemet.

Det fanns tre incitament till att införa BW som Respondenten tycker är väldigt viktiga:

- Informationskvalitet, dvs. att alla ser samma data.
- Det skall vara lätt att komma åt informationen, antingen via webben eller Excel. Det var de gränssnitt företaget ville ha och därför var BW ett väldigt bra val.
- Det skall vara snabbt att komma åt, det skall gå fort. Till exempel har användare i Hongkong ungefär samma svarstider som de i Sverige trots att de i Sverige sitter mycket närmare servern.

De kuber företaget använder har de skapat själva förutom en som är standard i BW. De har börjat med att skapa kuber för säljområdet där de ser det största behovet. De har än så länge fyra kuber som de kör skarpt med. Den ena kuben kallar de sales per day och med den kan de följa försäljningen per dag på ganska hög aggregeringsnivå. Kuben innehåller väldigt aggregerad data och är till för personer på ledningsnivå. Den andra kuben, sales per month, används för månadsrapportering och i den finns siffror som varje månad har stämts av mot korrekta siffror som stämmer med månadsbokslutet. Det tredje kuben heter daily sales och är mer operationell uppföljning än sales per day-kuben. Den kuben, som skall börja användas i sommar, innehåller information om bland annat enskilda säljare, distrikt och marknadsorganisation. T.ex. kan användarna se kundfordringar per säljare. Den fjärde och i dagsläget sista kuben är discountkuben. Det är en kub som innehåller 18 dimensioner som det går att vrida och vända på för att utföra rabattanalyser. Bland annat hämtas data till kuben från alla fakturor i R/3.

I sales per day-kuben får användaren en ögonblicksbild över hur de ligger till per gårdagen. De ser sin budget som är lagd per månad och får en graf över månaden där de ser försäljning ställt mot budget. En användare kan till exempel se i en rapport hur en produktgrupp går och kan genom att använda drill down gå ner i detaljnivå och se vilken produkt som går dåligt. Det går även att koppla till möjligheten att se vilka kunder som har köpt den specifika produkten, men företaget har valt att inte göra det.

När användare använder discountkuben för att ta fram rapporter kan de till exempel lägga olika regioner mot varandra och jämföra varför de ger mer rabatter i en region än i en annan. De kan se alla bruttosiffror och alla rabatter så att de ser vilken typ av rabatt som är lämnad. De har en mängd olika rabatter som t.ex. kassarabatt, rabatt p.g.a. avtal, fria produkter, lojalitetsprogram och det är även olika vilka rabatter som används i vilka länder. De ser också totalt hur mycket som är bortrabatterat. De kan välja i rapporten att inte se alla poster, t.ex. de rabatter som inte används i det land som rapporten tas fram för. De kan se vilka produktgrupper som rabatteras, eller vilka kunder och kundgrupper som får rabatt.

Respondenten tycker att det är en jättestark rapport. Användarna kan spara den med de nyckeltal de vill ha med i sina favoriter. Det skall kunna gå att gå ner ända till vilken faktura rabatterna gäller men den möjligheten finns inte riktigt ännu. De kan inte heller se vilken säljare eller distrikt det gäller, men det har efterfrågats och kommer att komma framåt hösten. Den här rapporten är ett jättebra hjälpmedel i företaget och det är något de inte kunde ta fram på något enkelt sätt tidigare. Det är t.ex. ett bra verktyg för att se om vissa säljare är väldigt generösa med rabatter, det kan de inte få några signaler om idag.

Säljare eller säljchefer kan se daily sales antingen på dagsbasis eller månadsbasis, per distrikt, årsjämförelse med ackumulerad siffra, kundlista, kan titta på enheter, det är vad en säljare eller säljchef behöver. Det är en ganska enkelt och ren rapport med uppskattade funktioner som de inte hade tillgängligt tidigare. Olika år kan jämföras mot varandra, användarna kan välja vilka perioder de vill jämföra och de kan se information per distrikt och per säljare.

När daily sales-kuben blir klar kommer de att vara klara med försäljningsområdet och gå vidare med att skapa kuber för andra delar av företaget. Nästa steg är att titta på alla logistikflöden i företaget och bygga ett antal olika kuber för det. Därefter är tanken att kombinera kuber för försäljnings- och inköpsvärden för att göra analyser av inköp ställt mot försäljningen. Respondenten uttryckte det som ”det är det som är hela grejjen med detta”.

Tidigare har de använt R/3 Sales information system (SIS) för operativ försäljningsstatistik. Det är inte användarvänligt och användaren måste veta mycket när ett urval skall göras och något skall sökas efter. Användaren måste för varje gång veta hur de skall gå till väga, vilken data som inte skall vara med om t.ex. inte alla kundgrupper skall vara med osv. I BW har de redan filtrerat bort sådant när datan går från datalagret till kuberna. Det gör det mycket enklare för användarna som nu bara behöver välja vilket år och månad de vill titta på.

Det tar inte lång tid att bygga rapporterna, det stora jobbet ligger i att ta fram dem. Det är viktigt att tänka på vad de vill ha ut ur sina rapporter innan de designar modeller. Det är, enligt Respondenten, många som går i den klassiska fällan att de börjar ladda upp en massa data och sen tänker vi får se vad vi skall ha. ”Så det är bättre att vända på det. Man behöver inte låsa in sig i något hörn men kan ändå tänka ganska stort, vad är det för något som vi vill se? I ett projekt måste man tänka på slutet ganska tidigt.”

Användarna kan titta på rapporter via Excel-gränssnittet som är ett av de sätt som erbjuds för att titta på rapporter. Det finns en add-in till Excel så att användaren får extra ikoner för rapporterna. Användaren kan välja att titta på vad som kallas för en workbook eller en query.

En workbook i det här sammanhanget är samma sak som en query men lite mer grafisk tilltalande med olika typsnitt och liknande. Användarna kan utforma sina rapporter som de vill ha dem och spara dem som sina egna favoriter. De kan välja att rapporterna skall uppdateras varje gång de öppnas och på så sätt innehåller de alltid alla nya värden. Ingen annan ser de här rapporterna som de själva skapar men de kan skicka dem till sina kollegor som kan öppna dem om de har BW. Har de behörighet att jobba vidare med dem så får de göra det. En användare som inte har behörighet till dessa rapporter kan se dem som en statisk Excelrapport men inte jobba vidare med dem.

Det är lätt för användare att bygga på rapporter med egna rapporter, men de som administrerar systemet kan välja hur långt en användare skall kunna bygga vidare på rapporterna. Företaget tillåter inte att användarna kommer in i själva query-designern för att skapa egna rapporter. Respondenten hänvisar till erfarenheter från andra företag som har tillåtit det och det resulterar snabbt i väldigt många rapporter som dessutom ofta är snarlika och det blir ohanterligt att administrera.

De har en ”bruttorapport” som användaren kan gå in och skapa egna varianter av och spara undan i sina egna favoriter, och sen har de standardrapporter som Respondenten skapar. Nu kan användarna själv skapa rapporter och är inte lika beroende av SAP gruppen som var de som oftast skapade rapporter i R/3.

När användaren har loggat in får han eller hon välja vilken workbook de vill titta på. Det finns olika begränsningar för vad en användare får och inte får se. En del får titta på vinstmarginaler, en del får inte det. Vissa säljare får inte veta vad företaget har för marginaler, de får bara veta hur mycket de får lämna i rabatt. Andra, som till exempel controllers får förstås veta vad marginalerna är.

Det syns alltid i rapporterna vilken period användaren jobbar med, vad querien heter, vilken kub den går mot, vilket tekniskt namn den har och när den senast var laddad, om någon behöver ifrågasätta det någon gång.

Inom företaget styr de väldigt hårt på behörigheten i BW, användaren ser bara det de får se. Det är inte vidöppet, en tysk säljare kan till exempel inte se vad de har sålt i Holland utan ser bara sitt distrikt. Däremot kan en säljchef se alla distrikt i sitt land men inga andra länder. Respondenten tycker att det är ett väldigt bra behörighetskoncept bakom BW. Olika användare får också se olika rapportgrupper med olika innehåll beroende på vilken användare det är. Ingen ser mer än de har tillgång till.

Företagets VD ser en risk med att användarna skall ägna för mycket tid åt BW och rapporterna. Säljare skall inte sitta framför datorn utan de skall ha ett enkelt verktyg och de

skall sälja. Men Respondenten tror inte det är någon risk för det utan ser BW som ett analysverktyg i samband med boksluten varje månad. Det finns en funktion i BW där det går att se vem som utnyttjar vad i systemet och när, men det har de inte kopplat in ännu. Det kan vara intressant för att se vilka frågor som används mest och då kan de även hålla koll på vilka som sitter där för mycket istället för att sälja.

Ny data till kuberna laddas från R/3 varje morgon, de har ingen realtidsdata i BW. Några bolag i koncernen använder inte R/3 av olika anledningar och flatfiler från de systemen laddas in i BW en gång i månaden.

Det finns alltid en tidstämpel i rapporten för när datan är laddad så användaren hela tiden får en bekräftelse på vilken data han eller hon jobbar med. All data som är med i R/3 fram till den tidpunkten är med i rapporten.

Respondenten tycker att den bästa lösningen är att använda R/3 till det som det är bra på, att sköta och stödja den operativa verksamheten, medan BW används till att göra analyserna. Rapporten med alla dimensionerna som kan köras i BW tar kanske 20 sekunder medan det kanske tar en och en halv timme i R/3. Däremot så finns det viss information som en användare kan vilja ha i realtid och då skall R/3 användas till det. För de analyser som företaget gör i dag räcker det att ladda datan en gång per dag. Eventuellt skulle de kunna göra två laddningar per dag, men det är inte säkert att det tillför något.

I BW finns det två typer av laddningar till datalagret; init- och deltaladdningar. ”Det är det som är jättesnyggt med det är extraheringsverktyget, ETL, det finns en deltamekanism och när man sätter upp systemet eller när man gör en förändring i strukturen eller lägger till dimensioner så gör man en initial laddning, en initladdning.” När en deltaladdning görs laddas bara nya poster in till kuberna. Senast företaget gjorde en intiladdning var på julafton eftersom det måste ske när det är en lucka i användandet. Men det kan i princip göras på vilken helg som helst, när ingen användare är inloggad. Företaget har problemet att de har en server och användare som jobbar från Australien till Sydamerika och det innebär att de har användare på i princip hela dygnet. De har ett fönster från 03.00 lördag morgon när amerikanarna går av till 23.00 på söndag när australiensarna går på. Underhållsfönstret är inte speciellt stort och därför får de använda helgerna till underhåll och de har underhållshelg en gång i månaden. Företaget gör inget sådant arbete själva utan de har anlitat ett externt företag som sköter all drift och backup. Inom företaget har de själva den stora applikationskompetensen. De är idag 12 personer och skall bli 14, de har alltså rätt så mycket kompetens i huset. Många som jobbar med det här inom företaget är gamla SAP-konsulter och har jobbat med de här verktygen länge. På BW och CRM (beslutstöd för

kundrelationshantering) tycker Respondenten att de är lite tunna än så länge men det håller de på att lära sig.

När en initial laddning körs handlar det om ca 5 miljoner poster. Sen är det 9000 poster per dag som går upp i kuben. De går inte till datalagret utan till det som är aggregerat, det är aggregerad data. Enskilda försäljningar osv. ser användarna i datalagret där all data finns. Där är det betydligt fler och större transaktioner. ”Vi jobbar alltså med ca 9000 poster om dan när vi slicar och dicar.” En deltakö körs och det återanvänds till de flesta kuberna. Samma data behöver inte laddas på flera ställen utan den laddas bara in och pushas sedan ut.

Något som är negativt med BW är att när en initladdning måste göras, när något skall ändras eller läggas till, om en dimensions skall läggas till måste den initdatan som redan finns i kuberna tas bort. Det är ett ganska stort arbete. Men, säger respondenten, så är det ju i alla system. Om strukturen ändras så måste en ny initladdning göras för att få deltamekanismen att fungera sen. Men det gör att det är lite sårbart och man drar sig för att ändra strukturen.

Däremot tycker han om deltamekanismen så att fulla laddningar inte behöver köras hela tiden. Det är inte hållbart, det är alldeles för stora poster för det. Även om det här är ett litet företag så håller det inte. Deltamekanismen i sig är väldigt stark men om strukturen behöver ändras så får man tänka sig för lite grann. En deltaladdning tar idag ca 2-3 minuter.

Den mesta datan kommer från R/3 och den datan gör inte företag något med, de tvättar den inte. De kallar samma saker för samma namn i BW som R/3 för att användarna skal känna igen sig.Om de hade haft många olika källsystem hade de kanske fått specificera t.ex. vad en kund är så att de jämför äpplen med äpplen. Företaget har redan sådana specifikationer klara i R/3 och forslar det bara vidare. Andra företag som har många system som de för in i BW får jobba mer med uppdateringsregler och tala om vad som är vad.

Vissa kundgrupper och produktgrupper som de inte vill ha med sig upp till kuberna plockas bort, det har de uppdateringsregler som styr. Det går aldrig någon data från R/3 direkt upp till kuben. Datalagret blir som en snapshot av R/3 vid jämna tidpunkter.

Respondenten är väldigt positiv till SAP men menar att han är medveten om bristerna i deras produkter. I BW tycker han inte att han än så länge har hittat några direkta brister. Det skulle i så fall kanske vara prestandan ibland men han säger att då får de jobba med index och aggregat för att lösa det.

Ett prestandaproblem som kan uppstå är om kuberna innehåller för mycket data. Företaget har en del kuber som går lite långsammare än vad som kanske är önskvärt men de tycker ändå att det är tillräckligt snabbt för att vara acceptabelt. För att de skall gå fortare kan de jobba ytterligare med aggregat.

Discountkuben är lite långsammare eftersom den innehåller ganska mycket data. Där skall de lägga in index, det håller de på med just nu. Den kuben används mest på månadsbasis och i samband med rapportering och analys, men användarna blir bortskämda och vill att det skall ta max 5 sekunder och sen skall den vara uppe.

Det största prestandaproblemet, enligt Respondenten, är att de inte har hittat rätt aggregationsnivå upp till kuberna och att de har för mycket detaljerad information med sig. ”Man vill göra för mycket.” Han tycker ändå att de har lyckats rätt så bra med det. De kan också använda sig av drill down-teknik för att jobba från kuben ner till datalagret. Det gör de en del, och sen kan de gå vidare ändå in i affärssystemet, R/3. Rapporterna tar lång tid att köra om det finns för mycket detaljer i kuberna. Respondenten säger att rent metodmässigt skall det inte vara för mycket detaljer i en kub. Det skall vara så aggregerad nivå som möjligt och vill användarna se mer detaljer från de gå ner till datalagret istället.

Alla detaljer finns i datalagret men det är ingen realtidsdata där. Där finns det dokumentinformation men det gör det inte i kuberna. Om en användare till exempel vill se fakturor som ligger bakom en produkt så göra de en drill down från kuben till datalagret. Men, säger respondenten, datalagret är så våldsamt stort och där måste de jobba mycket med index.

Prestandaproblem kan uppstå om datan förs över till kuberna utan att aggregeras. Det är beroende av hur stora servrar systemet körs på, hur mycket diskutrymme som finns bland annat. Företag A har inte några problem med prestanda än så länge.

När frågor ställs mot datalagret är det vikigt att hitta de fält som är indexerade. Om frågor ställs mot ett fält som inte är indexerat så kan det ta väldigt lång tid eftersom det handlar om stora mängder data. Det är inte tänkt att rapporter skall köras mot datalagret. Men när frågan ställs mot kuben går det sen att borra sig ner till detaljerna. Oftast nöjer användaren sig med aggregerad information.

Något som skulle kunna bli ett problem i förlängningen är att de kör BW och ett annat system på samma server och delar diskarea och annan hårdvara. BW borde flyttas till en helt separat server med anpassat diskutrymme där de får fler användare. Nu går det bra eftersom de fortfarande har få användare, men sen tar de hjälpa av företaget som sköter driften för att lösa det, men det är en kostnadsfråga.

Vi frågade om det går att säkerställa att olika användare ser samma siffror så att de kommunicerar om samma sak. Respondenten säger att när Excel används finns det aldrig någon garanti på att det är rätt data, det går att förvanska datan hur lätt som helst. Företaget använder sig av en tidstämpel för att tala om när datan är laddad men de kan inte ta ansvar för

om någon gör beräkningar på datan. De begränsningar de har gjort är att användarna inte kan ändra frågan. De kan se hur frågan är uppbyggd i ett displayläge och se vilka nyckeltal som finns med, men de kan inte göra nåt med det. Sen kan de lägga till vilka fält de vill eller skriva vilka kommentarer de vill i Excel men det är inget som går att begränsa.

Respondenten tycker inte att han kan svara på frågan om hur bra beslutsstöd som BW ger men säger att han ser en hel del nöjda användare. Däremot kan han inte uttala sig om det är beslutsstödet de är nöjda med eller att de i och med BW får datan på ett smidigare och snabbare sätt. Det är i alla fall en ganska tydlig och snabb beslutsinformation. De är nöjda med rapporterna och känner att de kan lita på dem. Han berättade om en användare som hade hört av sig eftersom han hade fått fram fel siffror i en rapport, men när de tog reda på vad det berodde på så var det fel i källan och inte i rapporten. Det ser Respondenten som ett bra betyg.

Det finns många fördelar redan idag och ändå har de inte börjat med data mining, hitta korrelationer och utnyttja BW fullt ut. ”Det är väldigt basic än så länge men det räcker oftast.”

Den största vinsten som respondenten ser med att använda BW är att det går fort att ta fram rapporter och att användarna nu kan göra sina egna rapporter som de sparar som favoriter.

En annan vinst är att det blir en informationskälla men dit har de inte nått än, men det skall bli så. Med en källa behöver de inte vara oroliga för var siffran kommer ifrån och användarna kommer inte på mötet med olika siffror.

Analysen med rabatter hade de inte tidigare och det ser han som en bonus de har fått med det här verktyget. Det hade säkert gått att få fram i R/3 men det hade varit mycket jobbigare. Nu får de bättre koll på siffrorna.

3.3 Datalager i ett bredare perspektiv

Vi har intervjuat ytterligare en leverantör av datalager för att få ett annat perspektiv på hur datalager kan användas. Dessutom har vi intervjuat en forskare vid Handelshögskolan i Stockholm som har undersökt hur ledningen i svenska storföretag använder applikationer för beslutsstöd. I det här delkapitlet redogör vi för de intervjuerna.

3.3.1 Redogörelse för intervjuer med Daniel Ringquist, SAS Institute

SAS Institute är ett privatägt företag som är specialiserat på att utveckla mjukvara för bland annat statistiska analyser och beslutsstöd. Företaget grundades 1976 och har idag lite drygt 9 500 anställda på 329 kontor världen över. De har kunder i 109 länder och ca 40 000 installationer av sin programvara. Företagets huvudkontor ligger i Cary, North Carolina, i USA. 2004 hade företaget en omsättning på 1,5 miljarder dollar. Vi har intervjuat Daniel

Ringquist som är produktchef för Business Intelligence för att få ett bredare perspektiv på vad datalager är och vad de kan användas till.

Enligt Ringquist blir ett datalager aldrig komplett utan skapas för en generisk situation, ett datalager kan bara utvecklas för allmänna behov som går att förutse. Ett datalager utgör en gemensam källa att utgå från vid rapportering och analyser vilket är bra, men kostnaden och tiden det tar att anpassa datalagret för ad hoc behov blir oöverstiglig. Det är viktigt att hämta data från ett och samma ställe men det är inte säkert att det är ett datalager som behövs. Det går inte att utöka ett datalager med något som enbart behövs vid ett enstaka tillfälle eftersom det är komplicerat att förändra en datalagermodell. Ofta blir strukturen för datalagret obegriplig då den innehåller många olika typer av data och väldigt många tabeller. Det inte är någon idé att tro att ett datalager kan uppdateras så fort ett nytt behov uppstår. IT-avdelningen som skall genomföra förändringarna i datalagermodellen har ofta mycket att göra men användarna kan ha behov av att använda lösningen med en gång. Då går det inte att ägna flera veckor åt att implementera stöd för en enstaka analys. Användarna har inget behov av ett datalager, de behöver rapporter och analyser, man måste vända på resonemanget säger Ringquist. Han ger exempel på företag som investerat miljontals kronor men efter projekt som tar flera år så får de ändå inte ut det de behöver och förväntar sig.

Analys är det som tillför mest värde för kunderna och används för att försöka förstå varför något blev på ett visst sätt. Det är de rapporter som ligger utanför de vanliga 80% som skapar värde för användarna och Ringquist ifrågasätter att den data som behövs för de analyserna behöver finnas med i datalagret. För att ta fram de rapporterna behövs analytiker som vet vilken data som behöver analyseras. Det är viktigt att analytikerna själva kan ta fram den data som de behöver utan att vara beroende av IT-avdelningens stöd, samtidigt måste IT-avdelningen förstå vad det innebär att göra analyser och vad det är som behöver analyseras. Ofta har analytikerna svårt att förklara sina behov för IT-avdelningen och de kan t.ex. inte uttrycka sina problem på egen hand med SQL. SAS Institutes egna databashanterare med tillhörande frågespråk är anpassat för den typen av ändamål, analytikerna kan på ett enklare sätt än med SQL ställa avancerade statistiska frågor mot databasen. Enligt Ringquist är det inte möjligt att få ut det analytikerna behöver ur databasen med vanlig SQL och dessutom måste databasen vara optimerad för att kunna läsa stora mängder data snabbt. SQL är till för att extrahera rader med data ur databasen men det går inte att få svar på det som användaren egentligen undrar. Ringquist berättar att det är vanligt att det inte går att använda kundernas befintliga datalager för att utföra analyserna eftersom SQL är för begränsat och traditionella

databaser inte är anpassade för analyser, då duplicerar de databasen med sin egen databashanterare.

Som exempel på analyser och lösningar som SAS Institute har varit med och tagit fram nämner Ringquist en dagligvarukedja som vill ta fram prognoser på hur mycket varor som skall skickas ut till de olika butikerna. Om det skickas ut för mycket varor måste en tom bil åka för att hämta tillbaka det som blir över. Det är inte lönsamt och därför vill de försöka optimera leveranserna av varenda artikel till alla butiker. Behovet av analyser är, enligt Ringquist, även stort inom offentlig sektor eftersom det är nödvändigt att veta vart pengarna tar vägen, dessutom är det viktigt att veta var och när verksamheten skall förstärkas eller när besparingar kan göras. När organisationen förstår vad som har hänt tidigare så kan de med prediktiva modeller ta fram ett bästa och ett värsta tänkbara scenario för att försöka förutse vad som kommer att hända och anpassa verksamheten och organisationen efter det. Andra områden där det ställs stora krav på analyser och rapporter är inom bank-, försäkrings- och läkemedelsbranschen, lagstiftningen kräver att företagen skall kunna redogöra för eventuella risker för att kunderna skall skyddas.

All analys och beslutsstöd handlar egentligen om att hitta samband. Ett företag är inte intresserade av om en specifik produkt fungerar eller inte, frågan är varför helt plötsligt 50 av 10 000 inte fungerar och om de kommer att sluta fungera om en månad igen eller om det bara var en engångsföreteelse. För att få reda på det kan matematiska modeller användas för att sätta upp ett antal antaganden och datorn utnyttjas för att gå igenom alla tänkbara variabler som t.ex. längd, gods, temperatur och sen använda sannolikhetsteori för att beräkna vilka parametrar som behöver kontrolleras noggrannare. Det går inte att genomföra okulärt, att t.ex. analysera och jämföra hundratals grafer och försöka hitta samband mellan dem är inte möjligt. Ringquist tar upp att spårbarheten i datan är viktig för företagen eftersom vissa verksamheter styrs av sträng lagstiftning och måste kunna redovisa externt hur ett visst resultat har beräknats.

Det finns inte en sanning för vilken arkitektur som skall användas för ett datalager. Ringquist betonar vikten av att utgå från varje enskild organisations behov och förutsättningar. Han anser att det bland annat beror på företagets storlek och komplexitet vilken arkitektur som skall väljas. Vissa har enbart behov av ett fåtal OLAP-kuber medan andra kan behöva fler. Det vikigaste är att utgå från vad organisationen behöver och mäktar med.

Ringquist säger att det kan vara omöjligt att på förhand avgöra vilken detaljnivå som är lämplig, men den måste utgå från organisationens behov. Det finns dock en risk med att bara

utgå från stundens behov och inte fundera över framtida användningsområden. Det är viktigt att verksamheten bestämmer sig för vad de vill ha med i datalagret vid utvecklingen, vilken data som skall gå att få ut ur systemet och vad de vill göra med den. Det gäller att vara pragmatisk.

Angående Dates (2004) kritik mot designprinciperna bakom starscheman så säger Ringquist att Date säkert har rätt ur ett teoretiskt perspektiv, men han anser att man måste kunna lösa kundens problem. Ringquist tycker att det är viktigt med den logiska designen, det är en nödvändig förutsättning, men det är enligt honom inte möjligt om ett datalager byggs helt enligt teorin. Om inte användarna kan få ut det de behöver ur databasen spelar det ingen roll vad Date förespråkar. För det första är det inte säkert att företaget har råd eller tid att vänta på en korrekt datamodell. För det andra är det möjligt att Date skulle kunna skapa datamodeller enligt teorin som är en bra lösning, men de som designar lösningen ute i verksamheten har inte samma kompetens som honom. För det tredje skall användarna förstå modellen och kunna ställa frågor mot den. Användarna kan till exempel ofta inte uttrycka det de behöver få fram i SQL för att själva lösa sina problem. Det är viktigt att vara pragmatisk och lösa kundens problem oavsett vilken modell som används.

Ringquist säger att användarna inte vet varifrån datan kommer, de vet att det t.ex. är försäljningskuben de arbetar mot och det räcker för dem. Datan har då gått från datalagret till en data mart till en kub till en rapport. Det är rapporten som är viktig för användaren och inte hur datan struktureras i databasen. Målen är att tillfredsställa användarnas behov och möjliggöra hög prestanda. OLAP-kuber är väldigt bra för resultatrapportering, det är lätt att bryta upp ett år i kvartal och perioder och det är lätt att jämföra saker. För rapportering i allmänhet är kuber bra.

3.3.2 Redogörelse för intervju med Björn Thodenius, Handelshögskolan

Björn Thodenius är forskare vid sektionen för Information Management på Handelshögskolan i Stockholm och har nyligen skrivit en avhandling om användandet av ledningsinformationssystem i Sveriges 200 största företag (Thodenius, 2005). Grundtanken med ledningsinformationssystem är att de skall utgöra ett beslutsstöd med inbyggda metoder och analysmodeller för att föreslå beslut. Ledningsinformationssystem går under många namn och det finns flera definitioner, vi hänvisar till Thodenius bok för en utförligare beskrivning. Thodenius har nästan uteslutande undersökt hur koncernledningen använder de här systemen som beslutsstöd och vi vill uppmärksamma läsaren på att ett datalager kan ha betydligt fler

användningsområden, t.ex. skapandet av rapporter av chefer på mellannivå (se intervjun med Företag A), men är ändå en viktig datakälla för beslutsstödssystem.

Thodenius säger att han inte tror på användningen av beslutsstöd där det finns någon form av kvalitet i besluten, de är t.ex. inte lämpliga för strategiskt viktiga beslut. De är bra på att automatisera t.ex. kreditbedömning där kundens inkomstuppgifter kan analyseras i en kreditprövning. Det är en låg andel kunder som de systemen inte passar för vilket gör att nästan hela kreditbedömningen kan automatiseras vilket sparar personal och resurser. Om vissa kunder ändå har speciella förutsättningar kan de hanteras separat. För strategiska beslut är själva beslutsprocessen mycket viktig för att beslutsfattaren skall vara insatt i och förstå problematiken. Om de besluten skulle automatiseras förloras mycket nödvändig kunskap till slut då ingen vet vad besluten egentligen grundas på. Enligt Thodenius är det mycket viktigt för beslutsfattarna att veta hur en rapport har tagits fram och hur innehållet har beräknats för att ett beslut skall kunna fattas. Det måste dessutom finnas någon som ansvarar för den och dess innehåll som det går att vända sig till vid tveksamheter. Med de förutsättningarna uppfyllda kan ett datalager användas för att samla in data som underlag för rapporter och analyser.

Högsta ledningen har behov av att välja ut nyckelvariabler som väl återspeglar verksamheten för att få en bild av hur verksamheten går och sedan använda beslutsstödssystemet för att följa hur värdet på de variablerna förändras. Det går inte att generellt säga vad de variablerna skall representera utan det är något ledningen av erfarenhet vet är en bra mätare på något som är viktigt för verksamheten. Ett exempel skulle kunna vara att ett flygbolag vill se beläggningen i sina flygplan för att jämföra förhållandet mellan platser som sålts till ordinarie pris med rabatterade platser. Det går inte heller att generalisera om hur detaljerad information som behövs, för data som är viktig för att styra verksamheten kan hög detaljnivå vara nödvändig, eller vilket historiskt perspektiv som är viktigt. Viss data kanske behövs samma dag eller dagen efter något har inträffat men datan blir kanske inte heller för gammal för att vara intressant vid senare analyser. Historisk data är viktig för strategiska beslut och det kan vara bra att kunna ta fram detaljerad data vid behov.

Företagen vill alltid ha bättre information än konkurrenterna men Thodenius anser att företagen samlar på sig för mycket data och säger att det är viktigt att tänka igenom vad rapporterna skall användas till. Han säger att det är viktigt att organisationen har någon form av strategi för vad de vill ha och de måste bedöma vilken detaljnivå som är rimligt. Det går inte att spara alla detaljer, det bli ohanterligt för ett stort företag. Ibland utökas rapporterna för att någon vid något tillfälle ville ha ett nyckeltal om t.ex. kunder, som då läggs till i rapporten.

Till slut kanske det finns 30 nyckeltal med i rapporten medan det bara är 10 som används. Det kan till viss del bero på att användarna inte riktigt vet vad de har behov av. Då är det bra att sätta sig ner och gå igenom vad som är viktigt, vad det är som används. Thodenius säger att det kan vara ganska svårt att ha en strategi från början, det går inte att förutsäga alla informationsbehov i en organisation eller vad informationen skall användas till, men företagen måste bedöma vad som är rimligt att spara.

Slutligen anser Thodenius att det största problemet med ett datalager är att definiera begrepp. Det är inte ett tekniskt problem men han tror att det är omöjligt för ett stort globalt företag att ha ett företagsövergripande datalager med gemensamma definitioner på alla begrepp (för mer detaljer se Thodenius, 2005). Däremot fungerar datalager avdelningsvis, för olika produktområden eller divisioner.

4 Analys och slutsatser

I det här delkapitlet jämför vi den teori vi har sammanställt med det vi har fått fram i våra intervjuer för att belysa de likheter och skillnader som finns mellan teori och praktik. Därefter redovisar vi våra slutsatser som utgår från analysen.

4.1 Analys

I analysen utgår vi i huvudsak från teorikapitlets tre huvuddelar för att tydligt jämföra de olika teorierna med motsvarande empiri.

4.1.1 Datalager

Stora, globala organisationer har ett mycket stort behov av att samla in data från olika typer av transaktionssystem till ett datalager för att möjliggöra analyser för beslutsunderlag. Det ställs mycket speciella krav på den här typen av system och det är extremt stora mängder data som behöver hanteras. En mängd olika aspekter bidrar tillsammans till att det är svårt och komplext att konstruera ett datalager. De kräver i princip ingen ny teknisk innovation utan snarare ett effektivt utnyttjande av befintliga tekniker för att skapa en arkitektur som är flexibel nog att möta beslutsfattarnas ständigt förändrade behov, men ändå kraftfull nog att möjliggöra analyser av flera TB data inom rimlig tid.

Inmons övergripande arkitektur verkar vara en lämplig utgångspunkt för ett datalager. När data marts byggs ovanpå datalagernivån finns det en gemensam grund som analyser och rapporter baseras på. Den grundläggande datalagernivån gör att det blir lättare att lägga till nya ämnesområden att samla in data om och bygga data marts för. Om ett datalager ses som en samling data marts, enligt Kimballs synsätt, finns det en risk för att fördelen med att utgå från samma data går förlorad.

Inmons arkitektur är en flexibel struktur för att utöka ämnesområden men också för att skapa data marts som kombinerar olika ämnesområden för att göra övergripande analyser, exempelvis göra kombinerade analyser av inköp ställt mot försäljningen som Företag A har tänkt göra.

Samtidigt är det ett stort och komplicerat arbete att bygga ett omfattande datalager. Det är viktigt att börja med det som är viktigast för företaget men ändå ha en bra genomtänkt arkitektur som tillåter att nya ämnesområden läggs till efter behov. Vi tror att det är

nödvändigt att snabbt kunna komma igång och sen bygga ut datalagret allt eftersom behoven ökar.

Daniel Ringquist på SAS Institute menar att det inte finns *en* sanning för vilken arkitektur som skall användas för ett datalager. Han betonar vikten av att utgå från varje enskild organisations behov och förutsättningar. I SAPs BW finns det inte heller några tekniska hinder för att bygga datalagret enligt Kimballs teorier. I BW utgörs Inmons datalagernivå av en logiskt sammanhängande mängd ODS-objekt och därför är det möjligt att bygga datalagret utifrån andra principer.

Data i ett datalager är sammanställd av detaljerad data från flera operativa system vilket underlättar för en verksamhetsanalytiker att göra analyser eftersom de inte behöver känna till vilka system som datan ursprungligen kommer ifrån (Inmon, 2002). Eftersom en analytiker behöver arbeta utforskande och heuristiskt så anser vi att friheten ökar med ett datalager. Alternativet är att använda många små program som extraherar data ur de operativa systemen.

En viktig fördel med ett datalager är att det utgör en gemensam, enhetlig datakälla för alla rapporter och analyser. För Företag A är det en stor vinst att deras rapporter kommer från en och samma informationskälla och de behöver då inte fundera på var siffrorna kommer ifrån. Det här är också något som Inmon betonar i sina böcker.

I ett datalager finns gemensamma datadefinitioner och det uppstår inte förvirring på samma sätt som tidigare om vad datan verkligen representerar. Dessutom kan olika användare ta fram rapporter med samma data och risken att olika data används försvinner. Företag A ser det som en väldigt stor fördel att alla användare ser samma data och att den går snabbt och enkelt att komma åt.

Ett annat problem som ett datalager kan lösa är bristen på historisk data i de operativa systemen (Inmon, 2002). I ett datalager kan historisk data sparas så länge den behövs. Enligt Björn Thodenius från Handelshögskolan i Stockholm är historisk data viktig för strategiska beslut och det kan vara bra att kunna ta fram detaljerad data vid behov. I ett datalager finns en tidstämpel på datan så att det vid jämförelser går att veta att datan kommer från samma tidpunkt. Företag A har information i alla sina rapporter om bland annat när datan är laddad och vilken kub den kommer ifrån så att inga diskussioner skall uppstå om vilken data som används. De är nöjda med rapporterna och har förtroende för dem. Vi ser det som en stor fördel att historisk data finns lättillgänglig eftersom det går att studera transaktioner i detalj och skapa rapporter där det går att se utvecklingen under en lång tid.

Som Söderström (1997) tar upp är en fördel med ett datalager att beslutsfattare själva kan välja den data som de vill ha vid ett givet tillfälle och få den presenterad på det sätt som

önskas. Enligt Thodenius är det mycket viktigt för beslutsfattarna att veta hur en rapport har tagits fram och hur innehållet har beräknats för att ett beslut skall kunna fattas. Vi tror att när rapporten baseras på data som kommer från en och samma källa och det går att kontrollera hur dataaggregationerna går till går det också att lita på den, förutsatt att datan är riktig vid laddningen till datalagret. I ett datalager aggregeras data på samma sätt varje gång vilket gör resultatet förutsägbart. Även Ringquist tar upp att spårbarhet är viktigt för företagen eftersom vissa verksamheter styrs av sträng lagstiftning och måste kunna redovisa externt hur ett visst resultat har beräknats.

Högsta ledningen har enligt Thodenius behov av att välja ut nyckelvariabler som väl återspeglar verksamheten för att få en bild av hur verksamheten går. Då tror vi att ett datalager kan göra stor nytta eftersom de variabler som presenteras för ledningen kan vara baserade på många olika typer av detaljerad data, bland annat historisk data för att se hur variablerna har utvecklats över tiden. Eftersom de viktigaste nyckeltalen för verksamheten behövs så nära realtid som möjligt tror vi att det är lämpligt att använda ODS:en för dem när det inte finns behov för det historiska perspektivet. Ett exempel skulle kunna vara att ett flygbolag kan se beläggningen i sina flygplan. Enligt Thodenius går det inte att förutsäga alla informationsbehov i en organisation eller vad informationen skall användas till, därför är ett datalager en fördel eftersom där kan all data samlas och efterfrågas vid behov.

Det är viktigt att vara medveten om vilka begränsningar det finns med ett datalager. Ringquist säger att det tar lång tid att förändra i ett datalager eftersom datamodellen är väldigt komplex. Dessutom kan det vara en stor del av rapporterna som det inte är någon idé att basera på ett datalager. Det kan vara rapporter som antingen behövs snabbare än förändringarna kan genomföras eller bara behövs vid ett fåtal tillfällen och då är det ingen idé att förändra i datalagret. Av den anledningen är det viktigt att ta ställning till vad som inte skall vara med i datalagret eftersom det inte får ta för lång tid att bygga, och datalagret i sig får inte heller vara onödigt komplext.

4.1.2 Dimensionsmodellen

Date (2004) beklagar att så vitt han vet finns det ingen databashanterare på marknaden som har implementerat stöd för relationsteorin i sin helhet. Eftersom han inte kommer med praktiska förslag på hur beslutsstödsdatabaser skall designas utan bara förklarar varför vissa designprinciper är fel rent teoretiskt så anser vi inte att hans argument väger tungt när företagen ställs inför praktiska problem.

Relationsmodellens styrka är att den ger största möjliga flexibilitet. Den logiska databasdesignen skall inte vara beroende av den avsedda användningen, samma regler skall gälla oavsett användning (Date, 2004). Relationsmodellen beskriver data och relationerna som de är, det finns inga problem med tvetydigheter förutsatt att designen är riktig.

Enligt Kimball et al. (1998) är enkelhet för användarna och prestanda de två primära skälen till att använda dimensionsmodellen. Vi håller inte med Kimball om att dimensionsmodellen primärt skall användas för att användare skall ha lättare att förstå diagrammen, i så fall hade det varit bättre att ha någon form av användargränssnitt mot databasen som påminner om starschemats enklare design. Det är orimligt att skapa databaser med enorma volymer redundant data bara för att användarna inte förstår hur de skall ställa frågor. Däremot skall dimensionsmodellen användas om den ger bättre prestanda och löser problemen på ett bättre sätt.

Ringquist säger att användarna inte vet varifrån datan kommer, de vet att det t.ex. är försäljningskuben de arbetar mot och det räcker för dem. Datan har då gått från datalagret till en data mart till en kub till en rapport. Det är rapporten som är viktig för användaren och inte hur datan struktureras i databasen. Målen är att tillfredsställa användarnas behov och möjliggöra hög prestanda.

Många som har försökt leverera data till slutanvändare har sett det som omöjligt att presentera enormt komplexa E/R-diagram för dem och har börjat om med en enklare design (Kimball et al., 1998). I exemplet i avsnitt 2.2.3.10 är det bara en faktatabell och fyra dimensioner som normaliseras till 3NF och enbart utifrån det fås ett E/R-diagram med 11 relationer, vilket är mer än en fördubbling. Det är lätt att tänka sig hur komplext det blir med så många som 15 dimensioner som Kimball säger är vanligt eller t.o.m. 18 som Företag A använder i en kub, som dessutom kan ha många gånger fler attribut och hierarkier än dimensionstabellerna i ovan nämnda exempel. Utifrån det går det att förstå Kimballs argument om att det inte är användarvänligt att framställa tabellerna i ett normaliserat E/R-diagram, men det kan inte vara det primära skälet till att välja dimensionsmodellen.

Dessutom ställer användarna ofta inte egna frågor i traditionell bemärkelse mot databasen utan använder verktyg för slice och dice, roll up och drill down. De utgår från fördefinierade rapporter som är en sorts dynamisk vy och behöver därför inte ställa komplexa frågor eller vara medvetna om vad som händer i bakgrunden. Användarna skapar sina frågor i en query-designer och ser därmed inte koden som genereras, de behöver inte förstå frågespråket som används. Hos Företag A är det de ansvariga för BW som skapar kuber och rapporter som användarna kan skapa egna rapporter utifrån. Även om användarna kombinerar kuber och

rapporter och tar in extern data så är den interna datastrukturen fortfarande inte exponerad mot dem.

Ovanstående resonemang gäller för den största delen av rapportering och användning av datalagret med tillhörande data marts. När mer komplexa analyser skall göras och frågor ställas mer ad hoc behöver användarna kunna göra mer saker på egen hand. Ringquist säger att analytikerna har svårt att förklara sina behov för IT-avdelningen och kan ofta inte uttrycka sina problem på egen hand med SQL. SAS Institutes egna databashanterare med tillhörande frågespråk är anpassat för den typen av ändamål och Ringquist ifrågasätter att all data behöver finnas med i datalagret. Ett datalager kan enligt honom bara ta fasta på det som går att förutse, det allmänna behovet.

Date (2004) betonar att den logiska databasdesignen skall göras innan den fysiska. Christer Ingemarsson på SAP håller med om Dates resonemang och säger att det är viktigt att utgå från en logisk datamodell när kuberna skapas, den logiska datamodellen behövs eftersom det inte går att få någon gemensam syn med kuberna. Även Ringquist tycker att det är viktigt med den logiska designen, det är en nödvändig förutsättning.

Vi håller med Inmon som anser att datalagret skall vara normaliserat. Primitiv data från de operativa systemen i datalagret, dvs. den data som aggregationerna skall utföras på, skall vara normaliserad för att vara så flexibel som möjligt och vara lätt att komma åt på alla olika sätt som användaren önskar. Det skall vara lätt att skapa nya kuber. Med ett datalager går det att modellera om kuberna hur som helst, enligt Ingemarsson. Det behövs ett flexibelt datalager som stödjer många användningsområden. På datalagernivån är det viktigt att utnyttja relationsmodellens flexibilitet och oberoende av användningsområde.

I en relation har varje tupel ett bestämt värde utifrån tabellens definition, dvs. varje tupel är en instans av den definitionen och alla tupler har därmed samma betydelse. I ett starschema som tillåter NULL-värden, nivåer, de olika posterna betyder olika saker och det inte finns en gemensam definition kan det vara förvirrande för användaren vad en given post betyder. I ett starschema kan ett attribut läggas till enbart av prestandaskäl för att undvika att utföra en join. I en relationsdatabas, där alla tabeller är perfekt normaliserade, kan det å andra sidan vara förvirrande för en användare att se ett sammanhang eftersom det ofta är många relationer att koppla samman, t.ex. kan ett stort affärssystem ha tusentals tabeller. Men i relationsdatabasen är allt perfekt, otvetydigt, definierat. De här för- och nackdelarna måste vägas mot varandra när designbeslut skall fattas.

För att bygga en data mart som är optimerad för en speciell avdelnings krav och behov måste det finnas mycket kunskap om hur den skall användas. När det väl är känt kan en data

mart utformas med en optimal starschemastruktur (Inmon, 2002). Eftersom Kimball ser ett datalager som en samling data marts och inget annat, det finns inget datalager i hans arkitektur, så är i princip all data denormaliserad om ett datalager byggs utifrån hans synsätt. Han betonar vikten av att inte använda snowflake, och det är ändå bara en låg grad av normalisering. Det tycker inte vi är ett bra synsätt, den data som kan vara normaliserad utan att prestanda och alltför mycket flexibilitet påverkas skall vara det.

Ett denormaliserat starschema har begränsningar i hur datan kan användas men för ett väldefinierat syfte fyller de en funktion och bidrar till ökad prestanda. Starschemat är anpassat för att göra roll up, drill down och ha hierarkier, vilket till exempel Företag A använder för sina rapporter. Kuber är väldigt bra för resultatrapportering, enligt Ringquist, det är lätt att bryta upp ett år i kvartal och perioder och det är lätt att jämföra olika värden. För rapportering i allmänhet är kuber bra.

Angående Dates (2004) kritik mot designprinciperna bakom starscheman så säger Ringquist att han säkert har rätt ur ett teoretiskt perspektiv, men han anser att det måste gå att lösa kundens problem. Det är enligt honom inte möjligt om ett datalager byggs helt enligt teorin. Om inte användarna kan få ut det de behöver ur databasen så spelar det ingen roll vad Date anser. För det första är det inte säkert att företaget har råd eller tid att vänta på en korrekt datamodell. För det andra är det möjligt att Date skulle kunna skapa datamodeller enligt teorin som är en bra lösning, men de som designar lösningen ute i verksamheten har inte samma kompetens som honom. För det tredje skall användarna förstå modellen och kunna ställa frågor mot den. Användarna kan till exempel ofta inte uttrycka det de behöver få fram i SQL för att själva lösa sina problem.

Vi håller med Ringquist i hans resonemang, det går inte att slaviskt följa en princip som går ut över användarvänlighet och prestanda. Vi tycker att prestanda skall gå först där det behövs. Där prestanda inte får gå före är på datalagernivån eftersom den är tänkt att vara flexibel och anpassningsbar och där skall det gå att ställa oväntade frågor. Därför är det inte lämpligt att låsa sig vid en viss design eftersom hela konceptet med datalager faller om flexibiliteten inte bevaras, relationsmodellen erbjuder maximal flexibilitet. Både Ingemarsson och Ringquist anser att det är viktigt att vara pragmatisk och lösa kundens problem oavsett vilken modell som används.

I BW används något som SAP kallar för extended starschema där vissa dimensioner ligger i kuberna men data som ändras sällan och används av flera kuber, som de kallar masterdata, ligger utanför och är gemensam för olika kuber. Det finns, som vi ser det, två fördelar med att göra på det sättet. För det första så behöver inte den data som är gemensam uppdateras varje

gång kuben laddas om. För det andra så behöver bara den gemensamma datan laddas om på ett ställe när den behöver uppdateras och då slår det igenom för alla kuber. Ingemarsson säger att det ger större flexibilitet, när masterdatan uppdateras har kuberna automatiskt tillgång till rätt data. Nackdelen är att blir extra läsningar för varje gång datan behövs, en join behövs för varje tabell med masterdata. Ett val måste då göras mellan prestanda och flexibilitet. För att få bättre prestanda kan attributen läggas in i kuberna som då blir en vanlig kub. Vi ser SAPs utökade starschema som en variant av ett snowflakeschema.

Date (2004) kritiserar att det ofta uttrycks att relationstabeller är platta och tvådimensionella. Det är sant att tabeller är tvådimensionella men det är inte relationer, de är n-dimensionella. Under tiden vi har arbetat med uppsatsen har vi ständigt i olika sammanhang stött på de här påståendena som Date är kritisk mot. Relationer framställs som platta och det reflekteras sällan över den flexibilitet som relationsmodellen faktiskt möjliggör. Naturligtvis håller vi med Date i hans kritik och vi skulle hellre vilja se en mer nyanserad jämförelse mellan modellerna i fler sammanhang. Vi vill dock tillägga att de vi intervjuat inte har uttryckt sig på det sättet.

4.1.3 Prestanda och optimering

Effektivt utnyttjande av index kan radikalt förbättra prestanda och minska I/O-belastningen (Inmon et al., 1999; Date, 2004; Imhoff et al., 2003). Det är inte bara viktigt att indexera sina databaser utan också att anpassa indexeringen efter de speciella förutsättningar som finns i varje databas och vilken typ av användning den är till för. Det bekräftas av Respondenten på Företag A som börjar ha behov av utökad indexering för att förkorta responstiderna. Deras mest detaljerade kub innehåller ganska mycket data och behöver indexeras för att det skall gå fortare att göra utsökningar mot den. Det är inte konstigt att det behövs effektiv indexering när de har 18 dimensioner i en kub och frågor som ställs mot den kan kräva många join-operationer.

Respondenten tar upp att det tar lång tid att ställa frågor om de ställs mot fält i datalagret som inte är indexerade, eftersom det finns så stora mängder data där. Men tanken är inte att frågor skall ställas mot datalagret utan mot kuben. Det kan ändå vara viktigt att även indexera datalagret om frågor inkluderar data därifrån. Om det finns behov att komma åt detaljerad data från datalagret så är det rimligt att tro att det ofta, relativt sett, är stora datamängder som efterfrågas och då är ett effektivt index värdefullt.

B-trädindex är bra att använda när det går att förutsäga vilka frågor som ställs (Imhoff et al., 2003). Bitmapindex är bra för oförutsägbara frågemönster och passar därför bra i en data

mart. Det är viktigt att välja indexeringsteknik utifrån användningsområde och det är inte självklart att bitmapindex är det som passar bäst i alla lägen, speciellt om attributen kan anta många värden. Eftersom poster i datalager och data marts inte uppdateras är inte B-trädets flexibilitet lika betydande som i ett operativt system. Dessutom är bitmapindexens svagheter vid uppdateringar inte lika påtagliga eftersom poster läggs till i stora mängder åt gången och indexen beräknas om.

Längden på en bitmapvektor är direkt relaterad till storleken på tabellen (Imhoff et al., 2003). Indexen kan bli väldigt stora, speciellt om tabellerna är stora. Index och aggregat för prestandaoptimeringar ökar datavolymerna kraftigt i ett datalager och kan tillsammans utgöra 5 - 6 gånger storleken av den ursprungliga affärsdatan (Winter Corporation, 2003). När datalagret optimeras bör det finnas en medvetenhet om förhållandet mellan storleken på affärsdatan och det utrymme som krävs för prestandaoptimeringar. En avvägning av kostnaden för det extra lagringsutrymmet mot behovet av ökad prestanda behöver göras.

Eftersom data normalt inte uppdateras i datalagret kan det vid den fysiska datadesignen tas hänsyn till hur många I/O-förfrågningar som krävs för att läsa in data (Inmon, 2002). Data som har hög sannolikhet att efterfrågas samtidigt placeras tillsammans på disk. Eftersom data i ett datalager alltid är relevant ur något tidsperspektiv är det ofta naturligt att, baserat på en lämplig tidsenhet, lagra data tillsammans. Det går att spara mycket I/O-belastning på att fysiskt spara data tillsammans om den också ofta läses tillsammans, det är självklart att om en stor mängd poster kan läsas sekvensiellt från disk blir det betydligt färre läsningar. Eftersom nästan all data har en tidstämpel och ofta efterfrågas ur ett tidsperspektiv ser vi en stor potential i att optimera diskläsningarna. En fördel är att det inte påverkar den logiska designen.

Det går även att dela upp tabeller i delar baserat på hur ofta olika attribut efterfrågas, attribut som mycket mer sällan efterfrågas kan flyttas ut i en egen tabell för att poster med attribut som efterfrågas ofta skall hamna mer kompakt tillsammans på disk (Inmon, 2002). Att dela upp tabeller för att kunna lagra de mest efterfrågade attributen kompaktare verkar vara effektiv metod så länge konsistensproblemen beaktas. Även i det här fallet skulle I/O-belastningen minska eftersom varje läsning ger fler poster. Vi antar att Inmon menar att det skulle bli ett enkelt ett till ett-förhållande mellan delarna i den uppdelade tabellen vilket kanske inte i sig gör en E/R-modell speciellt mycket mindre överskådlig, men vi ser en risk med att modellen skulle bli helt obegriplig om ett stort antal tabeller delades upp på det sättet. Eftersom den logiska designen påverkas tycker vi att det här sättet bör användas efter noga övervägande om var det går att spara mest I/O-förfrågningar.

Inmon (2002) anser att det går att minska antalet I/O-operationer genom att lägga till relaterade attribut redundant i tabeller där de ofta efterfrågas tillsammans med andra attribut. Även om den här metoden också lider av att det logiska schemat förändras och blir mer otydligt så finns det fördelar. Den utnyttjar förutsättningen att data normalt inte uppdateras i ett datalager och att det därför inte uppstår konsistensproblem mellan de redundanta attributen och att det då går att minska antalet join-operationer. Eftersom ingen join behövs är sannolikheten för att datan ligger nära varandra på disken stor.

Om vissa beräkningar utförs regelbundet kan kolumner läggas till i tabellen för att spara de aktuella värdena första gången de beräknas (Inmon, 2002). Om det i förväg är känt att användarna regelbundet är intresserade av specifik information går det att utföra beräkningarna medan primitiv data laddas in till ett datalager. Eftersom all data ändå skall behandlas vid inläsningen krävs det relativt lite extra datorkraft för att utföra de här beräkningarna. Det är en mycket effektivt att använda aggregerad data på de här sätten. Naturligtvis blir det en avvägning mot hur mycket extra diskutrymme som krävs, men de här båda sätten sparar uppenbart mycket beräkningkraft och I/O-belastning om det är aggregerad data som motsvarar vanligt efterfrågad data. Däremot kan vi hålla med Date om att det är bra att spara aggregerad data skiljt från primitiv data.

Att partitionering av data i ett datalager är viktigt framgår både av Date (2004) och Inmon (2002). I BW utnyttjas databashanterarnas inbyggda funktionalitet för partitionering och dessutom finns möjlighet för logisk partitionering. För att göra de stora datavolymerna hanterbara är det möjligt att dela in tabeller i partitioner baserat på t.ex. vilket år datan härrör från. Data som efterfrågas ofta kan tilldelas en egen diskuppsättning och en egen processor, vi ser att det är en stor fördel om nyare data har mycket större sannolikhet för att efterfrågas än äldre. Naturligtvis kan det finnas andra lämpliga sätt att partitionera beroende på hur datan används.

Hur prestanda ökar om dimensionstabeller används i stället för relationstabeller beror enligt Ingemarsson på hur stora dimensionstabellerna blir. Eftersom optimeringen bygger på att den kartesiska produkten beräknas på dimensionstabellerna för att sedan joinas med faktatabellen så blir produkten mycket stor om dimensionstabellerna är stora. Då är det bättre att använda en annan optimeringsstrategi.

4.1.4 Detaljerad data

Thodenius anser att företagen samlar på sig för mycket data och säger att det är viktigt att tänka igenom vad rapporterna skall användas till. Han säger att det är viktigt att

organisationen har någon form av strategi för vad de vill ha och de måste bedöma vilken detaljnivå som är rimligt. Det går inte att spara alla detaljer, det bli ohanterligt för ett stort företag.

Ibland utökas rapporterna för att någon vid något tillfälle ville ha något nyckeltal om t.ex. kunder, då läggs det till i rapporten. Till slut kanske det finns 30 nyckeltal med i rapporten medan det bara är 10 som används. Det kan till viss del bero på att användarna inte riktigt vet vad de har behov av. Då är det bra att sätta sig ner och gå igenom vad som är viktigt, vad det är som används. Vår bedömning är att det kan innebära att företagen regelbundet omprövar vilken data man skall samla i sitt datalager.

Thodenius säger att det kan vara ganska svårt att ha en strategi från början, nya situationer kan uppstå och då behövs viss information, men företagen måste bedöma vad som är rimligt att spara. Även Inmon (2002) och Ringquist säger att det kan vara omöjligt att på förhand avgöra vilken detaljnivå som är lämplig. Det är, som Inmon säger, först när användarna ser vad de kan göra som de kan säga vad de verkligen behöver. Han föreslår att profile records tas fram, där detaljnivån minskas i iterationer, eller att flytta över detaljerad, sällan använd data på billigare lagringsmedium. Att arkivera data är flexibelt då det alltid går att ta fram detaljerna vid behov. När datan blir äldre minskar sannolikheten för att den skall efterfrågas, en stor del av den historiska datan kan aggregeras eller arkiveras.

Hög detaljrikedom kräver inte bara mycket lagringsutrymme utan påverkar även prestandan då många fler poster måste gås igenom (Inmon, 2002). Det krävs att en noggrann avvägning görs baserat på vilken typ av frågor som det finns behov av att ställa mot databasen.

Ingemarsson säger att det går att ha hur detaljerad data som helst i ett datalager och menar att trenden går mot att spara mer detaljerad data. Hur detaljerad data organisationerna har eller vill ha kan kanske behöva begränsas av prestandaskäl eftersom det blir stora volymer data i datalagret. BW tillåter att användaren borrar sig ner från kuberna till ODS-objekten vid behov vilket minskar behovet av att lagra detaljerad data i kuben.

Ringquist säger att detaljnivån måste utgå från organisationens behov men säger att det finns en risk med att bara utgå från stundens behov och inte fundera över framtida användningsområden. Det är viktigt att verksamheten bestämmer sig för vad de vill ha med i datalagret vid utvecklingen, vilken data som skall gå att få ut ur systemet och vad de vill göra med den. Det gäller att ha ett pragmatiskt angreppssätt.

Respondenten på Företag A säger att han inte har hittat några direkta brister i BW men om han skulle nämna något så skulle det vara prestandan, men säger att de kan använda sig

betydligt mer av index och aggregat för att förbättra prestandan. Största prestandaproblemet är att de inte har hittat rätt aggregationsnivå upp till kuberna som i dagsläget innehåller för mycket detaljerad information. Rapporterna tar för lång tid att köra när det finns för mycket detaljer i kuberna.

Enligt Thodenius går det inte att generalisera om vilken detaljnivå för datan som är lämplig, för data som är viktig för att styra verksamheten kan hög detaljnivå vara nödvändig. Han säger också att informationen aldrig blir för gammal, men att det kan vara lämpligt att arkivera den detaljerade datan och plocka fram den vid behov. Företagen vill alltid ha bättre information än konkurrenterna.

4.2 Slutsatser

Vi har försökt ge en bred översikt över vad ett datalager är och används till ur ett tekniskt perspektiv. Vi har inte hittat vetenskaplig litteratur som behandlar hela problematiken kring hur ett datalager i praktiken bör designas och implementeras. Den litteratur vi har gått igenom är antingen brett inriktad men tar bara upp ett synsätt eller enbart fokuserad på specifika delar.

Utifrån vår analys har vi kommit fram till följande slutsatser som svarar mot vår problemformulering och vårt syfte men även sträcker sig över delar vi inte kunde förutse när vi påbörjade arbetet.

- Det finns ett mycket stort behov av att samla in stora mängder data från olika typer av operativa system till ett datalager för att möjliggöra analyser för beslutsunderlag. Även om det initialt är ett omfattande arbete som kräver mycket resurser ger det stor nytta för verksamheten. När datalagret är på plats finns en gemensam databas, med gemensam semantik, som kan användas till många ändamål och som är flexibel nog att anpassas efter framtida behov.
- Datalager utnyttjas mest av beslutsfattare och chefer på mellannivå, den högsta ledningen är mer intresserad av ett fåtal utvalda variabler att hålla reda på. De vanligaste användningsområdena är att ta fram rapporter och sammanställningar men också att göra analyser för att förstå olika samband. Det finns även ett behov av att ta fram prognoser baserat på analyserna för att anpassa verksamheten efter olika förhållanden.
- Det är viktigt att ha en pragmatisk inställning när datalagrets modeller skall utformas, användarna är intresserade av rapporter och information, inte av datamodeller. Det viktigaste med ett datalager är att ge användarna tillgång till

detaljerad historisk information inom rimlig tid och erbjuda en gemensam grund för rapporter och analyser.

- Datalagernivån behöver vara normaliserad för att vara så flexibel som möjligt och oberoende av användningsområde. I datalagret konsolideras data från de operativa systemen och utgör en gemensam grund för alla applikationer som utnyttjar datalagret. Rapporter som baseras på data från datalagret utgår från data med samma innebörd och värden vilket ger beslutsfattarna en enhetlig bild av verksamheten.
- På data mart-nivån, där prestanda är det primära, behöver tabellerna inte vara normaliserade eftersom den interna strukturen i kuberna inte påverkar andra användningsområden. Modelleringen av kuberna kan optimeras för prestanda och användarvänlighet. Starschemat är bra till rapportering och funktionaliteten för roll up, drill down och hierarkier förenklar arbetet för användaren eftersom rapporterna blir mer flexibla. Däremot måste problem med datainkonsistens beaktas noga för att nya problem inte skall uppstå.
- Eftersom volymerna i ett datalager blir enormt stora på grund av detaljrikedomen krävs en strategi för vilken typ av data som skall samlas in. Användarnas behov förändras över tiden och därför är det även viktigt att ompröva sina beslut för att kunna fortsätta använda datalagrets funktionalitet och kapacitet optimalt.
- Det är viktigt att ta hänsyn till prestandaaspekter för att inte få ett datalager som växer okontrollerat och till slut upphör att vara flexibelt. Till exempel behöver data indexeras, aggregeras, partitioneras och arkiveras. Men det gäller att vara uppmärksam på att indexering och aggregationer ökar lagringsbehovet kraftigt, det kan till och med kräva mer utrymme än den primitiva datan i sig.
- Vi saknar en mer nyanserad jämförelse mellan relationsmodellen och dimensionsmodellen. Vi tror att det finns ett behov av att utreda sambanden mellan redundans, prestanda och datavolymer. Går det att optimera förhållandet dem emellan? Lösningarna på prestandaproblemet får enorma konsekvenser för datalagringen idag, vilket i sin tur till slut ger återverkningar på prestandan. Det finns inget resonemang kring, och väldigt lite reflektioner över, vad det innebär att använda dimensionsmodellen och när det är bäst att avstå. Det skulle kanske gå att optimera kuberna ännu mer för prestanda än vad som görs idag. Det är

anmärkningsvärt att det finns så lite vetenskaplig litteratur om ett ämne som det redan för 50 år sedan uppstod behov för.

- Även om vi anser att dimensionsmodellen är bättre att använda i data marts än relationsmodellen tycker vi att det behövs utförligare teoretiska jämförelser och prestandatester mellan de båda modellerna för att tydligare se vilka vinster dimensionsmodellen ger och om det finns någon brytpunkt för när den ena modellen skall användas istället för den andra.

Referenser

Berild, S. (1997) *Data Warehouse – en introduktion*, SISU publikation 97:15, Kista

Christensen, L., Andersson, N., Carlsson, C. & Haglund, L. (1998) *Marknadsundersökning – en handbok*, Studentlitteratur, Lund

Date, C.J. (2004) *An Introduction to Database Systems* (8:e upplagan), Pearson Education, Inc./Addison Wesley, Boston

Devlin, B. (1997) *Data Warehouse – from Architecture to Implementation*, Addison Wesley Longman, Inc., Reading, Massachusetts

Elmasri, R. & Navathe, S.B (2003) *Fundamentals of Database Systems* (4:e upplagan), Pearson Education, Inc./Addison Wesley, Boston

Imhoff, C., Galemmo, N. & Geiger, J.G. (2003) *Mastering Data Warehouse Design – Relational and Dimensional Techniques*, Wiley Publishing, Inc., Indianapolis, Indiana

Inmon, W.H. (2002) *Building the Data Warehouse* (3:e upplagan), John Wiley & Sons, Inc., New York

Inmon, W.H., Rudin, K., Buss, C.K & Sousa, R. (1999) *Data Warehouse Performance*, John Wiley & Sons, Inc., New York

Inmon, W.H., Welch, J.D. & Glassey, K.L. (1997) *Managing the Data Warehouse*, John Wiley & Sons, Inc., New York

Kimball, R., Reeves, L., Ross, M. & Thornthwaite, W. (1998) *The Data Warehouse Lifecycle Toolkit – Expert Methods for Designing, Developing, and Deploying Data Warehouses*, John Wiley & Sons, Inc., New York

Kimball, R. & Ross, M. (2002) *The Data Warehouse Toolkit – The Complete Guide to Dimensional Modelling* (2:a upplagan), John Wiley & Sons, Inc., New York

Patel, R. & Davidson, B. (2003) *Forskningsmetodikens grunder* (3:e upplagan), Studentlitteratur, Lund

SAP (2002) *Data Warehousing with mySAP Business Intelligence*, Version 1.1

SAP (2003) *Enterprise Data Warehousing with SAP BW – An Overview*, Version 2.0

SAP (2004) *SAP Business Information Warehouse – Functions in Detail*, Version 3.0

Siberschatz, A., Galvin, P.B. & Gagne, G. (2003) *Operating System Concepts* (6:e upplagan), John Wiley & Sons, Inc., New York

Söderström, P. (1997) *"Data Warehouse" – Datalager: Verksamhet, Metod, Teknik*, Studentlitteratur, Lund

Thodenius, B. (2005) *Användning av ledningsinformationssystem – en longitudinell studie av svenska storföretag*, Ekonomiska Forskningsinstitutet vid Handelshögskolan i Stockholm, Stockholm

Wallén, G. (1996) *Vetenskapsteori och forskningsmetodik* (2:a upplagan), Studentlitteratur, Lund

Winter Corporation (2003) *SAP Business Information Warehouse – Multi-terabyte evaluation & feasibility test*

Internetkällor

Computer Swedens språkwebb (2005) http://computersweden.idg.se, avläst 2005-04-26

Lee, C-H. (2004) http://www.1keydata.com/datawarehousing/molap-rolap.html, avläst 2004-09-21

TechEncyklopedia (2005) http://www.techweb.com/encyclopedia, avläst 2005-04-26

Padron-McCarthy, T. (2003) En webbkurs om databaser,
http://www.ida.liu.se/~tompa/databaser, avläst 2005-04-29

Intervjuer

Anonym respondent (2005-04-13) BW-ansvarig, Företag A

Ingemarsson, C (2005-05-03) Nordic Business Development Manager, Business Intelligence & Analytics, SAP

Ingemarsson, C (2005-05-13) Nordic Business Development Manager, Business Intelligence & Analytics, SAP

Ringquist, D. (2005-03-21) Produktchef Business Intelligence, SAS Institute

Ringquist, D. (2005-05-18) Produktchef Business Intelligence, SAS Institute

Thodenius, B. (2005-05-06) Forskare vid sektionen för Information Management på Handelshögskolan i Stockholm