Datavetenskap

Mikael Forsberg

Mikael Morell

Optimering av SQL-frågor

för analys i QlikView

Examensarbete, C-nivå

2006:04

Optimering av SQL-frågor

för analys i QlikView

Mikael Forsberg

Mikael Morell

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.

Mikael Forsberg

Mikael Morell

Godkänd, 2006-01-18

Handledare: Katarina Asplund

Examinator: Stefan Lindskog

Sammanfattning

Huvudsyftet med detta examensarbete är att för vår uppdragsgivare ÅF:s räkning, analysera huruvida det är möjligt att, rent tidsmässigt, hämta lagrad information från en av företagets kunders databaser online samt presentera detta grafiskt, som statistisk information. Hänsyn måste tas till server- och nätverksbelastning. Vid analysen optimerade vi SQL-frågor, vilket resulterade i en minskning av den totala exekveringstiden på serversidan med 24 sekunder.

Den databasoberoende applikationen QlikView med sina inbyggda funktioner visade sig vara ett mycket användbart rapporteringsverktyg. Genom att utveckla ett grafiskt gränssnitt, tydliggjorde vi olika fråge- och svarsalternativ, vilka fungerar som underlag för sammanställning av produktionsdata.

Optimizing SQL-statements for analysis in QlikView

Abstract

The main purpose of this thesis is, on the behalf of ÅF, to examine how stored information can be gathered online in a timely fashion. Consideration must be taken to the server and network load. During our investigation we were able to optimize SQL queries that resulted in a reduction of the total execution time by 24 seconds at the server side. One analysis tool that was very useful during our assignment was QlikView, which turned out to be a database independent analysis and reporting tool. We have clarified some alternatives of the questions and answers by developing a graphical interface, which is the basis for putting together the information.

Innehållsförteckning

Figurförteckning

Tabellförteckning

1 Inledning

I dagens IT-baserade samhälle råder det ingen brist på information i organisationers databaser. Här samlas allt från kundregister till företagsekonomi beroende på ändamålet. Mängden av den information som lagras i databaserna gör att det är praktiskt taget omöjligt att analysera all information utan automatiserade tekniker.

Att extrahera och presentera statistisk information från de ansenliga datamängderna, som finns lagrade, har kommit att bli en viktig del för olika företag och organisationer. För att ha möjlighet att jämföra uppsatta mål och förväntningar, har kravet på lättåtkomliga informationsmöjligheter visat sig att vara av stort intresse inte minst med tanke på den tidsbesparande aspekten, som av naturliga skäl mynnar ut i ett ekonomiskt tänkande.

I utbildningen på dataingenjörsprogrammet ingår ett examensarbete. Vi har fått i uppdrag av ÅF System i Karlstad att undersöka huruvida det är möjligt att hämta statistisk information i onlineläge, eller om detta måste ske offline, genom överföring av batchfiler. För att grafiskt utveckla mätare har vi använt oss av sex frågor avsedda att användas vid en presentation av förväntningar och mål, vid ett företag eller en organisation. I vår redogörelse kommer vi att ta hänsyn till de av ÅF ställda kraven på sekretess och skyddande av företagets kunder.

1.1 ÅF System

[17] ÅF System är en av tre divisioner inom AB Ångpanneföreningen. Divisionerna i ÅF är System AB, ÅF Infrastruktur AB och ÅF Process AB. ÅF-koncernen har cirka 2750 medarbetare, som återfinns på ett femtiotal orter i Sverige, samt i ett tiotal länder i Europa.

Verksamhetsområdena inom ÅF System är produktutveckling och verksamhets- och affärssystem, där kundkategorin inom produktutveckling är den tillverkande industrin, samt inom verksamhets- och affärssystem. Division System har 350 medarbetare på nio kontor i Sverige och Norge.

De dotterbolag som finns i Karlstad är ÅF-Infrastruktur AB, ÅF-Kontroll AB och ÅF-Process AB. ÅF-Infrastruktur AB arbetar med olika infrastrukturella behov som finns i samhälle och näringsliv. Dess affärsområden är elkraft, installation, telecom, samt vägar och järnvägar. ÅF-Kontroll AB erbjuder ett komplett utbud av konsulttjänster inom kontroll, besiktning, provning, beräkning, certifiering och utbildning. ÅF-Process AB arbetar med att

säkerställa en kostnads- och miljöeffektiv energiförsörjning till näringsliv och samhälle. Uppdragsgivarna finns bland alla typer av energibolag, industrier, statliga myndigheter, verk, departement samt biståndsorganisationer.

1.2 Syfte och mål

Syftet med detta examensarbete är att undersöka huruvida det finns teknik för att hämta information via ODBC-uppkoppling, för att presentera denna statistiskt från en speciell databas. Målet är att för företaget presentera ett förslag till att hämta tidskrävande information online från hårt belastade databaser, eller om detta måste ske offline i intervaller via en batchfil. En annan målsättning är att utveckla olika mätare som statistiskt presenterar den inhämtade informationen.

1.3 Uppdragsbeskrivning

Vår uppdragsgivare har gett oss i uppgift att undersöka vilket tillvägagångssätt som är det bästa vad gäller att hämta upp och presentera statistisk information från en specifik kunds databaser. Hänsyn måste tas till både nätverk och serverbelastning.

Data skall samlas in från två delsystem baserade på Oracle. Mängden av information som finns i de underliggande databaserna uppskattas vara av storleksordningen 30 Terabyte.

Vår undersökning kommer att behandla huruvida det är möjligt rent tidsmässigt att presentera data online, eller om detta måste ske offline, med uppdatering av data i vissa tidsintervall per dygn.

Analysen av informationsinsamlingen kommer att utföras i PL/SQL Developer, vilket är ett utvecklingsverktyg för Oracle. I vårt arbete koncentrerar vi oss på sex frågor, vilka skall presenteras grafiskt med avseende på slutanvändarens behörighet. Via ett grafiskt gränssnitt kommer data att presenteras i form av diagram, mätare med visare eller liknande. Dock ska systemet via inloggning presentera nödvändig och övergripande information för respektive enhet och användare. Möjlighet till ”drill down” (se avsnitt 2.3.3) skall finnas för att se detaljnivå. Det grafiska gränssnittet kommer att utvecklas i verktyget QlikView från Qliktech.

1.4 Disposition av uppsatsen

Uppsatsen är i huvudsak uppdelad i sju huvuddelar: introduktions-, teori-, optimerings-, systembeskrivnings-, utveckling av mätare-, resultat- och diskussionsdel. Kapitel 2 är

teoridelen, där olika begrepp, tekniker, metoder och teorier beskrivs. I kapitel 3 presenterar vi rekommendationer för optimering av SQL-satser. I kapitel 4 förklarar vi det befintliga systemet samt olika systemlösningar. 5:e kapitlet beskriver utvecklingen av mätare i QlikView. I kapitel 6 presenterar vi de olika tester vi utfört genom optimering av de frågor som ställs för att generera batch-filer. Slutligen i kapitel 7 förs en diskussion av resultat, sammanfattning och presentation av förslag på lösning.

2 Teori

I detta kapitel redovisar vi det teoretiska ramverk vi använder oss av under resultat- och diskussiondelen. Vi förklarar begreppen datalager (2.1), OLAP (2.3), ETL (2.2) och hur de är relaterade till varandra genom ETL (2.4). Kapitlet introducerar även en presentation av QlikView (2.5) vilket har teknik för att hämta och sammanställa insamlad data från flera olika källor. En presentation av databasen Oracle (2.6) och programspråk, samt en utvecklingsmiljö som är avsedd för detta språk görs även.

2.1 Datalager

Ett datalager är, i sin enklaste perception, inget mer än en samling nyckelbitar av information som används för att hantera och styra affärsverksamheter mot fördelaktiga resultat. Generellt sett kan ett datalager beskrivas som en samling av beslutsfattande teknologier, riktat att möjliggöra för verkställande ledare att fatta fördelaktigare och snabbare beslut.

2.1.1 Definition

Bill Inmon [5], en pionjär och en av de mest omnämnda i datalagersammanhang, definierar ett datalager på följande sätt:

”A Data Warehouse is a subject oriented, integrated, time variant, and non-volatile collection of data in support of managerment’s decision and making process”, vilket fritt översatt betyder: Ett datalager är ett subjektinriktat, integrerat, tidsvarierat och en icke föränderlig samling av datastöd i en beslutsfattande utförandeprocess.

Nedan följer en beskrivning av innebörden för nyckelorden. [6]

- Subjektorienterad (Subjectoriented): Ett datalager är organiserat runt subjekt som till exempel kund, leverantör, produkt och försäljning. I stället för att koncentrera sig på de dagliga transaktionerna fokuserar ett datalager på modellering och analys av data för beslutsfattare.
- Integrerad (Integrated): Ett datalager har vanligtvis konstruerats genom integration av data från flera olika källor, exempelvis relationsdatabaser, textfiler eller från transaktionsdata. Datatvätt (se avsnitt 2.2) och dataintegrationstekniker utförs för

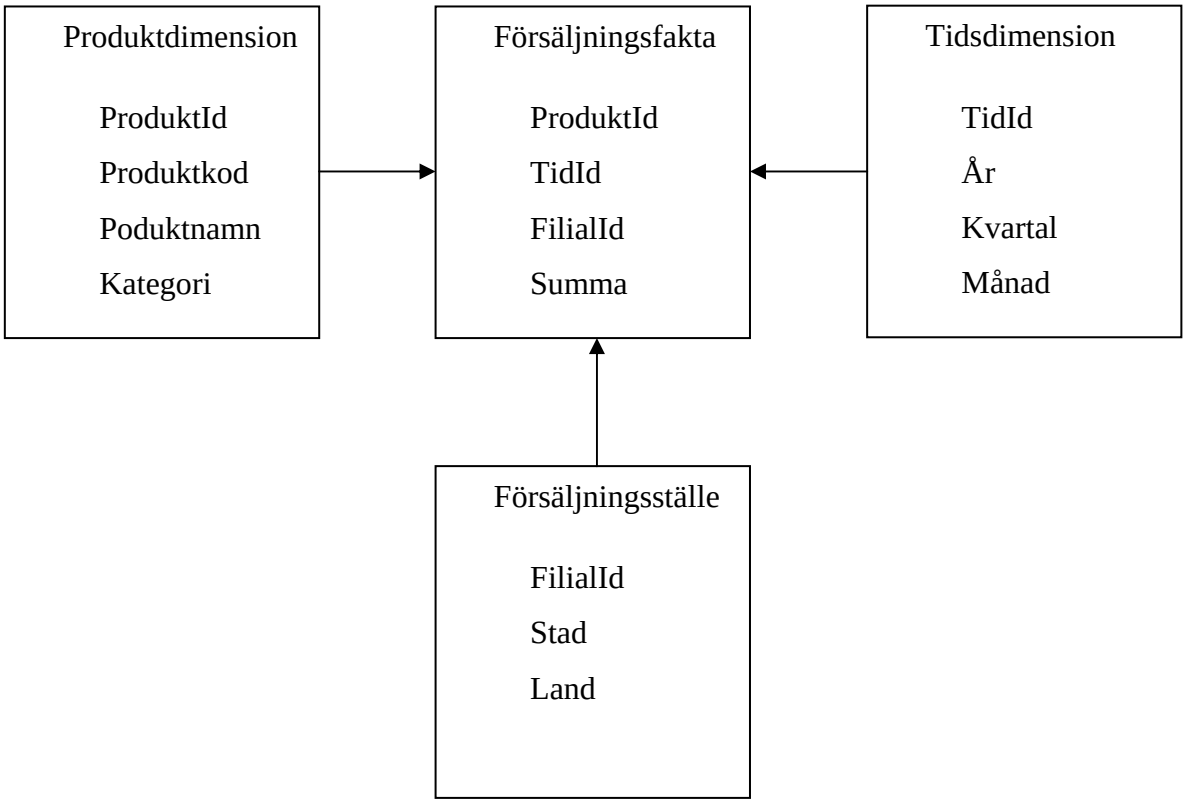
att lösa problemet med namnkonflikter och inkonsistenta mätvärden till ett enhetligt format.

- Tidsvariant (Timevariant): Den information som lagras i ett datalager tillhandahåller uppgifter i ett historiskt perspektiv, exempelvis 5-10 år tillbaka i tiden. Varje nyckelstruktur i datalagret innehåller, endera implicit eller explicit, ett element av tid.
- Icke ombytlig (Non-volatile): Data som lagrats i ett datalager ska inte uppdateras eller ändras. Uppdateringen sker i operationsmiljön innan data överförs till ett datalager.

Ett datalager är en central samlingsplats för all eller utvalda delar av data som ett företags affärssystem samlar in. Data från olika transaktionsprocesser och andra källor selekteras och organiseras i ett datalager, vilka analyseras via applikationer och frågor från användare. Datalagret betonar insamling och selektering av data från olika källor för värdefull analys.

2.1.2 Datalagerschema

Målet med ett datalagerschema är att konstruera en plan vilken presenterar en vy av summeringar eller aggregeringar av transaktionsdata. Vid kartläggning av dimensioner (se avsnitt 2.3.3) till tabeller är det viktigt att dimensionerna har ett enkelt fält med den primära nyckeln. Stjärnschemat är det vanligaste schemat som används i ett datalager och har fått sitt namn genom att gruppera flera dimensioner, runt en faktatabell, som en stjärna [9]. I figur 1 nedan har dimensionstabellerna länkats samman till faktatabellen genom främmandenycklar.



Figur 1: Stjärnschema

För att tydliggöra en summering av den totala försäljningen för till exempel produkt B under det första kvartalet, är det nödvändigt med en sammanslagning av de tre dimensionstabellerna. Nedan följer ett exempel på en SQL-sats för att utföra detta.

```
SELECT SUM(Försäljningsfakta.Summa)
FROM Tidsdimension, Försäljningsställe, Produktdimension
WHERE Produktdimension.ProduktId = Försäljningsfakta.ProduktId
  AND Försäljningsställe.FilialId = Försäljningsfakta.FilialId
  AND Tidsdimension.TidId = Försäljningsfakta.TidId
AND ProduktDimension.ProduktNamn = ’ProduktB’
AND TidsDimension.Kvartal = 1;
```

2.2 ETL

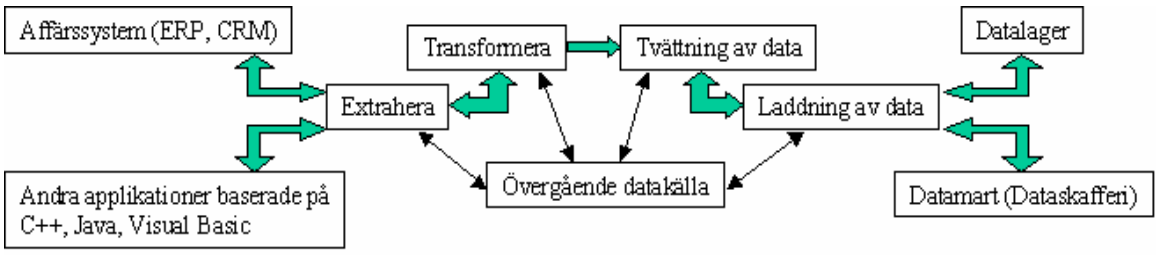
Företag har ofta data, i en eller flera databaser, som behöver kunna bli förflyttade från en plats till en annan, till exempel från ett affärssystem till ett annat, eller till ett datalager för att sedan kunna analyseras.

ETL (Extract Transform Load) [12] är processen som möjliggör för företag att förflytta data från många källor till en annan databas, eller till ett datalager (se Figur 2). Ett av momenten som ingår är att formatera om datamängden och göra den entydig, för senare analys. Problemet är att data kan ligga på ett antal olika system och därför i ett stort antal format. För att lösa detta problem använder sig företagen ofta av en ETL-mjukvara, vilket innebär inläsning av data från dess källa, omstrukturering av den för att vidare skickas till destinationen.

Data i en ETL-process kan komma från en mängd olika datakällor, till exempel en applikation som ligger på en stordator, ett affärssystem, en text-fil eller ett Excel-blad.

Extraktionen kan göras med hjälp av JDBC (Java Database Connectivety) eller något annat databaskopplings-program. Ett annat sätt är att skapa vanliga textfiler. Efter extraktionen förändras datan så att den kan skickas till sin destination.

Extraktionen kan utföras på flera sätt. Ibland kanske data endast behöver omformateras, men de flesta ETL-operationer innebär att dupliceringar måste tas bort för att tvinga fram en konsistent datamängd. Applikationen sätter upp regler för datafälten så att data konverteras till det format som destinationen kräver. Till exempel kan ordet manlig representeras som M, Manlig eller Man. ETL-verktyget kan konfigureras för definition av dessa ord vilka har samma betydelse, detta kallas för datatvätt. Orden konverteras sedan till det format som benämns i destinationen. Transformationen uppstår när data från olika källor struktureras så att data kan bindas samman, till exempel att fordringar blir bundna till fakturor. Efter det att data har bundits till varandra transporteras och laddas data in i datalager för senare analys med avseende på till exempel tidsperioder och uteblivna fordringar.



Figur 2: ETL-process

Av de största leverantörerna av ETL-system kan nämnas Microsoft som erbjuder data-transformation i sin SQL-server databas. Oracle har inkluderat viss ETL-funktionalitet i sin version 10g och IBM erbjuder en integrationskomponent för DB2 i sina datalagervarianter.

Det finns även tredjepartsprodukter som erbjuder integration mellan ett större antal applikationer och datastrukturer, såsom Informatica, Ascential Software Corp och Mass and Hummingbird Ltd.

2.3 OLAP

Termen OLAP [10] står för Online Analytical Processing och handlar om att få tillgång till aktuell data online för analys. OLAP är beteckningen på multidimensionella analysverktyg, som skapats av E F Codd[1]. Fördelarna med dessa produkter är att de kan köras i en vanlig PC, medan de förhållandevis snabbt kan hantera stora datamängder. Dessa analysinstrument är populära hos bl.a. ekonomer.

En annan av fördelarna med OLAP är att användargränssnitten, för flertalet produkter, är förhållandevis enkla och kan användas av personer som saknar teknikerutbildning. Data kan inhämtas från datalager eller operativa system in i arbetsstationen, som i flertalet fall är en PC, och lagras i denna.

Att ha till uppgift att specificera dataupphämtning till OLAP-verktyg kan medföra ett omfattande programmeringsarbete. Man kan emellertid utföra arbetet direkt i OLAP-verktygen. I vissa system kan man finna inbyggda funktioner för inhämtande av data. Ett vanligt använt upphämtningssätt är så kallad drill-through[2] där data förs över från en relationsdatabas via det vanliga OLAP-gränssnittet.

De strukturella egenskaperna för OLAP är emellertid begränsade. OLAP-verktygen hanterar helst data som är av ren stjärn-struktur, vilket innebär att dessa helst ska bestå av en sorts fakta som beskrivs med en kvantitet av dimensioner. Data består nästan alltid av olika typer av fakta. I ett OLAP-system beskriver man detta i ett kubsystem (se Figur 3). För att fungera på ett tillfredsställande sätt vill de enklare verktygen inte ha mer än fyra till sex dimensioner. Vad gäller aggregeringsnivåer bör dessa begränsas till tre till sex. Ytterligare begränsningar som man har att ta hänsyn till är antalet diskreta värden som återfinns på varje nivå. Om belastningen blir för hög för OLAP-verktygen får man använda sig av MOLAP och ROLAP vilka beskrivs nedan.

[1] E F Codd är samma person som skapade begreppet relationsdatabas.
[2] Drill through betyder att data förflyttas horisontellt mellan två noder, via en relaterad länk.

2.3.1 MOLAP

[9] Multidimensionell OLAP (MOLAP), ofta kallad MOLAP-server, är benämningen på ett system med OLAP-egenskaper där lagringen sker i en multidimensionell kub. MOLAP-kuber är strukturerade för att kunna lagra stora mängder data samt för att hämtning av data ska ske snabbt. Data ligger inte i arbetsstationen utan på en server, som kan vara en stordator.

2.3.2 ROLAP

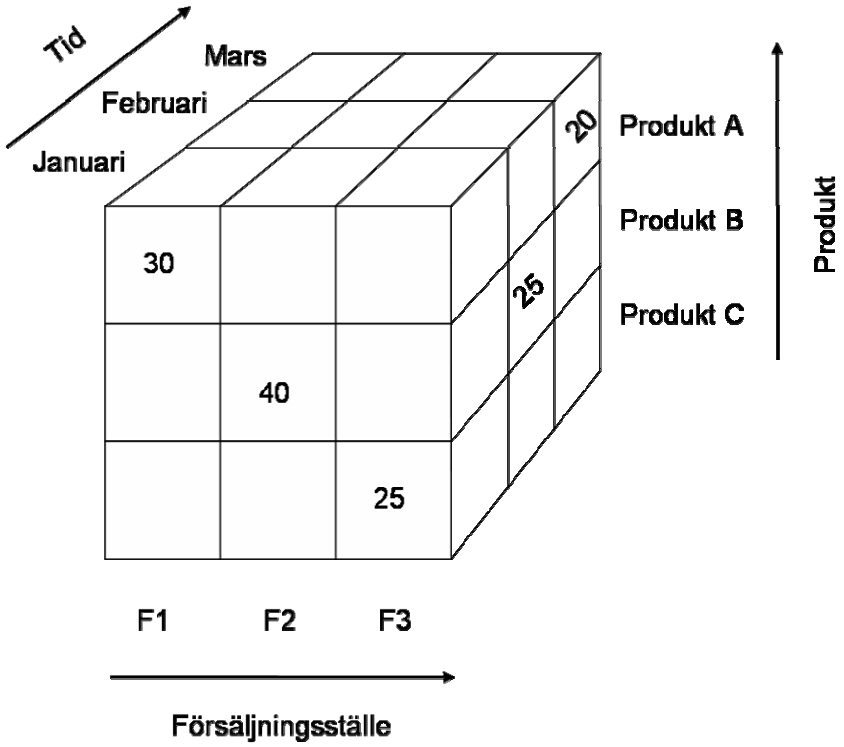
[10] Relational OLAP (ROLAP) är benämningen på ett system med OLAP-egenskaper där lagringen sker i en relationsdatabas. Gränssnittet mot användaren finns i klienten och servern är den som utför databasoperationerna. Förfaringssättet är mycket gynnsamt om volymen av data är omfattande och om typerna av sammanställningarna är varierande. Den effektiva databashanteraren söker och sammanställer data och man är inte begränsad till att arbeta med kubsystemet (enskilda kuber) då man har tillgång till en relationsdatabas. Stjärn-strukturen förespråkas även här.

Databashanterarens egenskaper utgör skillnaden mellan de olika systemen MOLAP och ROLAP. Det sistnämnda gör att man slipper ifrån strukturella begränsningar samt får tillgång till den kapacitet som finns i en relationsdatabas.

2.3.3 OLAP-kuber

Alla OLAP-system är beroende av en sammanslagen vy av transaktionsdata och är känd som en multidimensionell konceptuell vy.

Relationsdatamodellen är den modell som används i traditionella databaser, vilken beskriver data som en samling tabeller och kan ses som en tvådimensionell matris. Vad beträffar datalager utökas modellen med flera dimensioner för att konstruera multidimensionella matriser så kallade hyperkuber [1]. Oftast benämns dessa kuber som OLAP-kuber.



Figur 3: OLAP-kub[3]

I en OLAP-kub är dimensioner och fakta två viktiga begrepp. Fakta är själva datan som man vill analysera och dimensionerna är det sätt man vill analysera datan på. I figur 3 ovan motsvarar varje cell i kuben antalet sålda produkter. Dimensionerna tid, produkt och försäljningsställe motsvarar olika grupperingssätt att summera data efter. Kuben är tänkt att analysera ett företags produktförsäljning under tidsperioden januari till mars. Företaget har tre försäljningsställen (F1, F2 och F3) som alla säljer produkten A, B och C. Tidsdimensionen anger när produkterna såldes och ger en möjlighet att gruppera produktförsäljningen vid olika tidsperioder. Det totala antalet sålda produkter under januari till mars är summan av alla celler, vilket här är 140 st. Genom att gruppera efter olika dimensioner kan man borra sig ner (Drill down) för att se en mer detaljerad nivå. För att analysera den totala försäljningen av produkt B under en tidsperiod, kan tillvägagångssättet ses som om man skär ett snitt ur kuben. Via en gruppering av produkt B blir summan av cellerna 65, vilket är den totala försäljningen av produkt B under perioden januari till mars.

2.3.4 Leverans av data till OLAP-kuber

Serverbaserade OLAP-produkter är komponenter som blir mer och mer populära att använda i datalager. OLAP-servern innehåller två primära funktioner:

[3] Bilden är ett modifierat exempel ur boken [9]

- Prestanda på frågor (select i SQL)

  Använda aggregering, specialiserad indexering och lagringsstrukturer. OLAP-servern hanterar automatiskt aggregering och indexering, en fördel som kommer till sin rätt genom att hantera aggregerings-tabeller på rätt sätt

- Analyserbar data

  Att använda språk som, olikt SQL, är designade för komplex analys. OLAP-servern har även mekanismer för att lagra komplexa beräkningar och inställningar för säkerheten på servern. Vissa OLAP-verktyg har data-mining (datautvinning) integrerat.

2.3.5 Datakällor i en kub

Den bästa källan för OLAP-kuben är ett dimensionerat datalager [7], vilket är lagrat i en relationsdatabas. Språket för ett dimensionerat datalager, dimensioner, nycklar, hierarkier, attribut och fakta översätts exakt till OLAP-dimensionen. OLAP-motorer utvecklas först och främst för att kunna hantera snabba och komplexa frågor i dimensionerade strukturer. Olikt relationsdatabasen och ETL-verktyget, så är inte OLAP-motorn designad för att rensa data eller att försäkra sig om referensintegritet. Även om det är möjligt att skapa en kub direkt från en källa där transaktioner utförs, ska man undvika detta. Man bör istället skapa ett datalager med relationer, som man sedan ställer frågor till.

De varierande server-baserade OLAP-produkterna och versioner som finns att tillgå erbjuder olika funktionalitet. En av skillnaderna är de skiftande källorna, vilka för över data till OLAP-kuben.

En del produkter kräver att data kommer från en textfil, andra kräver att källan är en relationsdatabas, medan andra tillåter information kommer från olika källor. Om källan är textfiler, är det sista steget i ETL-processen att plocka ut de relevanta seten av data. Även om det är sämre prestanda och mindre elegant att ladda data från en fil än direkt från en relationsdatabas så är det inte säkert att det är det sämsta valet. Det är möjligt att skriva ut resultatet till en fil i alla relationsdatabaser och ETL-verktyg. Det är viktigt att se till att alla frågor som kuben ställer mot datalagret är optimerade. Om det är nödvändigt att lägga in index eller vyer i datalagret, så bör man göra detta.

2.3.6 Vanliga fel och problem

[7] Ett av de vanligaste felen som kan uppstå under processen att ladda fakta till kuben, är felaktig referensintegritet. En rad som laddats in i faktatabellen har ingen tillhörande dimensionsmedlem. Om man följer de råd som finns i de böcker som skrivits om detta ämne och använder surrogatnycklar, så kommer man inte att konfronteras av detta problem.

2.4 Integrering av OLAP-processen i ETL-systemet

Om datalagret innehåller OLAP-kuber [7], ska dessa administreras lika professionellt som alla andra delar av systemet. Detta betyder att man ska ha avtal med leverantören angående dataladdning och systemets tillgänglighet. Man brukar föredra att publicera data med samma schema som övriga relationsdatabaser, det kan även vara acceptabelt att förnya data i en kub med längre intervall än i relationsdatabasen. Det viktigaste är att man kommer överens om ett intervall med användarna och håller sig till detta. Tillverkarna av OLAP-servrar har inte gjort de framsteg som man kan förvänta sig med att tillhandahålla verktyg för att hantera OLAP-databaser på ett professionellt sätt.

2.5 QlikView

Qliktech är ett svenskt företag som etablerat sig på marknaden under de senaste åren och som erbjuder produkten QlikView [16]. QlikView använder en patenterad teknik, Associative Query Logic (AQL), som innebär att istället för att aggregera data till en icke-flexibel kub-struktur (se avsnitt 2.3.3), definieras de tabeller och fält som man vill läsa in i applikationen via ett skript. Vid exekvering av skriptet läses data in ner på transaktionsnivå och sparas i applikationen. Resultatet kallas för ett ”datamoln” som genom AQL kan användas för att göra en avancerad analys helt oberoende av fördefinierad aggregering. AQL-tekniken beskrivs vidare i nästa avsnitt 2.5.1.

QlikView bygger på att aggregera data i datorns primärminne, vilket medför en mycket snabb åtkomst av data. En annan fördel med QlikView är att den integrerar alla typer av data. Det gäller allt från att hämta data från relationsdatabaser, text-format, Excel-blad till XML. Via enkla instruktioner eller genom en guide för ett ETL-script läses data in, tvättas och lagras i QlikView. Enligt Qliktech anpassas applikationen till organisationens strukturer och inte tvärtom. Vare sig data är lagrat i ett datalager eller i en transaktionsdatabas, så finns de alltid tillgängliga för QlikView.

2.5.1 AQL

AQL [16] är en metod och en enhet för att extrahera information från en databas, där ett flertal dataposter inkluderas. Varje post innefattar minst två element vilka definieras av en typ och ett värde. I första steget läses dataposterna in i datorns primärminne och all bearbetning kan ske offline. I nästa steg kodas dataelementen binärt och lagras i en eller flera tabeller. När selektion av minst ett dataelementvärde upptäcks kommer alla binärkodade värden att granskas för att inhämta alla dataposter som utförts i urvalet. Med hjälp av binärkodning kommer dessa poster att mycket snabbt hittas. Slutligen erhålls de dataelementvärden, som innefattas av urvalen, vilka hålls reda på i en statusmängd.

2.6 Oracle databas

Oracle [14] är en av världens största dataföretag och är också den största tillverkaren av databashanterare vilka har samma namn som företaget. För snart trettio år sedan såg Larry Ellison en möjlighet som andra företag hade missat när han kom över en beskrivning av en fungerande prototyp för en relationsdatabas. Han upptäckte också att inget annat företag tidigare hade tagit ställning till att kommersialisera teknologin. Ellison och hans medgrundare Bob Miner och Ed Oates insåg att det fanns en enorm affärspotential för relationsdatabasmodellen, men det som de inte insåg var att det skulle ändra utseendet för affärsberäkningar för alltid.

2.6.1 PL/SQL

PL/SQL (Procedural Language/SQL) [2] är ett programspråk framtaget av företaget Oracle baserat på SQL. Jämfört med SQL har språket vissa tillägg och modifikationer som är specifika för Oracles databashanterare. I PL/SQL ges det möjligheter att skriva procedurer, funktioner och paket som lagras i Oracle-databasen. Skillnaden mellan en procedur och en funktion är att en funktion alltid returnerar ett värde, som alltid måste tas om hand, medan en procedur inte returnerar något värde.

2.6.2 Utvecklingsverktyg för Oracle

PL/SQL Developer är en integrerad utvecklingsmiljö (IDE) för att utveckla lagrade programenheter i en Oracle-databas. Applikationen erbjuder många möjligheter för en programmerare, t ex. optimering av SQL-satser för programenheter via funktionen ”Explain Plan” (se avsnitt 3.3.1), som visar statistisk information för den exekverande SQL-satsen i form av total exekveringstid och CPU-tid.

3 Optimering

Ett av de viktigaste områdena inom databastekniken är optimering, beroende på att användaren har ett mycket stort intresse av att på så kort tid som möjligt få svar på sina frågor. Eftersom vi ska utreda om det är möjligt att köra de berörda SQL-frågorna online, så har vi valt att ta upp några olika rekommendationer för optimering

Detta kapitel tar upp prestandakrävande operationer (3.1), prioritering vid datainsamling (3.2) och slutligen riktlinjer vid optimering av SQL-frågor (3.3), vilket innehåller de olika metoder som vi använt oss av för att optimera med.

3.1 Prestandakrävande operationer

En av de största faktorerna när det gäller prestanda för databaser och SQL-satser är typen av bearbetning som äger rum i en databas. När man talar om transaktionell bearbetning refereras det till två olika typer, användarinmatning och batchladdning.

Vanliga inmatningar från användare innehåller SQL-satser som INSERT, UPDATE och DELETE. Dessa typer av transaktioner utövas ofta av användare eller kund via ett grafiskt gränssnitt, vilket innebär att användare sällan kommer i kontakt med SQL-satserna på ett synligt sätt.

En batchladdning innehåller mängder av transaktioner mot en databas på en gång. Till exempel kan man tänka sig en överföring av senaste årets data till en massiv historisk tabell. Batchladdningar är kända för att vara betungande för system och databasresurser. Dessa resurser kan innehålla tabellaccesser, systemkatalogaccess, rollback-segment och sortering av areautrymme. I stort sett tar batchladdningar det mesta av processorkraft och minnesutrymme beroende på operativsystem och databas.

3.2 Prioritering vid datainsamling

Oracle [9] har tagit fram en prioriteringslista med rekommendationer som kan fungera som handledning för att effektivisera arbetsprocessen.

- **Förändra affärsreglerna**. Det kan till exempel finnas en regel i företaget att en chef måste attestera en utbetalning som överstiger 2000 kronor. Detta kan medföra att det

kommer att finnas en kö hos denna chef på attesteringar. Det kanske är möjligt att regeln höjs till 10000 kronor.

- **Datadesign**. Vissa värden kanske beräknas varje gång som det efterfrågas och detta sker regelbundet. Det är då kanske bättre att lagra det beräknade värdet i en egen kolumn.
- **Applikationens design**. Ett exempel här är att ett värde inte ändras så ofta. Då kan det vara värt att endast hämta detta värde en gång om dagen, till exempel på morgonen.
- **Den logiska strukturen i databas**, vilket innebär att se till att en tabell varken är över- eller underindexerad.
- **Operationer i databasen**. Det gäller att vara säker på att applikationen utnyttjar de fördelar som finns i SQL för att applikationens process ska bli så snabb som möjligt. Att förstå hur SQL-uttrycken exekverar i applikationen, kan leda till förbättringar i uttrycken.
- **Accessvägar**. Försäkra sig om att det finns effektiva accessvägar. Om en applikation har en låg nivå av träffar i en stor tabell, kan en hash-struktur vara lösningen. Om det alltid byggs ett temporärt index på en icke indexerad tabell, kan lösningen vara att detta index sätts permanent.
- **Minnesallokering.** Minnesallokeringen kan explicit sättas för buffertens cache, loggningsbuffertens cache samt den sekventiella cachen. Att utföra detta på ett korrekt sätt kan ha positiv effekt på prestandan.
- **I/O och fysisk struktur**. Att distribuera data över ett antal diskar, så att I/O distribueras och att koncentrationen på en disk undviks kan till exempel öka prestandan. Man bör även se till att tabellerna får tillräckligt stort utrymme från början och inte utföras detta dynamiskt, då detta påverkar prestandan negativt hos OLTP[4]-applikationer.
- **Konkurrens av resurserna**. Många användare kan leda till konkurrens av resurser. Dessa typer av konkurrens bör beaktas, block, delad pool och låsning. Det finns en hel del diagnosiska verktyg för att upptäcka dessa problem.
- **Den underliggande plattformen**. Detta är specifikt för implementationens hårdvara, operativsystem och nätverkskonfigurationen. På Linux är det till exempel möjligt att fininställa buffertens cache.

[4] OLTP (On-Line Transaction Processing.) samma som transaktionsutförande

3.3 Riktlinjer optimering av SQL-frågor

[13] Optimeraren kan inte använda en accessväg utan att denna gjorts tillgänglig för SQL genom indexering. Nedan följer några tips på hur man kan välja en viss accessväg.

3.3.1 Optimeringsverktyg i Oracle

[13] Om en SQL-sats inte är optimerad kan den vara ineffektiv, även om Oracle-databasen i sig är väl fungerande. Att en fråga är optimerad innebär bland annat att urvalet först görs på nycklar i stigande ordning och sedan på de övriga villkoren. I Oracle Server finns två användbara verktyg med benämningen EXPLAIN PLAN (förklara exekveringsplan), samt TKPROF (Transient Kernel Profiler) [15]. Dessa verktyg kan hjälpa till att öka prestanda på SQL-satsen. Här beskrivs dessa verktyg.

EXPLAIN PLAN [13] hjälper till att upptäcka accessvägen som används av SQL. Genom att skriva ”EXPLAIN PLAN FOR” före urvalet i PL/SQL, kommer urvalets plan att skrivas till en tabell som benämns PLAN_TABLE i databasen. Om inte denna tabell finns kommer PL/SQL att fråga om den ska skapas. Planen innehåller de urval som kommer att göras, samt de primärnycklar som kommer att eftersökas och i vilken ordning detta kommer att ske. Denna plan kommer att skapas och lagras i det delade lagringsutrymmet i Oracle både när man skriver data till databasen och när man gör ett urval i databasen.

[15] TKPROF skriver aktuell prestanda på SQL-frågan, med avseende på cpu-användning, till en loggfil. För att kunna använda sig av TKPROF, måste man aktivera ”SQL TRACE”, vilket enklast görs genom att skriva ”alter session set sql_trace=true” i kommandofönstret. Man kan få Oracle att inkludera information om exekveringstider i de genererade filerna. Detta görs genom att skriva ”alter session set timed_statistics=true”.

3.3.2 Kostnadsbaserad optimering och statistik

En kostnadsbaserad optimerare [8] fungerar genom att utgiften för olika exekveringsplaner beräknas. Viktigt är att välja den exekveringsplan som har den lägst beräknade kostnaden. Kostnadsmodellen är en viktig komponent i optimeraren, eftersom noggrannheten hos denna direkt påverkar förmågan att välja de bästa exekveringsplanerna. Avgiften baseras på noggrann modellering av varje komponent i exekveringsplanen. Kostnadsmodellen innehåller detaljerad information om alla accessmetoder och databasstrukturer i Oracle, så att en korrekt kostnad kan genereras för varje typ av operation. Modellen förlitar sig på optimerarens statistik, som beskriver objekten i databasen, samt hårdvaruprestandan. Det måste finnas aktuell statistik för att denna modell ska fungera. Det finns många verktyg i Oracle som

underlättar och automatisererar hämtningen av statistiken. Det finns inte endast en definition av kostnad. För vissa applikationer är målet att minimera tidsåtgången att hämta första raden, eller den första uppsättningen av rader, medan målet i andra applikationer är att returnera hela resultatet av frågan på en minsta möjlig tid. Oracles kostnadsmodell stödjer båda dessa mål genom att beräkna olika typer av kostnader, baserad på prestanda hos databasen.

3.3.3 Regelbaserad optimering

Oracle har dessutom en regelbaserad optimeringsmodell [8]. Oracle slutade att ge stöd till denna modell i Oracle 10g version 1, medan den stöds i version 2 för att det ska finnas bakåtkompabilitet. Eftersom de flesta av Oracles kunder använder den kostnadsbaserade optimeringen, så har vi endast beskrivit denna.

3.3.4 Hints

[11] Man kan använda kommentarer i SQL genom att skicka instruktioner, eller hints till optimeraren i Oracle. Optimeraren använder dessa hints som förslag för att välja exekveringsplanen för uttrycket. Ett uttryck kan endast innehålla en kommentar med hints, denna kommentar måste komma efter nyckelorden SELECT, UPDATE, INSERT eller DELETE

Här visas de olika stilar på syntax som Oracle stödjer

{DELETE|INSERT|SELECT|UPDATE} /*+ hint [text] [hint[text]]... */

eller

{DELETE|INSERT|SELECT|UPDATE} --+ hint [text] [hint[text]]...

”hints” som används för optimering:

/*+ ALL_ROWS */	Väljer explicit det kostnadsbaserade sättet att optimera ett uttrycksblock, med målet bästa genomströmning (minimal användning av resurser)
/*+ CHOSE */	Detta får optimeraren att välja mellan kostnadsbaserade och regelbaserade sättet att optimera ett uttrycksblock, baserat på statistik för de tabeller som uttrycket berör.
/*+ FIRST_ROWS */	Väljer explicit det kostnadsbaserade sättet att optimera ett uttrycksblock, med målet bästa svarstid (minimal användning av resurser, för att returnera de första raderna)
/*+ RULE */	Väljer explicit det regelbaserade sättet att optimera ett uttrycksblock

Det här sättet att optimera uttryck bör användas med stor försiktighet, eftersom det inte är säkert att de finns kvar i nästa version av Oracle. Eftersom detta är specifikt till för en Oracle-databas, så kan vi inte använda detta i det här fallet.

3.3.5 Förstå sig på optimeraren

SQL [13] kan exekveras med optimering baserad på regler eller kostnad. Optimering baserad på regler är vanligast i gamla applikationer, många utvecklare har använt sig av denna typ av optimering i åratal och är nöjda med detta, men man bör använda sig av optimering baserad på kostnad i nyare applikationer. Oracle uppdaterar optimeringen som är baserad på kostnad till varje ny version, för att den ska bli mer och mer stabil och pålitlig. Man bör analysera schemat med jämna mellanrum om man väljer att använda detta sätt. Analysen lagrar statistik om databasen i datauppslagningstabellerna, som sedan kommer att användas av optimering baserad på kostnad. SQL kan endast optimeras med optimering baserad på kostnad. Om man planerar att uppdatera från optimering baserad på regler till optimering baserad på kostnad bör man utföra prestandatester på alla SQL-uttryck i hela databasen.

3.3.6 Tänk globalt när man utvecklar lokalt

[13] Man bör alltid tänka på att varje förändring som man utför för att öka prestanda på ett SQL-uttryck kan påverka andra uttryck som används av applikationer och användare.

3.3.7 Urvalet är det kritiska i SQL-satsen

[13] Det är viktigt att tänka på vilken typ av fråga man ställer till en databas. Generellt gäller att alla frågor som innehåller tecknen som innehåller jämförelseoperatorer inte kommer att indexeras. Indexering kommer heller inte att ske vid kontroll om värde saknas (null) eller om värde inte saknas. Indexering kommer inte att ske vid kontroll om ett värde inte finns i en värdelista eller vid kontroll av olikheter. Indexering kommer inte att ske om man kontrollerar om en sträng liknar en sträng som börjar med ett jokertecken (tex. ’%’), till skillnad från om strängen inte börjar med jokertecknet, då frågan kommer att indexeras. Indexering kommer att ske vid kontroll om värde existerar i en underfråga eller vid gruppering och kontroll om en eller flera kolumner har ett visst värde. Exempel på kod som kommer att indexeras, resp. inte kommer att indexeras finns i bilaga A.10 och A.11.

3.3.8 Specificera den först indexerade kolumnen i en sammansatt primärnyckel i urvalet

För en sammansatt nyckel, kommer urvalet att använda sig av index så länge som den inledande nyckeln används i urvalet.

I följande exempel [13] består primärnyckeln av PART_NUM, PRODUCT_ID:

Detta exempel använder sig av index:

```
SELECT  *
  FROM PARTS
WHERE PART_NUM  =  100;
```

Medan detta exempel inte använder sig av index:

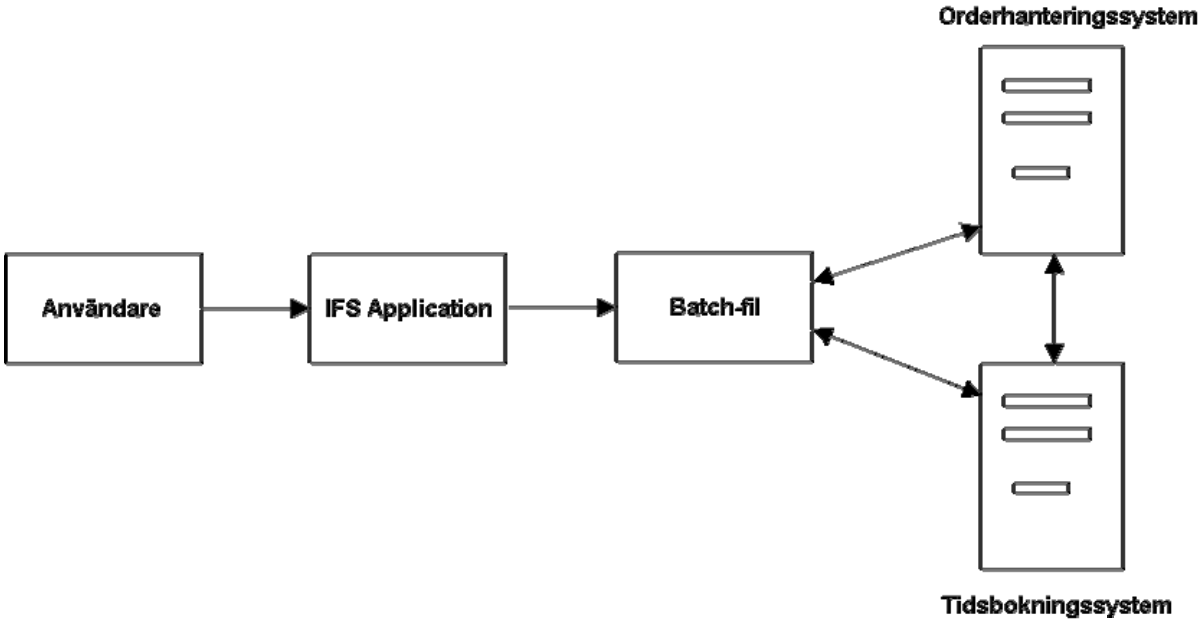
```
SELECT *
  FROM PARTS
WHERE PRODUCT_ID =  5555;
```

4 Befintligt system

I det här kapitlet presenterar vi en översikt av de system som vi har kommit i kontakt med under vårt arbete. Avsnitt 4.1 ger en översikt av det befintliga systemet. I 4.2 beskriver vi hur rapportering av data sker i det nuvarande systemet. Avsnitt 4.3 presenterar alternativa systemlösningar som ligger till grund för vår undersökning.

4.1 Systemöversikt

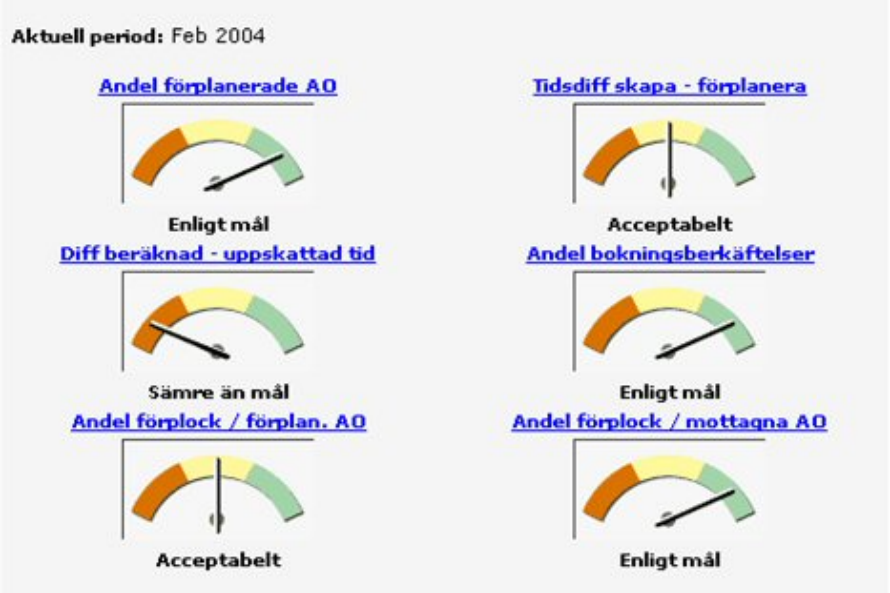
Det befintliga system som används idag är ett orderhanteringssystem som är sammankopplat med ett tidsbokningssytem (se Figur 4). Orderhanteringssystemet består av ett antal integrerade system. De databaser som används i systemen i dag är baserade på Oracle 8 och 10. Emellertid kommunicerar systemet med andra underliggande system som vi inte kommer att nämna vid namn på grund av sekretesskäl och som inte innefattas i vår uppgift.



Figur 4: Översikt av befintligt system

4.2 Rapportering av data

Den befintliga webbapplikation som används för analys och rapportering har utvecklats av IFS (Industrial Financial Systems) och bygger på en IFS-applikation. Den inre strukturen är inte känd på grund av att IFS har patent på detta affärssystem. Dock har vi fått tillgång till slutprodukten via en skärmdump vilken visas i figur 5 nedan.



Figur 5: Skärmdump från befintligt system

Det som visas i figur 5 ovan är ett sammanställt resultat av sex frågor som ställs mot de olika databaserna. Visas rött[5], vid analysens utförande, är resultatet sämre än företagets målsättning. Gult är ett acceptabelt resultat, medan det gröna fältet motsvarar uppställda förväntningar. Man bör här beakta de olika filialernas varierande målsättningar.

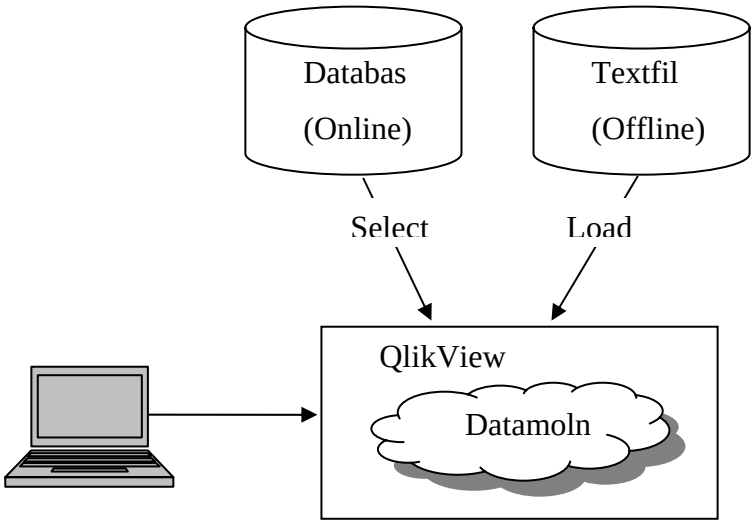
4.3 Alternativa systemlösningar

De nya lösningarna innebär följande: Databasinformation överförs från orderhanteringssystemet och tidsbokningssystemet in i analysverktyget Qlikview, vilket kan ske online eller offline.

Vid exekvering av inladdningsskript läses data in och sparas i applikationen. Resultatet kallas för ett ”datamoln” som genom AQL kan användas för att göra en avancerad analys helt

[5] De olika mätartavlorna är graderade i rött, gult och grönt.

oberoende av fördefinierad aggregering (se 2.5.1), vilket åskådliggörs genom bilden nedan (Figur 6).



Figur 6: Systemlösning med QlikView

5 Utveckling av mätare i QlikView

Ett delmoment i vårt uppdrag är att grafiskt presentera statistiskt material för användaren. Kapitlet tar upp i avsnitt 5.1 tabellkonstruktionen och i avsnitt 5.2 presenteras implementationen av det grafiska gränssnittet.

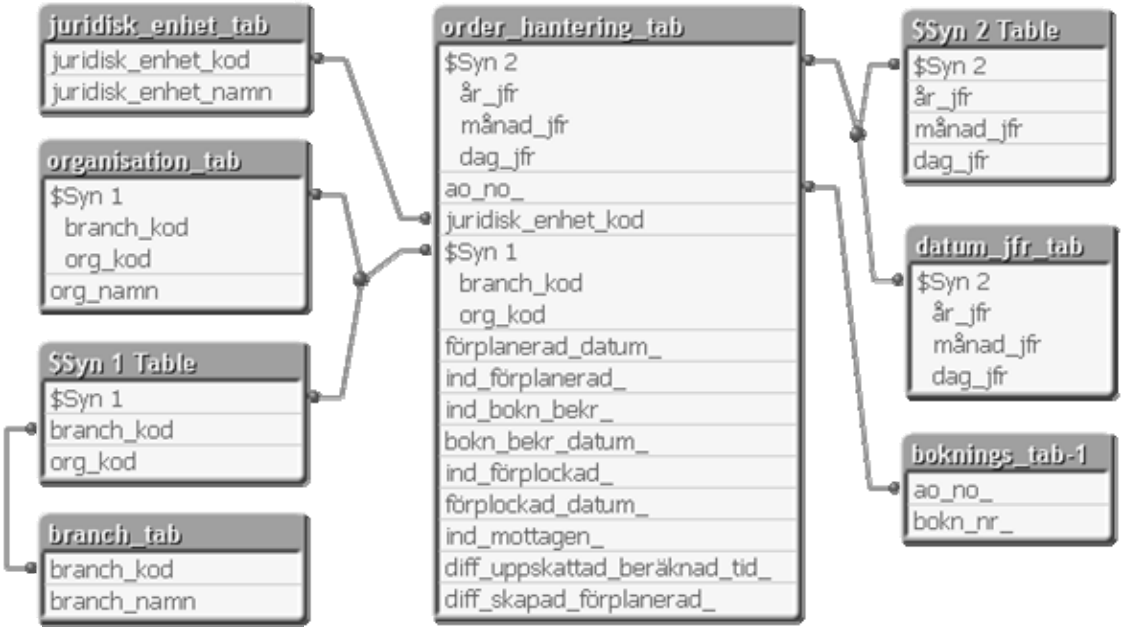
När det gäller presentationen av data i verktyget QlikView så har vi i första hand försökt att efterlikna de mätare som använts i den befintliga applikationen från IFS (Kap 4 figur 5).

Att utveckla i QlikView (se avsnitt 2.5), är fördelaktigt eftersom det har egenskaper från datalager, OLAP och ETL (se kapitel 2). En av fördelarna med QlikView är dess möjlighet att grafiskt presentera förändringar av data och sammanställa dessa till en helhet, vilket innebär att t.ex. en arbetsorder kan lagras mer än en gång i likhet med en OLAP-kub (se avsnitt 2.3.3).

5.1 Tabellkonstruktion

Det inledande momentet var att utforma tabeller i QlikView utifrån de textfiler som vi fick tillgång till vid arbetets början. Filerna innehåller den information som överförs via batch vilket är en ögonblicksbild för hur databasen såg ut vid en viss tidpunkt.

Konstruktionen av tabellerna skapades i det ETL-skript som finns i QlikView (bilaga A.3, A.4). I skriptet definierades olika namn på tabeller och kolumner. Genom att använda ett alias med samma namn för kolumner kommer QlikView automatiskt att skapa kopplingar mellan tabellerna. I figur 7 nedan presenteras den interna tabellvyn, vilket visar datatabellerna så som de lagras av QlikView. För de tabeller som delar fler än ett fält skapas automatiskt komposita syntetiska nycklar, vilka länkas samman via syntetiska tabeller. Det är denna vy som bäst förklarar QlikView-logiken.



Figur 7: Tabellvy i QlikView

5.2 Grafisk implementation

I detta avsnitt presenteras de cirkelmätare som vi utvecklat, vilken visar en sammanställning av insamlad information från orderhanteringssystemet och tidsbokningssystemet [3]. När man skall visa en tabell som innehåller dimensionsvariabler i ett diagram, så måste man använda aggregeringsfunktioner för att visa en summering av värden. Nedan visas en av de aggregeringsfunktioner som vi använder för att beräkna andelen förplanerade arbetsorder av det totala antalet arbetsorder i systemet (se avsnitt 5.2.1).

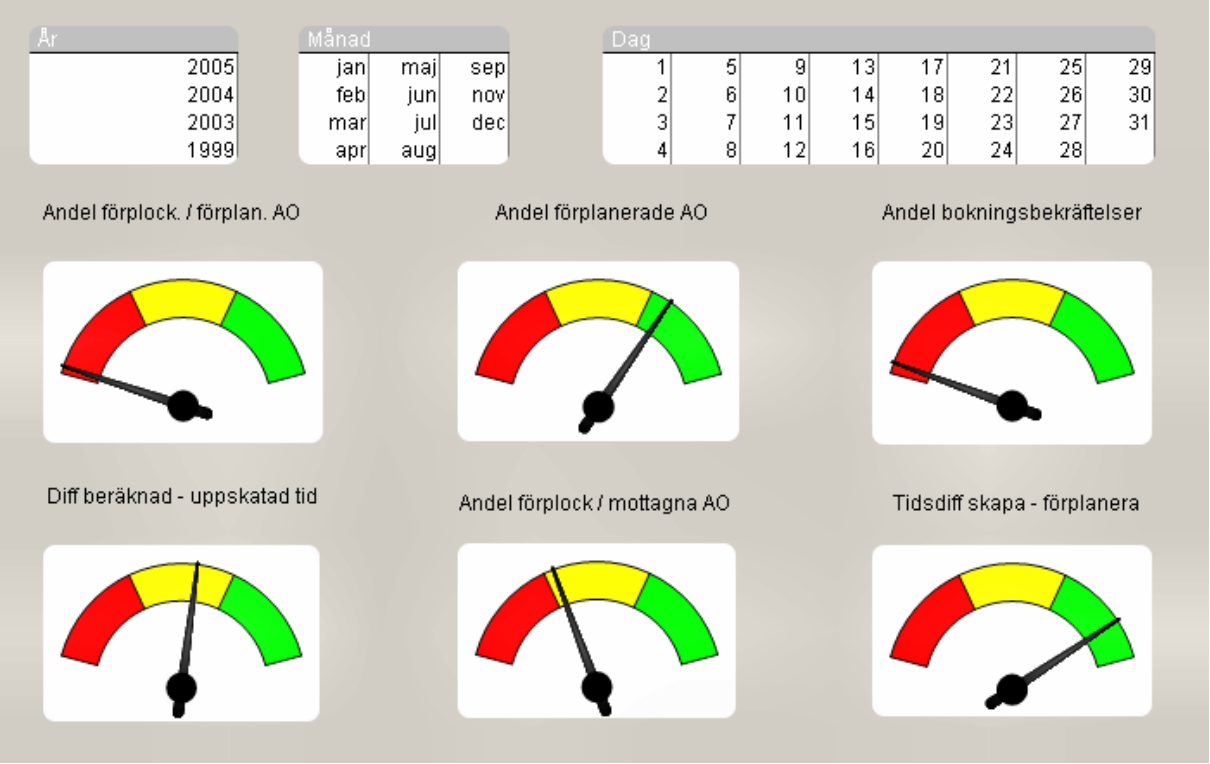
```
IF(COUNT(ALL DISTINCT ah_ao_no) = 0,
   0,
   COUNT(TOTAL ao_ind_förplanerad)/COUNT(ALL DISTINCT ah_ao_no)
)
```

För att funktionen alltid ska returnera ett relevant värde har ett villkor inkluderats i funktionen, vilket innebär att om antalet arbetsorder är större än noll returneras andelen, annars noll.

Koden för implementationen av det grafiska gränssnittet (Figur 8) finns i bilaga A.9.

5.2.1 Slutligt användargränssnitt i QlikView

Vid start av applikationen visas nedanstående cirkelmätare. Genom att välja år, månad eller dag utförs en analys på mer detaljerad nivå.



Figur 8: Skärmdump av användargränssnittet

Beskrivning av de olika mätarnas innebörd (Figur 8):

- Andel förplanerade AO

  Hur stor del av de order som är inlagda i tidsbokningssystemet, är förplanerade i ordersystemet

- Tidssdiff. skapa – förplanera

  Hur lång tid tog det från att ordern skapades tills dess att den förplanerades

- Diff. beräknad – uppskattad tid

  Hur stor andel av de order som är registrerade i tidsbokningssystemet, har en differens mellan uppskattad och beräknad tid.

- Andel bokningsbekräftelser

  Hur stor andel av alla order i tidsbokningssystemet har en indikator på att bokningsbekräftelse har skickats i ordersystemet

- Andel förplockade / förplanerade AO

Hur stor del av de order som är inlagda i tidsbokningssystemet och har varit förplanerade, är förplockade

- Andel förplockade / mottagna AO

  Hur stor del av de order som är inlagda i tidsbokningssystemet och är mottagna, är förplockade.

6 Analys av batch / online

I inledningen av uppsatsen kan man läsa att vår undersökning kommer att behandla huruvida det är möjligt rent tidsmässigt att presentera data online, eller om detta måste ske offline, med uppdatering av data i vissa tidsintervall per dygn. Med online-läge i detta sammanhang avses en ODBC[6]-uppkoppling direkt in i databaser för att hämta in data. I detta kapitel presenterar vi resultat för utförda tester samt olika optimeringssätt för de frågor som ställs mot ordersystemet och tidsbokningssystemet vid dataöverföring. I avsnitt 6.1, 6.3 och 6.5 redovisas de olika tester som utförts, samt i avsnitt 6.2, 6.4 och 6.6 redovisas slutsatser och analyser av dessa tester.

6.1 Testutförande 1

Det inledande testet utfördes för att avgöra om man ska använda batch-körning, eller om det går bra att köra online med frågor direkt till databasen.

Under vårt arbete har vi fått tillgång till de batchskript (bilaga A.1, A.2) som körs vid överföring av förändringar i databasen. Dessa skript har sedan analyseras i PL/SQL Developer där det finns en möjlighet att testa skriptet för att mäta total exekveringstid. I tabell 1 och tabell 2 visas våra mätresultat. Övriga mätresultat finns i bilaga B. Detta test har gjorts på två olika sätt. Det första testet tar med alla resulterande bokningar, medan det andra testet endast tar med de bokningar som har utförts sedan senaste batchkörning.

Antal rader	Tid i sekunder	Endast förändrad data
102	0,280	Nej
39	0.203	Ja

Tabell 1: Resultat online-körning tidsbokningssystem

[6] ODBC (Open Database Connectivity) är ett API (Application Programming Interface), som beskriver koppling till en databas och utvecklades ursprungligen av Microsoft baserat SQL och X/Open? och ISO/IEC [Call Level Interface]? CLI. De flesta databaser stödjer idag ODBC vilket gör att ett datorprogram som använder sig av ODBC-interface kan kopplas till många olika databassystem.

Antal rader	Tid i sekunder	Från datum
13	19.579	2005-11-01
1451	27.796	2005-05-01
1510	28.703	2005-01-01

Tabell 2: Resultat online-körning hämtning av orderhuvud

6.2 Utvärdering av testkörning 1

Att hämta data i online-läge kommer inte att vara acceptabelt för slutanvändare eller vara en skonsam belastning för en databasserver, med anledning av de uppmätta exekveringstiderna i ordersystemet. De varierande exekveringstiderna mellan 20 till 35 sekunder anser vi vara en alldeles för lång tid, eftersom vi upplysts om att exekveringstider runt 5 sekunder inte accepteras av användaren. Vid en närmare granskning av den befintliga proceduren, vilken exekveras på serversidan, konstaterade vi att den var alldeles för avancerad för att köras online. Den ställda frågan mot databasen innehöll en avkodning, 14 villkor och allt för många funktionsanrop, vilket berör minst 10 paket inklusive det egna paketet. Vårt fortsatta arbete i analysen för online gav oss en ide om att en möjlig lösning kan vara att skapa vyer i det berörda paketet för att få ner exekveringstiderna.

6.3 Testutförande 2

Den befintliga koden är skriven med en traditionell cursor[7] (bilaga A.6). Vi har testat att hämta data från databasen med hjälp av en cursor som loopar sig igenom alla data som har förändrat sig från och med den senaste batchkörningen. Resultatet av denna test är en exekveringstid på cirka 30 sekunder

[7] Cursor, engelska för markör / pekare.

Skulle det löna sig att skriva om cursorn så att den hanteras med s.k. ”BULK COLLECT” istället? ”BULK COLLECT” innebär ett uppsnabbat sätt att hämta data. Detta görs genom att upphämtade data lagras i temporära PL/SQL-tabeller, som man sedan har direkt åtkomst till. Att använda bulk på stora datamängder rekommenderas inte, eftersom detta resulterar i en stor processoranvändning, samt att varje urval lagrar en temporär tabell. Man bör därför använda sig av en övre gräns för hur stort resultatet från cursorns fråga får vara, och sedan använda den flera gånger, om det behövs. Testet har genomförts genom att ställa en fråga på traditionellt sätt med en vanlig cursor, samt att ställa en fråga med en cursor som lagrar data med hjälp av ”BULK COLLECT” (bilaga A.5).

6.4 Utvärdering av testkörning 2

Resultatet av testet med en traditionell cursor blev en exekveringstid på cirka 30 sekunder. Resultatet med en cursor som hanteras med ”BULK COLLECT” blev cirka 28 sekunders svarstid. Det är alltså inte möjligt att förbättra prestandan med hjälp av en cursor som hanteras med ”BULK COLLECT”. I tabell 3 visas våra mätresultat. Övriga mätresultat finns i bilaga B.

Antal rader	Tid I sekunder	Från datum
6 rader	18,610 s	2005-12-01
6 rader	17,485 s	2005-12-01
1510 rader	27,672 s	2005-01-01
1510 rader	29,890 s	2005-01-01

Tabell 3: Resultat med ursprungliga urvalet och cursor som hanteras med bulk collect

Antal rader	Tid i sekunder	Från datum
6	17,875	2005-12-01
6	18,532	2005-12-01
1510	32,109	2005-01-01
1510	28,390	2005-01-01

Tabell 4: Resultat med ursprungliga urvalet och traditionell cursor

Vi har även utfört tester med användning av hints (Kap 6.3.4), vilket inte påverkat resultatet nämnvärt. Resultaten av dessa tester finns i bilaga B.

6.5 Testutförande 3

I detta test har vi skrivit om frågan som ställs till databasen och lagrat denna som en vy (bilaga A.7). Vi har skrivit om frågan till nästan ren SQL, dvs. vyn innehåller minimalt antal funktionsanrop, sedan har vi även skrivit om frågan som en sammanslagen fråga med UNION (bilaga A.8). Detta har medfört att vi har lyckats att få ner exekveringstiden från ca 30 sekunder till ca 4 sekunder, oberoende av antal order som hämtas.

6.6 Utvärdering av testkörning 3

Resultatet visar på att det är fullt möjligt att köra den sammanslagna frågan online. Testresultatet tyder på att tiden inte är beroende av antalet order som hämtas samt att det är möjligt att hämta data online i detta system. I tabell 5 visas våra mätresultat med en optimerad fråga mot en optimerad vy. Övriga mätresultat finns i bilaga B.

Antal rader	Tid i sekunder	Från datum
6	3.845	2005-12-01
6	3.693	2005-12-01
6	3.743	2005-12-01
1510	3.878	2005-01-01
1510	3.673	2005-01-01
1510	3.756	2005-01-01

Tabell 5: Resultat online-körning med sammanslagen fråga mot vy

6.7 Summering av utförda tester

Av Tabell 1 kan man utläsa att exekveringstiden är acceptabel med 0,280 – 0,203 sek, vid inhämtande av data i ett onlineläge. Tabell 2 visar de första resultaten vid datainhämtning online av information från orderhanteringssystemet, vilka har en oacceptabel exekveringstid på cirka 28 sekunder. I nästa tabell (Tabell 3) kan man se resultaten av andra försöket i ”BULK COLLECT” där man inte kan påvisa någon nämnvärd förändring. Den sista tabellen (Tabell 5) har upprättats efter onlinekörning av en sammanslagen fråga mot en vy. Resultatet blev en minskning av exekveringstiden på serversidan med 24 sekunder jämfört med den första provkörningen (Tabell 2).

7 Slutsats och diskussion

I avsnitt 7.1 presenteras vårt förslag till uppdragsgivaren. I avsnitt 7.2 presenteras förslag till vidare arbete. I avsnitt 7.3 tar vi upp de problem som vi mött. I avsnitt 7.4 tar vi upp våra nyförvärvade kunskaper. I avsnitt 7.5 ges förslag till alternativa lösningar.

7.1 Rekommendationer

Vi har efter att ha utfört en del tester dragit slutsatsen att det är möjligt att skriva om den befintliga koden, så att den kommer att kunna köras online på ett acceptabelt sätt. Dock måste man även beakta tiden det tar för QlikView att läsa in och strukturera data i internminnet. Vid hämtning av en större datamängd i QlikView uppskattas den totala laddningstiden till 40 sekunder. Tiden det tar att läsa in en större datamängd i QlikView är även beroende av prestandan hos klienten. Visserligen har vi minskat exekveringstiderna på serversidan avsevärt, men vi upplever att svarstiderna i QlikView är alldeles för långa. Överföring av data genom batch-filer anser vi vara det bästa sättet, med tanke på databasens enorma informationsinnehåll. Vi rekommenderar därför ett offline-läge. Alternativa lösningar presenteras i avsnitt 7.5.

7.2 Vidare arbete

Resultaten från våra mätningar får oss att spekulera i hur lång tid det skulle ta i en riktig slutmiljö. Troligtvis kommer svarstiderna att vara längre, eftersom testdatabaserna inte innehåller lika mycket data som finns i kundens databaser. När det gäller nätverksbelastningen behövs en speciell sorts mjukvara som har förmågan att simulera och mäta belastningen i ett nätverk med många användare.

Begränsningen ligger inte på serversidan utan i QlikViews förmåga att ladda in data. Anledningen till de långa svarstiderna tror vi är beroende av prestanda på slutanvändarens dator.

Ett moment som kvarstår är att presentera informationen med avseende på användarens behörighet i QlikView, vilket vi inte har utrett vidare på grund av tidsbrist.

7.3 Problem

Utförande av vårt uppdrag har inte varit problemfritt. Den dator vi har haft tillgång till har inte mäktat med nedladdningar av större datamängder, vilket medfört att programmet kraschat. Ytterligare ett problem har varit att man inte kan ha filer öppna för redigering, samtidigt som inladdning sker i QlikView. Syntaxen i QlikView har, i vissa fall, skiljt sig från traditionella programspråk.

I arbetets inledning kunde vi inte utföra tester på tillräckligt stora datamängder, vilket medförde restultat som inte kändes rättvisande. Den första databasen som vi fick tillgång till på ÅF i Karlstad innehöll alldeles för lite data för att kunna utföra prestandatester. Denna databas innehöll inte mer än 37 rader. Det bör även nämnas att vid denna tidpunkt skulle leverans till kund ske, vilket medförde att vi inte fick tillåtelse att lägga in fler data i systemet. Vi kom då överens med affärsområdeschefen Niklas Siljeblad att det bästa vore om vi kunde utföra våra tester på kontoret i Åmål, där det finns bättre möjligheter att testa SQL-frågor mot databasen.

7.4 Lärdomar

Under examensarbetets gång har vi fått utökade kunskaper i databashantering. Insikten i att optimering är ett stort kapitel, med många möjligheter till fördjupning, har gett oss nyttiga kunskaper för framtida utveckling. Efter att på olika sätt ha gjort försök till att optimera en av de frågor som ställs till databasen, har vi kommit fram till att många faktorer påverkar exekveringstiderna, speciellt vid förfrågningar mot de större databaserna. I vissa fall kan det vara möjligt att optimera med s.k. hints (se kapitel 3). Emellertid kunde vi, i detta fall, inte se några större skillnader. Det finns även olika sätt att skriva en cursor på i PL/SQL. Efter att ha testat att skriva en cursor med ”BULK COLLECT” kunde vi konstatera att inte heller detta påverkade våra resultat nämnvärt. Vi blev överraskade av möjligheten att optimera genom att undvika funktionsanrop. Viktigt är också att tänka på ***hur*** man ställer en sammanslagen fråga mot databasen, eftersom det finns en mängd olika sätt att utföra detta på. Ett av sätten är att kontrollera om ett urval finns i en underfråga med hjälp av ”EXISTS”, vilket oftast är ett effektivt sätt att skriva en fråga på. Våra tester visar att det inte är en regel utan undantag. Vi har även fått kunskaper om termerna OLAP och datalager, vilket är en alternativ lösning till QlikView.

7.5 Alternativa lösningar

7.5.1 Alternativa verktyg

Vi anser att detta arbete långt ifrån är avslutat i och med att vi har slutfört examensarbetet. Troligen har den fråga som har den sämsta prestandan optimerats maximalt. Om man från uppdragsgivarens sida anser att vår optimering är tillräckligt bra, rekommenderar vi att använda alternativa OLAP-verktyg istället för QlikView. Det finns inbyggt stöd för datalager och OLAP i Oracle 10g.

7.5.2 Server-lösning

I våra mätningar konstaterades det att överföringen av information inte kan utföras i onlineläge på grund av en för lång exekveringtid. En tänkbar lösning är att installera en QlikView-server, vilken uppdateras via batch-filer under natten eller flera gånger per dag. Användaren ges då en möjlighet att köra i ett online-läge mot servern för att hämta den senaste informationen via ett QlikView-dokument, alternativt att servern exekverar dokumentet för direkt analys i webbläsaren. Genom server-lösningen kan användarnas behörighet kontrolleras bättre, genom rättigheter till filen med avseende på dennes befattning i verksamhet.

7.5.3 Hybrid-lösning

Vi har haft tankegångar på lösning av både online och offline. Informationen i QlikView-dokumentet uppdateras under natten via batch. När användaren vill utföra en analys sker en ODBC-uppkoppling för att hämta de senaste händelserna från den senaste bachkörningen. Denna ide slogs i spillror när online testerna resulterade i en alldeles för lång exekveringstid av SQL-skriptet i QlikView.

Referenser

Böcker:

[1] Elmasri Ramez, Navathe Shamkant B, Fundamentals of Database Systems, Fourth edition, 2003

[2] Greene Joseph B, Oracle DBA Survival Guide, 1995

[3] HIC, KHN, JNN, MSJ, QlikView Reference Manual, Book 1 Installation, Script & Macros, Second Edition, QlikTech International AB, 2003

[4] HIC, KHN, JNN, MSJ, QlikView Reference Manual, Book 2 Layout, Second Edition, QlikTech International AB, 2003

[5] Inmon W H, Building the Data Warehouse, Wiley 1996

[6] Kamber Micheline, Han Jiawei, Data Mining Concepts and Techniques, Academic Press 2001

[7] Kimball Ralph, Caserta Joe, The Data Warehouse ETL Toolkit, Wiley, 2004

[8] Lumpkin George, Jacobsson Hakan, Oracle Corporation, Query Optimization in Oracle Database 10g Release 2, 2005

[9] Smith William, Systems building with Oracle, 2004

[10] Söderström Peter, "Data Warehouse" Datalager Verkasmhet, Metod, Teknik, studentlitteratur1997

Internet:

[11] Oracle SQL Hints, http://www.adp-gmbh.ch/ora/sql/hints.html, 2006-01-19

[12] Computerworld, http://www.computerworld.com/databasetopics/businessintelligence/datawarehouse/story/0,10801,89534,00.html, 2005-12-10

[13] Database programming and design, http://www.dbpd.com/vault/9801xtra.htm, 2005-12-05

[14] Oracle Corporation, www.oracle.com, 2005-11-29

[15] Oracle's TKPROF, http://www.adp-gmbh.ch/ora/tuning/tkprof/index.html, 2006-01-19

[16] QlichTech Nordic, http://www.qliktech.com, 2005-12-28

[17] ÅF | Ångpanneföreningen | The AF Group, http://www.af.se, 2005-09-13

A Bilaga Kod

A.1 Del av befintlig urvalsfråga, hämtning av orderhuvud

```
SELECT      h.ao_nr, …
    FROM    ao_huvuden h,
            kostnadsställe ko
    WHERE   (
                (
                    TRUNC(h.tidsstämpel) >= from_date
                    AND TRUNC(h.tidsstämpel) <= to_date
                )
                OR  (
                    EXISTS (
                        SELECT  j.ao_nr
                        FROM    ao_jobs j
                        WHERE   TRUNC(j.tidsstämpel) >= from_date
                        AND     TRUNC(j.tidsstämpel) <= to_date
                        AND     j.återförsäljare_kod =  h.återförsäljare_kod
                        AND     j.ao_nr = h.ao_nr
                    )

                )
            )
    AND     h.återförsäljare_kod = återförsäljare_kod
    AND     (
                h.branch_kod = ko.branch_kod (+)
                AND h.återförsäljare_kod = ko.återförsäljare_kod (+)
                AND h.avdelning_kod = ko.avdelnig_kod (+)
    );
```

A.2 Del av befintlig procedur, hämtning av arbetsorder-information

```
DECLARE
   buffert_              VARCHAR2(2000);
   partner_              VARCHAR2(8);
   branch_               VARCHAR2(20);
   plats_                VARCHAR2(10);
   namn_                 VARCHAR2(100);
   min_skrivbara_datum_  DATE := '1900-01-01';

   CURSOR hämta_ao IS
      SELECT             ao_no,
                         TRUNC(reg_datum)
                         reg_datum,
                         rapporterad_av_id,
                         rapporterad_av,
                         ah_ao_no,
                         org_kod
         FROM            aktiv_separera_tab
         WHERE           batch_skapad = 0
         AND             TRUNC(reg_datum) >= TRUNC(min_skrivbara_datum_)
         ORDER BY        org_kod;
   BEGIN
      FOR ao_ IN hämta_ao LOOP
         buffer_   :=    NULL;
         branch_   :=    org_api.hämta_branch(ao_.org_kod);
         plats_    :=    branch_api.hämta_plats(branch_);
         partner_  :=    Anslutning_API.Hämta_Partner(plats_);
         namn_     :=    Person_Info_API.Hämta_Name(ao_.rapporterad_av);
         buffer_   :=    '501;'||
                         partner_||';'||
                         plats_||';'||
                         branch_||';'||
                         ao_.organisations_kod||';'||
```

```
                    ao_.rapporterad_av_id||';'||
                    namn_||';'||
                    ao_.ao_no||';'||
                    ao_.ah_ao_no||';'||
                    ao_.reg_datum;
        dbms_output.put_line(buffer_);
    END LOOP;
END;
```

A.3 ETL-skript I QlikView för inläsning av batch-fil

```
LOAD    DISTINCT branch_kod,
        org_kod,
        org_namn
  FROM  file.txt (ansi, txt, delimiter is ';', embedded labels);
```

A.4 ETL-script i QlikView med en sammanslagen fråga, ställd mot en ODBC-källa

```
// Hämta förändrade arbetsordrar
SELECT      ao_no,
            förplanerad_datum,
            skapad_datum,
            uppskattad_tid,
            ind_bokn_bekr,
            bokn_bekr_datum,
            status,
            ind_förplockad,
            förplockad_datum,
            juridisk_enhet_kod,
            branch_kod,
            org_kod,
            ind_alla_jobb_fakturerade,
```

```
                beräknad_tid,
                fakturerade_jobb,
                totalt_antal_jobb
    FROM        qlik_view h
    WHERE       TRUNC(h.tidsstämpel) >=
                   TO_DATE('$(från_dt_)', 'YYYY-MM-DD')
    AND         TRUNC(h.tidsstämpel) <= '$(till_dt_)'
    AND         h.återförsäljare_kod = '$(återförsäljare_kod)'
UNION
// Hämta arbetsordrar med förändrade jobb
SELECT          ao_no,
                förplanerad_datum,
                skapad_datum,
                uppskattad_tid,
                ind_bokn_bekr,
                bokn_bekr_datum,
                status,
                ind_förplockad,
                förplockad_datum,
                juridisk_enhet_kod,
                branch_kod,
                org_kod,
                ind_alla_jobb_fakturerade,
                beräknad_tid,
                fakturerade_jobb,
                totalt_antal_jobb
    FROM        qlik_view h, ao_jobb j
    WHERE       ao_no = j.ao_no
    AND         TRUNC(j.tidsstämpel) >=
                TO_DATE('$(från_dt)', 'YYYY-MM-DD')
    AND         TRUNC(j.tidsstämpel) <= '$(till_dt_)'
    AND         j.återförsäljare_kod = '$(återförsäljare_kod)';
```

A.5 Procedur med en cursor innehållande bulk collect

```
DECLARE
    TYPE ao_no_typ IS
        TABLE OF ao_huvuden.ao_no%TYPE
        INDEX BY BINARY_INTEGER;
    Forts typdeklarationer…
    ao_no_      ao_no_typ;
    Forts parameterdeklarationer…
    från_dt_        DATE        :=  TO_DATE('2005-01-01', 'YYYY-MM-DD');
    till_dt_        DATE        :=  SYSDATE;
    återförsäljare_kod_ ao_huvuden.återförsäljare_kod%TYPE := 'PRD';
    i               INTEGER     :=  1;

    CURSOR ao_rec(
        från_dt_                DATE,
        till_dt_                DATE,
        återförsäljare_kod_     ao_huvuden.återförsäljare_kod%TYPE) IS
            SELECT      ao_no,
                        Forts urval…
                FROM    ao_huvuden h,
                        kostnadsställe ko
                WHERE (
                            (
                                TRUNC(h.tidsstämpel) >= från_dt_
                                AND TRUNC(h.tidsstämpel) <= till_dt_
                            )
                            OR
                            (
                                EXISTS (
                                    SELECT      j.ao_no
                                    FROM        ao_jobs j
```

```
                                    WHERE    TRUNC(j.tidsstämpel) >= från_dt_
                                    AND      TRUNC(j.tidsstämpel) <= till_dt_
                                    AND      j.återförsäljare_kod =
                                             h.återförsäljare_kod
                                    AND      j.ao_no = h.ao_no
                                )
                            )
                )
                AND     h.återförsäljare_kod = återförsäljare_kod_
                AND (
                        h.branch_kod = ko.branch_kod (+)
                        AND     h.återförsäljare_kod = ko.återförsäljare_kod (+)
                        AND     h.avdelning_kod = ko.avdelning_kod (+)
                );
    BEGIN
        OPEN ao_rec(från_dt_, till_dt_, återförsäljare_kod_);
            FETCH ao_rec BULK COLLECT INTO
                ao_no_,
                branch_kod_,
                återförsäljare_kod_,
                Forts. urval till parametrar…+ eventuellt en övre gräns
        CLOSE ao_rec;
        FOR i IN ao_no_.FIRST .. ao_no_.LAST LOOP
            dbms_output.put_line(ao_no_(i) || ' ' || återförsäljare_kod_(i));
        END LOOP;
    END;
```

A.6 Procedur med en vanlig cursor

```
DECLARE
    från_dt_        DATE        :=      TO_DATE('2005-01-01', 'YYYY-MM-DD');
    till_dt_        DATE        :=      SYSDATE;
```

```
återförsäljare_kod_     ao_huvuden.återförsäljare_kod%TYPE := 'PRD';
i                 INTEGER := 1;
CURSOR get_ao (
    från_dt_          DATE,
    till_dt_          DATE,
    återförsäljare_kod_ ao_huvuden.återförsäljare_kod%TYPE) IS
    SELECT            ao_no,
                      Forts. Urval…
        FROM          ao_huvuden h,
                      kostnadsställe ko
        WHERE         (
                          (
                              TRUNC(h.tidsstämpel) >= från_dt_
                              AND TRUNC(h.tidsstämpel) <= till_dt_
                          )
                          OR (
                              EXISTS (
                                          SELECT    j.ao_no
                                          FROM      ao_jobs j
                                          WHERE     TRUNC(j.tidsstämpel) >=
                                                    från_dt_
                                          AND       TRUNC(j.tidsstämpel) <=
                                                    till_dt_
                                          AND       j.återförsäljare_kod =
                                                    h.återförsäljare_kod
                                          AND       j.ao_no = h.ao_no
                              )
                          )
                      )
        AND
                      h.återförsäljare_kod = återförsäljare_kod_
        AND           (
                          h.branch_kod = ko.branch_kod (+)
```

```
                    AND     h.återförsäljare_kod = ko.återförsäljare_kod (+)
                    AND     h.avdelning_kod = ko.avdelning_kod (+)
    );

BEGIN
    FOR ao_rec IN get_ao(från_dt_, till_dt_, återförsäljare_kod_) LOOP
        dbms_output.put_lineao    (i || ' ' || _rec.återförsäljare_kod);
        i := i+1;
    END LOOP;
END;
```

A.7 Vyn som används i stället för proceduren med cursorn

```
CREATE OR REPLACE VIEW qlik_view AS
    SELECT      ao_no,
                …
                DECODE (status,
                            '20',
                            (SELECT COUNT(*)
                                FROM        ao_jobb j
                                WHERE       j.återförsäljare_kod = h.återförsäljare_kod
                                AND         j.branch_kod = h.branch_kod
                                AND         j.ao_no = h.ao_no
                            ),
                            0
                ) förplanerade_jobb,
                …
                (SELECT     förplanerad_tid
                   FROM     kostnadsställe ko
                   WHERE    ko.återförsäljare_kod = h.återförsäljare_kod
                   AND      ko.branch_kod = h.branch_kod
                   AND      ko.avdelning_kod = h.avdelning
```

```
                ) förplanerad_tid,
                …
        FROM    ao_huvud h
WITH READ ONLY
```

A.8 Den sammanslagna frågan som används mot vyn

```
SELECT      ao_no,
            ...
            förplanerad_tid,
            ...
    FROM    qlik_view h
    WHERE   TRUNC(h.tstamp) >= TO_DATE(från_dt_, 'YYYY-MM-DD')
    AND     TRUNC(h.tstamp) <= till_dt_
    AND     h.återförsäljare_kod = återförsäljare_kod_
UNION
SELECT      ao_no,
            ...
            förplanerad_tid,
            ...
    FROM    qlik_view h, ao_jobs j
    WHERE   ao_no = j.ao_no
    AND     TRUNC(j.tidsstämpel) >= TO_DATE(från_dt_, 'YYYY-MM-DD')
    AND     TRUNC(j.tidsstämpel) <= till_dt_
    AND     j.återförsäljare_kod = återförsäljare_kod_
```

I denna kod är från_dt_, till_dt_ och återförsäljare_kod_ variabler.

A.9 Implementation av mätare i QlikView

```
IF(COUNT(ALL DISTINCT ah_ao_no) = 0,
   0,
   COUNT(TOTAL ao_ind_förplanerad)/COUNT(ALL DISTINCT ah_ao_no)
)
```

```
IF(COUNT(TOTAL ao_ind_förplanerad) = 0,
   0,
   COUNT(TOTAL ao_diff_skapad_förplanerad)/COUNT(TOTAL ao_ind_förplanerad)
)

IF(COUNT(TOTAL ao_ind_förplanerad) = 0,
   0,
   COUNT(TOTAL ao_diff_kalk_ber_tid)/COUNT(TOTAL ao_ind_förplanerad)
)

IF(COUNT(ALL DISTINCT ah_ao_no) = 0,
   0,
   COUNT(TOTAL ao_ind_bokn_bekr)/COUNT(ALL DISTINCT ah_ao_no)
)

IF(COUNT(TOTAL ao_ind_förplanerad) = 0,
   0,
   COUNT(TOTAL ao_ind_förplockad)/COUNT(TOTAL ao_ind_förplanerad)
)

IF(COUNT(TOTAL ao_ind_mottagen) = 0,
   0,
   COUNT(TOTAL ao_ind_förplockad)/COUNT(TOTAL ao_ind_mottagen)
)
```

A.10 Exempel på kod som kommer att indexeras:

col1 **LIKE** ’sträng%’,

col1 **LIKE** ’sträng%sträng%’, som använder sig av index

SELECT … **FROM** … **WHERE EXISTS** (underfråga)

SELECT … **GROUP BY** … **HAVING** …

SELECT … **WHERE** … **GROUP BY** …

A.11 Exempel på kod som inte kommer att indexeras:

col1>col2

col1<col2

col1>=col2

col1<=col2

col1 **IS NULL**

col1 **IS NOT NULL** (index sparar inte identiteten på rader som saknar värden)

col1 **NOT IN** (value1, value2)

col1 != uttryck

col1 **LIKE** ‘%sträng’

B Bilaga Testkörningsresultat

Antal objekt i databasen	
Antal orderhuvuden	140 859
Antal jobb	275 338
Antal kostnadsställen	610

B.1 Ursprungliga urvalet

B.1.1 Med traditionell cursor

6 rader	17,875 s	1510 rader	32,109 s
6 rader	18,532 s	1510 rader	28,390 s
6 rader	17,922 s	1510 rader	27,719 s
6 rader	18,235 s	1510 rader	28,015 s

B.1.2 Med Hint /*+ FIRST_ROWS */ i både grundurvalet och exists

6 rader	14,985 s	1510 rader	25,328 s
6 rader	15,109 s	1510 rader	26,032 s
6 rader	14,938 s	1510 rader	25,547 s
6 rader	14,906 s	1510 rader	25,328 s

B.1.3 Med Hint /*+ RULE */ i både grundurvalet och exists

6 rader	19,218 s	1510 rader	26,344 s
6 rader	16,234 s	1510 rader	26,297 s
6 rader	16,250 s	1510 rader	26,047 s
6 rader	16,156 s	1510 rader	27,359 s

B.1.4 Med Hint /*+ CHOSE */ i både grundurvalet och exists

6 rader	17,969 s	1510 rader	28,234 s
6 rader	18,875 s	1510 rader	28,437 s
6 rader	19,406 s	1510 rader	27,937 s
6 rader	20,641 s	1510 rader	28,625 s

B.1.5 Utan alla funktionsanrop

6 rader	13,484 s	1510 rader	13,047 s
6 rader	12,875 s	1510 rader	13,516 s
6 rader	13,063 s	1510 rader	13,860 s
6 rader	13,109 s	1510 rader	13,313 s

B.1.6 Enda funktionsanropet är det som hämtar externt värde

1510 rader	18,422 s
	19,391 s
	19,313 s
	19,500 s

B.1.7 Enda funktionsanropet är det som hämtar kalkylerad tid

1510 rader	21,734 s
	21,672 s
	21,765 s
	24,750 s

B.1.8 Enda funktionsanropet är det som hämtar antalet fakturerade arbeten

1510 rader	18,438 s
	18,563 s
	18,750 s
	18,985 s

B.1.9 Enda funktionsanropet är det som hämtar förplanerade arbeten

1510 rader	20,015 s
	18,281 s
	18,000 s
	18,656 s

B.1.10 Enda funktionsanropet är det som hämtar det totala antalet arbeten

1510 rader	22,297 s
	17,844 s

18,922 s

18,813 s

B.1.11 Enda funktionsanropet är det som hämtar antalet aktiviteter

1510 rader 19,328 s

17,812 s

19,500 s

18,172 s

B.1.12 Enda funktionsanropet är det som hämtar antalet aktiviteter med tiden 0

1510 rader 18,656 s

18,266 s

18,656 s

18,579 s

B.1.13 Enda funktionsanropet är det som hämtar artikelrader

1510 rader 18,141 s

17,922 s

18,219 s

18,063 s

B.1.14 Enda funktionsanropet är det som hämtar betalningssätt

1510 rader 19,094 s

18,485 s

19,156 s

18,734 s

B.1.15 Enda funktionsanropet är det som hämtar namnet på den anställde

1510 rader 18,532 s

20,922 s

18,828 s

18,812 s

B.1.16 Enda funktionsanropet är det som hämtar anställningsnummer

1510 rader 18,453 s

18,344 s

18,765 s

19,000 s

B.1.17 Utan exists

6 rader	1,266 s	1510 rader	11,109 s
6 rader	1,203 s	1510 rader	11,344 s
6 rader	1,203 s	1510 rader	11,188 s
6 rader	1,125 s	1510 rader	11,141 s

B.2 Cursor med bulk collect

B.2.1 Med exists, utan alla funktionsanrop

6 rader	13,062 s	1510 rader	13,000 s
6 rader	12,969 s	1510 rader	12,781 s
6 rader	12,765 s	1510 rader	12,719 s
6 rader	13,125 s	1510 rader	12,734 s

B.2.2 Utan exists, med alla funktionsanrop

6 rader	1,203 s	1510 rader	10,437 s
6 rader	1,172 s	1510 rader	10,453 s
6 rader	1,187 s	1510 rader	10,485 s
6 rader	1,172 s	1510 rader	10,344 s

B.2.3 Med exists, med alla funktionsanrop

6 rader	18,610 s	1510 rader	27,109 s
6 rader	17,485 s	1510 rader	27,672 s
6 rader	18,078 s	1510 rader	29,890 s
6 rader	17,515 s	1510 rader	26,875 s

B.2.4 Ursprungliga urvalet, med hint /*+ FIRST_ROWS */ i både grundurvalet och exists

6 rader	18,610 s	1510 rader	24,016 s
6 rader	17,485 s	1510 rader	24,578 s
6 rader	18,078 s	1510 rader	24,172 s
6 rader	17,515 s	1510 rader	23,953 s

B.3 Ren SQL (utan funktionsanrop)

B.3.1 Sammanslagen fråga ställd mot en vy som bygger på ren SQL

6 rader	3.845 s	1510 rader	3.878 s
6 rader	3.693 s	1510 rader	3.673 s
6 rader	3.743 s	1510 rader	3.756 s
6 rader	3.786 s	1510 rader	3.965 s

Bilaga databasstruktur

B.4 Intern tabellvy