Databasmigration från SQL Server till Amazon Redshift

VIKTOR LINDGREN
BACHELOR´S THESIS
DEPARTMENT OF ELECTRICAL AND INFORMATION TECHNOLOGY
FACULTY OF ENGINEERING | LTH | LUND UNIVERSITY

Examensarbete

Databasmigration från SQL Server till Amazon Redshift

Av

Viktor Lindgren

Department of Electrical and Information Technology
Faculty of Engineering, LTH, Lund University
SE-221 00 Lund, Sweden

Sammanfattning

Växande databaser skapar behov av bättre lagrings- och analysmöjligheter, och ett företag med ett sådana behov är Bring SCM. De har sökt en lösning på hur de bättre skulle kunna analysera sin affärsdata, vilken för tillfället finns på en SQL Server-databas som sköts av Elastic Mobile. För att förbättra denna lösning undersöks i detta arbete en datalager-lösning, mer specifikt Amazon Redshift som är ett molnbaserat datalager. Utöver detta undersöks Amazons Database Migration Service som ett alternativ till den nuvarande migrationslösningen.

För Database Migration Service testas först kopieringen av den fulla databasen och sedan testas löpande replikation av ändringar. För Redshift testas prestandan vid exekvering av åtta av de SQL-frågor som Bring SCM använder sig av i dagsläget, dels på ett lagringsoptimerat kluster och dels på ett beräkningsoptimerat kluster. Slutligen sätts resultaten i relation till kostnaden för det testade systemet jämfört med det befintliga.

Database Migration Service visade stor potential jämfört med det existerande migrationssystemet, och det lagringsoptimerade Redshift-klustret gav både god prestanda och kostnadseffektivitet. Det beräkningsoptimerade klustret presterade under förväntan, särskilt i relation till kostnaden.

Redshift och Database Migration Service är båda mycket lovande alternativ till den nuvarande SQL Server-lösningen för analysarbete, men vid en migration bör man se över vilka möjligheter som finns för optimering av datastrukturen för att bättre passa datalager.

Nyckelord: Redshift, Databas, Databasmigration, Datalager, IT-moln, Big Data

Abstract

Growing databases create a need for improved storage and analysis solutions. One company with that need is Bring SCM, who have been looking for a solution to better analyze their business data, which is currently hosted on a SQL Server-database by Elastic Mobile. To improve this solution, this paper will examine a data warehouse solution to the problem, specifically using the cloud-based Amazon Redshift data warehouse. Additionally, Amazon's Database Migration Service will be examined as an alternative to the current migration solution.

For Database Migration Service, both the migration of the full database as well as the continuous data replication are tested. Redshift is tested based on performance for eight SQL statements that Bring SCM currently use in their work. This is performed both on a storage-optimized cluster and a computation-optimized one. Finally, the costs of running the different clusters are used to evaluate the cost-effectiveness of the solutions.

Database Migration Service showed great potential compared to the current migration solution, and the storage-optimized Redshift-cluster gave both good performance and cost-efficiency. The computation-optimized cluster performed worse than expected, especially in relation to the cost.

Redshift and Database Migration Service are very promising alternatives to the current SQL Server-solution for analysis purposes, but when implementing such a system, consideration should be taken regarding optimizing the data structure for data warehouse use.

Keywords: Redshift, Database, Database Migration, Data Warehouse, IT-cloud, Big Data

Tack

Jag vill rikta tack först och främst till Elastic Mobile samt Bring SCM för att ha bistått både med stöd i arbetet, men också med de verktyg respektive den testdata som gjort arbetet möjligt. Specifikt tack till Joakim Bülow som handlett arbetet från Elastic Mobiles sida, samt Johannes Mueller vid Bring SCM.

Vidare vill jag tacka LTH:s handledare Christin Lindholm, och examinator Christian Nyberg. Slutligen vill jag tacka Estrid Ericson Borggren, Karolina Ström samt Kerstin Svensson för korrekturläsning.

Innehållsförteckning

1. Introduktion

Det är ingen tvekan om att data är en värdefull vara. För företag som Facebook och Google är det en viktig del av affärsidén att använda användardata för bland annat riktad reklam, och i senaste amerikanska valet byggde kampanjerna mycket på dataanalyser; Hilary Clintons kampanj gick ut med att den var ”data-driven” [1], och Donald Trumps kampanj stöddes av storskaliga dataanalyser av målgrupper utförda av Cambridge Analytica [2]. Storskaliga dataanalyser av den här typen möjliggörs av att det genereras enorma mängder data i världen – totala mängden genererad data årligen uppskattades för 2013 till 4,4 Zettabyte – $4{,}4 * 10^{12}$ Gigabyte, och det beräknas öka med en faktor 10 till 2020 [3]. Behandlingen av enorma datamängder har också samlats under termen ”big data”.

I takt med att mängden data som skapas ökar så ökar även behovet av lagring och följaktligen beräkningskapacitet för att söka i den ökande mängden data. En av lösningarna kommer ifrån att IT-infrastrukturen gått mer och mer mot att leverera så mycket som möjligt som en service; allt ifrån fysisk infrastruktur, plattform, mjukvara och till och med själva datan – det under samlingsnamnet molntjänster. Genom att använda molnbaserad infrastruktur med centraliserad hårdvara som delas mellan en mängd olika virtuella maskiner underlättas arbete med stora mängder data. Detta då det är mer skalbart och kan ofta bli billigare än att ha en separat server för till exempel en databas.

1.1. Bakgrund

Elastic Mobile är ett Lundabaserat företag som jobbar med att flytta företags lokala IT-lösningar till molnlösningar – primärt med hjälp av Amazons samling av molntjänster Amazon Web Services (AWS). De har som ett steg i att utöka sina tjänster för databashantering börjat intressera sig för datalager eller data warehouses som det heter på engelska. Datalager är system som innebär att man aggregerar data från en eller flera datakällor – ofta relationsdatabaser, samt inkluderar verktyg för att utföra analyser av datan i samma system. Då de i dagsläget inte använder sig av datalagerlösningarna i AWS såg de ett

examensarbete som en god möjlighet att utforska potentialen i dessa [4]. Det sammanfaller också väl med ett behov hos en av deras kunder – Bring SCM (Supply Chain Management), ett dotterbolag till Bring Frigo som sysslar med logistik och transport i Norden. En av molnservrarna som Elastic Mobile sköter åt Bring SCM innehåller nämligen en analysdatabas som Business Intelligence (BI)-avdelningen hos Bring SCM använder sig av för att analysera affärsdata. De ser ett antal möjligheter till förbättring av sitt nuvarande system och är intresserade av att undersöka användningen av ett datalager för att åstadkomma det, och ur det växte det här arbete fram.

1.2. Syfte

Syftet med arbetet är att undersöka på vilka sätt det går att ta steget från en befintlig lösning med en SQL Server-databas till en datalager-lösning, samt vilka begränsningar och möjligheter det innebär. Fokus ligger på användarvänlighet, belastning på systemen vid datamigration, samt vilka möjligheter till analys som ges. Situationen Bring SCM befinner sig i används som utgångspunkt för att värdera resultaten, och förutom de faktiska prestandaresultaten ska även kostnadsaspekter tas i beaktande.

1.3. Målformulering

Målet med arbetet är att genomföra en förstudie, samt att skapa ett antal prototyper för olika lösningar för datamigration från det befintliga systemet till ett molnbaserat datalager. Ett datalager skall konfigureras för att kunna ta emot och lagra den data som i dagsläget lagras på SQL-servrar, och prototyper för att överföra datan ska tas fram. Prototyperna ska jämföras utifrån vilken belastning de skapar på SQL-databaserna samt hur komplex implementation de kräver. Därefter ska en utvärdering av datalagret samt de migrationslösningar som skapats ske där de jämförs med det befintliga systemet. Resultatet av arbetet ska presenteras för Business Intelligence (BI)-ansvariga på Bring SCM samt molningenjörer hos Elastic Mobile för att ge ett bra underlag för eventuell framtida implementering.

1.4. Problemformulering

Utifrån det mål som beskrivits i föregående avsnitt har ett antal konkreta frågeställningar formulerats, vilka arbetet kommer att försöka besvara. De är som följer:

1. Vilka utmaningar finns vid skapandet av ett datalager gjort för att ta emot data från SQL Server-databaser?
2. Hur migreras en befintlig SQL Server-databas lämpligast till en datalager-lösning?
3. Vilka utmaningar finns vid upprättandet av en löpande datamigration från befintligt system till ett datalager.
4. Vilka flaskhalsar finns i det framtagna systemet?
5. Hur presterar datalagret jämfört med existerande system vid olika typer av SQL-frågor?
6. Vilka kostnader finns, och hur förhåller de sig till kostnaderna för det befintliga systemet?

1.5. Motivering

Som nämnts i inledningen till kapitlet genereras mer och mer data på en global skala, men det är ju inte bara nya datakällor, utan givetvis ökar även mängden data i existerande databaser snabbt. Att hitta bra sätt att skala upp dessa databaser på är då också ett växande behov, och att förlägga de till molndatabaser är en populär lösning. Andelen data som passerat molnet beräknades 2013 vara under 20%, men prognosen för 2020 är närmre 40% [3]. Microsoft SQL Server som är den SQL-version Bring SCM använder är den näst mest populära databashanteraren för kommersiella system och den tredje mest populära av alla databashanterare [5]. En lösning som är anpassad utifrån den är därför relevant för en stor andel av alla databaser som används – och då andra versioner av SQL delar många av de grundläggande egenskaperna hos SQL server kan det även vara relevant för dem.

Motiveringen för Elastic Mobiles del ligger i att utforska fler delar av AWS ekosystem och kunna avgöra om det är något de vill inkludera i sitt utbud [4]. För Bring SCM:s del ligger värdet av det här arbetet

primärt i möjligheten att förbättra och förenkla sitt dagliga arbete med databaserna, men det finns även ett visst intresse för den potential som ett datalager ger för att integrera datan med andra datakällor [6].

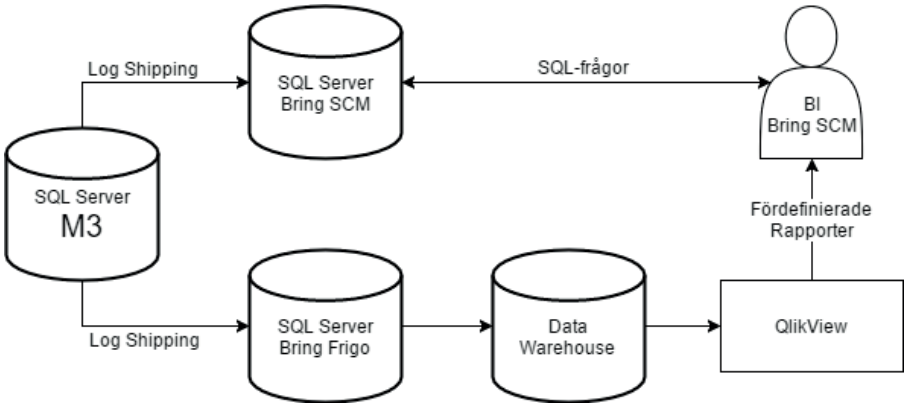
Personligen har data och hur man omvandlar den till information länge varit intressant. Redan i gymnasiet för tio år sen valde jag att i en projektkurs ge mig på att skapa ett program för att visualisera statistik - vilket snabbt visade sig vara alltför omfattande för mina dåvarande förutsättningar. Oavsett tidigare utgång känns det kul att nu kunna göra någonting mer konkret med en liknande inriktning, och dessutom något med hög relevans i dagsläget.

2. Teknisk Bakgrund

I det här kapitlet ges en kunskapsbas kring det befintliga systemet hos Bring SCM samt de verktyg som kommer användas. Först beskrivs den situation Bring SCM befinner sig i med sina servrar, följt av en specifikation av vilka behov ett datalager behöver uppfylla för det här arbetet samt en beskrivning av de verktyg och system som används under arbetes gång. Avslutningsvis ges en kort överblick över vilka alternativ som valts bort för arbetet, samt en redovisning av forskningsläget. Ordet ”arbetet” i bestämd form används här och genom rapporten, och det syftar alltid på denna rapport samt de undersökningar som gjorts till den.

2.1. Befintligt system

I nuläget finns ett system med en serie servrar som är uppdelat mellan Bring SCM och Bring Frigos datalager-avdelning, avbildat i fig. 1.



Figur 1: Nuvarande server-struktur.

Den ursprungliga datakällan för hela systemet består av en molnserver med Microsoft SQL Server, där all data som genereras i Brings affärssystem M3 lagras. En gång i timmen synkroniseras innehållet på den servern med två andra SQL-databaser; en som Bring SCM använder sig av, och en som Bring Frigo använder sig av. För att göra

det används Log Shipping, en process som skapar och uppdaterar kopior av en SQL Server-databas genom att kopiera transaktionsloggen med alla förändringar som skett i källdatabasen, skicka den med filöverföring till målservern, och från loggen replikera ändringarna i måldatabasen. Även måldatabaserna använder sig av Microsoft SQL Server och befinner sig i Amazons moln på virtuella servrar [6].

Bring Frigo har till sin Log Shipping-server kopplat ett datalager av okänd typ, dit de hämtar in och transformerar data från sin SQL-databas. Till det har de kopplat QlikView, ett BI-verktyg som används till att analysera data och presentera den i form av rapporter.

BI-personalen på Bring SCM har för sina analyser tillgång till sin egen SQL-databas som de ställer rena SQL-frågor till, samt ett urval av rapporterna från Frigos datalager. Problemen med systemet kan delas in i tre delar. Först och främst så kräver Log Shipping att SQL-databasen ställs i ”recovery”-läge varje timme när den ska uppdateras, vilket förhindrar att frågor ställs. Förutom att uppdateringen kan störa arbetet i allmänhet kan vissa av frågorna ta väldigt lång tid att ställa – runt en halvtimme för de mest omfattande, och då är en låsning av servern en gång i timmen ett stort störmoment [7]. Vidare är de rapporter de får från Bring Frigo begränsade i det att de inte själva kan göra ändringar – till exempel om de vill lägga till ytterligare en kolumn data – utan för att förändra något måste en förfrågan skickas till Bring Frigo, vilket kan ta onödigt lång tid. Dessutom är Log Shipping inte utan problem, utan då och då blir något fel i överföringen, vilket måste lösas manuellt från Elastic Mobiles sida. Slutligen finns en önskan om att kunna aggregera datan med andra datakällor, något som inte är möjligt i det befintliga systemet.

2.2. Datalager

Ett datalager – data warehouse på engelska – är ett system gjort för att samla in och analysera stora mängder data, ofta ackumulerad från ett flertal källor över en längre tidsperiod. Den främsta användningen av datalager är som datakälla till olika typer av beslutsstödsystem, mer eller mindre automatiserade system som används av företag och organisationer för att ge underlag för välgrundade beslut – till exempel att identifiera målgrupper, trender och dylikt [8].

Datalager är inte helt olika databaser för en användare, men innehåller ofta flera integrerade system utöver själva databasen. En anledning till att man gör skillnad på datalager och databaser är att även om ett datalager använder någon form av databas att lagra sin data på så är en databas optimerad för att hantera transaktioner av data. Ett datalager är däremot optimerat för att hantera sökning i och analys av datan. Generellt sett är ett datalager också större och innehåller data från flera olika databaser. Ytterligare en skillnad är att man vanligtvis väljer att strukturera informationen i ett datalager på annorlunda sätt än hur den är strukturerad i en databas, alternativt använder sig av helt ostrukturerad data [9]. Hur strukturen eller bristen därav väljs beror helt på vilka behov av sökning och analys datalagret skall uppfylla, men valfriheten i designstadiet är ofta större än för en databas då begränsningar så som till exempel primärnycklar inte alltid är ett krav – den typen av krav tenderar att istället ställas på datakällorna som datalagret kopplas mot.

Databasen i ett datalager kan vara en helt vanlig relationsdatabas, men man frångår ofta normalisering och använder annorlunda tabellstrukturer för att optimera prestandan med avseende på analysfrågor. Ett av de vanligare sätten att strukturera datan i datalager som använder sig av relationsdatabaser är stjärn- eller snöflinge-scheman, döpta efter deras schematiska utseende [9]. Grundtanken i båda dessa upplägg är att man har en central tabell, ”fact table” där den viktigaste datan finns, och alla övriga tabeller på något sätt relaterar till den. Skillnaden mellan stjärn- och snöflinge-scheman är att i stjärn-scheman måste alla tabeller vara direkt kopplade mot den centrala tabellen, medans i snöflinge-scheman får varje tabell ha ytterligare tabeller kopplade till sig istället för direkt till den centrala tabellen. Detta för att minska mängden duplicerad data som annars kan bli stor när man frångår normalisering. I detta arbete kommer optimerade schemastrukturer av inte undersökas.

För ett rudimentärt datalager krävs egentligen bara två komponenter eller system utöver datakällorna: ETL eller ELT samt en databas [9]. ETL är ett begrepp inom datalager som står för Extract, Transform, Load; och handlar som det låter om att extrahera data från datakällor, transformera datan till ett format som passar datalagret, och slutligen ladda in den transformerade datan i datalagret. ELT står för samma saker, men i annan ordning – här sker transformationen av data

först när den är inne i datalagret. Utöver dessa två delar kan en mängd andra verktyg kopplas in, analysverktyg, rapport- och revisionssystem, multipla ETL-processer och mycket annat. Då det systemet som används i det här arbetet endast använder sig av en enkel ETL utan någon transformation samt en databas så ligger det mesta av de delarna utanför arbetets syfte. Fokus kommer alltså att ligga på en ETL-process, en databas samt koppling av analysverktyg till databasen.

2.3. Verktyg och system

Det finns en uppsjö av datalager-lösningar tillgängliga på marknaden för alla steg i processen: lagring, ETL, analys etcetera. En övergripande analys av de tillgängliga alternativen hade varit alltför omfattande för ett sådant här arbete, så en begränsning får göras. Då målet är att ge ett underlag för hur en migration till ett datalager kan ske blir det en rimlig avgränsning att använda sig av endast ett destinationssystem, och fokusera på att utforska de alternativ som finns för migrationen dit.

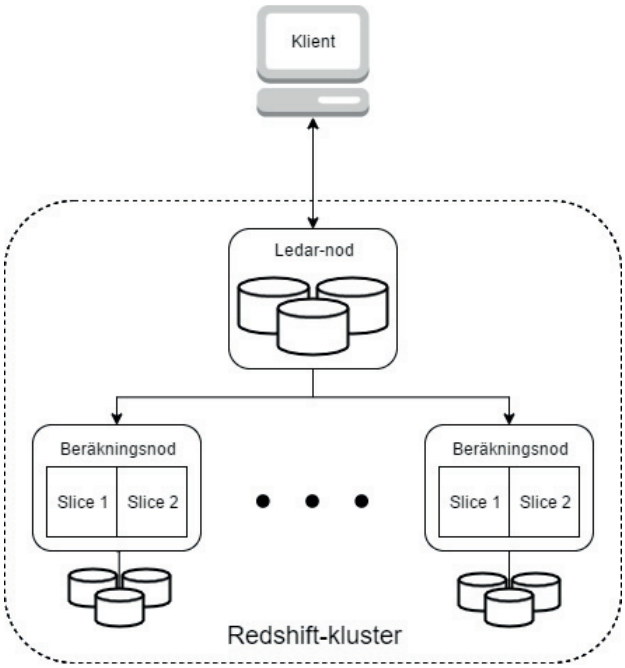
2.3.1. Redshift

Det datalager-system som valts för det här arbete är Amazon Redshift, av ett flertal anledningar. Först och främst så är Bring SCM:s befintliga servrar redan hostade i Amazons molnnätverk, vilket ger en rad fördelar för Redshift. Det ger ökad datasäkerhet då servrarna befinner sig inom samma nätverk och till och med samma serverpark [10], och trafiken mellan servrarna behöver inte skickas över externa nätverk. Av samma anledning har det potential att minska de löpande kostnaderna en del, då extern nätverkstrafik kostar, medans trafik inom samma nätverk i regel är avgiftsfri [11]. Utöver det så förenklar det även arbetet genom att konsolidera de tjänster som används till en leverantör, och det ger möjlighet att använda en del av Amazons tjänster som inte hade varit tillgängliga annars. Mer specifikt är det AWS Database Migration Service (DMS) som är intressant, vilket är en lösning Amazon har för just databasmigration.

Systemet är uppbyggt för att från klientens sida bete sig precis som en databas, och går att kommunicera med genom ett valfritt av två olika API:er (Application Programming Interface), antingen JDBC (Java Database Connectivity) eller ODBC (Open Database Connectivity) – precis samma typ av API som används för vanliga databaser [12].

Redshift använder sig av en modifierad version av PostgreSQL 8.0.2 och kan i mångt och mycket behandlas som om det vore en PostgreSQL-databas vad det gäller SQL-frågor, men på hårdvaru- och infrastruktur-nivå finns det däremot en del väsentliga skillnader.

Redshift använder sig inte av en separat server, utan ett Redshift-datalager består av ett serverkluster med en eller flera noder – visualiserat i fig. 2 – som var och en har dedikerad hårdvara [12].



Figur 2: Redshifts övergripande arkitektur. Inspirerad av en bild från Redshift-dokumentationen.

Om klustret har mer än en nod är en av noderna alltid en ledar-nod, vilken sköter kommunikationen till och från klienter och avgör om den kan besvara SQL-frågorna själv eller behöver ta hjälp av övriga noder. De övriga noderna är beräkningsnoder som används för att parallellisera mer omfattande beräkningar som kräver tillgång till många kolumner, dessa noder kommer i två generella kategorier, beräkningsoptimerade noder – ”dc1.” – samt lagrings-optimerade

noder – ”ds2.”. Inom varje kategori finns ett antal olika typer att välja på, med varierande beräknings- och lagrings-kapacitet man kan välja utefter behov, men alla noder inom samma kluster måste dock vara av samma typ. För att åstadkomma parallellisering kompilerar ledarnoden kod som sedan distribueras till övriga noder, exekveras, och resultaten skickas tillbaka och sammanställs i ledarnoden.

Själva databasen lagras uppdelad över hela klustret och är lagrad sorterad efter kolumn för att vara bättre anpassad för analysfrågor istället för sorterad efter rad som de flesta SQL-databaser tenderar att vara, då det lämpar sig bättre för läs- och skriv-frågor [13]. Hur uppdelningen sker går att bestämma manuellt genom att bestämma vilken kolumn som används som nyckel för att sortera, alternativt går det att låta Redshift själv avgöra. Varje nod är i sig uppdelad i två eller flera ”Slices” – se Slice 1, Slice 2 i bilden ovan – som fungerar som virtuella instanser där varje instans allokeras en del av nodens beräkningsresurser samt en bestämd del av databasen.

Varje kolumn i databasen är också komprimerad för att minimera lagringsutrymmet, något som är högst relevant när man börjar aggregera stora mängder data då det minskar hårddiskanvändningen [14]. Denna komprimering kan väljas manuellt, men det rekommenderas att låta Redshift själv välja komprimering, vilket sker per automatik om COPY-kommandot används – mer om detta kommando senare. Komprimering sker per kolumn, och varje kolumn delas sorterad upp i ett antal olika fragment så att endast de relevanta fragmenten för en SQL-fråga behöver läsas.

2.3.2. Övriga verktyg och system

Utöver Redshift kommer ett antal olika system och verktyg användas till arbetet, varav merparten av dem är system inom Amazons moln. Valet att hålla sig till Amazons tjänster är delvis baserat på tillgången till omfattande dokumentation, samt rekommendationer ifrån den enda akademiska källa som specifikt avhandlar migration mellan SQL Server och Redshift [15].

Som datakälla finns en m4.large EC2-instans med Microsoft SQL Server 2008 R2 och en EBS-volym på 2 TB. SQL Server är en databashanterare som använder sig av Transact-SQL (T-SQL), och som nämnts i inledningen en av de mest använda databashanterarna i

världen. Den finns en uppsjö av funktioner och verktyg inkluderat i den, men det är inget som kommer användas i detta arbetet.

EC2 och EBS står för Elastic Compute Cloud respektive Elastic Block Storage, och de är två AWS-tjänster som används tillsammans för att skapa virtuella maskiner på Amazons molnnätverk. m4.large specificerar vilken hårdvara instansen använder, i det här fallet två Intel Xeon-processorer med 8 GB primärminne [16]. Det här är något nerskalat ifrån den server Bring SCM använder i produktion, vilken använder sig av en r3.xlarge-instans med fyra motsvarande processorer och 30,5 GB minne, samt en mer minnesoptimerad arkitektur. Den EC2-instans som används är innehållsmässigt en exakt avbild av den server Bring SCM använder sig av för dataanalys, som i sin tur är en avbild av den SQL-databas de använder sig av i produktion. Undantaget att den inte har någon aktiv koppling till produktions-databasen M3 använder sig av, utan den befinner sig i det tillstånd som produktionsdatabasen befann sig i februari 2017.

AWS Simple Storage Service (S3) används för mellanlagring. S3 är en molnbaserad datalagringslösning för enkel data-lagring i vad de kallar ”buckets”, som använder en platt filstruktur – det vill säga utan mappstrukturer eller liknande [17]. S3 använder sig inte av några fördefinierade storlekar utan skalas automatiskt efter hur mycket som lagras. För att ladda in data i Redshift använder sig DMS av en för användaren dold S3-bucket, men utöver det kommer även en separat S3-bucket användas.

Som tidigare nämnt kommer också Database Migration Service (DMS) användas under arbetet. Det är en tjänst som Amazon skapat för att förenkla migrationer mellan databaser när minst en av ändpunkterna befinner sig inom AWS:s moln. För att utföra migrationer används en replikationsinstans som utför alla beräkningar som behövs i ETL-processen. För att hämta data från källdatabasen läser DMS transaktionsloggen ifrån SQL Server-databasen på samma sätt som Log Shipping gör [18], återskapar därefter tabellerna i en S3-bucket som skapas automatiskt av DMS, för att därefter ladda upp tabellerna till Redshift-klustret [19]. För att övervaka överföringarna används AWS CloudWatch, som samlar de loggar DMS skapar vid överföring.

Samtliga ovan nämnda system är del av AWS, och befinner sig inom samma Virtual Private Cloud (VPC). Ett VPC fungerar precis som ett lokalt nätverk fast med logiska istället för fysiska avgränsningar utåt [20]. Det innebär att man har interna IP-adresser inom ett VPC, och kan ha en samlad brandvägg och anslutnings-policy mot extern trafik.

Slutligen har valet gjorts att använda SQL Workbench/J för alla moment som involverar ställandet av SQL-frågor. SQL Workbench/J är en klient med fokus på scripts och textbaserad interaktion som går att koppla till ett flertal olika databastyper – däribland SQL Server och Redshift som den kommer användas till här [21].

2.4. Alternativ

Det finns en uppsjö av mer eller mindre liknande datalager-lösningar tillgängliga, och ännu fler sätt att sköta ETL-processer in i dessa [22]. Vad det gäller själva datalagret är det absolut mest relevanta konkurrerande alternativet Microsofts Azure SQL Data Warehouse [23]. Det är även det ett molnbaserat datalager byggt på en SQL-variant, i det här fallet en version av SQL Server. Särskilt om det skulle vara en stor fördel att ha kvar SQL Server istället för att byta till PostgreSQL är detta ett mycket aktuellt alternativ. Den direkt synliga nackdelen är att data behöver skickas mellan olika nätverk, vilket kan innebära både säkerhetsrisker och kostnader, men det är inget som hindrar SQL Data Warehouse från att vara ett högst relevant alternativ.

Utöver fulla datalager-lösningar finns även mer lågnivå-fokuserade lösningar som Hadoop [24] och Spark [25]. Den här typen av lösningar är ofta väldigt kraftfulla, men har den stora nackdelen att de är mycket mer komplexa i och med att de är ramverk som behöver implementeras på mjukvarunivå. Ett annat alternativ är att använda sig av analysdatabaser förlagda på vanliga servrar, som till exempel Oracle erbjuder [26]. Detta tenderar att vara avsevärt dyrare och är därför ett mindre relevant alternativ om det inte finns specifika behov av att datan måste lagras lokalt.

Vad det gäller ETL-lösningar är utbudet ännu större, och det är dessutom väldigt svårt att skapa en uppfattning om vilka för- och nackdelar som finns hos de olika systemen utan att genomföra faktiska

tester. Exempel på lösningar som kan vara intressanta är ETLeap [27], Flyway [28] samt Matillion [29]. Matillon är en helhetslösning som bygger på befintliga AWS-system, Flyway fokuserar primärt på databasscheman, och ETLeap fokuserar på att samla flera datakällor. Amazon själva har också tillkännagett en kommande lösning för ETL kallad Glue, som kan vara intressant att titta på när det blir tillgängligt [30].

2.5. Forskningsläge

En sökning på en vanlig sökmotor ger snabbt en uppsjö av bloggar och hemsidor som behandlar hur man bör eller inte bör flytta en databas till datalager-lösningar. Söker man däremot bland akademiska källor är det betydligt mer begränsat med material, och fokus ligger mer på hur man designar datastrukturerna från grunden snarare än hur man migrera data. Detta är fullt rimligt och mer eller mindre väntat – innan data laddas in i datalagret behöver det finnas en struktur för hur den ska lagras väl där. Tills man kommit till någon koncensus om detta riskerar det som bygger vidare på det att stå på ostadiga grunder.

Vad det betyder för detta arbete är att det finns en akademisk grund att stå på i form av hur ett datalager bör se ut, men samtidigt behandlas ett specifikt ämne där det finns väldigt begränsat med akademisk historia.

3. Metod

I följande kapitel beskrivs och diskuteras de metoder som använts under arbetet. Övergripande utförs arbetet i fyra huvudsakliga delar som motsvarar kapitel 3.1, 3.2.1, 3.2.2 och 3.4. Först och främst skapas ett Redshift-kluster som konfigureras för att innehålla en databas med innehåll motsvarande den SQL Server-databas som Bring SCM använder sig av. Nästa steg är en migration av den befintliga databasen där allt innehåll samt tabellstruktur överförs, och därefter följer upprättandet av en kontinuerlig överföring som löpande replikerar förändringar ifrån källdatabasen till Redshift. Slutligen genomförs en serie tester på det upprättade systemet. Kapitel 3.1 till och med 3.3 avhandlar migrationen av data från den befintliga SQL-servern in i Redshift, och därefter beskrivs de tester som sker på datan när den väl finns i Redshift i kapitel 3.4. En sammanfattning av testmiljön ges i kapitel 3.5, och en kostnadsanalys beskrivs i kapitel 3.6. Avslutningsvis sker en reflektion över tillförlitligheten hos de källor som använts för arbetet i kapitel 3.7.

Något som också bör nämnas är att inte alla tester inte är genomförda, då de tester som redovisas i kapitel 3.3 i ett tidigt stadium visade sig vara olämpliga.

Det finns generellt sett tre tillvägagångssätt för att interagera med tjänster inom AWS [31]; via webbaserad kontrollpanel, Command-Line Interface (CLI), eller API – alla med sina för- och nackdelar. Då API används för att integrera kommunikationen i kod, och det inte finns ett behov av att skriva ett program för att genomföra det här arbetet är API inte lämpligt för arbetet. Såväl CLI som den webbaserade kontrollpanelen kan vara lämpliga att använda, och ger motsvarande grader av kontroll och information. Den webbaserade kontrollpanelen är att föredra, då CLI kräver mer förarbete med till exempel konfiguration av brandväggar till molnnätverket samt installation av en klient.

3.1. Upprättande av Redshift-kluster

För att skapa ett Redshift-kluster i AWS:s kontrollpanel följer man en guidad process där man först specificerar identifikationsdata så som

klustrets namn, huvudanvändare, och dylikt. Därefter väljer man hur många noder klustret ska bestå av, samt vilken typ av noder dessa ska vara. Redshift använder sig som beskrivet i föregående kapitel av två typer av noder, beräkningsoptimerade DC1-noder, samt lagrings-optimerade DS2-noder. För att avgöra vilket alternativ som passar behoven bäst skapas ett kluster av varje typ. För det lagringsoptimerade klustret används en enda dc2.xlarge-nod, och för det beräkningsoptimerade klustret används initialt åtta stycken dc1.large-noder, vilket sedan utökades till tio då åtta visade sig ge otillräckligt lagringsutrymme. I båda fallen är de ursprungligen dimensionerade utefter den nodstorlek som har minst lika mycket utrymme som SQL-databasens storlek.

Därefter följer en serie inställningar kring säkerhets- och nätverks-konfiguration, varav endast en är relevant här, och det är valet huruvida databasen ska vara krypterad. Då källdatabasen befinner sig på samma nätverk och inte är krypterad får behovet av kryptering vid en framtida implementation anses obefintligt.

När ett Redshift-kluster väl konfigurerats och startats går det inte att stänga ner och starta upp igen som man kanske förväntar sig av de flesta typer av servrar. För att stoppa ett Redshift-kluster måste hela klustret raderas [32]. Man kan dock spara ner innehållet samt klusterkonfigurationen i ett ”Snapshot” lagrat i en S3-bucket, och vilket man kan använda för att upprätta ett nytt kluster med samma innehåll. För att testa denna funktionalitet skapas ett Snapshot av klustrets innehåll efter att hela SQL-serverns innehåll överförts, klustret raderas, för att sedan återskapas igen, och tiden som går åt till de båda momenten mäts.

Slutligen görs ett enkelt test där ett kluster fylls med data för att sedan utökas med ytterligare en nod, för att mäta hur snabbt ett kluster kan skalas. Detta sker på klustret som består av dc1.large-noder.

3.2. Database Migration Service

För att migrera en databas till Redshift kan man använda DMS, Amazons egna migrationsservice. För att göra det använder man en replikationsinstans – en server som sköter ETL-processen mellan databaserna. Denna instans är i sig inte kopplad till några specifika

databaser, utan varje ETL-process kopplas till en valfri replikationsinstans som sedan sköter de beräkningar och databas-frågor som behövs. Dessa ETL-processer beskrivs i något som DMS kallar för ”tasks”, i vilka källa, destination, samt eventuell transformering specificeras. ”Task” kommer nedan att översättas till ”överföringsprocess”.

Replikationsinstansen som används i det här arbetet används för samtliga överföringsprocesser är av typen dms.t2.medium, som är ett av de mindre kraftfulla alternativen. Skalan går från 1 virtuell CPU (vCPU) och 1 GB primärminne upp till 16 vCPU och 30 GB primärminne, och en dms.t2.medium-instans har 2 vCPU och 4 GB minne [33]. Det valet motiveras av att det i arbetet inte finns någon stor tidspress, samt att en utgångspunkt bör befinna sig en bit in på skalan för att ge bättre indikationer på vilken storlek en framtida fullskalig implementation skulle behöva.

Kopplat till DMS finns även ett verktyg som sköter ETL-processen för själva databasschemat, AWS Schema Conversion Tool [34]. Det är ett program som automatiserar konverteringen och överföringen av data-typer och strukturer mellan två databaser eller datalager, och ger ett grafiskt gränssnitt för att modifiera konverteringarna, samt manuellt specificera de konverteringar som inte kunde automatiseras. Schema Conversion Tool används till att säkerställa att schemat kan konverteras korrekt.

3.2.1. Initial migration

Den initiala migrationen av hela databasen kan ske på flera sätt, då varje överföringsprocess kan innehålla allt från ett enda fält till en hel databas beroende på inställningar, samt kan utföra transformeringar av datan. I det här fallet testas två stycken tillvägagångssätt: Först görs en migration av hela databasen i en överföringsprocess, sedan görs en migration där varje databasschema placeras i en egen överföringsprocess.

Valet av hur omfattande varje överföringsprocess är sker genom att välja vilka scheman som inkluderas, och med möjlighet att filtrera baserat både på tabeller och även kolumninnehåll [35]. Det är dock bara filtrering och transformering av namn som är möjligt, det går inte att skapa några mer avancerade ETL-processer i DMS. I den samlade

överföringsprocessen läggs alla scheman till, och i de uppdelade väljs ett schema per överföringsprocess. Utöver det finns en del övergripande inställningar för migrationen vars detaljer till största del är oväsentliga för det här fallet. Värt att nämna är dock att man kan ställa in antalet tabeller som laddas parallellt, och där används standardinställningen med åtta parallella tabeller.

Det finns inget krav på att definiera databasschema och tabeller i Redshift-klustret i förväg, dessa kommer skapas allt eftersom de behövs under överföringen. Skulle det redan finnas ett fördefinierat schema går det att antingen fortsätta skriva ny data till tabeller med samma namn, ta bort hela tabellen med DELETE före skrivning, alternativt använda SQL:s TRUNCATE för att ta bort alla referenser till tabelldatan. TRUNCATE motsvarar en borttagning med DELETE, men går snabbare och lämnar inga spår i loggarna, vilket innebär att det inte går att återställa tabellen ifrån loggarna.

Hela överföringen loggas med hjälp av Amazon CloudWatch [36] – ett system för att övervaka AWS-tjänster – för att upptäcka eventuella fel. För att mäta tiden överföringen tar används de tider som DMS själv visar upp i kontrollpanelen för varje överföringsprocess.

3.2.2. Löpande replikation

För den kontinuerliga replikationen skapas överföringsprocesserna på samma sätt som tidigare, med skillnaden att migrationstypen är vald till ”replicate data changes only” så den data som fanns före skapandet av överföringsprocessen inte tas med – då den har blivit flyttad i föregående steg.

Då systemet är helt frikopplat från produktionssystemen kommer inga uppdateringar från annat håll att ske i databasen, utan de transaktioner som ska replikeras får skapas för hand i källdatabasen. För att testa detta skapas en enkel tabell i ett nytt schema enligt följande kommando:

```
CREATE SCHEMA testschema;
CREATE TABLE testschema.test(
  Number int,
  Text1 varchar(255),
  Text2 nvarchar(255),
  Date1 date,
  Date2 datetime,
  PRIMARY KEY (Number)
);
```

En överföringsprocess skapas i DMS och konfigureras för att läsa kontinuerligt endast ifrån det schemat, varefter 1000 rader data skrivs med enkla INSERT-kommandon till tabellen med hjälp av SQL Workbench/J för att simulera aktivitet. Vid ett senare tillfälle tas tabellinnehållet bort med DELETE-kommandot, och när den även försvunnit ifrån Redshift laddas 1000 rader data igen. Datan som skrivs är genererad med hjälp av mockaroo.com [37], ett verktyg gjort för att skapa testdata.

Då det är en kontinuerlig process går det inte att använda tiderna i DMS:s kontrollpanel för mätning, utan istället används Amazon CloudWatch för att samla loggarna och se hur lång fördröjning det är mellan skrivning och läsning. Då det inte framkommer några som helst indikationer på att Redshift skulle vara en flaskhals sker testet endast mot det ena – lagringsoptimerade – klustret.

3.2.3. Simple Storage Service

Sedan mars 2017 stödjer DMS användningen av Simple Storage Service (S3) som destination [38]. S3 är en rudimentär lagringsservice i AWS som bygger på ett platt filsystem där filer lagras i ”buckets”, motsvarande hårddiskar men utan fördefinierad storlek. Då möjlighet redan finns att använda Redshift som destination i DMS är behovet av använda mellanlagring så som S3 reducerat. Det kan dock fortfarande vara relevant för mer komplexa ETL-processer att mellanlagra data mellan olika steg, alternativt för att använda som säkerhetskopiering integrerad i ETL-processen, vilket gör att det fortfarande är relevant att undersöka.

Tillvägagångssättet här är identiskt med hur DMS används för initial migration med Redshift som destinationssystem. En migration görs där samtliga scheman samlas i en överföringsprocess, och en annan migration sker där varje schema får en överföringsprocess. Även här används CloudWatch för att logga processerna.

3.3. Alternativa metoder

Utöver Database Migration Service finns det en mängd olika alternativ för att utföra arbetet. De som är mest intressanta att undersöka för det här arbetet är Redshifts COPY-kommando samt Log Shipping då de båda är väldigt närliggande mål- respektive käll-system.

3.3.1. Redshifts COPY-kommando

För att ladda in data i ett Redshift-kluster är det kutym att använda sig av Copy-kommandot som finns i den modifierade version av PostgreSQL som Redshift använder sig av [39]. Det gäller såväl när DMS laddar in data som när man ska interagera med ett kluster på en lägre abstraktionsnivå till exempel om man skriver egen kod för att sköta uppladdningar.

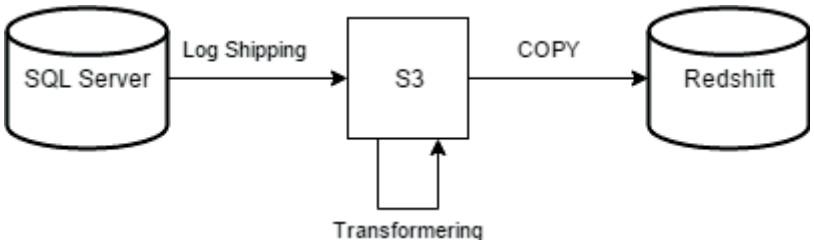
För alla former av implementeringar som inte använder sig av en färdig plattform är COPY-kommandot någonting man kommer behöva använda sig av. Då en djupgående analys av det här tillvägagångssättet kräver att det skapas ett program eller script från grunden, inklusive eventuella diagnostik-verktyg för att kunna göra meningsfulla mätningar är det inget som får plats inom ramen för arbetet.

Den ursprungliga tanken var att skriva ett script för att testauppladdning av data från S3 till Redshift. På grund av att mellanlagring i S3 inte gav önskat resultat – vilket redovisas i nästa kapitel – kommer detta test inte genomföras.

3.3.2. Log Shipping

Systemet som undersöks använder sig av Log Shipping, vilket gör det intressant att undersöka möjligheten att se om det går att använda sig av även i det här fallet. Redshift har inget stöd för att läsa transaktionsloggar från SQL-databaser, utan en lösning som bygger på Log Shipping behöver något sätt att läsa av transaktionsloggarna och

mellanlagra databasinnehållet någonstans. Då Redshift är designat för att använda S3 som primär datakälla vore det ett lämpligt val. ETL-processen skulle då se ut som visas i fig. 3.



Figur 3: Föreslagen systemarkitektur för Log Shipping till Redshift.

Det stora problemet med ett sådant system är att det är komplext samt att det inte går att hitta någon information om liknande lösningar, vilket indikerar att det inte är en eftersträvansvärd lösning. Fördelen – och vad som skulle vara det enda skälet att välja en sådan här lösning – är att Log Shipping redan är konfigurerat på M3-servern, och en implementation som använder sig av det då gör minimal påverkan på källsystemet. Då läsning av transaktionsloggar är ett förhållandevis komplicerat och omfattande arbete kommer det inte testas, men det är relevant att ha i åtanke om övriga lösningar skulle vara problematiska att implementera med M3-servern som källa.

3.4. SQL-frågor

För att testa systemet används åtta stycken SQL-frågor som Bring SCM bidragit med, och som de använder i sitt arbete – dessa finns i sin helhet i appendix A. Genom att använda de frågorna får man ett testscenario som är identiskt på både käll- och mål-system, och som dessutom är snarlikt den verkliga användningen. Frågorna ger inte bara ett mått på hur snabbt de olika systemen kan exekvera dem, utan resultaten fungerar även som stickprov på att datan finns och är korrekt formaterad i Redshift-klustret. För att ställa frågorna används SQL Workbench/J, och frågornas exekveringstider avläses ifrån samma program. Då databashanterare ofta har funktioner för att optimera

prestandan av frågor som ställs ofta kommer varje fråga ställas tre gånger i följd för att se effekten av denna optimering.

De här frågorna är skrivna för SQL Server, och några mindre förändringar krävdes för att översätta frågorna till PostgreSQL för Redshift. Förändringarna är dock minimala, och bör i sig rimligtvis inte ge någon märkbar påverkan på exekveringstiden. Det som förändrats är tre olika saker. Först och främst använder sig SQL Server av ”WITH (NOLOCK)” som suffix på vissa rader för att specificera att frågan kan läsa även ifrån rader som är låsta av en annan SQL-transaktion. I Redshift finns inget sådant lås från grunden [40], och tillägget kan helt enkelt tas bort. Vidare var formateringen av data vid vissa WHERE-satser i SQL-frågorna otillräcklig för Redshifts skarpare krav på formatering, och på ett antal platser fick citationstecken läggas till eller tas bort runt värden för att möta dessa. Slutligen innehöll en av frågorna ett CONVERT-kommando som konverterade datum till ett annat format. PostgreSQL saknar den variant på CONVERT-kommandot som SQL-frågan använde sig av, och valet gjordes att helt enkelt ta bort konverteringen då den inte var central för SQL-frågan.

3.5. Testmiljö

Sammanfattningsvis består testmiljön av en SQL Server-databas, en DMS-migrations-instans, två Redshift-kluster, tre S3-buckets – varav två automatiserade av de två Redshift-klustren – och utöver detta en Windows-PC med SQL Workbench/J för att styra allting. För att minimera antalet externa variabler sker testerna helt sekventiellt, med endast ett Redshift-kluster eller en S3-bucket kopplad till övriga system åt gången.

Att SQL Server-databasen som är datakälla är en kopia på en av Bring SCM:s servrar är till stor hjälp för arbetet. Först och främst så är datan identisk med den som används i produktion, vilket innebär att testerna sker på data som är lik produktionsdata både i omfattning och hur den är strukturerad. Servern är förlagd i samma Amazon-datacenter som den faktiska produktionsservern, vilket innebär att även infrastrukturen efterliknar produktionsmiljön väl. Skillnaden mellan servrarna är vilken typ av serverinstans de ligger på, där Bring SCM:s server är betydligt kraftfullare, som specificerats tidigare. Valet av en mindre kraftfull serverinstans för testning är gjort av Elastic Mobile

och är inget som går att förändra, utan det är något som får tas i beaktande vid tester som använder sig av denna server.

Användandet av två Redshift-kluster motiveras av att det ger en bild av hur väl Redshift skalar, och ett bra underlag för vilken typ och storlek av kluster som är lämplig att implementera.

Hela systemet befinner sig inom samma VPC på ett av Amazons datacenter, vilket dels innebär att de olika komponenterna har en fysisk närhet till varandra som minskar kommunikationstiden mellan dem och dels att ingen trafik rör sig utanför det interna nätverket.

3.6. Kostnader

En högst relevant aspekt för den här typ av system är monetära kostnader. Då molnbaserade system bygger på att stora mängder infrastruktur delas upp på mindre servrar är den egentliga begränsningen där vad som är rimligt att betala för då det nästan alltid går att välja bättre hårdvara. Kostnaden för lösningar som kräver mjukvaruutveckling eller på annat sätt är i huvudsak begränsade av arbete snarare än infrastruktur är mycket mer svåruppskattade. Därför har fokus lagts på en kostnadsjämförelse mellan de tjänster som används hos Amazon i dagsläget samt vid eventuellt upprättande av ett Redshift-kluster i produktionsmiljö med Database Migration Service för att replikera innehållet ur produktionsdatabasen.

Fyra system kommer jämföras. Primärt är det den EC2-server Bring SCM använder sig av som ska jämföras med de två typerna av Redshift-kluster, men för att kunna sätta kostnader i relation till prestandajämförelsen kommer även den EC2-server som används för testerna att inkluderas i prisjämförelsen. De relevanta delarna av konfigurationerna som jämförs är som följer:

- Bring SCM:s server, som är en r3.xlarge EC2-instans, EBS-optimerad med Windows och SQL Server Standard. Kopplad till denna är 1600GB General Purpose EBS.
- Testservern som använts, som är en m4.large EC2-instans, EBS-optimerad med Windows och SQL Server Standard. Kopplad till denna är 1600GB General Purpose EBS.

- Det lagringsoptimerade Redshift-klustret, som använder en ds2.xlarge-nod. DMS med en dms.t2.medium-instans.
- Det beräkningsoptimerade Redshift-klustret, som använder åtta stycken dc1.large-noder. DMS med en dms.t2.medium -instans.

AWS egna verktyg för prisuppskattning [41] kommer att användas, och jämförelsen sker på icke reducerade priser – i en implementation kan dessa komma att rabatteras med avtal, men ett icke reducerat pris är enklare att jämföra då det är svårt att veta exakt vilken rabattering som kommer ske. Samtliga prisberäkningar sker inom en och samma AWS-region, då priserna kan variera något mellan olika regioner i världen.

3.7. Källkritik

Kunskapen som inhämtats för detta arbete kommer från en mängd olika avsändare. Först och främst är det dokumentationen för och föredrag om Amazon-tjänster som är den huvudsakliga källan för hur det praktiska arbetet genomförs. Till dessa hör [10-14], [16-20], [30-36], [38-41] samt [46]. De är över lag aktuella, och deras riktighet bekräftas delvis av arbetets lyckade resultat. Däremot har Amazon givetvis ett intresse av att deras produkter framstår i god dager¸ och vid ett antal tillfällen noterades det att fördelar med tjänsterna nämndes långt mycket tydligare än nackdelar. Över lag är materialet dock informativt och sakligt så länge man tar det i beaktning.

För de mer teoretiska aspekterna av arbetet har huvudsakligen två böcker används, *”Building a Data Warehouse, with examples on SQL Server”* av V. Rainardi [8] samt *”Advanced Data Warehouse Design, From Conventional to Spatial and Temporal Applications”* av E. Malinowski och E. Zimányi [9]. Rainardi har lång erfarenhet av att arbeta med flertalet relevanta system [42], och både Malinowski samt Zimányi är båda professorer inom datavetenskap och har omfattande publicering inom ämnet [43][44]. Den kritik som skulle kunna riktas mot dessa källor är att de är få till antalet. Det stämmer, och det är primärt på grund av arbetets mer praktiska fokus, då majoriteten av den

vetenskapliga litteratur som finns tillgänglig har ett mycket snävare fokus än vad som är applicerbart för detta arbete.

Två tidningsartiklar – [1] och [2] – från Internet används i inledningen som källor. De är troligtvis de minst tillförlitliga av arbetets källor, då det kan inte uteslutas att det skulle finnas någon partiskhet eller agenda bakom dessa. Påståendena där stämde dock överens med vad andra källor påstod, och då ingen vikt av arbetet i stort läggs på informationen därifrån anses det oproblematiskt om de inte skulle vara fullt korrekta. Ytterligare en artiken från Internet – [15] – användes, denna är dock mer tillförligtlig då författaren har gott om erfarenhet i det han skriver om. Andra Internet-källor som används är en mängd hemsidor för olika produkter och tjänster: [21-29], [37], [47]. De är officiella informationskanaler för respektive företag, så de får anses tillitliga, dock med insikten om att de kan vara något säljande.

Vidare finns det en del personlig kommunikation – [4], [6], [7] – med i arbetet involverade parter, där det inte finns någon anledning att tvivla på sanningshalten. Två källor refererar till lagtext [48][49] och dessa ses utan tvekan även de som goda källor. Slutligen finns det två undersökningar, [3] och [5]. Den första är gjord av en stor global och etablerad IT-fokuserad analys-firma och får anses tillförlitlig. För den andra är avsändaren inte lika välkänd, men den redovisar å andra sidan utförligt sin metod, vilket stärker tillförlitligheten mycket.

För värdering av Internet-källor har guiden för källkritik på Internet som Internetstiftelsen i Sverige [45] gett ut använts som hjälp.

4. Resultat

I detta kapitel presenteras resultaten av de undersökningar och tester som beskrevs i föregående kapitel. Kapitelindelningen följer metodkapitlets indelning och ordning, med undantag för kapitel 3.3 och 3.5 som inte gett upphov till några resultat att redovisa och därför saknar motsvarighet här.

4.1. Redshift-klustret

Skapandet av Redshift-klustren krävde en del arbete med konfigurationen, men det skedde utan några problem. Första försöket fördröjdes av otillräckliga rättigheter för användarkontot, men när de korrekta rättigheterna tilldelats flöt arbetet på problemfritt. Tidsåtgången var endast ett fåtal minuter både för konfiguration och för uppstart av klustret.

Att spara ner ett ”snapshot” av klustret med hela databasen överförd fungerade klanderfritt och gick även det på ett fåtal minuter. Att återskapa klustret tog något längre tid – cirka 50 minuter för det lagringsoptimerade klustret – och var problemfritt. Dock identifierades någonting som inte påverkade det här testet, men som kan vara relevant för användning av Redshift i skarp läge; Det går att ändra många inställningar vid återställning av ett sparat kluster, men det går inte att ändra nodtyp eller antal noder, utan detta kan endast göras på ett kluster som är aktivt.

Tillägg av ytterligare en nod i det fulla klustret var mycket enkel att starta, och tog cirka 6 timmar.

4.2. Database Migration Service

Det visade sig omedelbart att Schema Conversion Tool inte var applicerbart på det här arbetet. Det såg ursprungligen ut att vara användbart för arbetet, men det visade sig så fort verktyget skulle användas att det endast stödjer konvertering mellan homogena system, det vill säga två relationsdatabaser eller två datalager. Då arbetet gäller konvertering mellan ett system av vardera typ gjorde det att det inte är

något som gick att använda i det här fallet, utan arbetet fick genomföras utan detta verktyg.

4.2.1. Initial migration

Efter att replikationsinstansen konfigurerats och upprättats utan problem skapades och startades överföringsprocesserna. Vid första försöket med överföring till det lagringsoptimerade klustret misslyckades både försöket med flera överföringsprocesser samt det med en överföringsprocess – en stor andel av tabellerna, flera av de stora, gick inte att överföra. Orsaken till detta var svår att tyda ur loggarna. Det fanns inga mönster som tydde på att någon särskild typ av data eller tabell gav upphov till problemen, men den gemensamma nämnaren var att problemen uppstod av att replikationsinstansen inte lyckades läsa tabellerna ifrån SQL Server-databasen. Efter support från Elastic Mobile och utökade användar-rättigheter i AWS gjordes ett nytt försök, och denna gång lyckades samtliga schema-uppdelade överföringsprocesser utan varningar, medan den samlade överföringsprocessen återigen misslyckades för det lagrings-optimerade klustret.

För det beräkningsfokuserade klustret visade det sig att storleken på klustret var marginellt för liten – 4 tabeller på cirka 100Gb total fick inte plats vid testet med en samlad överföringsprocess. Att skala upp klustret beräknades ta nästan lika lång tid som den ursprungliga överföringen utifrån det tidigare testet, så av tidsskäl identifierades istället ett antal större tabeller, och dessa togs bort ifrån Redshift för att ge plats åt de som inte överförts. Överföringstiden för övriga tabeller adderades till resultatet, det fås det ta i beaktning att den faktiska tidsåtgången kan skilja sig, och detta endast är ett estimat. Den schemavis uppdelade överföringen gjordes till detta nya kluster, och skedde utan problem.

Resultatet av dessa tester presenteras i Tabell I nedan, uppdelad mellan samlad respektive uppdelade överföringsprocesser, samt till vilken klustertyp migrationen skedde.

TABELL I. TIDSÅTGÅNG FÖR INITIAL MIGRATION

Metod	*Klustertyp*	*Resultat*
Samlad	Beräkning	26h 19m
Schemavis	Beräkning	31h 36m
Samlad	Lagring	11h 52m
Schemavis	Lagring	14h 45m

Undersökningen visade också att filstorleken på databasen varierade stort beroende på system. I SQL-server upptar den cirka 1200 GB, det lagringsoptimerade klustret behöver ungefär 200GB och det beräkningsoptimerade klustret behövde nästan 1600 GB.

4.2.2. Löpande replikation

Den löpande replikationen fungerade mycket väl. Det krävdes att SQL Server-databasen ställdes in till att publicera sina transaktionsloggar, men utöver det uppstod inga problem under genomförandet. Både INSERT och DELETE-kommandon tog bara ett fåtal sekunder innan ändringarna replikerades, och det gick utmärkt att läsa och skriva till båda databaserna parallellt med detta – varken CPU eller läsning/skrivnings-kapaciteten belastades nämnvärt.

4.2.3. Simple Storage Service

Tester med S3 gav liksom den initiala migrationen varierande resultat till att börja med. Både den samlade och de schemauppdelade överföringsprocesserna misslyckades vid upprepade försök med att överföra de två största scheman som fanns i källdatabasen. Manuella SQL-frågor ställda till käll-databasen visade att tabellerna gick utmärkt att läsa därifrån, och detta skedde efter att användar-rättigheterna utökades för att lösa överföringen till Redshift tidigare. Det visade sig dessutom att de scheman som överföringsprocessen lyckats överföra inte placerats så som specificerats utan lagts osorterade. När ett ytterligare försök gjordes med den samlade överföringsprocessen och samma specifikationer en vecka senare fungerade det helt problemfritt och tog 4 timmar 22 minuter.

Det visade sig dock att en migration till S3 endast resulterar i filer av typen .csv (Comma-Separated Values) som innehåller datan i tabellerna, ingen information om datatyper eller annan tabell-information så som kolumnnamn sparas. Givet detta ansågs en lösning som inkluderade S3 som alltför omfattande för detta arbete. För att det ska vara aktuellt behöver en annan lösning för att överföra databasschemat hittas, och därför genomförs testet med separata överföringsprocesser inte.

4.3. SQL-frågor

Testning med hjälp av SQL-frågorna som Bring SCM bistått med visade tydliga prestandaskillnader mellan SQL Server och Redshift. Resultaten redovisas nedan med referensvärdena från SQL Server, lagringsoptimerad Redshift samt beräkningsoptimerad Redshift i Tabell II, III respektive IV nedan.

TABELL II. RESULTAT AV SQL-FRÅGOR, SQL SERVER

Fråga	*Test 1*	*Test 2*	*Test 3*
1	2,67s	2,63s	2,66s
2a	7,04s	6,65s	7,68s
2b	5436s	5857s	5715s
3	26,87s	20,27s	19,5s
4	90s	15,74s	14,39s
5	219s	218s	218s
6	210s	210s	210s
7	1,59s	0,23s	0,28s

TABELL III. RESULTAT AV SQL-FRÅGOR, REDSHIFT (LAGRING)

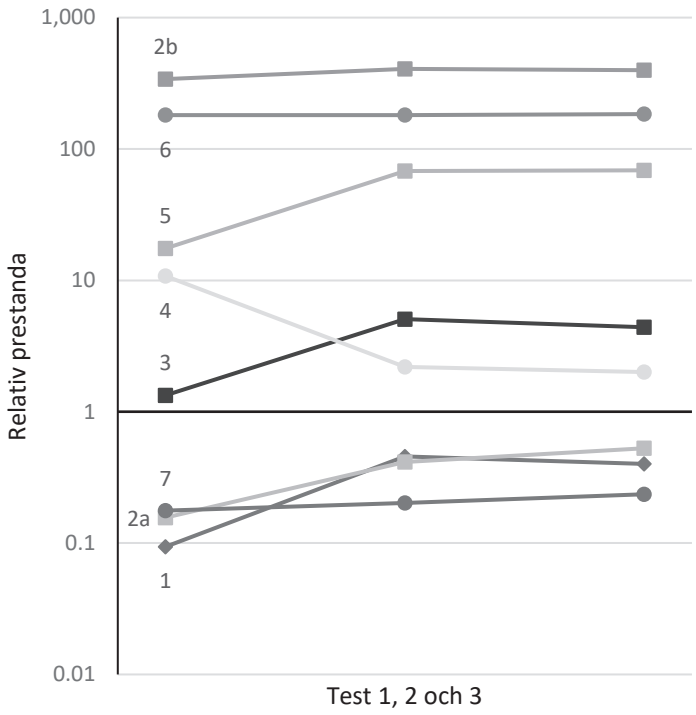
Fråga	*Test 1*	*Test 2*	*Test 3*
1	28,46s	5,76s	6,63s
2a	45,03s	16,07s	14,58s
2b	15,97s	14,39s	14,37s
3	20,15s	4s	4,43s
4	8,35s	7,18s	7,18s
5	12,51s	3,21s	3,17s
6	1,16s	1,16s	1,14s
7	9s	1,14s	1,19s

TABELL IV. RESULTAT AV SQL-FRÅGOR, REDSHIFT (BERÄKNING)

Fråga	*Test 1*	*Test 2*	*Test 3*
1	32,52s	5,75s	5,58s
2a	59,53s	28,22s	28,21s
2b	39,19s	31,88s	32,19s
3	26,94s	5,58s	5,62s
4	16,47s	4,43s	4,01s
5	15,96s	3,59s	3,55s
6	10,59s	1,1s	1s
7	9,8s	1,41s	1,32s

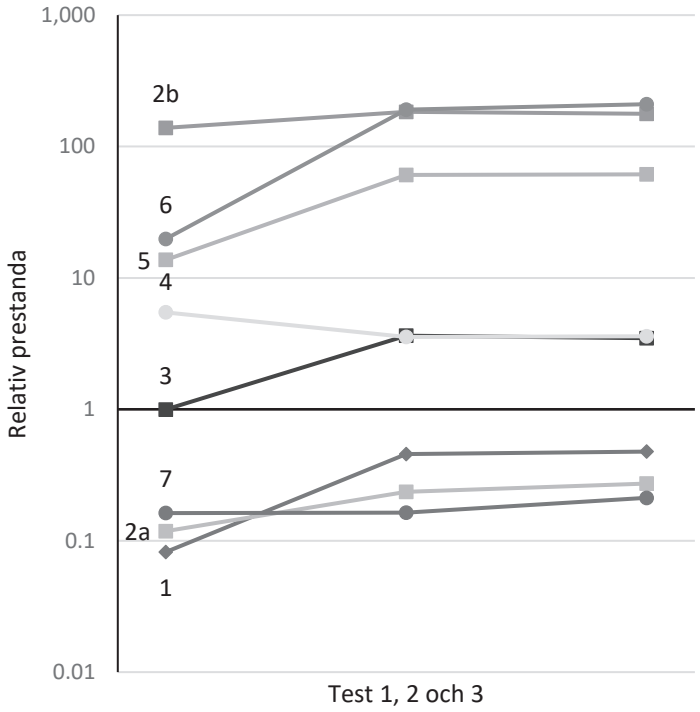
I fig. 4 nedan visualiseras den prestandaförändring som det lagringsoptimerade (ds) Redshift-klustret gav jämfört med SQL Server. Värdena gavs av att dividera exekveringstiden i SQL Server med exekveringstiden i Redshift.

Över lag uppvisade det lagringsoptimerade Redshift-klustret förbättringar, men en del frågor – 1, 2a samt 7 – gick fortfarande snabbare på SQL Server. För fråga 1 och 7 är dock försämringen endast mellan 0,13 och 4,06 sekunder som minst respektive mest, vilket inte är signifikant då externa faktorer som en fördröjning i nätverkskommunikation kan ge upphov till liknande försämringar, och det därför inte kan anses signifikant nog för att grunda ett beslut på. Fråga 2a tog mellan 12,15 och 14,5 sekunder längre, vilket är en tydlig försämring. Samtliga frågor visade antingen en direkt förbättring eller försämring konsekvent genom alla exekveringar, optimeringen hos ettdera systemet gjorde alltså inte avgörande skillnad för någon fråga.



Figur 4: Relativ prestanda i Redshift (ds) jämfört med SQL Server vid de tre testerna. Värden över 1 är till fördel för Redshift, skalan är logaritmisk.

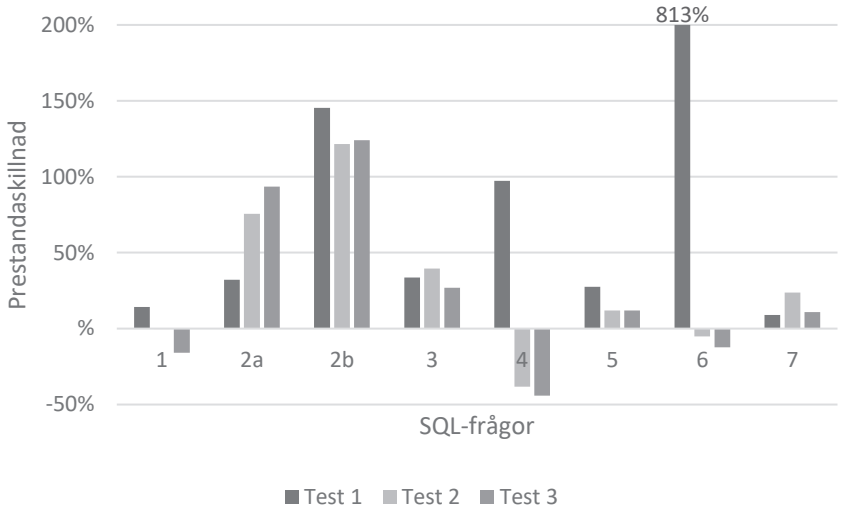
Det beräkningsoptimerade (dc) Redshift-klustret gav liknande resultat, redovisade i fig. 5 nedan. Det är ett snarlikt mönster i prestandaskillnaden för respektive fråga jämfört med det lagringsoptimerade klustret, men noterbart är att det lagrings-optimerade klustret över lag ger en större förbättring än det beräkningsoptimerade klustret.



Figur 5: Relativ prestanda i Redshift (dc) jämfört med SQL Server vid de tre testerna. Värden över 1 är till fördel för Redshift, skalan är logaritmisk.

En jämförelse av de två Redshift-klustren redovisas i fig. 6 nedan. Värdena ges av formeln $t_{dc}/t_{ds} - 1$ där t-värdena är exekveringstiden för en fråga på dc- respektive ds-kluster. På första exekveringen var det lagringsoptimerade klustret snabbare på samtliga frågor, men det beräknings-optimerade klustret presterade något bättre vid efterföljande exekveringar. Det var dock endast på fråga 1, 4 samt 6 som det faktiskt blev snabbare än det lagringsoptimerade klustret efter

optimering, och då med betydligt mindre marginaler än hur mycket bättre det lagringsoptimerade klustret presterade på övriga frågor.



Figur 6: Prestandaskillnad mellan dc- och ds-kluster i Redshift vid de tre testerna. Värden över 0 är till fördel för det lagringsoptimerade klustret. Stapeln för Test 1 på fråga 6 är avhuggen av läslighetsskäl.

4.4. Kostnader

Priserna är beräknade för en månads användning av respektive tjänst, avrundat till heltal. De är räknade i USD, vilket är den valuta Amazon använder sig av för alla sina beräkningar. DMS fanns inte med i beräkningsverktyget, utan det priset är baserat på timpriset för instansen [46] multiplicerat med 732 timmar (30,5 dagar), samma som används för övriga tjänster i beräkningsverktyget. Priset för SQL-servrarna redovisas nedan i tabell V, och priset för Redshift-kluster i tabell VI.

TABELL V. PRIS FÖR SQL SERVER

Databas	*EC2*	*EBS*	*Totalt*
Bring SCM	$1022	$175	$1198
Testserver	$683	$175	$859

TABELL VI. PRIS FÖR REDSHIFT-KLUSTER

Kluster	*Pris/nod*	*Noder*	*DMS*	*Totalt*
Lagring	$695	1	$114	$809
Beräkning	$220	10	$114	$2314

Noterbart här är att det lagringoptimerade Redshift-klustret visade sig vara billigare än alla andra alternativ, och det beräkningsoptimerade klustret är väldigt mycket dyrare än övriga alternativ. Det bör också nämnas att det lagringsoptimerade klustret endast använt cirka 10% av lagringsutrymmet, och det beräkningsoptimerade klustret använt över 95%. Skulle databasen behöva mer lagringsutrymme kommer det beräkningsoptimerade klustret därför snabbt att bli ännu dyrare.

5. Diskussion

Här kommer resultaten och vad de innebär i relation till de ursprungliga frågeställningarna diskuteras. Kapitlet är till skillnad från föregående två kapitel inte strukturerat efter moment, utan utgår istället i huvudsak ifrån de ursprungliga frågeställningarna som lades fram i kapitel 1.4. Utöver detta inkluderas en reflektion över vilka begränsningar som finns i slutsatserna i kapitel 5.7 samt en kort reflektion över arbetets etiska aspekter.

5.1. Skapandet av datalager

Att upprätta ett Redshift-kluster visade sig vara förvånansvärt enkelt, likaså att spara ner och återställa kopior av klustren. Även skalning av kluster är smidigt, om än något tidskrävande. I stort stämmer dessa egenskaper väl överens med tanken om att Redshift är någonting som är gjort att använda för analyser, i och med hur enkelt det är att skapa ett nytt kluster att använda för till exempel en stor undersökning.

Det viktigaste att ta med sig här är hur flexibelt det är med molnservrar jämfört med klassiska lösningar. Att skapa en ny server, ändra antalet servrar eller storleken på en av dem är ett markant större arbete när man behöver förhålla sig till den faktiska fysiska hårdvaran.

5.2. Initial migration

Det fanns en del problem med DMS, men de – tillsammans med de flesta problem med arbete – gick samtliga att spåra till rättigheter för användare och servrar i AWS som var svåra att hitta en korrekt konfiguration för. Det är ligger inte inom detta arbetes huvudsakliga område, men det är värt att nämna då det blir något som påverkar den som använder dessa system.

En annan sak att notera är hur olika stor plats en databas kan ta i olika system. Även om Redshift komprimerar datan så finns det duplicerad data på kluster med mer än en beräkningsnod och en uppskattning av storleksbehovet visade sig vara svår att göra. Det är svårt att uppskatta hur stark den är ifrån den data som finns, men det

tycks finnas en positiv korrelation mellan antalet noder och det lagringsutrymme som behövs för databasen.

Att använda S3 som mellanlagring visade sig vara möjligt, men inte utan problem. Det kan definitivt vara ett alternativ om behovet finns och om man bygger upp en egen datastruktur i Redshift, men bristen på tabell- och schema-information är annars problematisk. Viktigt att nämna är dock att i skrivande stund är användandet av DMS för att flytta data till S3 fortfarande i ett väldigt tidigt stadium, de första testerna skedde i dagarna efter att tjänsten släppts officiellt. Detta skulle kunna vara anledningen till att de två i övrigt identiska testerna vid skilda tillfällen gav olika resultat.

Sammanfattningsvis är det fullt möjligt och relativt enkelt att använda DMS för datamigrationen, men det är definitivt relevant att se över vilka övriga alternativ som finns, särskilt om man vill använda sig av en ETL-process till skillnad från en ELT-process. Dock verkar ELT passa Redshift bättre [47] på grund av den stora beräkningskapaciteten som erbjuds.

5.3. Löpande migration

Den löpande migrationen med DMS ser väldigt lovande ut. Den var enkel, hade bra konfigurationsmöjligheter och var snabb utan att belasta någondera av databaserna märkbart. Att den dessutom inte låser måldatabasen för läsning – eller ens för skrivning från andra källor – är något som explicit eftersökts. Nackdelen jämfört med ett Log Shipping-system är ju att det krävs ytterligare en server i systemet, men det får anses vara en väldigt liten börda jämfört med de fördelar som ges.

5.4. Flaskhalsar

Generellt sett så fanns det få tecken på att det skulle finnas några stora flaskhalsar i systemet. Den största begränsningen verkade vara EC2-instansen som SQL Server-databasen kördes på, som såg ut att arbeta på maximal kapacitet vid ett antal tillfällen under arbetet. Då inga kritiska problem uppstod på grund av den och den dessutom är rejält mycket mindre kraftfull än den server som skulle användas som

datakälla i skarpt läge får risken för att det skulle ställa till problem anses som låg.

Ett problem som inte uppstod, men som skulle kunna komma att uppstå under andra förutsättningar är att DMS-instansen såg ut att kunna vara en begränsning vid den initiala migratonen, så vilket behov av kapacitet som finns där är något som bör undersökas. DMS saknar också inbyggt stöd för schemaläggning, så skulle det finnas ett behov av att datan hämtades i omgångar kan detta bli omständligt att lösa där.

5.5. Prestanda vid SQL-frågor

Märkbart är att de frågor med omfattande nästlade SQL-frågor som SQL Server var långsammast på också är de frågor som gav störst prestandaökning. Det antyder att det är något som Redshift är bättre på än vad SQL Server är. Mindre väntat var att fråga 2b gick snabbare än 2a i det lagringsoptimerade klustret, trots att 2b endast är en mer komplex variant av fråga 2a. Förklaringen för detta får antas vara ordningen frågorna exekverades i, och att den optimering som skett för fråga 2a i Redshift efter första gången till stor grad också påverkat exekveringen av fråga 2b. Ytterligare ett noterbart fynd är att fråga 7 gav markant försämring trots att den strukturellt liknar de frågor som generellt gav bra förbättringar. Ingen förklaring kunde hittas till denna anomali.

För det beräkningsoptimerade klustret såg resultaten relativt lika ut. Det är anmärkningsvärt att prestandan över lag var sämre här än för det lagringsoptimerade klustret. Det i kombination med att det för en del av frågorna blev bättre än det lagringsoptimerade klustret indikerar att problemet kan ligga i hur datan strukturerats. Det stämmer också bra med de rekommendationer som konsekvent ges om att strukturera om att använda andra datastrukturer för datalager.

5.6. Kostnader

Som visat i resultatkapitlet så finns det stor variation för vad ett Redshift-kluster kan kosta. Det mest slående är hur hög kostnaden blev för beräkningsoptimerade kluster då varje nod fick plats med relativt lite data, och delar av datan till synes har lagrats på mer än ett ställe i

och med det ökade behovet av lagringsutrymme jämfört med SQL Server.

Lagringsoptimerade kluster ser ut som ett bättre alternativ, då de fortfarande kan ge en markant prestandaökning jämfört med SQL Server, men dessutom vara billigare med bra marginal för att skala upp – den minsta instansen använde endast cirka 10% av lagringsutrymmet för att rymma hela databasen.

DMS har räknats som en fast kostnad här, då endast en instanstyp använts. Det är fullt rimligt att anta att en annan storlek kan vara lämplig att använda sig av, och då kan den kostnaden skala från en fjärdedel så mycket upp till flera storleksordningar större – allt beror på hur mycket data som behöver laddas parallellt vid drift.

Den sammanfattade slutsatsen av det hela är att lagringsoptimerade kluster ser ut att vara en väldigt bra lösning, med både bättre prestanda och lägre pris än alla övriga alternativ. En annan lösning kan vara att använda sig av mer än ett kluster, där man har ett lagringsoptimerat kluster med en oförändrad datastruktur, och skapar beräkningsoptimerade kluster att använda till specifika behov så som kuber – mindre databaser strukturerade för en bestämd typ av fråga [9] – vid behov. Då får man en enkel migration och lägre löpande kostnad, men fortfarande möjlighet att utnyttja de mer specialiserade funktionerna i datalager.

5.7. Begränsningar i slutsatserna

Detta arbete har avhandlat en stor mängd system och aspekter av dessa. Ändå finns det givetvis en hel del saker som inte fått plats eller som testerna inte lyckats säga tillräckligt mycket om. Testerna bygger på data som är strukturerad på ett icke optimalt sätt för ett datalager, och även om det blir förbättringar trots detta så är det fortfarande okänt hur stora förbättringar det skulle bli om optimering skedde där. Den SQL Server-databas som använts är också placerad på en mindre kraftfull EC2-instans än den som används i produktion, så de jämförelser som gjorts där behöver varken nödvändigtvis eller troligtvis ge lika bra resultat till Redshifts fördel som testerna visar på här.

Testerna har vidare endast skett på en specifik mängd data, så det går att ifrågasätta hur allmänna de slutsatser som dras blir. Troligtvis

kan man anta att de kan anses likna andra M3-system, men utöver det är det svårt att säga så mycket.

5.8. Etiska aspekter

Att flytta en databas kanske i sig inte medför några alltför stora behov av etiska överväganden utöver att se till så person- och affärsdata för inblandade parter fortsätter vara skyddad. Men de ökade analysmöjligheterna som datalager innebär bör begrundas något djupare än så – kanske inte specifikt för den data som används i detta arbete, men definitivt i större kontext. Big Data i allmänhet möjliggör väldigt mycket nya möjligheter, och ett bra exempel på detta är data mining.

Data mining handlar om att hitta mönster i data som man kanske inte nödvändigtvis letade efter eller antog var där, ofta genom att jämföra all tillgänglig data och se vilka korrelationer som framträder. Exemplet i introduktionen med hur big data användes i det amerikanska valet är ett bra exempel på detta, där väljare profilerades utifrån ett antal faktorer och politiska budskap skräddarsyddes för målgrupper. Skälet att det förknippas med just big data är dels att ju mer data man har tillgänglig desto fler och starkare samband går det att identifiera, men också att det krävs väldigt hög beräkningskapacitet för att göra detta.

De som datan handlar om har oftast väldigt begränsad insikt i och kontroll över exakt hur datan används, vilken data som lagras och särskilt vilka samband som dras utifrån den. Därför ligger ansvaret fullt på den som behandlar datan att se till att det sköts på ett etiskt försvarbart sätt.

Det finns givetvis en del lagstiftning kring vad som får lagras, och det är primärt personuppgiftslagen [48] som är relevant där, samt dataskyddsreformen [49] i EU som börjar tillämpas 2018. De kan dock inte anses vara fulltäckande då teknisk utveckling rör sig markant mycket snabbare än vad lagändringar kan hänga med i – det är till exempel 20 år mellan dessa två lagar. Fokus ligger också primärt på vilken data som får lagras men inte på hur den får aggregeras med annan data. Utöver detta är det givetvis en stor skillnad på juridik och etik då lagen måste ge större tolkningsutrymme på grund av att den är

koppad till påföljder, och ett etiskt ramverk dessutom rimligtvis bör hålla en högre standard än att bara undvika att bryta mot lagen.

Data mining och annan storskalig dataanalys är givetvis ingenting som per definition är dåligt etiskt. Men det är dock ett väldigt kraftfullt verktyg som kan användas till många olika syften, och bör därför behandlas med eftertanke.

6. Vidare arbete

Detta arbete har varit både väldigt begränsat i valet av vilka tjänster som undersöks, samtidigt som det har varit spretigt i vilka aspekter som undersökts. Hade tiden funnits hade en del av dessa obesvarade frågor kunnat inkluderats, men istället får dörren lämnas öppen för ytterligare undersökningar i annat arbete på bred front.

Det med mest allmännyttigt intresse att undersöka borde rimligtvis vara en jämförelse med de övriga datalager- och ETL/ELT-alternativen på marknaden eller någon form av metastudie på den data som redan finns tillgänglig kring olika alternativ. Att datalager är användbara för analys är bortom tvivel, så frågorna som återstår är vilket system man väljer, och hur man designar det. Just för ETL/ELT-verktyg finns det väldigt många alternativ men väldigt begränsad överblick.

Designen – mer specifikt databasdesignen – är någonting som detta arbete endast snuddat vid, och som definitivt borde undersökas mer, särskilt när man tittar på kluster med många noder. Att identifiera vilka eventuella prestandavinster som kan fås av snöflingeschema, stjärnschema eller liknande anpassning så som att dela upp databasen i flera mindre är högst intressant, särskilt om man vill titta på mer storskaliga analyser och verktyg så som data mining och upprättandet av kuber som Bring SCM visat intresse för.

Fokus har legat på en mer teknisk och prestandafokuserad analys, primärt av upprättandet av ett kluster. Det lämnar rum för mer användar-fokuserade analyser och att titta på hur Redshift är för slutanvändaren jämfört med SQL Server, till exempel kring möjlighet att använda sig av BI-/analysverktyg.

Slutligen är det flera system som alla har en mängd möjliga inställningar vardera som undersökts. Omfattningen har inte tillåtit att alla möjliga och relevanta permutationer undersökts, så en del har nedprioriterats. Exempelvis hade relationen mellan antal kluster och beräkningshastighet, prestanda vid simultana SQL-frågor, krypterings påverkan på prestandan och den inverkan på DMS som replikationsserverns storlek ger alla varit högst intressanta att undersöka.

Referenser

[1] Was Donald Trump's Surprise Win a Failure of Big Data? Not Really. Hämtad 2017-04-28 från http://fortune.com/2016/11/14/donald-trump-big-data-polls/

[2] Trump Campaign Turns to 'Psychographic' Data Firm Used by Cruz. Hämtad 2017-04-28 från http://www.nationalreview.com/article/438739/trump-campaigns-data-firm-partner-cambridge-analytica-worked-cruz

[3] The Digital Universe of Opportunities: Rich Data and the Increasing Value of the Internet of Things. Hämtad 2017-04-28 från https://www.emc.com/leadership/digital-universe/2014iview/executive-summary.htm

[4] Nilson, Richard; Elastic Mobile, 2017. Möte 19 januari.

[5] Popularity of open source DBMS versus commercial DBMS. Hämtad 2017-04-26 från https://db-engines.com/en/ranking_osvsc

[6] Mueller, Johannes; Bring SCM, 2017. Möte 15 februari.

[7] Mueller, Johannes; Bring SCM, 2017. Möte 5 april.

[8] E. Malinowski och E. Zimányi, *"Advanced Data Warehouse Design, From Conventional to Spatial and Temporal Applications"*, Berlin Heidelberg, Springer-Verlag, ISBN 978-3-540-74404-7, 2009.

[9] V. Rainardi, *"Building a Data Warehouse, with examples on SQL Server"*, Springer-Verlag, New York, ISBN 978-1-59059-931-0, 2008.

[10] AWS Global Infrastructure. Hämtad 2017-04-28 från https://aws.amazon.com/about-aws/global-infrastructure/

[11] Amazon EC2 Pricing. Hämtad 2017-04-28 från https://aws.amazon.com/ec2/pricing/on-demand/

[12] Data Warehouse System Architecture. Hämtad 2017-04-28 från https://docs.aws.amazon.com/redshift/latest/dg/c_high_level_system_architecture.html

[13] V. Srinivasan, J. Cunningham. 2015. Building your Data Warehouse with Amazon Redshift. [online]. Hämtad 2017-04-28 från https://www.youtube.com/watch?v=GhHXVVFju9Q

[14] Choosing a Compression Type. Hämtad 2017-05-05 från https://docs.aws.amazon.com/redshift/latest/dg/t_Compressing_data_on_disk.html

[15] A. Haines, 2014. Migrating a Big Data Warehouse to the Cloud. Baseline, *Baseline* 5.

[16] Amazon EC2 Instance Types. Hämtad 2017-04-28 från https://aws.amazon.com/ec2/instance-types/

[17] Amazon Simple Storage Service (S3) FAQs. Hämtad 2017-04-28 från https://aws.amazon.com/s3/faqs/

[18] Using a Microsoft SQL Server Database as a Source for AWS Database Migration Service. Hämtad 2017-04-28 från https://docs.aws.amazon.com/dms/latest/userguide/CHAP_Source.SQLServer.html

[19] Using an Amazon Redshift Database as a Target for AWS Database Migration Service. Hämtad 2017-04-28 från https://docs.aws.amazon.com/dms/latest/userguide/CHAP_Target.Redshift.html

[20] Amazon Virtual Private Cloud (VPC). Hämtad 2017-04-28 från https://aws.amazon.com/vpc/

[21] SQL Workbench/J – Compatible DBMS. Hämtad 2017-04-28 från http://www.sql-workbench.net/databases.html

[22] Which data warehouse should you use?. Hämtad 2017-05-09 från http://www.metabase.com/blog/which-data-warehouse-to-use

[23] SQL Data Warehouse. Hämtad 2017-05-09 från https://azure.microsoft.com/en-us/services/sql-data-warehouse/

[24] Welcome to Apache™ Hadoop®!. Hämtad 2017-05-09 från https://hadoop.apache.org/

[25] Apache Spark™. Hämtad 2017-05-09 från https://spark.apache.org/

[26] Data Warehouse | Oracle. Hämtad 2017-05-09 från https://www.oracle.com/database/data-warehouse/index.html

[27] Etleap: ETL for Big Data. Hämtad 2017-05-09 från https://etleap.com/

[28] Flyway by Boxfuse. Hämtad 2017-05-09 från https://flywaydb.org/

[29] Matillion ETL for Amazon Redshift. Hämtad 2017-05-09 från https://www.matillion.com/etl-for-redshift/etl-redshift/

[30] AWS Glue (coming soon). Hämtad 2017-05-09 från https://aws.amazon.com/glue/

[31] Getting Started with AWS. Hämtad 2017-05-15 från https://docs.aws.amazon.com/gettingstarted/latest/awsgsg-intro/gsg-aws-intro.html

[32] Amazon Redshift Clusters. Hämtad 2017-05-15 från https://docs.aws.amazon.com/redshift/latest/mgmt/working-with-clusters.html

[33] Replication Instances for Database Migration Service. Hämtad 2017-05-15 från https://docs.aws.amazon.com/dms/latest/userguide/CHAP_Introduction.ReplicationInstance.html

[34] What is the AWS Schema Conversion Tool?. Hämtad 2017-05-15 från https://docs.aws.amazon.com/SchemaConversionTool/latest/userguide/Welcome.html

[35] Using Table Mapping with an Task to Select and Filter Data. Hämtad 2017-05-05 från https://docs.aws.amazon.com/dms/latest/userguide/CHAP_Tasks.CustomizingTasks.TableMapping.html

[36] Amazon CloudWatch. Hämtad 2017-05-15 från https://aws.amazon.com/cloudwatch/

[37] Mockaroo – Random Data Generator. Hämtad 2017-05-15 från mockaroo.com

[38] AWS Database Migration Service Adds Amazon Simple Storage Service (S3) as a Target. Hämtad 2017-05-15 från https://aws.amazon.com/about-aws/whats-new/2017/03/aws-database-migration-service-adds-amazon-simple-storage-service-s3-as-a-target/

[39] Amazon Redshift Best Practices for Loading Data. Hämtad 2017-05-15 från https://docs.aws.amazon.com/redshift/latest/dg/c_loading-data-best-practices.html

[40] Managing Concurrent Write Operations. Hämtad 2017-05-15 från https://docs.aws.amazon.com/redshift/latest/dg/c_Concurrent_writes.html

[41] Amazon Web Services Simple Monthly Calculator. Hämtad 2017-05-15 från https://calculator.s3.amazonaws.com/index.html

[42] About me | Data Warehousing and Business Intelligence. Hämtad 2017-05-09 från https://dwbi1.wordpress.com/about/

[43] Elzbieta Malinowski (University of Costa Rica , San José) on Research Gate. Hämtad 2017-05-09 från https://www.researchgate.net/profile/Elzbieta_Malinowski

[44] Esteban Zimanyi (Université Libre de Bruxelles , Brussels) On Research Gate. Hämtad 2017-05-09 från https://www.researchgate.net/profile/Esteban_Zimanyi

[45] Källkritik på internet. Hämtad 2017-05-19 från https://www.iis.se/lar-dig-mer/guider/kallkritik-pa-internet/

[46] AWS Database Migration Service Pricing. Hämtad 2017-05-15 från https://aws.amazon.com/dms/pricing/

[47] Better, faster, smarter: ETL vs. ELT. Hämtad 2017-05-15 från https://www.matillion.com/redshift/better-faster-smarter-elt-vs-etl/

[48] Personuppgiftslag (1998:204) hämtad 2017-05-19 från http://www.riksdagen.se/sv/dokument-lagar/dokument/svensk-forfattningssamling/personuppgiftslag-1998204_sfs-1998-204

[49] Home Page of EU GDPR. Hämtad 2017-05-19 från http://ec.europa.eu/justice/data-protection/reform/files/regulation_oj_en.pdf

Förkortningar

AWS	Amazon Web Services
S3	Simple Storage Service
EC2	Elastic Compute Cloud
EBS	Elastic BeanStalk
ETL	Extract, Transform, Load
ELT	Extract, Load, Transform
DMS	Database Migration Service
dc	Dense Computing
ds	Dense Storage

Appendix A: SQL-frågor

Fråga 1

```
SELECT
MMITGR as "NAT/GLO"
,T1.MBSTAT AS "Status"
,MWWHNM AS "Lager"
,T8.CTTX40 AS "CLASIFICATION"
,T1.MBWHLO AS "Lagerställe"
,T1.MBITNO AS "Art.nr"
,T2.MMITDS AS "Art.namn"
,T2.MMBUAR AS "Business are"
,MMSTCN AS "Lagerkrav"
,MBBUYE AS "Planner"
,CAST ((T1.MBSTQT) AS INT) AS "LHU på lager"
,CAST ((T1.MBSTQT-T1.MBREQT) AS INT) AS "Disponibelt Saldo"
,CAST ((T1.MBSTQT-T1.MBREQT)/MMCFI2 AS DEC(38,1)) as "Pall netto"
,CAST ((T1.MBSTQT-T1.MBREQT) AS INT) AS "kvt netto"
,T3.M9UCOS AS "Pris lokal valuta"
,T5.CULOCD AS "Lokal valuta"
,T5.CUARAT AS "Senaste valutakurs"
,T3.M9UCOS/T5.CUARAT AS "Pris SEK"
,SUM (MHSOQT) AS "Såld kvt"
,SUM (MHFOQT) AS "Prognos kvt"
,SUM (MHMOVN) AS "Neg.flytt"
,COUNT (MHCYP6)  AS "Antal veckor"
,T7.Quantity
,T1.MBSSQT AS "Säkerhetslager qty"
,T1.MBSSQT/MMCFI2 as "SL pall"
,MMCFI2 AS "LHU per pall"
FROM
MVXJDTA.MITBAL T1
INNER JOIN
MVXJDTA.MITWHL on T1.MBWHLO=MWWHLO AND T1.MBCONO=MWCONO
INNER JOIN
MVXJDTA.MITMAS T2 ON T1.MBCONO=T2.MMCONO AND T1.MBITNO=T2.MMITNO
INNER JOIN
```

```
MVXJDTA.MITFAC T3 ON T1.MBCONO=T3.M9CONO
AND T1.MBITNO=T3.M9ITNO AND MWFACI=T3.M9FACI
INNER JOIN
MVXJDTA.MITSTA T4 ON T1.MBCONO=T4.MHCONO AND
T1.MBWHLO=T4.MHWHLO AND T1.MBITNO=T4.MHITNO
INNER JOIN
MVXJDTA.CMNDIV ON T1.MBCONO=CCCONO AND T3.M9FACI=CCFACI
LEFT OUTER JOIN
MVXJDTA.CSYTAB T8 ON T2.MMCONO=T8.CTCONO AND
T2.MMMES4=T8.CTSTKY AND T8.CTSTCO='MES0'
INNER JOIN
MVXJDTA.CCURRA T5 ON T1.MBCONO=T5.CUCONO AND
CCDIVI=T5.CUDIVI AND CUCRTP=1
LEFT OUTER JOIN
(
SELECT MLCONO, MLWHLO, MLITNO,
SUM (MLSTQT) AS Quantity
FROM MVXJDTA.MITLOC
WHERE MLWHSL LIKE ('010=>%')
GROUP BY MLCONO, MLWHLO, MLITNO
) AS T7 ON
T1.MBCONO=T7.MLCONO AND T1.MBWHLO=T7.MLWHLO
AND T1.MBITNO=T7.MLITNO
JOIN
(
SELECT CUCONO, CUDIVI, CUCUCD,
CULOCD, MAX (CUCUTD) AS CUTD
FROM MVXJDTA.CCURRA
WHERE CUCRTP=1
GROUP BY CUCONO, CUDIVI, CUCUCD, CULOCD
) AS T6 ON
T5.CUCONO=T6.CUCONO AND T5.CUDIVI=T6.CUDIVI AND
T5.CUCUCD=T6.CUCUCD AND T5.CULOCD=T6.CULOCD AND
T5.CUCUTD=T6.CUTD AND T5.CUCUCD='SEK'
WHERE
T1.MBCONO=2 AND T2.MMITTY='LAG'
AND MMHIE1='IKE' AND MMCFI2>0
AND CCDIVI<>0 AND MWWHSY<>'F'
```

```
AND T1.MBSTAT IN ('20','50')
AND (RIGHT (MHCYP6,2)<>'00')
AND T1.MBSTQT>0 AND
T4.MHCYP6 BETWEEN 201301 AND 201612
GROUP BY
MMITGR ,T1.MBSTAT ,MWWHNM
,T1.MBWHLO ,MWWHSY ,T1.MBITNO
,T2.MMITDS ,MBBUYE ,T5.CULOCD
,T5.CUARAT ,T3.M9UCOS/T5.CUARAT
,CAST ((T1.MBSTQT) AS INT)
,CAST ((T1.MBSTQT-T1.MBREQT) AS INT)
,CAST ((T1.MBSTQT-T1.MBREQT)/MMCFI2 AS DEC(38,1))
,CAST ((T1.MBSTQT-T1.MBREQT) AS INT)
,T1.MBSTQT ,T3.M9UCOS ,T7.Quantity
,T1.MBSSQT ,MMCFI2 ,T8.CTTX40
,T1.MBSSQT ,T2.MMBUAR ,MMSTCN
```

Fråga 2a

```
SELECT
MZWHLO AS "Lagerställe"
,MZCUNO AS "Bring kundnr"
,OKALCU AS "Picadeli kundnr"
,OKCUNM AS "Butiksnamn"
,MZRIDN AS "Ordernr"
,OACUOR AS "Kunds ordernr"
,MZRIDI AS "Leveransnr"
,MZRIDL AS "Orderrad"
,OBCODT AS "Bekräftat leveransdatum"
,OBORQA AS "Beställd kvt"
,OBDLQA+OBIVQA AS "Levererad kvt"
,OBSPUN AS "Försäljningsenhet"
,IDSUNM AS "Leverantör"
,MZZDNO AS "Claimnr"
,MZZDLN AS "Claimrad"
,MZZREP AS "Registrerad av"
,MZRGDT AS "Registreringsdatum"
,MZZDSL AS "Claimstatus"
,MZTX60 AS "Fritext"
,MZITNO AS "Bring artikelnr"
,MZPOPN AS "Picadeli artikelnr"
,MMITDS AS "Artikelnamn"
,MMITTY AS "Artikeltyp"
,MMSTCN AS "Temperatur"
,MZTRQA AS "Kvantitet"
,CASE
WHEN OBSPUN='KG' THEN MZTRQA
ELSE (MZTRQA*MMGRWE)
END AS "Bruttovikt (kg)"
,MZALUN AS "Enhet"
,ROUND((M9APPR*MZTRQA),2) AS "Genomsnittspris per rad"
,MZZRC0 AS "Rapporterad orsakskod"
,T1.CTTX60 AS "Rapporterad orsak"
,MZZRC1 AS "Bekräftad orsakskod"
,T2.CTTX60 AS "Bekräftad orsak"
```

```
,MZZCF1 AS "Kostnadsbärare"
,MZZCF2 AS "Ytterförpackning OK"
,MZZCF3 AS "BBD"
,MZZCF4 AS "Batch"
,MZZCF5 AS "Dokumentation saknas"
,NZSEQN AS "Åtgärdsrad"
,NZZDAC AS "Åtgärdskod"
,NZZDAT AS "Åtgärdstyp"
,CATX60 AS "Åtgärd"
,NZITNO AS "Bring artikelnr"
,ORPOPN AS "Picadeli artikelnr"
,NZRIDN AS "Batchordernr"
,NZRIDL AS "Batchorderrad"
,EVORNR AS "Kreditordernr"
FROM AMGJDTA.CZDVLN --deviation lines / claimrader
JOIN AMGJDTA.CZDVLA ON MZCONO=NZCONO AND
MZZDNO=NZZDNO AND MZZDLN=NZZDLN --action lines / åtgärdsrader
JOIN AMGJDTA.OZCLCO T1 ON MZCONO=T1.CTCONO
AND MZZRC0=T1.CTZCRC -- claimorsaker rapporterat
JOIN AMGJDTA.OZCLCO T2 ON MZCONO=T2.CTCONO
AND MZZRC0=T2.CTZCRC -- claimorsaker bekräftat
JOIN AMGJDTA.OZACCO ON MZCONO=CACONO AND MZWHLO=CAWHLO AND
NZZDAC=CAZACD
JOIN MVXJDTA.OCUSMA ON MZCONO=OKCONO AND MZCUNO=OKCUNO
LEFT JOIN MVXJDTA.OOHEAD ON MZCONO=OACONO AND MZRIDN=OAORNO
LEFT JOIN MVXJDTA.OOLINE ON MZCONO=OBCONO AND MZWHLO=OBWHLO
AND OAORNO=OBORNO AND MZRIDL=OBPONR AND MZRIDX=OBPOSX
JOIN MVXJDTA.MITBAL ON MZCONO=MBCONO AND MZWHLO=MBWHLO AND
MZITNO=MBITNO
JOIN MVXJDTA.CIDMAS ON MZCONO=IDCONO AND MBSUNO=IDSUNO
JOIN MVXJDTA.MITMAS ON MZCONO=MMCONO AND MZITNO=MMITNO
JOIN MVXJDTA.MITFAC ON MZCONO=M9CONO AND MZITNO=M9ITNO AND
MZFACI=M9FACI
LEFT JOIN MVXJDTA.OCUSIT ON MZCONO=ORCONO AND NZITNO=ORITNO AND
ORCUNO='KREDIT'
LEFT JOIN MVXJDTA.OXCNTR ON NZCONO=EVCONO AND NZFACI=EVFACI AND
NZRIDN=EVORNO
WHERE
MZWHLO IN ('040','046') AND MZCONO=4 AND
MZRGDT BETWEEN 20160101 AND 20161231
```

Fråga 2b

```
SELECT
MZWHLO AS "Lagerställe"
,MZCUNO AS "Bring kundnr"
,OKALCU AS "Picadeli kundnr"
,OKCUNM AS "Butiksnamn"
,T3.H6DLIX AS "Leveransnr"
,MZRIDN AS "Ordernr"
,OACUOR AS "Kunds ordernr"
,MZRIDI AS "Leveransnr"
,MZRIDL AS "Orderrad"
,OBCODT AS "Bekräftat leveransdatum"
,OBORQA AS "Beställd kvt"
,OBDLQA+OBIVQA AS "Levererad kvt"
,OBSPUN AS "Försäljningsenhet"
,IDSUNM AS "Leverantör"
,MZZDNO AS "Claimnr"
,MZZDLN AS "Claimrad"
,MZZREP AS "Registrerad av"
,MZRGDT AS "Registreringsdatum"
,MZZDSL AS "Claimstatus"
,MZTX60 AS "Fritext"
,MZITNO AS "Bring artikelnr"
,MZPOPN AS "Picadeli artikelnr"
,MMITDS AS "Artikelnamn"
,MMITTY AS "Artikeltyp"
,MMSTCN AS "Temperatur"
,MZTRQA AS "Kvantitet"
,CASE
WHEN OBSPUN='KG' THEN MZTRQA
ELSE (MZTRQA*MMGRWE)
END AS "Bruttovikt (kg)"
,MZALUN AS "Enhet"
,ROUND((M9APPR*MZTRQA),2) AS "Genomsnittspris per rad"
,MZZRC0 AS "Rapporterad orsakskod"
,T1.CTTX60 AS "Rapporterad orsak"
,MZZRC1 AS "Bekräftad orsakskod"
```

```
,T2.CTTX60 AS "Bekräftad orsak"
,MZZCF1 AS "Kostnadsbärare"
,MZZCF2 AS "Ytterförpackning OK"
,MZZCF3 AS "BBD"
,MZZCF4 AS "Batch"
,MZZCF5 AS "Dokumentation saknas"
,NZSEQN AS "Åtgärdsrad"
,NZZDAC AS "Åtgärdskod"
,NZZDAT AS "Åtgärdstyp"
,CATX60 AS "Åtgärd"
,NZITNO AS "Bring artikelnr"
,ORPOPN AS "Picadeli artikelnr"
,NZRIDN AS "Batchordernr"
,NZRIDL AS "Batchorderrad"
,EVORNR AS "Kreditordernr"
FROM AMGJDTA.CZDVLN --deviation lines / claimrader
JOIN AMGJDTA.CZDVLA ON MZCONO=NZCONO AND MZZDNO=NZZDNO AND
MZZDLN=NZZDLN --action lines / åtgärdsrader
JOIN AMGJDTA.OZCLCO T1 ON MZCONO=T1.CTCONO AND MZZRC0=T1.CTZCRC --
claimorsaker rapporterat
JOIN AMGJDTA.OZCLCO T2 ON MZCONO=T2.CTCONO AND MZZRC0=T2.CTZCRC --
claimorsaker bekräftat
JOIN AMGJDTA.OZACCO ON MZCONO=CACONO AND MZWHLO=CAWHLO AND
NZZDAC=CAZACD
JOIN MVXJDTA.OCUSMA ON MZCONO=OKCONO AND MZCUNO=OKCUNO
LEFT JOIN MVXJDTA.OOHEAD ON MZCONO=OACONO AND MZRIDN=OAORNO
LEFT JOIN MVXJDTA.OOLINE ON MZCONO=OBCONO AND MZWHLO=OBWHLO AND
OAORNO=OBORNO AND MZRIDL=OBPONR AND MZRIDX=OBPOSX
JOIN MVXJDTA.MITBAL ON MZCONO=MBCONO AND MZWHLO=MBWHLO AND
MZITNO=MBITNO
JOIN MVXJDTA.CIDMAS ON MZCONO=IDCONO AND MBSUNO=IDSUNO
JOIN MVXJDTA.MITMAS ON MZCONO=MMCONO AND MZITNO=MMITNO
JOIN MVXJDTA.MITFAC ON MZCONO=M9CONO AND MZITNO=M9ITNO AND
MZFACI=M9FACI
LEFT JOIN MVXJDTA.OCUSIT ON MZCONO=ORCONO AND NZITNO=ORITNO AND
ORCUNO='KREDIT'
LEFT JOIN MVXJDTA.OXCNTR ON NZCONO=EVCONO AND NZFACI=EVFACI AND
NZRIDN=EVORNO
LEFT JOIN
(
SELECT H6CONO, H6WHLO, H6RIDN, MIN(H6DLIX) AS "H6DLIX"
```

```
FROM MVXJDTA.MHPICD
WHERE H6CONO=4
GROUP BY H6CONO, H6WHLO, H6RIDN
) AS T3
ON MZCONO=T3.H6CONO AND MZWHLO=T3.H6WHLO AND MZRIDN=T3.H6RIDN
WHERE
MZWHLO IN ('040','046') AND MZCONO=4 AND
MZRGDT BETWEEN 20160101 AND 20161231
```

Fråga 3

```
SELECT
T5.MWWHNM AS "Warehouse"
,T1.MBITNO AS "Art.nbr"
,T3.MMITDS AS "Art.namn"
,S1.MONTH AS "Month"
,T7.JUTX40 AS "Planner"
,SUM(S1.FOREC) AS "Forecast"
,SUM(S1.SOL) AS "Sold"
,T6.IDSUNM AS "Supplier"
,T3.MMCFI5 AS "Service level"
,T1.MBPRCD AS "Prognosmetod"
FROM MVXJDTA.MITBAL T1
INNER JOIN
(
SELECT
S1.MHCONO, S1.MHWHLO, S1.MHITNO, T2.CDCYP1 AS MONTH
,((SUM (S1.MHFOQT)/5)*T2.CNTCYP2) AS FOREC
,((SUM (S1.MHSOQT)/5)*T2.CNTCYP2) AS SOL
FROM MVXJDTA.MITSTA S1
INNER JOIN
(
SELECT T1.CDCONO,T1.CDDIVI
,T1.CDCYP1,T1.CDCYP2,T2.CNTCYP1
,COUNT (T1.CDYMD8) AS CNTCYP2
FROM MVXJDTA.CSYCAL T1
INNER JOIN
(
SELECT CDCONO,CDDIVI,CDCYP1
,COUNT (CDYMD8) AS "CNTCYP1"
FROM MVXJDTA.CSYCAL
WHERE CDDIVI='' AND CDWDPC='100'
GROUP BY CDCONO,CDDIVI,CDCYP1
) AS T2 ON
T1.CDCONO=T2.CDCONO AND T1.CDDIVI=T2.CDDIVI AND
T1.CDCYP1=T2.CDCYP1
WHERE T1.CDDIVI='' AND T1.CDWDPC='100'
```

```
AND T1.CDYMD8 BETWEEN 20160101 AND 20161231
GROUP BY T1.CDCONO,T1.CDDIVI
,T1.CDCYP1,T1.CDCYP2,T2.CNTCYP1
) AS T2 ON
S1.MHCONO=T2.CDCONO AND S1.MHCYP6=T2.CDCYP2
GROUP BY S1.MHCONO,S1.MHWHLO
,S1.MHITNO,T2.CDCYP1,T2.CNTCYP2
) AS S1 ON S1.MHCONO=T1.MBCONO AND
S1.MHITNO=T1.MBITNO AND S1.MHWHLO=T1.MBWHLO
INNER JOIN MVXJDTA.MITMAS T3 ON
T1.MBCONO=T3.MMCONO AND T1.MBITNO=T3.MMITNO
INNER JOIN MVXJDTA.MITWHL T5 ON
T1.MBCONO=T5.MWCONO AND T1.MBWHLO=T5.MWWHLO
LEFT OUTER JOIN MVXJDTA.CIDMAS T6 ON
T1.MBCONO=T6.IDCONO AND T3.MMSUNO=T6.IDSUNO
LEFT OUTER JOIN MVXJDTA.CMNUSR T7 ON T1.MBBUYE=T7.JUUSID
WHERE T1.MBCONO=2 AND T3.MMITTY='LAG' AND
T1.MBSTAT='20' AND T5.MWWHTY='IK' AND
T5.MWWHSY='M'
GROUP BY
T5.MWWHNM,T1.MBITNO,S1.MONTH,T3.MMITDS
,T7.JUTX40,T6.IDSUNM,T3.MMCFI5,T1.MBPRCD
```

Fråga 4

```
SELECT
MTCONO,MTWHLO AS "Lager"
,MTRSCD AS "Felkod"
,IDSUNO,IDSUNM,MTTRTP
,MTITNO,MTTTYP,MTTRTP
,LEFT(EZRGDT,6) AS "PERIOD"
,MTITNO AS "Artikel"
,EZAIT1,EZSENO
,SUM (MTTRQT) AS "Antal"
,SUM (EZACAM) AS "BELOPP"
FROM
MVXJDTA.CINACC INNER JOIN MVXJDTA.MITTRA ON
EZCONO=MTCONO AND EZITNO=MTITNO AND
EZANBR=MTANBR AND MTRFTX<>'MARTIN&SER 40010029'
INNER JOIN MVXJDTA.MITBAL ON
EZCONO=MBCONO AND EZITNO=MBITNO AND MTWHLO=MBWHLO
INNER JOIN MVXJDTA.MITVEN ON
EZCONO=IFCONO AND EZITNO=IFITNO AND MBSUNO=IFSUNO
INNER JOIN MVXJDTA.CIDMAS ON
IFCONO=IDCONO AND IFSUNO=IDSUNO
WHERE EZCONO=4 AND EZDIVI='040'
AND MTTTYP IN (40,41)
and MTRSCD NOT
IN('115','120','121','122','131','135','200','201','203','204','205','
206','207','996','997','998','999','RET')
AND IFSUNO<>'40010029'
AND MTWHLO='040'
GROUP BY
MTCONO,MTWHLO,IDSUNO
,IDSUNM,MTTRTP,EZRGDT
,MTTTYP,MTITNO,MTRSCD
,EZAIT1,EZSENO,MTTRTP
```

Fråga 5

```
select
T3.MWWHNM as "DCname",
cast(SUM(T2.MMVOL3*T1.MTTRQT)/1000 as decimal(9,1)) as "M3",
cast(TS.year as INT) as "Year",
cast(TS.month as INT) as "Month"
from MVXJDTA.MITTRA T1
INNER JOIN MVXJDTA.MITMAS T2 ON
T1.MTCONO=T2.MMCONO AND T1.MTITNO=T2.MMITNO
INNER JOIN MVXJDTA.MITWHL T3 ON
T1.MTCONO=T3.MWCONO AND T1.MTWHLO=T3.MWWHLO
LEFT JOIN MVXJDTA.OOHEAD T4 on
T1.MTWHLO=T4.OAWHLO AND T1.MTCONO=T4.OACONO AND
T1.MTRIDN=T4.OAORNO
INNER JOIN
(
SELECT CDCONO,CDDIVI,CDYMD8,
LEFT(CAST(CDYMD8 AS VARCHAR),4) AS year,
SUBSTRING(CAST(CDYMD8 AS VARCHAR),5,2) AS month,
SUBSTRING(CAST(CDCYP2 AS VARCHAR),5,2) AS week,
SUBSTRING(CAST(CDYWD5 AS VARCHAR),5,1) AS weekday,
(LEFT(CAST(CDYMD8 AS VARCHAR),4))+(SUBSTRING(CAST(CDCYP2 AS
VARCHAR),5,2)) AS yearweek
FROM MVXJDTA.CSYCAL
WHERE CDCONO=2 AND CDDIVI='060'
) AS TS ON MMCONO=TS.CDCONO AND MTTRDT=TS.CDYMD8
where
T1.MTCONO=2 AND
T1.MTTRDT >20160101 AND                --Datum
T1.MTTTYP='31' AND
T1.MTWHLO
IN('001','010','100','110','120','150','155','160','165','169','210','
250','300','510','540','610','710','800')
group by
T3.MWWHNM,cast(TS.year as INT),cast(TS.month as INT)
```

Fråga 6

```
select
TS.yearmonth, MWWHNM as Lager,
SUM(MTTRQT) as Antal,
cast(SUM(MMVOL3*MTTRQT)/1000 as decimal(9,1)) as "M3"
from MVXJDTA.MITTRA
LEFT OUTER JOIN MVXJDTA.MITMAS ON
MTCONO=MMCONO AND MTITNO=MMITNO
LEFT OUTER JOIN MVXJDTA.MITWHL on
MTWHLO=MWWHLO and MTCONO=MWCONO
INNER JOIN
(
SELECT CDCONO,CDDIVI,CDYMD8,
LEFT(CAST(CDYMD8 AS VARCHAR),4) AS year,
SUBSTRING(CAST(CDYMD8 AS VARCHAR),5,2) AS month,
SUBSTRING(CAST(CDCYP2 AS VARCHAR),5,2) AS week,
SUBSTRING(CAST(CDYWD5 AS VARCHAR),5,1) AS weekday,
(LEFT(CAST(CDYMD8 AS VARCHAR),4))+(SUBSTRING(CAST(CDCYP2 AS
VARCHAR),5,2)) AS yearweek,
(LEFT(CAST(CDYMD8 AS VARCHAR),4))+(SUBSTRING(CAST(CDYMD8 AS
VARCHAR),5,2)) AS yearmonth
FROM MVXJDTA.CSYCAL
WHERE CDCONO=2 and CDDIVI='060'
) AS TS ON
MMCONO=TS.CDCONO AND MTTRDT=TS.CDYMD8
where MWWHTY='IK' AND MTCONO=2 AND
MTTTYP=51 AND TS.yearmonth BETWEEN '201601' and '201612'
group by
MWWHNM,
MWWHLO,
TS.yearmonth,
MTTTYP
```

Fråga 7

```
select
cast(SUM(T2.MMVOL3*T1.MTTRQT)/1000 as decimal(9,1)) as M3,
TS.yearmonth as Month, T2.MMBUAR as BA,
MMCFI4 as "Consumer unit", SUM(T1.MTTRQT) as "LHU",
T2.MMITDS as ArtName
from MVXJDTA.MITTRA T1
INNER JOIN MVXJDTA.MITMAS T2 ON
T1.MTCONO=T2.MMCONO AND T1.MTITNO=T2.MMITNO
INNER JOIN MVXJDTA.MITWHL T3 ON
T1.MTCONO=T3.MWCONO AND T1.MTWHLO=T3.MWWHLO
INNER JOIN
(
SELECT CDCONO,CDDIVI,CDYMD8,
LEFT(CAST(CDYMD8 AS VARCHAR),4) AS year,
SUBSTRING(CAST(CDYMD8 AS VARCHAR),4,4) AS month,
SUBSTRING(CAST(CDCYP2 AS VARCHAR),5,2) AS week,
SUBSTRING(CAST(CDYWD5 AS VARCHAR),5,1) AS weekday,
LEFT(CAST(CDYMD8 AS VARCHAR),4)+SUBSTRING(CAST(CDCYP2 AS VARCHAR),5,2)
AS yearweek,
LEFT(CAST(CDYMD8 AS VARCHAR),6) AS yearmonth
FROM MVXJDTA.CSYCAL
WHERE CDCONO=2 AND CDDIVI='060'
) AS TS ON
MMCONO=TS.CDCONO AND MTTRDT=TS.CDYMD8
WHERE
T1.MTCONO=2 AND
T1.MTTRDT > '20160101' AND                --Datum
T1.MTTTYP='31' AND T3.MWWHNM='TOKYO'
GROUP BY
T3.MWWHNM,TS.yearmonth,MMBUAR
,MMCFI4,T2.MMITDSTDS
```

Printed by Tryckeriet i E-huset, Lund 2017

LUND
UNIVERSITY

Series of Bachelor's theses
Department of Electrical and Information Technology
LU/LTH-EIT 2017-586
http://www.eit.lth.se