Institutionen för datavetenskap
Department of Computer and Information Science

Examensarbete

Implementering av PostgreSQL som databashanterare för MONITOR

av

Jesper Axelsson

LIU-IDA/LITH-EX-G--14/083--SE

2014-10-23

Linköpings universitet
SE-581 83 Linköping, Sweden

Linköpings universitet
581 83 Linköping

Linköpings universitet
Institutionen för datavetenskap

Examensarbete

Implementering av PostgreSQL som databashanterare för MONITOR

av

Jesper Axelsson

LIU-IDA/LITH-EX-G--14/083--SE

2014-10-23

Handledare: Patrick Lambrix
Examinator: Patrick Lambrix

Sammanfattning

Monitors affärssystem MONITOR är under ständig utveckling och i och med detta ville man kolla upp huruvida PostgreSQL skulle kunna användas som DBMS istället för det nuvarande; Sybase SQL Anywhere. Examensarbete har därför bestått av en jämförelse hur PostgreSQL står sig jämte andra DBMS:er, en implementering utav en PostgreSQLdatabas som MONITOR arbetar mot samt ett prestandatest utav skapandet av databasen.

I många avseenden verkar PostgreSQL vara ett alternativ till SQL Anywhere;

1. Alla datatyper finns i båda dialekterna.
2. Backup av data finns i olika utföranden och går att automatisera.
3. Enkelt att installera och uppdatera.
4. Ingen licensieringskostnad existerar.
5. Support finns tillgänglig i olika former.

Dock så är inte PostgreSQL ett bra DBMS att byta till i dagsläget då systemet inte fungerade på grund av att vissa uttryck inte översattes ordentligt samt att ingen motsvarighet till LIST existerar. Ännu större är dock problemet med tiden det tar att flytta data till en PostgreSQL-databas då det inte är intressant att lösa problem med funktioner i systemet om det ändå inte går att använda på grund utav att konvertering av data tar så lång tid som det gör.

Innehåll

Kapitel 1

Inledning

Detta kapitel innehåller en överblick av vad rapporten innehåller och dess syfte. Här hittar man även en sammanställning av hur examensarbetet genomfördes.

1.1 Motivering

1.1.1 Monitor - Företaget

Monitor ERP System (hädanefter Monitor) är ett Hudiksvallsbaserat företag som arbetar inom tillverkningsindustrin, den egna verksamheten består dock inte utav tillverkning av fysiska produkter utan man tillhandahåller affärssystemet MONITOR. Ute i drift har man konsulter, support och även utbildningar i systemet för att förenkla användandet för kunderna, detta ger snabb feedback på hur användarna vill att programmet ska se ut och fungera. Av denna anledning har de även valt att inrikta sig på att sälja produkten till små och mellanstora företag så att inte alla resurser läggs på att få ett system skräddarsytt till ett stort företag utan kan användas utav många med små eventuella modifikationer, vilket har visat sig vara ett vinnande koncept. Mer än 2300 företag använder sig utav MONITOR, de flesta av dessa är företag som verkar i Sverige men däri finns även svenska företag som migrerat sin produktion utomlands samt nya kunder som värvats i andra länder.

I dagsläget har de runt 120 anställda varav merparten sitter i Hudiksvall där utvecklingen av programvaran sker, både uppgradering av det befintliga systemet och skapandet av nya versioner. Det är i samband med det senare som detta examensarbete uppkommit.[1]

1.1.2 MONITOR - Affärssystemet

Då ett affärssystem innehåller stora mängder information är det viktigt att denna sparas på ett säkert men enkelanvänt sätt. Så i och med att en ny version av MONITOR är under utveckling ville Monitor kolla över hur andra databashanterare (DBMS[1]:er) står sig i jämförelse med den de använder idag; Sybase SQL Anywhere. Anledningen till varför det blev aktuellt att kolla över detta ligger delvis i att Sybase har köpts upp utav SAP AG vilket är ett multinationellt företag som också sysslar med att tillverka affärssystem. Uppköpet har framförallt gjorts för att komma åt de mobila varianter på databashantering som Sybase utvecklat vilket gör framtiden för SQL Anywhere svårförutsägbar. En annan anledning är att eftersom vissa av de kunder man har är väldigt små företag skulle det vara bra att ha ett alternativ där man slapp licenskostnader för databashanteraren och på så vis kunna leverera en ännu mer prisvärd produkt. Av dessa anledningar ville Monitor undersöka vilka alternativa DBMS:er som finns, hur dessa presterar samt även få en inblick i hur databasoberoende koden som skrivits i dagsläget är.

1.2 Syfte

Rapporten kommer att undersöka förutsättningar för att ersätta det nuvarande databashanteringssystemet som Monitor använder med PostgreSQL. Vad som kommer utvärderas är enkelhet, prestanda och vilka eventuella ändringar som behöver göras för att en sådan övergång ska kunna göras.

1.3 Frågeställning

- Vad krävs för att installera PostgreSQL?
- Är det enkelt att hålla PostgreSQL uppdaterat?
- Vad finns det för backup alternativ?
- Hur står sig PostgreSQL jämtemot SQL Anywhere;
  - Syntaxmässigt?
  - Prestandamässigt?
- Vilka ändringar/tillägg behöver göras i MONITORs kod för att få det fungerande tillsammans med PostgreSQL?

[1] Database Management System

1.4 Metod

Examensarbetet började med en studie utav PostgreSQL där fokus las på att jämföra PostgreSQL med andra databashanteringssystem. Därefter påbörjades programmering genom att få PostgreSQL att fungera mot både Entity Framework samt NHibernate. Efter det att grundläggande funktionalitet testats i de två ORM[2]:en (Entity Framework och NHibernate) initierades ett försök att få MONITOR med Enitity Framework fungerande, detta fungerade dock inte önskvärt. Sedan testades NHibernateversionen utav MONITOR vilket klarade av att starta programmet. I detta skede uppstod dock flera problem då nästintill ingen funktionalitet som programmerats in i systemet fungerade, varför så var fallet utforskades delvis. Sedan genomfördes även en prestandatestning då det var viktigt att mäta tiden för skapandet av databasen.

1.5 Rapportinnehåll

Kapitel ett innehåller en motivering till varför detta examensarbete har uppstått varpå det andra kapitlet ger en bakgrund till vad PostgreSQL är och vad som behövs för att kunna använda detta tillsammans med Monitors kod.

Tredje kapitlet beskriver de krav som satts upp på PostgreSQL medan det fjärde kapitlet går igenom delar av dessa krav då jämförelser med andra DBMS:er görs.

Femte kapitlet innehåller den programmering som gjorts under examensarbetet och består av de ändringar och tillägg som gjorts i framförallt Monitors kod. Häri ryms även en genomgång av det tidsexperiment som gjorts och resultaten presenteras.

I kapitel sex hittar man sedan en återkoppling till frågeställningen för examensarbetet samt de rekommendationer som ges för framtida utforskning utav användandet av PostgreSQL tillsammans med MONITOR.

1.6 Avgränsningar

Det finns givetvis andra typer av prestanda än just insättning av data som Monitor hade velat undersöka med PostgreSQL, t.ex. hämtning av data, CPU-användning m.fl. Några sådana tester gick dock inte att inkludera i arbetet inom den tidsram som fanns.

[2]Begreppet beskrivs närmare i avsnitt 2.2 och 2.3 på sidorna 11 respektive 12

1.7 Förtydligande

Ordet konvertering som används i rapporten kan ses tvetydig men syftar på ändringar som görs i affärssystemet MONITOR mellan olika versioner. Detta leder till att de jämförelser som görs i rapporten handlar om konverteringar från den tidigare versionen (då bara SQL Anywhere används) till en senare version (då SQL Anywhere används men PostgreSQL och SQL Server testas).

Kapitel 2

Det teoretiska

Detta kapitel introducerar läsaren till begrepp och uttryck som används flitigt i rapporten. Här ryms även en genomgång av olika licenstyper samt information om PostgreSQL.

2.1 Databaser

Databaser används för att kunna lagra (stora mängder) data på ett lättåtkomligt sätt. För att hantera datan i en databas använder man sig utav ett DBMS, med det kan man då till exempel lägg in ny data, hämta data man är intresserad av m.m.

Det finns olika sätt för hur man väljer att representera data:n i databasen men det vanligaste är att man använder sig av en relationsdatabas. I en sådan skapar man relationer som beskriver det man vill lagra och sedan hur dessa relationer är relaterade till varandra. Enklaste sättet för att sedan presentera datan sker med hjälp av SQL vilket returnerar en tabell med den information man frågat efter, relationen **Person** skulle då kunna se ut som i tabell 2.1 om den skrivs ut. Skulle man däremot vilja ge personer

förnamn	efternamn	intresse
Jesper	Axelsson	data
Anders	Haraldsson	LISP

Tabell 2.1: Exempel på hur relationen **Person** skulle kunna se ut

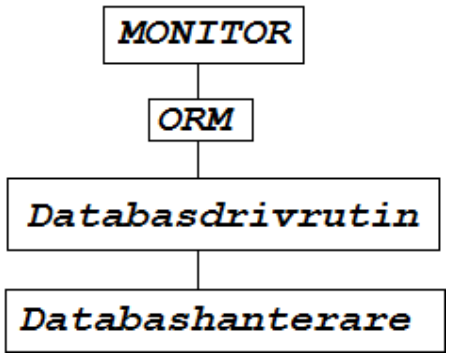
möjligheten att ha flera intressen bör det snarare representeras som i tabell 2.1 där den andra personen (Haraldsson) nu också är intresserad av data.[9]

2.2 Databashantering - Vad behövs?

För att slippa hantera en databas direkt i koden man skriver brukar man använda sig utav ett ORM[1]. Det man gör är att man beskriver i klasserna hur de lagras i databasen (alt. hur man vill att de ska lagras) varpå ORM:et sedan skapar översättning mellan datan (i databasen) och objekten (i det aktiva programmet). ORM:et kommunicerar med databashanteraren med hjälp av en databasdrivrutin, detta görs för att man ska behöva skriva så lite kod som möjligt som är beroende på hur den underliggande strukturen ser ut.

Det man gärna uppnår genom att använda sig utav ett ORM är att oavsett vilket DBMS man använder så ska inte koden man skrivit behöva ändras, koden har då blivit databasoberoende. En annan väldigt positiv sak är att bygger man program med de strukturer som man får tillgång till via ORM:et så hjälper kompilatorn till vid eventuella felskrivningar och man kan även få hjälp med autokomplettering.

En överblick av de delarna man behöver för att hantera en databas ser man i figur 2.1. I denna figur vill man göra så få ändringar som möjligt ju



Figur 2.1: Struktur för databashantering

längre upp man kommer, det vill säga om man byter databashanterare ska man förhoppningsvis slippa göra allt för stora ändringar i programvarans

[1] Object-Relational Mapping

id	förnamn	efternamn
1	Jesper	Axelsson
2	Anders	Haraldsson

person	intresse
1	1
2	1
2	2

id	intresse
1	data
2	LISP

Tabell 2.2: Reltionerna; **Person**, **Intresserad av** och **Intresse**

kod. I fallet att koden skulle vara databasoberoende behöver man i sådana fall bara skriva i koden att man använder en annan DBMS och eventuellt lägga till en annan databasdrivrutin till projektet.

En avgörande faktor för val av ORM är vilket språk som man utvecklar programvaran i och i fallet med MONITOR så har man använt sig av C♯ och .NET. I tidigare versioner av MONITOR har man använt sig utav NHibernate men inför utvecklingen av den nya generationen av systemet har man valt att även kolla på hur Entity Framework fungerar.

2.3 ORM

Ett ORM används i två huvudsyften:

1. Skapa tabeller dynamiskt utifrån klasser
2. Läsande, uppdaterande av data etc. ska se mer ut som ”vanlig” kod

Den första punkten är den där själva kopplingen mellan klasserna i koden och relationerna i databasen görs. Just denna delen är inte helt trivial och kan skapa sådana problem att man kanske väljer att använda sig av egen hårdkodad SQL för att skapa alla tabeller. För små databaser med få tabeller går det nog ändå att hålla reda på och uppdatera de ändringar som behöver göras under utvecklingens gång men när antalet tabeller ökar (i Monitors fall var det över 350 tabeller) kommer något ORM vara en självklar och även behövd lösning.

Den andra delen handlar om själva användandet utav datan i databasen, hur man kommer åt den, uppdaterar den m.m. Oavsett hur man kommer åt datan behöver man koppla den till de objekt som finns i programmet så använder man sig inte utav ett existerande ORM så kommer man mer eller mindre att ha skapat ett i slutändan.

2.3.1 NHibernate

NHibernate är utvecklat av JBoss som är ett företag som utvecklar ”programvara till program för kommunikation med operativsystemet” (engelska: middleware). JBoss har dock blivit en del av Red Hat som utvecklar Hibernate men är mest kända för att vara det största företaget som bidrar till utvecklingen av Linux. Första versionen släpptes 2005 och var en portning utav framförallt Hibernate 2.1, redan från början var dock NHibernate öppen källkod, släppt under LGPL[2], vilket gjorde att man snabbt överlät mycket utav utvecklandet till allmänheten.[2]

[2]Se 2.5 på sidan 13 för mer information angående licenser

2.3.2 Entity Framework

Entity Framework är ett ORM som är utvecklat av Microsoft med .NET i åtanke. Den första versionen släpptes 2008 och den senaste stora uppgraderingen, version 6.0, kom 2013. I och med att version 6.0 släpptes ändrade man även licensieringen så projektet är numera öppen källkod och är släppt under Apache License v2[2].[3]

För just PostgreSQL, ska det visa sig, är det inget bra ORM i dagsläget.[3]

2.4 Databasdrivrutiner

För att ORM:et ska kunna kommunicera med DBMS:et krävs en drivrutin som översätter de abstraherade ORM-uttrycken till körbar kod för den specifika databasen. Oavsett ORM så använder man sig av drivrutinen Npgsql om man ska kommunicera med PostgreSQL ifrån en .NET-miljö.

2.5 Licenser

För att kunna göra jämförelsen mellan olika databashanteringssystem[4] behövs kunskap om några olika mjukvarulicenser, så nedan gås de olika licenserna som jag stött på igenom kortfattat.

Apache-license

Vid vidareutveckling är all oförändrad kod fortfarande under Apache-licens, i filer där ändringar gjorts ska det finnas ett meddelande om att koden inte är originalkod.

Public domain

Får användas som man vill men om man ska bidra till projektet måste man avsäga sig ägandeskap på det man bidragit med.

GPL

Står gör GNU General Public License. Man får inte ändra licensiering om något är släppt under GPL, kod som använder sig av GPL-licensierade bibliotek eller liknande måste också släppas under GPL. Källkod ska kunna kommas åt av användare.

LGPL

Står för GNU Lesser General Public License. Får användas som externt bibliotek utan att programmet som skapas behöver släppas med LGPL-licens. Om programmet som är licensierat modifieras måste det släppas under LGPL.

[3] Se avsnitt 4.3.1 på sidan 22
[4] Avsnitt 4.2 på sidan 21

EPL

Står för Eclipse Public License. Man kan modifiera kod och släppa ny programvara där de egna bidragen inte behöver släppas under EPL om det är patenterade eller liknande, utan de delarna kan då ligga under andra licenser. Annars är den som LGPL i det att man kan länka in det i program utan att programmet släpps under EPL.

IPL

Står för IBM Public License och är till för att försöka underlätta kommersiellt bruk av open-source mjukvara, detta genom att lägga större ansvar hos de som utvecklar den licensierade produkten.

BSD

Står för Berkeley Software Distribution. Man får göra vad man vill med det förutom att hävda att allt är egengjort och dra in skaparna i eventuella problem som uppkommer.

PostgreSQL-license

En specificerad version av BSD för PostgreSQL.

2.6 PostgreSQL

Den första releasen av PostgreSQL hette postgres och släpptes 1995. Namnbytet till PostgreSQL skedde 1996 då man började stödja SQL istället för det tidigare QUEL. Idag utvecklas PostgreSQL utav PostgreSQL Global Development Group som består av både företag och drivna privatpersoner. Det som skiljer PostgreSQL från mängden är deras inriktning på stöd för olika typer av indexering och säkerheten mot databasen. Exempelvis går det att skapa användare som bara kan komma åt vissa kolumner i några tabeller.

En stor del av vad som blivit PostgreSQL framgång är inte bara hur databashanteraren i sig utvecklats utan även alla de tillägg som finns skapade till PostgreSQL. Då licensen gör det möjligt att göra kommersiella produkter med PostgreSQL finns det många stora företag och webbsidor som ändrat källkoden så att databashanteraren fungerar perfekt efter deras behov men samtidigt finns möjligheten att lägga till och köra nya tillägg som utvecklas. Exempel på dessa är Instagram och Yahoo!.

2.6.1 Backup

PostgreSQL stöder tre olika backup-tekniker[4]:

- Dumpa databas som SQL-kommandon
- Kopiera filer

- Kontinuerlig arkivering

SQL-kommandon

pg_dump är ett PostgreSQL kommando för att dumpa en eller flera databas(-er) som SQL-kommandon. Detta är det enda backup-sättet som fungerar om man ska uppgradera PostgreSQL till nästa major version eller om man ska byta från 32- till 64-bitsstruktur. pg_dump går att använda sig av medan servern är igång och slutresultatet blir en ögonblicksbild av hur databasen såg ut när pg_dump kördes. En negativ aspekt med pg_dump är att när man ska återställa databasen(-erna) kommer det att ta tid om det är många rader som ska in, dessutom sparas inte roller och tablespaces utan vill man ha den informationen får man använda sig av extra flaggor eller pg_dumpall.

Kopiering

Enklaste sättet är ju att bara kopiera filerna som lagrar all information, den stora nackdelen är att servern behöver antingen skapa en frusen ögonblicksbild eller stängas ner. Skapar man ögonblicksbilden behöver man även kopiera WAL(Write Ahead Log)-filerna, dessutom blir det svårare att skapa ju mer utspridd databasen är över diskar (partitioning) eftersom man behöver frysa allt samtidigt. Anledningen till att man behöver WAL-filerna om man inte stänger ner servern är för att eventuell buffring inte kommer med när man kopierar filerna.

Kontinuerlig arkivering

Den kontinuerliga arkiveringen är helt enkelt en automatisering utav den andra punkten där man arkiverar flertalet WAL-filer och lagrar dessa tillsammans med backup:er som tas med förinställt jämna mellanrum. Det finns även kommandon för att återställa ifrån dessa typer av backuper så att detta inte behöver göras helt manuellt.

2.6.2 Support

Då PostgreSQL är ett open source projekt är de främsta kontaktmedlen via mailinglistor och irc-kanaler. Tack vare sin popularitet finns det dock företag som man kan sluta avtal med för att få mer direkt hjälp.[5]

2.6.3 Underhåll

En major release släpps varje år i september och innehåller nya features. För att uppgradera till en ny major release behöver databasen dump:as och sedan reload:as när man har gjort uppgraderingen eftersom den interna strukturen kommer att ändras. Alternativt kan man använda pg_upgrade för att göra en sådan uppgradering. Minor releases fixar mindre buggar och för

[5] Se *http://www.postgresql.org/support* för mer information

att göra en sådan uppgradering behöver man inte dump/reload:a databasen utan bara stanna servern och lägga in de nya binärerna.

Avläser man tabellen i Figur 2.2 så verkar det som att antalet minor releases stagnerar och gissningsvis behövs sådana mindre uppgraderingar göras varannan till varje månad.

[6]Bilden tagen från http://www.postgresql.org/support/versioning

Version	Current minor	Supported	First release date
9.3	9.3.2	Yes	September 2013
9.2	9.2.6	Yes	September 2012
9.1	9.1.11	Yes	September 2011
9.0	9.0.15	Yes	September 2010
8.4	8.4.19	Yes	July 2009
8.3	8.3.23	No	February 2008
8.2	8.2.23	No	December 2006
8.1	8.1.23	No	November 2005
8.0	8.0.26	No	January 2005
7.4	7.4.30	No	November 2003
7.3	7.3.21	No	November 2002
7.2	7.2.8	No	February 2002
7.1	7.1.3	No	April 2001
7.0	7.0.3	No	May 2000
6.5	6.5.3	No	June 1999
6.4	6.4.2	No	October 1998
6.3	6.3.2	No	March 1998

Figur 2.2: Versioner av PostgreSQL (2014-01-14)[6]

Kapitel 3

Kravspecifikation

En viktig del för att MONITOR ska fungera är att det finns stöd för UTF-8 kodning av tecken, detta då man redan idag har kunder vars kundbas finns i till exempel Kina. Dessutom i och med att Monitor börjar slå sig in på utländska marknader skapar det ännu större krav på att DBMS:et klarar av att lagra kundnamn.

Tiden det tar att konvertera en databas, till exempel ifrån det gamla systemet till det nya, får inte ta för lång tid. Lång tid är dock högst subjektivt men om konverteringen till PostgreSQL jämförelsevis tar tre gånger så lång tid, eller mer, som motsvarande konvertering till SQL Anywhere bör man nog kunna anse att det går för långsamt.

Att få den funktionalitet som programmerats in i MONITOR fungerande även när man kör det mot PostgreSQL och ifall det inte går identifiera de problem som behöver åtgärdas.

Kapitel 4

Analys - Jämförelser

Arbetet på företaget började med att jag gjorde en jämförelse mellan det systemet de använde i dagsläget (SQL Anywhere), det de höll på att testa (MS SQL Server) samt givetvis PostgreSQL. Främst gjordes detta för att ta reda på om det fattades några datatyper eller annat som de använde sig av i MONITOR som skulle göra en övergång svårhanterlig eller omöjlig. Det gjordes även en jämförelse mellan PostgreSQL och andra open-source DBMS:er för att se hur PostgreSQL står sig jämte dessa.

Fullständiga tabeller för dessa jämförelser återfinns i bilaga A.

4.1 Datatyper

4.1.1 Numeriska Datatyper

Som man kan se i Tabell 4.1 så innehåller PostgreSQL inte lika många av de vanliga numeriska typerna men har fyllt ut med en ny datatyp; SERIAL. PostgreSQL har även implementerat BOOLEAN, medan med de andra två får man använda sig av BIT eller liknande om man vill spara värden av typen sant\falskt.

NUMERIC är en datatyp som är bra att använda när man behöver spara värden med många decimaler då float/double och deras namnar kan drabbas av avrundningsfel. Priset man får betala är dock att operationer med dessa datatyper blir långsammare, värt att notera med just det är att i SQL Anywhere är SMALLMONEY/MONEY implementerat som NUMERIC(10,4)/NUMERIC(19,4) och inte en egen datatyp. I tabell 4.2 ser man lite skillnad på storleksomfånget på NUMERIC, och dess namne DECIMAL, i de olika dialekterna. Just för PostgreSQL existerar två varianter där den övre är en begränsning för när man specificerar värden på p och s, den undre när man inte gör det.

	SQL Anywhere	MS SQL Server	PostgreSQL
1 byte	TINYINT	TINYINT	-
2 bytes	SMALLINT	SMALLINT	SMALLINT
4 bytes	INTEGER	INT	INTEGER
8 bytes	BIGINT	BIGINT	BIGINT
4 bytes	REAL	REAL	REAL
8 bytes	DOUBLE	FLOAT([25..53])	DOUBLE PRECISION
4 or 8 bytes	FLOAT	FLOAT	-
2 bytes	-	-	SMALLSERIAL
4 bytes	-	-	SERIAL
8 bytes	-	-	BIGSERIAL
4 bytes	SMALLMONEY	SMALLMONEY	-
8 bytes	MONEY	MONEY	MONEY
	-	-	BOOLEAN

Tabell 4.1: Numeriska datatyper

	p (precision)	s (scale)
SQL Anywhere	1..127	$0..p$
MS SQL Server	1..38	$0..p$
PostgreSQL	1..1000	$0..p$
	1..147455	$0..z, z \leq p$ $z \leq 16383$

Tabell 4.2: NUMERIC(p, s)/DECIMAL(p, s)

4.1.2 Teckendatatyper

Den största skillnaden ligger nog dock i hur tecken och strängar är implementerat i de olika dialekterna, i PostgreSQL räknar man till exempel längder på strängar i antal tecken istället för i antal bytes.[1]

Något som finns i SQL Anywhere och MS SQL Server men inte i PostgreSQL är datatyper som utmärker att kolumnen ska vara UTF-8 kodad, närmare bestämt NCHAR med flera. PostgreSQL har istället valt att hålla sådant på databasnivå genom att använda *-E* flaggan när man skapar databasen.[2]

Vill man använda sig utav en speciell teckenkodning när man sorterar så använder man sig av *COLLATE "kod"* i PostgreSQL och MS SQL Server medan i SQL Anywhere använder man *SORTKEY(kod)*.

4.1.3 Datum och Tid

En ganska viktig datayp är datum och tid då man gärna vill hålla koll på när man t.ex. fått en beställning, när den levererats m.m. Enligt SQL2011-standarden bör man döpa datatypen för datum och tid till TIMESTAMP, MS SQL Server har av traditionella skäl valt att behålla DATETIME. SQL Anywhere har valt att acceptera båda sätten att skriva på.[3]

Ett tillägg som PostgreSQL även har gjort är INTERVAL så att man kan lägga till t.ex. en månad till ett givet datum. Gör man någon operation mellan TIMESTAMPs får man ut resultatet i ett INTERVAL som man sedan kan förvandla (med t.ex. YEAR TO MONTH) om det skulle vara intressant.

4.1.4 SQL funktioner

För att skriva funktioner i de olika dialekterna så ser inslagningen lite olika ut, framförallt eftersom i PostgreSQL så är man inte tvungen att använda sig utav ett givet språk utan man kan byta mellan SQL, PL/pgSQL, PL/Python m.fl.[4]

4.1.5 Övriga iakttagelser

Både SQL Anywhere och MS SQL Server använder sig utav IDENTITY som en handle för att skapa datatyper som automatiskt räknas upp, i PostgreSQL däremot så använder man sig utav datatypen SERIAL och behöver således inte använda sig av nyckelord (t.ex. AUTO_INCREMENT) för att göra datatypen självuppräknande.

[1] Se appendix A.9 på sidan 42
[2] *http://www.postgresql.org/docs/9.3/static/multibyte.html*
[3] Se appendix A.11 på sidan 44
[4] *http://www.postgresql.org/docs/9.3/static/external-pl.html*

4.2 Andra Open Source DBMS:er

Mitt sökande efter databashanterare ledde mig fram till en Wikipediasida där en större jämförelse av DBMS:er gjorts så jag valde att använda den som startpunkt.[5] Till att börja med gick jag igenom de ickekommersiella DBMS:erna och kollade hur pass ”öppna” deras licenser var för att se redan där vilka som gick att rensa bort.[6] Detta uteslöt de DBMS:er som i tabell

DBMS	Mjukvarulicens
Apache Derby	Apache licence
SQLite	Public domain
H2	EPL, modded MPL
Firebird	IPL, IDPL
PostgreSQL	PostgreSQL licence
HSQLDB	BSD
MonetDB	MonetDB Public licence v1.1
SmallSQL	LGPL
CUBRID	GPL v2
Drizzle	BSD, GPL v2
Ingres	GPL or Proprietary
LucidDB	GPL v2
MariaDB	GPL v2
MySQL	GPL or Proprietary
OpenLink Virtuoso	GPL or Proprietary

Tabell 4.3: Databashanterare och deras licenser

4.3 ligger under SmallSQL och utav de åtta kvarvarande var det tre stycken som inte hade eller hade bristfälligt stöd för .Net; HSQLDB, MonetDB och SmallSQL så dessa uteslöt jag utan vidare funderingar. Fortsatt granskning av tabellerna som finns på sidorna 36-40 ger en jämförelse där PostgreSQL inte riktigt når upp till de andra open source DBMS:erna med avseende på begränsningar som sätts på storlekar i databaser[7], dock finns en mängd andra fördelar; uppdelning av databasen på hårdvara[8], kontroll över hur databasen får kommas åt[9] och indexering[10]. Samma avvägningar verkar vara gjorda även när man jämför med de kommersiella alternativen SQL Anywhere och MS SQL Server.

[5] Se referens [8]
[6] Se avsnitt 2.5 på sidan 13 för en sammanfattning av licenstyper
[7] Se appendix A.4 på sidan 37
[8] Se appendix A.2 på sidan 36
[9] Se appendix A.6 på sidan 39
[10] Se appendix A.7 på sidan 40

4.3 Design

4.3.1 Val av ORM

En viktig del i utvecklingsarbetet utav ny programvara är att saker sker så smidigt som möjligt, i det här fallet att man dynamiskt kan skapa databasen med tabeller när ändringar gjorts i klasser. Då Monitor höll på att utforska möjligheten att använda sig av Entity Framework (EF) testades först huruvida det fungerade tillsammans med PostgreSQL. Tyvärr så funkade det inte att skapa tabeller dynamiskt med EF, detta eftersom när man ska skapa tabellerna måste man ta bort databasen och skapa den från grunden. PostgreSQL tillåter inte att man gör något sådant när man kopplat upp sig mot en databas utan det måste göras som superuser i ett separat skede.

I NHibernate har man dock möjlighet att skapa databasen i ett skede för att sedan skapa tabellerna för sig, man behöver således inte ta bort databasen helt mellan återskapandet utan det går att bara ta bort tabellerna och sedan skapa dessa separat.

Kapitel 5

Implementation

Detta kapitlet beskriver den programmering som utförts under examensarbetet och börjar med att presentera hur man installerar PostgreSQL. Därefter beskrivs de ändringar och tillägg som har behövts göras i Monitors och Npgsqls kod för att lyckas konvertera en databas ifrån det gamla systemet till det nya som var under utveckling. Efteråt beskrivs de försök som gjordes för att få funktionaliteten i MONITOR fungerande och kapitlet avslutas med det tidsexperiment som utfördes för att kunna analysera hastighetsdifferenserna mellan PostgreSQL och SQL Anywhere.

5.1 Versioner

Versioner av de olika mjukvarorna och drivrutinerna som användes:

- NHibernate 3.3.3
- Entity Framework 5.0.0
- PostgreSQL 9.3.2
- Npgsql 2.0.14.3

5.2 Installation av PostgreSQL

Att installera PostgreSQL görs enklast genom att ladda ner binärer[1], extrahera och lägga dessa där man vill ha dem varpå man sedan kör `initdb` för att skapa databasen. Ett exempel på hur detta skulle se ut syns i koden 5.1.

Listing 5.1: install_PostgreSQL.bat

```
@echo off
```

[1] *http://www.postgresql.org/download/*

```
MKDIR C:\pgsql
SET source=C:\Users\monjeax\Downloads\pgsql
SET postgresql_root=C:\pgsql

echo copying files
XCOPY /E /Q "%source%" "%postgresql_root%"
MKDIR "%postgresql_root%\data"
MKDIR "%postgresql_root%\log"

echo initialising database
cd "%postgresql_root%\bin"
initdb -U postgres -A password -E utf8 -W -D "%postgresql_root%\data"
echo database initialized and ready to use
```

5.3 Skapa databas

5.3.1 Tillägg till Monitorkoden

Som jag nämnt tidigare så hade man försökt att använda sig utav MS SQL Server som DBMS istället för SQL Anywhere så i källkoden fanns med andra ord en delvis lyckad konvertering till MS SQL Server implementerad. Som grund för de ändringar som jag behövde göra hade jag då dels de extra funktionerna som de skapat för att få till MS SQL Server som DBMS och dels de som används för att skapa SQL Anywhere varianten. Den första delen som jag angrep i den existerande koden var klassen `ConvertTestDatabases`[2] och skapade i den testmetoden `ConvertDemo3ToPostgreSQL()`. Det metoden gör är att skapa en ny databas för att sedan importera data ifrån en testdatabas som är utformad enligt den nuvarande generationen utav MONITOR och konvertera denna data så att den passar den nya utformningen utav systemet.

Själva skapandet av databasen och tabeller sker i klassen `PostgreSQLDatabaseCreator`[3] och är uppdelad i två steg. Först så skapar man databasen och de användare som ska finnas för att sedan ansluta till databasen som en av de användarna och skapar alla tabellerna.

Nästa steg blev att ta reda på var fler skillnader fanns mellan MS SQL Server implementationen och SQL Anywhere motsvarigheten. Första stoppet blev funktionen `NHibernateSessionBuilder` i klassen `SessionFactory`[4] där jag la till ett `else if` för att hantera PostgreSQL som databashanteringsdialekt, här ser man även en naturlig fortsättning: Att lägga till PostgreSQL som en dialekt i MONITORkoden. För detta la jag till PostgreSQL i en enum i `MonitorCompanyConfiguration`[5] och även som ett val att starta applikationen med i xml-filen med samma namn[6]. Informationen från den xml-filen

[2] Se appendix B.1 på sidan 46
[3] Se appendix B.2 på sidan 48
[4] Se appendix B.3 på sidan 52
[5] Se appendix B.4 på sidan 53
[6] Se appendix B.5 på sidan 53

används utav klassen `ConfigurationFileCompanySelector`[7] för att välja vilken databas som ska användas i applikationen. Här finns även koden för att ändra lite grann i NHibernates implementation utav PostgreSQLdialekten där den viktigaste delen är att DateTimeOffset kopplas till TimestampTZ.

Det sista tillägget som behövde göras i MONITORs kod för att få skapandet av databas och tabeller fungerande var att kommentera ut en halv rad i `PaymentPlanRowMapping`[8] eftersom PostgreSQL inte gillar specialtecken i kolumnnamn.

5.3.2 Tillägg till Npgsql

Som jag nämnde var det viktigaste tillägget för MONITORs PostgreSQLdialekt att DateTimeOffset kopplades till TimestampTZ, detta eftersom det inte var gjort i NHibernates variant. Anledningen till att det var så var för att Npgsqldrivern inte returnerade ett DateTimeOffset-objekt utan ett DateTime-objekt, med andra ord ett utan tidszon specificerat. Dock används DateTimeOffset flitigt i MONITORs kod så lösningen på problemet blev att lägga till kopplingen i den egna dialekten, ändra i Npgsqls källkod[9] och skapa en drivrutin utifrån den ändringen.

Ett annat tillägg jag gjorde var att lägga till `"APP"` som en förkortning för `"APPLICATIONNAME"` i koden som skapar anslutningssträngar.[10] Alternativt hade man kunnat ändra detta i MONITORkoden så att det stod just "applicationame" i anslutningssträngen istället, båda två fungerar utmärkt.

5.3.3 Ändringar

Som man kan se i MS SQL Server konverteringen av databasen[11] så finns det ett globalt sätt att stänga av "constraint check":s, nämligen:

```
command.CommandText = @"EXEC sp_msforeachtable ""ALTER
    TABLE ? CHECK CONSTRAINT ALL"";";
```

Detta inkluderar då framförallt "foreign key constraint check":s som annars avbryter importeringen av data ganska omgående i MONITORs fall, motsvarande funktionalitet finns i SQL Anywhere men tyvärr inte i PostgreSQL. Lyckligtvis så går det att skapa denna funktionalitet genom att loop:a igenom alla tabeller och slå av/på triggers för dessa vilket är vad funktionen `performConstraintChanger`[12] gör.

När dessa ändringar gjorts gick det att påbörja konvertering av data ifrån testdatabasen till den som skapats och det var här som ett allvarligt tidsproblem uppstod. SQL Anywhere konverterade datan på ungefär 3 minuter medan nästa felmeddelande vid PostgreSQLkonverteringen inte kom förrän

[7] Se appendix B.6 på sidan 53
[8] Se appendix B.7 på sidan 54
[9] Se appendix B.10 på sidan 56
[10] Se appendix B.9 på sidan 56
[11] Se rad 30 i ConvertDemo3ToMSSql() appendix B.1 på sidan 46
[12] Se rad 84 och framåt i appendix B.2 i `PostgreSQLDatabaseCreator` på sidan 48

vid runt 30 minuter.[13] Det hela försvårades ytterligare då jag via debuggning kunde få konverteringen att köras för när jag hittat problemområdet och lyckades stanna i debuggning innan fel uppstod kunde jag sedan stega mig fram och på det viset magiskt ta mig förbi problemet. När jag efter många försök gett upp med att försöka identifiera min Heisenbugg körde jag konverteringen med full loggning varav ett urklipp man ser i 5.2.

Listing 5.2: PartPreparation.log

```
2014-02-10 16:33:49 5456  None  Entering NpgsqlCommand.
    ReplaceParameterValue(INSERT INTO monitor.
    FormReportBuildingBlock (Name, Type, IsStandard, Definition,
     Id) VALUES (:p0, :p1, :p2, :p3, :p4), p0, $1::text)
...
2014-02-10 17:52:01 4628  None  Entering NpgsqlCommand.
    ReplaceParameterValue(select 'insert', p0, $1::int8)
```

Det anrop som inte såg ut som de andra och inte gick att köra i PostgreSQL var *select 'insert'* och återfanns i *PartPreparation.hbm.xml*[14][15]. För att lösa problemet ersattes *'insert'* och *'update'* med *?* så att koden kunde exekveras, förslagsvis går det även att kommentera bort den felande koden.

5.4 Access

Efter alla ändringar gjorts gick det att starta programmet, dock så fungerade inte de implementerade funktionerna i systemet. För att kunna söka efter företag behövde en koppling göras mellan SQL Anywheres `patindex` och en motsvarande funktion i PostgreSQL[16]. Betydligt knepigare blev det däremot att försöka få funktionen LIST att fungera. LIST är en SQL Anywhere funktion som används för att dynamiskt hämta ut ett eller flera kolumnnamn som man vill använda i en SELECT-sats, ett exempel på hur det används finns i 5.3. En sådan funktion existerar dock inte i PostgreSQL utan man skulle behöva bygga en sådan själv, vilket går att göra eftersom man kan definiera egna summerande funktioner. Ett sådant försök finns i funktionen `AddListFunction`[17].

Listing 5.3: Exempel på användning av List ur MONITOR

```
SELECT Alloy.Id, Code, List(DISTINCT (
select List(distinct desca.text, '\n') from monitor.
    DynamicPhraseTranslation desca where desca.DynamicPhraseId
    in(Alloy.DescriptionId,Unit.CodeId, Unit.DescriptionId)))
FROM monitor.Alloy LEFT OUTER JOIN monitor.Unit on Unit.Id =
    Alloy.UnitId
GROUP BY Alloy.Id, Code
```

[13] Förmodad anledning utforskas mer i 5.5.3
[14] .hbm.xml är en xml-fil som beskriver kopplingen mellan klass och relation
[15] Se appendix B.8 på sidan 55
[16] Se appendix B.6 på sidan 53 rad nummer 143
[17] Se appendix B.2 på sidan 48 rad nummer 152

5.5 Experiment

5.5.1 Databasskapande

Tabell 5.1 är en sammanställning av bilderna C.3 och C.4 på sidorna 80 respektive 81. Tiderna är tagna genom att avbryta `ConvertDemo3ToPostgreSQL`

DBMS	Skapandetid (sekunder)
SQL Anywhere	52
PostgreSQL	44

Tabell 5.1: Skapandetider för databas och tabeller

[18] på rad 207, och motsvarande för `ConvertDemo3`, precis efter att alla tabeller skapats men innan någon data konverterats. Här verkar PostgreSQL vara lite snabbare, dock så borde en del av tiden kunna förklaras då den server som PostgreSQLdatabasen skapas på var tvungen att vara uppstartad innan koden kördes. Detta i jämförelse med SQL Anywhere som kan starta upp en temporär server via flaggor i anropet.

5.5.2 Konvertering

Tittar man istället på exekveringstiderna för hela konverteringen, tabell 5.2, verkar det som att PostgreSQLkonverteringen är lite mer än 10 gånger långsammare. Tabellen är baserad på totaltiderna från B.11 och B.12, vars utskrifter egentligen är likadana som de för databas och tabellskapandet men då dessa var väldigt mycket längre valde jag att presentera dem i skrift istället för med bilder.[19][20]

DBMS	Konverteringstid
SQL Anywhere	211 s $\approx$ 3,5 min
PostgreSQL	2188 s $\approx$ 36 min

Tabell 5.2: Konverteringstider

I det här skedet kan man konstatera att någonting förmodligen är på tok när datan konverteras, så för att studera detta närmare gjordes de test som beskrivs i 5.5.3.

5.5.3 Insättning

Det gjordes två test som har med insättning att göra:

[18] Se appendix B.1 på sidan 46
[19] Se appendix B.11 på sidan 56
[20] Se appendix B.12 på sidan 63

1. Det ena för att se om någon skillnad mellan SQL Anywhere och PostgreSQL existerar i avseende på hastigheten som data sätts in i de olika systemen.

2. Det andra för att få en inblick i vilken metod som används när sparning av ett objekt görs i NHibernate.

Då NHibernate var det enda som fungerade att skapa databaser med i PostgreSQL valde jag att bara göra testen med det ORM:et. För att genomföra testen skapades en enkel relation `Person`[21].

Det första testet gjordes för att ta reda på om det är vid insättning av data som den stora differensen mellan konverteringstiderna dyker upp. För att undersöka detta skapades metoden `InsertTest`[22] med stödfunktioner, som insättningsmetod används NHibernates `Save` vilket är den man använder i MONITORkoden.

I tabell 5.3 kan man även här se en markant tidsskillnad mellan de två DBMS:erna, trots att datan i detta fall är väldigt simpel.[23]

Insättningar	SQL Anywhere	PostgreSQL
1000	0,2	0,3
10000	0,9	3,0
100000	10,6	27,5
1000000	105,8	222,7

Tabell 5.3: PostgreSQL vs SQL Anywhere (sekunder)

Efter det första testets resultat, som påvisar ett långsammare PostgreSQL, utformades det andra testet för att se om insättningar i PostgreSQL gick att snabba upp och för att eventuellt kunna avgöra vilket sätt som `Save` använder sig utav. Tabell 5.4 innehåller de tider som skrevs ut när `NpgsqlInsertTests`[24] kördes.[25]

Insättningar	Konkatenering	Förberedd	Kopiering
1000	0,27	0,32	0,03
10000	1,93	2,74	0,20
100000	20,52	25,45	2,04
1000000	203,31	317,87	21,16

Tabell 5.4: Körtider för insättningsmetoder i PostgreSQL (sekunder)

Jämför man tabellerna 5.3 och 5.4 så kan man i alla fall konstatera en sak, kopiering av datan används inte.

[21] Se appendix C.1 på sidan 71
[22] Se appendix C.2 på sidan 71 rad nummer 118
[23] Fullständig utskrift återfinns i figur C.2 på sidan 79
[24] Se appendix C.2 på sidan 71 rad nummer 248
[25] Fullständig utskrift återfinns i figur C.1 på sidan 79

Konkatenering

Det vanligaste och kanske enklaste sättet är att med strängar bygga ihop SQL-frågorna och återfinns i funktionen `CreatePersonsConcat`[26]:

```
cmd.CommandText = "insert into person (id, name)values ({0}, '{1}');", i, "test
"+ i);
```

Förberedd

För att använda sig av förberedda kommandon krävs det lite mer kodande vilket syns i funktionen `CreatePersonsPrepared`[27] men generellt sett så skapar man ett kommando:

```
cmd.CommandText = "insert into person (id, name)values (@id, @name);";
```

Varpå man sedan använder namnen för att byta ut värdena:

```
cmd.Parameters["id"].Value = i;
```

Dessa bör vara snabbare än konkateneringsmotsvarigheten men kommandona som ges måste vara tillräckligt komplexa för att en positiv skillnad skall märkas eftersom annars görs en massa arbete i onödan.

Kopiering

Kopiering är ett sätt att ladda in stora mängder data på kort tid och görs i PostgreSQL via kommandot *COPY*. Hela funktionen som kopierar en `Person` heter `CreatePersonsCopy`[28] där kopieringskommandot ser ut som följer:

```
cmd.CommandText = "COPY person(id, name)FROM STDIN;";
```

Detta används sedan i detta fallet genom att kopiera data till en ström så att PostgreSQL kan läsa därifrån.

```
copyIn.Start();
for (int i = start; i < end; i++)
{
  serializer.AddInt32(i);
  serializer.AddString("text" + i);
  serializer.EndRow();
  serializer.Flush();
}

copyIn.End();
```

[26] Se appendix C.2 på sidan 71 rad nummer 150
[27] Se appendix C.2 på sidan 71 rad nummer 173
[28] Se appendix C.2 på sidan 71 rad nummer 200

Kapitel 6

Diskussion - Är PostgreSQL ett alternativ?

Detta kapitel besvarar frågorna som ställdes i frågeställningen i kapitel 1, ger mina rekommendationer om vad man på Monitor bör tänka på i framtiden samt föreslår en idé på framtida arbete.

6.1 Frågeställning - En återkoppling

6.1.1 Vad krävs för att installera PostgreSQL?

För att installera PostgreSQL behövs inte mycket mer än binärer och att man kör en initiering av den.

6.1.2 Är det enkelt att hålla uppdaterat?

PostgreSQL uppgraderas med jämna mellanrum och där emellan släpper man bara mindre uppdateringar, det verkar också enkelt att underhålla då pg_upgrade sköter mycket utav jobbet.

6.1.3 Vad finns det för backup alternativ?

Här finns lite olika alternativ men viktigast är att det finns möjlighet att sätta upp kontinuerlig backup som dessutom är väldigt justerbar.

6.1.4 Hur står sig PostgreSQL jämtemot SQL Anywhere

Syntaxmässigt?

Generellt var det inga problem då allt som behövdes för att skapa alla tabeller fanns, problem uppstod dock när SQL Anywhere specifika funktioner användes. Detta går dock att lösa då det går att definiera egna summerande funktioner i PostgreSQL som i sådana fall kan efterlikna funktionaliteten.

Prestandamässigt?

Då jag diskuterade tidsåtgången för konvertering av databasen med min handledare på företaget insågs det snabbt att om det tar 10 gånger så lång tid så är det inte hållbart i varken ett körbart system eller en testmiljö. Ett exempel som min handledare gav var att de har kunddatabaser i dagsläget där konvertering till ny version med SQL Anywhere tar runt 12 timmar, vilket skulle motsvara 120h (5 dagar) för en konvertering till PostgreSQL.

En lösning som skulle vara tvungen att göras om PostgreSQL ska användas vore att se över om det inte går att antingen via `Save`metoden eller på annat sätt använda sig av kopiering för att föra över data. En sådan lösning skulle i sådana fall kunna ta bort mycket utav den tidsdifferensen som existerar idag mellan PostgreSQL och SQL Anywhere implementationerna. Om det är så som SQL Anywhere versionen fungerar med hjälp av deras drivrutiner det vet jag inte men om så inte är fallet skulle en tidsvinning eventuellt även existera för SQL Anywhere-versionen av konverteringen.

En annan viktig anledning som uppdagades i konversationen med handledaren var varför man använde sig av `Save`metoden och inte bara dumpade in all data. Detta gjordes så för att kunna kontrollera datan under konverteringen, därav benämningen, så att ingenting blev korrupt och ifall något skulle ske finns en större möjlighet att felsöka men även att bara återskapa den ursprungliga versionen.

6.1.5 Vilka ändringar/tillägg behöver göras i MONITORs kod för att få det fungerande tillsammans med PostgreSQL?

Ändringarna som gjordes beskrivs mer i kapitel 5 men övergripande går att säga att inga drastiska ändringar behövts göras i MONITORs kod, i alla fall inte för att få ett grundläggande system att starta.

Mycket problem uppstod dock när funktionaliteten skulle testas, detta på grund utav att mycket av den är skriven i SQL i koden. Detta skapade till exempel problem med vissa uträkningar, jag lyckades inte lista ut var exakt det blev fel men förmodligen har det att göra med hur Npgsql byter ut namngivna parametrar mot värden och att detta eventuellt sker i två steg.

6.2 Rekommendationer

Om man vill utforska huruvida en SQL-dialekt klarar av att köras i MONITOR bör man kolla upp om det finns någon motsvarighet för hårdkodade lösningar (t.ex. *LIST*), skulle det visa sig att detta inte är möjligt för fler dialekter än PostgreSQL skulle det kunna vara en idé att se över koden i sig och se om man kan hitta andra sätt att implementera lösningen. Detta så att man inte stenhårt låser sig till SQL Anywhere för all framtid.

Först och främst bör man dock göra ett hastighetstest så att man kan uppskatta hur lång tid en konvertering skulle ta. Är det så att det tar mer än 2 gånger så lång tid bör man ha i åtanke att en lösning på hastighetsproblemet borde fixas då det kan ta för lång tid i slutprodukten men framförallt kommer utvecklingen av programvaran att gå långsammare då även små testfall skapar mycket extra dötid.

6.3 Framtida arbete

Ta reda på exakt hur funktioner och argument evalueras i NHibernate. Detta inkluderade specifika funktioner som LIST men även mer generella uträkningar som skedde i koden. I fallet med LIST hade det varit nödvändigt att veta hur argument evaluerades då funktionaliteten hade varit tvungen att byggas och fall som rekursion samt återkommande anrop hade uppstått. De mer generella uträkningar i koden verkade inte köras korrekt på grund av att de evalueras i två steg och någonstans på vägen kommunicerade inte NHibernate och Npgsql som de skulle.

Litteraturförteckning

[1] Monitor ERP System AB, "Company Today", *Monitor*, november 2014. [Online] Tillgänglig: http://www.monitor.se/company/companytoday/ [hämtad: 1 november, 2014]

[2] Wikimedia Foundation, "NHibernate", *Wikipedia*, november 2014. [Online] Tillgänglig: http://en.wikipedia.org/wiki/NHibernate [hämtad: 1 november, 2014]

[3] Wikimedia Foundation, "Entity Framework", *Wikipedia*, november 2014. [Online] Tillgänglig: http://en.wikipedia.org/wiki/Entity_Framework [hämtad: 1 november, 2014]

[4] The PostgreSQL Global Development Group, "Backup", *PostgreSQL Documentation*, september 2013. [Online] Tillgänglig: http://www.postgresql.org/docs/9.3/static/backup.html [Hämtad: 1 november, 2014]

[5] The PostgreSQL Global Development Group, "Data Types", *PostgreSQL Documentation*, september 2013. [Online] Tillgänglig: http://www.postgresql.org/docs/9.3/static/datatype.html [Hämtad: 10 september, 2014]

[6] Microsoft, "Data Types", *Microsoft SQL Server Documentation*, mars 2012. [Online] Tillgänglig: http://technet.microsoft.com/en-us/library/ms187752.aspx [Hämtad: 10 september, 2014]

[7] iAnywhere Solutions, Inc., "SyBooks Online", *SQL Anywhere Documentation*, juni 2010. [Online] Tillgänglig: http://infocenter.sybase.com/help/topic/com.sybase.help.sqlanywhere.12.0.1/dbreference/rf-datatypes.html [Hämtad: 10 september, 2014]

[8] Wikimedia Foundation Inc., "Comparison of relational database management systems", *Wikipedia*, september 2014. [Online] Tillgänglig: http://en.wikipedia.org/wiki/Comparison_of_relational_database_management_systems [hämtad: 10 september, 2014]

[9] Elmasri, R. and Navathe, S. B. *Fundamentals of Database Systems* 5e uppl. Förlag: Addison-Wesley eller Pearson.

Bilaga A

Jämförelser

Tabellerna A.1-7 är en avskrivning av en wikipedia artikel [8] medan de resterande är en sammanställningen av [5, 6, 7]

Tabell A.1: Databashanterare

DBMS	Latest Relase	Software licence	Main Usage
Apache Derby	15/4/2013	Apache licence	Desktop applications
SQLite	3/9/2013	Public domain	Embedded systems and web applications
H2	17/3/2013	EPL, modded MPL	Desktop applications
Firebird	24/3/2013	IPL, IDPL	Desktop applications
PostgreSQL	5/12/2013	PostgreSQL licence	Desktop applications
HSQLDB	8/10/2013	BSD	Desktop applications
MonetDB	1/4/2012	MonetDB Public licence v1.1	Large databases (Data mining, GIS and more)
SmallSQL	1/12/2008	LGPL	Java Desktop (Embedded)
CUBRID	24/2/2012	GPL v2	
Drizzle	23/5/2012	BSD, GPL v2	
Ingres	12/10/2010	GPL or Proprietary	
LucidDB		GPL v2	
MariaDB	21/11/2013	GPL v2	
MySQL	30/7/2013	GPL or Proprietary	
OpenLink Virtuoso	5/8/2013	GPL or Proprietary	
Microsoft SQL Server	1/1/2012	Proprietary	Windows desktop applications
SQL Anywhere	9/7/2010	Proprietary	Desktop applications

Tabell A.2: Uppdelning

Partitioning	Apache Derby	SQLite	H2	Firebird	PostgreSQL	Microsoft SQL Server	SQL Anywhere
Range	No	No	No	No	Yes	Yes	No
Hash	No	No	No	No	Yes	No	No
Range+Hash	No	No	No	No	Yes	No	No
List	No	No	No	No	Yes	No	No

Tabell A.3: Operativsystem

Operating system	Apache Derby	SQLite	H2	Firebird	PostgreSQL	Microsoft SQL Server	SQL Anywhere
Windows	Yes	Yes	Yes	Yes	Yes	Yes	Yes
OS X	Yes	Yes	Yes	Yes	Yes	No	Yes
Linux	Yes	Yes	Yes	Yes	Yes	No	Yes
BSD	Yes	Yes	Yes	Yes	Yes	No	No
UNIX	Yes	Yes	Yes	Yes	Yes	No	Yes
iOS	?	Yes	No	No	No	No	No
Android	No	Yes	Yes	No	Yes	No	Yes

Tabell A.4: Begränsningar

Limits	Apache Derby	SQLite	H2	Firebird
Max DB size	Unlimited	128 TB	64 TB	Unlimited
Max table size	Unlimited	Limited by file size	2^{31} objects	32 TB
Max columns per row	1 012 (5 000 in views)	32767	2^{31} objects	Depends
Max Blob/Clob size	2^{31} chars	2 GB	64 TB	2 GB
Max column name size (char)	128	Unlimited	2^{31}	31

Limits	PostgreSQL	Microsoft SQL Server	SQL Anywhere
Max DB size	Unlimited	542 272 TB	104 TB
Max table size	32 TB	542 272 TB	Limited by file size
Max columns per row	250 - 1 600	30000	45000
Max Blob/Clob size	1GB (text), 4 TB (pg_largeobject)	2 GB	2 GB
Max column name size (char)	63	128	?

Tabell A.5: Funktionalitet

Capabilities	Apache Derby	SQLite	H2	Firebird
Outer joins	Yes	LEFT only	Yes	Yes
Intersect	Yes	Yes	Yes	?
Except	Yes	Yes	Yes	?
Inner selects	?	Yes	Yes	Yes
Merge joins	?	No	No	Yes
Blobs and Clobs	Yes	Yes	Yes	Yes
Common Table Expressions	?	No	?	Yes
Windowing Functions	No	No	No	Yes
Parallel Query	?	No	?	?
Materialized view	No	No	No	No
Data Domain	No	No	Yes	Yes
Cursor	Yes	No	No	Yes
Trigger	Yes	Yes	Yes	Yes
Function	Yes	No	Yes	Yes
Procedure	Yes	No	Yes	Yes
External Routine	Yes	Yes	Yes	Yes

Capabilities	PostgreSQL	Microsoft SQL Server	SQL Anywhere
Outer joins	Yes	Yes	Yes
Intersect	Yes	Yes	Yes
Except	Yes	Yes	Yes
Inner selects	Yes	Yes	Yes
Merge joins	Yes	Yes	Yes
Blobs and Clobs	Yes	Yes	Yes
Common Table Expressions	Yes	Yes	Yes
Windowing Functions	Yes	Yes	Yes
Parallel Query	No	Yes	Yes
Materialized view	Yes	Yes	Yes
Data Domain	Yes	Yes	Yes
Cursor	Yes	Yes	Yes
Trigger	Yes	Yes	Yes
Function	Yes	Yes	Yes
Procedure	Yes	Yes	Yes
External Routine	Yes	Yes	Yes

Tabell A.6: Åtkomst

Access control	Apache Derby	SQLite	H2	Firebird
Native network encryption	?	Not relevant	Yes	No
Brute-force protection	?	Not relevant	Yes	Yes
Enterprise directory compatibility	?	Not relevant	?	Yes
Password complexity rules	?	Not relevant	No	No
Patch access	?	Partial	?	Partial
Audit	?	Yes	?	No
Resource limit	?	Yes	Yes	No
RBAC	?	No	Yes	No
Security Certification	?	No	No	?

Access control	PostgreSQL	Microsoft SQL Server	SQL Anywhere
Native network encryption	Yes	Yes	Yes
Brute-force protection	Yes	?	?
Enterprise directory compatibility	Yes	Yes	Yes
Password complexity rules	Yes	Yes	Yes
Patch access	Yes	Yes	?
Audit	No	Yes	Yes
Resource limit	Yes	Yes	No
RBAC	Yes	Yes	Yes
Security Certification	EAL 1+	EAL 4+	EAL 3+

Tabell A.7: Indexering

Index	Apache Derby	SQLite	H2	Firebird	PostgreSQL	Microsoft SQL Server	SQL Anywhere
R-/R+ tree	No	Yes	No	No	Yes	?	No
Hash	No	No	Yes	No	Yes	Cluster & fill factor	No
Expression	No	No	No	Yes	Yes	Yes	No
Partial	No	Yes	No	No	Yes	Yes	No
Reverse	No	Yes	No	Yes	Yes	No	No
Bitmap	No	No	No	No	Yes	No	No
GiST	No	No	No	No	Yes	No	No
GIN	No	No	No	No	Yes	No	No
Full-text	No	Yes	Yes	No	Yes	Yes	Yes
Spatial	No	SpatiaLite	Yes	No	Yes	Yes	?

Tabell A.8: Numeriska datatyper

	SQL Anywhere	MS SQL Server	PostgreSQL
1 byte	TINYINT	TINYINT	-
2 bytes	SMALLINT	SMALLINT	SMALLINT
4 bytes	INTEGER	INT	INTEGER
8 bytes	BIGINT	BIGINT	BIGINT
4 bytes	REAL	REAL	REAL
8 bytes	DOUBLE	FLOAT([25..53])	DOUBLE PRECISION
4 or 8 bytes	FLOAT	FLOAT	-
2 bytes	-	-	SMALLSERIAL
4 bytes	-	-	SERIAL
8 bytes	-	-	BIGSERIAL
4 bytes	SMALLMONEY	SMALLMONEY	-
8 bytes	MONEY	MONEY	MONEY
-	-	BOOLEAN	

Tabell A.9: Teckendatatyper

SQL Anywhere	CHAR(n, [CHAR]), VARCHAR(n)	n = [1..32767], $\begin{cases} \text{CHAR} & n = \#\text{characters} \\ \text{VARCHAR} & n = \#\text{bytes} \end{cases}$
MS SQL Server	CHAR(n), CHARACTER(n)	n =#bytes, n = [1..8000], will be padded with trailing spaces up to n, fixed size
PostgreSQL	CHAR(n), CHARACTER(n)	n =#characters, upper limit of n varies, will be padded with trailing spaces up to n , fixed size
SQL Anywhere	LONG VARCHAR(n)	n = #bytes, $n = [1..2^{31} - 1]$
MS SQL Server	VARCHAR(n)	n = #bytes, $n = [1..2^{31} - 1]$
PostgreSQL	VARCHAR(n)	n = #characters, upper limit of n varies
SQL Anywhere	TEXT	Implemented as a LONG VARCHAR
MS SQL Server	TEXT	Upper limit of $2^{31} - 1$ bytes
PostgreSQL	TEXT	No upper limit
SQL Anywhere	NCHAR(n), NVARCHAR(n)	n = #characters, n = [1..32767], UTF-8 encoded characters, $n <$ bytes $< 4n$ storage size
MS SQL Server	NCHAR(n)	n = #characters, n = [1..4000], UTF-8 encoded characters, $n < bytes < 2n$ storage size, will be padded with trailing spaces up to n, fixed size
SQL Anywhere	LONG NVARCHAR(n)	n = #characters, maximum size = $2^{31} - 1$ bytes, UTF-8 encoded characters, $1 - 4$ bytes/character
MS SQL Server	NVARCHAR(n)	n = #characters, maximum size = $2^{31} - 1$ bytes, UTF-8 encoded characters, storage size = 2∗length(actual data)+2 bytes
SQL Anywhere	NTEXT	Implemented as a LONG NVARCHAR
MS SQL Server	NTEXT	Maximum string length = $2^{30} - 1$ bytes, UTF-8 encoded characters, storage size = 2∗length(string)

Tabell A.10: Binära Datatyper

SQL Anywhere	BINARY(n), VARBINARY(n)	n = [1..32767]
MS SQL Server	BINARY(n)	n = [1..8000], will be padded with trailing zeros up to n, fixed size
SQL Anywhere	LONG BINARY	Maximum size = 2^{31} – 1 bytes
MS SQL Server	VARBINARY	Maximum size = 2^{31} – 1 bytes, storage size= 2∗ length(data) +2 bytes
PostgreSQL	BYTEA	Storage size = length(binary string)+(1 or 4 bytes overhead)
SQL Anywhere	BIT	0 or 1, not null
MS SQL Server	BIT	0 or 1, $x \neq 0 \Rightarrow 1$
SQL Anywhere	VARBIT(n)	n = #bits, n = [1..32767]
PostgreSQL	BIT(n)	Will be padded with trailing zeros up to n, fixed size
SQL Anywhere	LONG VARBIT	Can store arbitrary length of 1s and 0s
PostgreSQL	BIT VARYING[n]	Stores an arbirary length of 1s and 0s unless n is specified

Tabell A.11: Datum och Tid

SQL Anywhere	MS SQL Server	PostgreSQL	Format
DATE	DATE	DATE	YYYY-MM-DD
TIME	TIME	TIME	hh:mm:ss[.nnnnnn]
-	-	TIME WTH TIME ZONE	hh:mm:ss[.nnnnnn] [±hh:mm]
TIMESTAMP, DATETIME	DATETIME	TIMESTAMP	YYYY-MM-DD hh.mm.ss[.nnnnnn]
TIMESTAMP WITH TIME ZONE, DATETIMEOFFSET	DATETIMEOFFSET	TIMESTAMP WITH TIME ZONE	YYYY-MM-DD hh:mm:ss[.nnnnnn] [±hh:mm]
-	-	INTERVAL	

Optional arguments for INTERVAL: YEAR, MONTH, DAY, HOUR, MINUTE, SECOND and some more

Tabell A.12: Skala och noggrannhet för NUMERIC och DECIMAL

	p (precision)	s (scale)	Storage size
SQL Anywhere	1..127	0..p	$2 + \text{INT}\left(\frac{BEFORE+1}{2}\right)$ $+\text{INT}\left(\frac{AFTER+1}{2}\right)$
MS SQL Server	1..38	0..p	$\begin{cases} 5 \text{ bytes} & p \in [1..9] \\ 9 \text{ bytes} & p \in [10..19] \\ 13 \text{ bytes} & p \in [20..28] \\ 17 \text{ bytes} & p \in [29..38] \end{cases}$
PostgreSQL	1..1000	0..p	3-8 bytes overhead + 2 bytes/ 4 digits
	1..147455	0..$z, z \leq p$ $z \leq 16383$	

BEFORE\AFTER = antal siffror innan\efter decimaltecken

Bilaga B

Skapande

B.1 Monitorkod ändringar

Listing B.1: ConvertTestDatabases.cs

```
151     [TestMethod]
152     public void ConvertDemo3ToMSSql()
153     {
154       var stopWatch = new Stopwatch();
155       stopWatch.Start();
156       const string databaseName = "Monitor";
157       var outPutDirectory = Path.Combine(MonwinTestPath, "
              ConvertingDemo3");
158
159       var dbPath = string.Format(@"{0}\{1}.mdf", outPutDirectory
              , databaseName);
160       var logPath = string.Format(@"{0}\{1}.ldf",
              outPutDirectory, databaseName);
161
162       DeleteFile(dbPath);
163       DeleteFile(logPath);
164
165       var creator = new SqlServerDatabaseCreator();
166       var dbInfo = creator.Create(databaseName, outPutDirectory)
              ;
167
168       var importer = new DataImporter(new TypeLocator(new[] {
              typeof(SqlServerDatabaseCreator).Assembly }), new
              DefaultImporterActivator());
169       importer.Import(dbInfo.InstallConnectString,
              MonitorDatabaseDialect.MicrosoftSqlServer);
170
171       ImportData(@"\\server1\mon_utv\db\TestG5\monitor.db", "
              ConvertingDemo3", MonitorDatabaseDialect.
              MicrosoftSqlServer, ExtraSqlForDemo3);
172
173       const string connectString = @"integrated security=SSPI;
              data source=.\SQLEXPRESS;persist security info=False;
              initial catalog=master";
```

```
174         using (var sessionBuilder =
                DefaultPersistenceSessionCreator.
                CreateNHibernateSessionBuilder(connectString, new
                Dictionary<Assembly, IEnumerable<string>>(), false,
                MonitorDatabaseDialect.MicrosoftSqlServer))
175         using (var session = sessionBuilder.OpenSession(new
                EmptyInterceptor()))
176         {
177           var command = session.Connection.CreateCommand();
178           command.CommandText = "USE Monitor";
179           command.ExecuteNonQuery();
180           command.CommandText = @"EXEC sp_msforeachtable ""ALTER
                  TABLE ? CHECK CONSTRAINT ALL"";";
181           command.ExecuteNonQuery();
182           command.CommandText = "DBCC SHRINKFILE (Monitor_Log, 1,
                  TRUNCATEONLY);";
183           command.ExecuteNonQuery();
184           command.CommandText = "USE master";
185           command.ExecuteNonQuery();
186           command.CommandText = "EXEC sp_detach_db 'Monitor', '
                  true'";
187           command.ExecuteNonQuery();
188         }
189         stopWatch.Stop();
190         Console.WriteLine("Total time: {0}", stopWatch.Elapsed.
                TotalSeconds);
191       }
192
193       [TestMethod]
194       public void ConvertDemo3ToPostgreSQL()
195       {
196         var stopwatch = new Stopwatch();
197         stopwatch.Start();
198         const string databaseName = "monitor";
199
200         var creator = new PostgreSQLDatabaseCreator();
201         var dbInfo = creator.Create(databaseName, @"C:\
                PostgresTest");
202
203         creator.DisableConstraints(databaseName);
204
205         var importer = new DataImporter(new TypeLocator(new[] {
                typeof(PostgreSQLDatabaseCreator).Assembly }), new
                DefaultImporterActivator());
206         importer.Import(dbInfo.InstallConnectString,
                MonitorDatabaseDialect.PostgreSQL);
207
208         ImportData(@"\\server1\mon_utv\db\TestG5\monitor.db", "
                ConvertingDemo3", MonitorDatabaseDialect.PostgreSQL,
                ExtraSqlForDemo3);
209
210         creator.EnableConstraints(databaseName);
211
212         stopwatch.Stop();
213         Console.WriteLine("Total time: {0}", stopwatch.Elapsed.
                TotalSeconds);
214       }
```

Listing B.2: PostgreSQLDatabaseCreator.cs

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6  using Monitor.ComponentModel;
7  using Monitor.Core.DefaultImplementations;
8  using Npgsql;
9  using System.Reflection;
10 using System.IO;
11
12 namespace Monitor.Server.Pil
13 {
14     public class PostgreSQLDatabaseCreator
15     {
16         private const string CreateDb = @"CREATE DATABASE {0}
               OWNER {1};";
17
18         public DatabaseInformation Create(string databaseName,
               string directory)
19         {
20             const string baseConnectionString = @"Server=
                   localhost;Port=5432;Database=postgres;User Id=
                   postgres;Password=H3j;";
21             var connectionString = "";
22
23             using (var connection = new NpgsqlConnection(
                   baseConnectionString))
24             {
25                 connection.Open();
26                 var command = connection.CreateCommand();
27
28                 // Remove database and "user", need to drop
                       database first
29                 command.CommandText = String.Format(@"DROP
                       DATABASE IF EXISTS {0};", databaseName);
30                 command.ExecuteNonQuery();
31                 command.CommandText = "DROP ROLE IF EXISTS
                       applicationserver;";
32                 command.ExecuteNonQuery();
33
34                 // Create user "ApplicationServer" with Create
                       DB rights
35                 command.CommandText = "CREATE USER
                       applicationserver CREATEDB PASSWORD '
                       smjmek29'";
36                 command.ExecuteNonQuery();
37
38                 // Create Database
39                 command.CommandText = String.Format(CreateDb,
                       databaseName, "applicationserver");
40                 command.ExecuteNonQuery();
41
42                 connection.Close();
43             }
44
```

```
45              // Login as that user instead
46              connectionString = String.Format(@"Server=localhost;
                    Database={0};User Id=applicationserver;Password=
                    smjmek29;", databaseName);
47
48              using (var sessionBuilder =
                    DefaultPersistenceSessionCreator.
                    CreateNHibernateSessionBuilder(connectionString,
                     new Dictionary<Assembly, IEnumerable<string>>()
                    , true, MonitorDatabaseDialect.PostgreSQL))
49              using (var session = sessionBuilder.OpenSession(null
                    ))
50              using (var conn = new NpgsqlConnection(
                    connectionString))
51              {
52                  conn.Open();
53
54                  var command = conn.CreateCommand();
55
56                  // Create monitor schema
57                  command.CommandText = "CREATE SCHEMA \"monitor
                        \";";
58                  command.ExecuteNonQuery();
59
60                  // Add it to the front of the schema search
                        order for the database
61                  command.CommandText = String.Format(@"ALTER
                        DATABASE {0} SET search_path=monitor,public;
                        ", databaseName);
62                  command.ExecuteNonQuery();
63
64                  var exporter = sessionBuilder.GetSchemaExporter
                        ();
65
66                  // Create all tables - Does not create any views
67                  exporter.Create(false, true);
68                  conn.Close();
69              }
70
71              return new DatabaseInformation {
                    InstallConnectString = connectionString };
72          }
73
74          private void performConstraintChanger(string
                databaseName, string enable_disable)
75          {
76              // Since there are no global way of disabling/
                    enabling checks it needs to be done at table
                    level.
77              // This function loops through all tables (actually
                    their names) and disables/enables triggers
78              var constraintsString =
79              @"CREATE FUNCTION tempFunc() RETURNS VOID AS $$
80                      DECLARE
81                          cur CURSOR FOR SELECT table_name FROM
                                information_schema.tables WHERE
                                table_schema='monitor';
```

```
82                          name information_schema.tables.
                                table_name%TYPE;
83                      BEGIN
84                          OPEN cur;
85                          LOOP
86                              FETCH cur INTO name;
87                              IF NOT FOUND THEN
88                                  EXIT;
89                              END IF;
90                              EXECUTE 'ALTER TABLE ' ||
                                    quote_ident(name) || ' {0}
                                    TRIGGER ALL;';
91                          END LOOP;
92                      END;
93                      $$ LANGUAGE plpgsql;";
94
95              string connectionString = String.Format(@"Server=
                    localhost;Database={0};User Id=postgres;Password
                    =H3j;", databaseName);
96
97              using (var connection = new NpgsqlConnection(
                    connectionString))
98              {
99                  connection.Open();
100
101                 var command = connection.CreateCommand();
102
103                 // Create the function
104                 command.CommandText = String.Format(
                        constraintsString, enable_disable);
105                 command.ExecuteNonQuery();
106
107                 // Use it...
108                 command.CommandText = "SELECT tempFunc();";
109                 command.ExecuteNonQuery();
110
111                 // ... and remove it
112                 command.CommandText = "DROP FUNCTION tempFunc()"
                        ;
113                 command.ExecuteNonQuery();
114
115                 connection.Close();
116             }
117         }
118
119         // Disable trigger checks (e.g. foreign keys, not-null)
120         public void DisableConstraints(string databaseName)
121         {
122             performConstraintChanger(databaseName, "DISABLE");
123         }
124
125         // Enable trigger checks (e.g. foreign keys, not-null)
126         public void EnableConstraints(string databaseName)
127         {
128             performConstraintChanger(databaseName, "ENABLE");
129         }
130     }
```

```
131 }
132
133         /*
134          * This is the SQL Anywhere function List partially
                implemented
135          *
136          * List(columnName, delimiter) -> string_agg(columnName,
                delimiter)
137          * List(DISTINCT columnName, delimiter) ->
                array_to_string(list_ditinct columnName, delimiter)
138          *
139          * To make this work the List-call would have to be
                mapped to a function that selects which
140          * of the implementations should be returned dependent
                on if the DISTINCT parameter is specified.
141          * Another problem arises due to that fact that there
                can be an expression instead of a column name,
142          * this _should_ probably solve itself but since I don't
                know the exact nature of how parameters are
143          * stored/replaced/executed between NHibernate and
                Npgsql only guesswork can be done so heavy testing
144          * would be needed.
145          *
146          * In the Monitor source code occurences of Select List(
                distinct (Select List(distinct ...
147          * are a way of dynamically determine which columns to
                use in a select statement.
148          *
149          * The following code would also breaks in the above
                example due to the fact that I don't think
150          * it'll handle multiple columns as an argument.
151          */
152         public void AddListFunction(string databaseName)
153         {
154             var appendString =
155                 @"CREATE OR REPLACE FUNCTION append_if_distinct(
                        anyarray, anyelement) RETURNS anyarray AS $$
156                     BEGIN
157                         IF (SELECT $2 = ANY($1)) THEN
158                             return $1;
159                         ELSE
160                             return array_append($1, $2);
161                         END IF;
162                     END;
163                     $$ LANGUAGE plpgsql;";
164
165             var aggregateString =
166                 @"CREATE AGGREGATE list_distinct(anyelement)
167                         (
168                             sfunc=append_if_distinct,
169                             stype=anyarray,
170                             initcond='{}'
171                         );";
172
173             string connectionString = String.Format(@"Server=
                    localhost;Database={0};User Id=postgres;Password
                    =H3j;", databaseName);
```

```
174
175             using (var connection = new NpgsqlConnection(
                    connectionString))
176             {
177                 connection.Open();
178
179                 var command = connection.CreateCommand();
180
181                 command.CommandText = appendString;
182                 command.ExecuteNonQuery();
183
184                 command.CommandText = aggregateString;
185                 command.ExecuteNonQuery();
186             }
187         }
188     }
189 }
```

Listing B.3: SessionFactory.cs

```
88      public static NHibernateSessionBuilder
            CreateNHibernateSessionBuilder(string connectStringIn,
            IDictionary<Assembly, IEnumerable<string>>
            externalMappings, bool debugMode, MonitorDatabaseDialect
             databaseDialect)
89      {
90        var coreAssembly = Assembly.GetAssembly(typeof(Part));
91        var monitorModelInspector = new MonitorModelInspector(
              coreAssembly);
92        var conventionMapper = new MonitorDbConventionModelMapper(
              monitorModelInspector, coreAssembly);
93
94        var properties = NHibernateDefaultPropertiesFactory.
              GetDefaultProperties();
95        string adaptedConnectString = connectStringIn;
96
97        if (databaseDialect == MonitorDatabaseDialect.
              MicrosoftSqlServer)
98        {
99          properties["dialect"] = "Monitor.Core.
                DefaultImplementations.MonitorMsSql2012Dialect,
                Monitor.Core.DefaultImplementations";
100         properties.Add("adonet.batch_size", "50");
101       }
102       else if (databaseDialect == MonitorDatabaseDialect.
              PostgreSQL)
103       {
104         properties["dialect"] = "Monitor.Core.
                DefaultImplementations.MonitorPostgreSQLDialect,
                Monitor.Core.DefaultImplementations";
105       }
106       else
107       {
108         adaptedConnectString = ConnectionStringAdapter.Adapt(
                connectStringIn);
109       }
110
```

```
111         return NHibernateSessionBuilderBuilder.BeginConfiguration
                ()
112           .SetNHibernateProperties(properties)
113           .SetConnectString(adaptedConnectString)
114           .SetDebugMode(debugMode)
115           .SetConventionModelMapper(conventionMapper)
116           .SetupAllMonitorNetMappings()
117           .AddAllMappingsFromResources(externalMappings)
118           .Build();
119       }
```

Listing B.4: MonitorCompanyConfiguration.cs

```
51    public enum MonitorDatabaseDialect
52    {
53      SqlAnywhere = 0,
54      MicrosoftSqlServer = 1,
55      PostgreSQL = 2
56    }
```

Listing B.5: MonitorCompanyConfiguration.xml

```
<?xml version="1.0" encoding="utf-8"?>
<Configuration>
  <DatabaseConfiguration ConnectString ="START=dbsrv12 -z -c 50M
       -gk ALL -gd ALL -x none;AUTOSTOP=NO;DRIVER={SQL Anywhere
      12};ENG=Lokal_G5" DurationBeforeDisconnect="36000" Dbms="
      ODBC" DatabaseDirectory="c:\monwintest\db\g5"/>
  <StartupConfiguration Language="1"/>
  <CompanyConfiguration>
    <Company Number="004" Name="Demo3" DatabaseDirectory="Demo3"
         Image="PerssonsMekaniska.bmp" Autostart="true"/>
    <Company Number="005" Name="Arkivator" DatabaseDirectory="
        Arkivator" Image="PerssonsMekaniska.bmp" Autostart="
        false"/>
    <Company Number="051" Name="Test"  DatabaseDirectory="Test"
        Image="midsummer.png"/>
    <Company Number="008" Name="MS SqlServer" DatabaseDialect="
        MicrosoftSqlServer" Autostart="false"
        ConnectStringExtension="integrated security=SSPI;data
        source=.\SQLEXPRESS;persist security info=False;"
        DatabaseDirectory="Monitor" Image="midsummer.png"/>
    <Company Number="007" Name="Postgres" DatabaseDialect="
        PostgreSQL" Autostart="false" ConnectStringExtension="
        Server=localhost;Database=monitor;Password=H3j;User Id=
        postgres" Image="midsummer.png"/>
  </CompanyConfiguration>
</Configuration>
```

Listing B.6: ConfigurationFileCompanySelector.cs

```
127   public class MonitorMsSql2012Dialect : MsSql2008Dialect
128   {
129     protected override void RegisterFunctions()
130     {
131       base.RegisterFunctions();
```

```
132         RegisterFunction("patindex", new StandardSQLFunction("
                patindex", NHibernateUtil.Int32));
133       }
134     }
135
136       public class MonitorPostgreSQLDialect : PostgreSQLDialect
137       {
138           public MonitorPostgreSQLDialect() : base()
139           {
140               RegisterColumnType(DbType.DateTimeOffset, "
                      timestamptz");
141               RegisterColumnType(DbType.Guid, "uuid");
142
143               RegisterFunction("patindex", new SQLFunctionTemplate
                      (NHibernateUtil.Int32, "position(?1 in ?2)"));
144           }
145
146           public override string GetDropTableString(string
                  tableName)
147           {
148               return String.Format("drop table if exists {0}",
                      tableName);
149           }
150
151           public override string GetDropSequenceString(string
                  sequenceName)
152           {
153               return "drop sequence if exists " + sequenceName;
154           }
155       }
```

Listing B.7: PaymentPlanRowMapping.cs

```
1  using Monitor.Core;
2  using Monitor.Persistence.NHibernateUserTypeFactories;
3
4  namespace Monitor.Persistence.MonitorDB.Mappings
5  {
6    public class PlanRowBaseMapping<TPlanType, TPLanRowType> :
         MappingBase<TPLanRowType>
7      where TPlanType : PlanBase<TPlanType, TPLanRowType>
8      where TPLanRowType : PlanRowBase, IPlanRow, new()
9    {
10     public PlanRowBaseMapping()
11     {
12       Table("PaymentPlanRow");
13       AssignedId();
14
15       Property(r => r.PartialInvoiceType);
16       Property(r => r.Percent); //, map=>map.Column("[Percent]")
             ); // PostgreSQL can't handle unquoted []
17       Property(r => r.RowNumber);
18       Property(r => r.DeductionType);
19
20       ManyToOne(r => r.ParentPlan, map =>
21                      {
22                        map.Class(typeof(TPlanType));
```

```
23                          map.Column("PaymentPlanId");
24                        });
25
26          Property(r => r.ServicePartId, map => { map.Column("PartId
                "); map.Type<BusinessKeyUserType<Part>>(); });
27        }
28      }
29
30      public class InvoicingPlanRowMapping : PlanRowBaseMapping<
            InvoicingPlan, InvoicingPlanRow>
31      {
32        public InvoicingPlanRowMapping()
33        {
34          Property(r => r.PaymentTermId);
35          Property(r => r.UnpaidAdvanceCheckAction);
36        }
37      }
38
39      public class PaymentPlanRowMapping : PlanRowBaseMapping<
            PaymentPlan, PaymentPlanRow>
40      {
41      }
42  }
```

Listing B.8: PartPreparation.hbm.xml

```
<class name="PartPreparation" table="part" schema-action="none
    ">
  <id name="Id" type="Monitor.Persistence.
      NHibernateUserTypeFactories.BusinessKeyUserType`1[[
      Monitor.Core.Part,Monitor.Core]], Monitor.Persistence" >
    <generator class="assigned" />
  </id>
  <list name="Materials" access="nosetter.camelcase" cascade="
      all-delete-orphan" table="PartMaterialRowList"  schema="
      monitor">
    <key column="MainPartId"/>
    <index column="RowIndex"/>
    <many-to-many column="MaterialRowId" class="MaterialRow" >
        </many-to-many>
  </list>

  <list name="Operations" access="nosetter.camelcase" cascade=
      "all-delete-orphan" table="PartOperationRowList" schema=
      "monitor" >
    <key column="MainPartId"/>
    <index column="RowIndex"/>
    <many-to-many column="OperationRowId" class="OperationRow"
         ></many-to-many>
  </list>

  <sql-insert check="none">select ?</sql-insert>
  <sql-delete check="none">select ?</sql-delete>
</class>
```

B.2 Npgsql källkod ändringar

Listing B.9: NpgsqlConnectionStringBuilder.cs

```
677         case "APP":
678         case "APPLICATIONNAME":
679           return Keywords.ApplicationName;
```

Listing B.10: NpgsqlTypesHelper.cs

```
617         yield return
618           new NpgsqlBackendTypeInfo(0, "timestamptz",
                  NpgsqlDbType.TimestampTZ, DbType.DateTimeOffset,
                  typeof(NpgsqlTimeStampTZ),
619             new ConvertBackendToNativeHandler(
                    ExtendedBackendToNativeTypeConverter.
                    ToTimeStampTZ),
620             typeof(DateTimeOffset), timestamptz => ((
                    DateTimeOffset)(NpgsqlTimeStampTZ)timestamptz).
                    ToLocalTime(), npgsqlTimestampTZ => (
                    npgsqlTimestampTZ is DateTime ? (
                    NpgsqlTimeStampTZ)(DateTime)npgsqlTimestampTZ :
                    npgsqlTimestampTZ is DateTimeOffset ? (
                    NpgsqlTimeStampTZ)(DateTimeOffset)
                    npgsqlTimestampTZ : npgsqlTimestampTZ));
```

B.3 Konverteringstid

Listing B.11: Körtider för konvertering till SQL Anywhere

```
1  Test Name:  ConvertDemo3
2  Test Outcome: Passed
3  Result StandardOutput:
4  Starting importer [Monitor.Server.Pil.MenuImporter] on thread
       [15]
5  Starting importer [Monitor.Server.Pil.LanguageImporter] on
       thread [20]
6  Starting importer [Monitor.Server.Pil.
       FormReportConfigurationImporter] on thread [18]
7  Starting importer [Monitor.Server.Pil.
       FormReportBuildingBlockImporter] on thread [19]
8  Timings for [Monitor.Server.Pil.LanguageImporter]: total
       :0,0647114
9  Starting importer [Monitor.Server.Pil.NumberSeriesImporter] on
       thread [20]
10 Timings for [Monitor.Server.Pil.FormReportBuildingBlockImporter
       ]: total:0,1583222
11 Timings for [Monitor.Server.Pil.FormReportConfigurationImporter
       ]: total:0,1584933
12 Timings for [Monitor.Server.Pil.NumberSeriesImporter]: total
       :0,0851305
13 Starting importer [Monitor.Server.Pil.PostalCodeImporter] on
       thread [18]
14 Starting importer [Monitor.Server.Pil.CodingDimesionImporter] on
        thread [20]
15 Starting importer [Monitor.Server.Pil.CalendarImporter] on
       thread [19]
16 Timings for [Monitor.Server.Pil.CodingDimesionImporter]: total
       :0,0534765
```

```
17 Timings for [Monitor.Server.Pil.CalendarImporter]: total
       :0,054051
18 Starting importer [Monitor.Server.Pil.
       CurrencyExchangeTypeImporter] on thread [20]
19 Starting importer [Monitor.Server.Pil.FormReportSettingImporter]
        on thread [19]
20 Timings for [Monitor.Server.Pil.CurrencyExchangeTypeImporter]:
       total:0,0135711
21 Starting importer [Monitor.Server.Pil.CustomerOrderTypeImporter]
        on thread [20]
22 Timings for [Monitor.Server.Pil.CustomerOrderTypeImporter]:
       total:0,0139099
23 Starting importer [Monitor.Server.Pil.
       FormReportTranslationGroupImporter] on thread [20]
24 Timings for [Monitor.Server.Pil.FormReportSettingImporter]:
       total:0,0576995
25 Starting importer [Monitor.Server.Pil.PaymentFormatImporter] on
       thread [19]
26 Timings for [Monitor.Server.Pil.PaymentFormatImporter]: total
       :0,0087753
27 Starting importer [Monitor.Server.Pil.PurchaseOrderTypeImporter]
        on thread [19]
28 Timings for [Monitor.Server.Pil.PurchaseOrderTypeImporter]:
       total:0,0092856
29 Starting importer [Monitor.Server.Pil.RoleImporter] on thread
       [19]
30 Timings for [Monitor.Server.Pil.MenuImporter]: total:0,3429641
31 Starting importer [Monitor.Server.Pil.ScheduledServiceImporter]
       on thread [15]
32 Timings for [Monitor.Server.Pil.RoleImporter]: total:0,1868447
33 Starting importer [Monitor.Server.Pil.CurrencyImporter] on
       thread [19]
34 Timings for [Monitor.Server.Pil.CurrencyImporter]: total
       :0,0345313
35 Starting importer [Monitor.Server.Pil.ApplicationUserImporter]
       on thread [14]
36 Timings for [Monitor.Server.Pil.ApplicationUserImporter]: total
       :0,0163005
37 Starting importer [Monitor.Server.Pil.CountryImporter] on thread
        [14]
38 Timings for [Monitor.Server.Pil.
       FormReportTranslationGroupImporter]: total:0,2994448
39 Starting importer [Monitor.Server.Pil.SystemParameterImporter]
       on thread [19]
40 Timings for [Monitor.Server.Pil.SystemParameterImporter]: total
       :0,0652721
41 Timings for [Monitor.Server.Pil.ScheduledServiceImporter]: total
       :0,5406473
42 Timings for [Monitor.Server.Pil.CountryImporter]: total
       :0,4766609
43 Timings for [Monitor.Server.Pil.PostalCodeImporter]: total
       :3,622539
44 Starting importer [Monitor.DatabaseConverter.Importer.
       BudgetChartImporter] on thread [19]
45 Starting importer [Monitor.DatabaseConverter.Importer.
       CategoryImporter] on thread [14]
```

```
46 Starting importer [Monitor.DatabaseConverter.Importer.
       CodingElementImporter] on thread [20]
47 Starting importer [Monitor.DatabaseConverter.Importer.
       AccountingYearImporter] on thread [18]
48 Timings for [Monitor.DatabaseConverter.Importer.
       AccountingYearImporter]: total:0,1093839
49 Timings for [Monitor.DatabaseConverter.Importer.
       BudgetChartImporter]: total:0,0694956
50 Starting importer [Monitor.DatabaseConverter.Importer.
       CentralBankCodeImporter] on thread [19]
51 Starting importer [Monitor.DatabaseConverter.Importer.
       GoodsTypeImporter] on thread [15]
52 Timings for [Monitor.DatabaseConverter.Importer.CategoryImporter
       ]: total:0,1322253
53 Starting importer [Monitor.DatabaseConverter.Importer.
       PackageTypeImporter] on thread [14]
54 Timings for [Monitor.DatabaseConverter.Importer.
       GoodsTypeImporter]: total:0,0403908
55 Starting importer [Monitor.DatabaseConverter.Importer.
       DeliveryTimeImporter] on thread [15]
56 Timings for [Monitor.DatabaseConverter.Importer.
       DeliveryTimeImporter]: total:0,0701166
57 Timings for [Monitor.DatabaseConverter.Importer.
       CodingElementImporter]: total:0,2280458
58 Starting importer [Monitor.DatabaseConverter.Importer.
       InterruptionCodeImporter] on thread [15]
59 Starting importer [Monitor.DatabaseConverter.Importer.
       LocationTypeImporter] on thread [18]
60 Timings for [Monitor.DatabaseConverter.Importer.
       LocationTypeImporter]: total:0,026818
61 Starting importer [Monitor.DatabaseConverter.Importer.
       PackingTermImporter] on thread [20]
62 Timings for [Monitor.DatabaseConverter.Importer.
       InterruptionCodeImporter]: total:0,0371666
63 Starting importer [Monitor.DatabaseConverter.Importer.
       ProjectImporter] on thread [15]
64 Timings for [Monitor.DatabaseConverter.Importer.ProjectImporter
       ]: total:0,0469454
65 Starting importer [Monitor.DatabaseConverter.Importer.
       RejectionCodeItemImporter] on thread [18]
66 Timings for [Monitor.DatabaseConverter.Importer.
       PackingTermImporter]: total:0,0586055
67 Starting importer [Monitor.DatabaseConverter.Importer.
       ResponseTimeImporter] on thread [15]
68 Timings for [Monitor.DatabaseConverter.Importer.
       ResponseTimeImporter]: total:0,0132931
69 Starting importer [Monitor.DatabaseConverter.Importer.
       LanguageSettingImporter] on thread [20]
70 Timings for [Monitor.DatabaseConverter.Importer.
       LanguageSettingImporter]: total:0,0156618
71 Starting importer [Monitor.DatabaseConverter.Importer.
       CurrencyImporter] on thread [20]
72 Timings for [Monitor.DatabaseConverter.Importer.
       RejectionCodeItemImporter]: total:0,0544556
73 Starting importer [Monitor.DatabaseConverter.Importer.
       CustomerDistrictImporter] on thread [18]
```

```
74  Timings for [Monitor.DatabaseConverter.Importer.
        CustomerDistrictImporter]: total:0,0264091
75  Starting importer [Monitor.DatabaseConverter.Importer.
        CustomerRelationshipActivityImporter] on thread [15]
76  Timings for [Monitor.DatabaseConverter.Importer.
        CustomerRelationshipActivityImporter]: total:0,0828088
77  Starting importer [Monitor.DatabaseConverter.Importer.
        CustomerStatusImporter] on thread [18]
78  Timings for [Monitor.DatabaseConverter.Importer.
        CustomerStatusImporter]: total:0,0353136
79  Starting importer [Monitor.DatabaseConverter.Importer.
        CustomerTypeImporter] on thread [18]
80  Timings for [Monitor.DatabaseConverter.Importer.
        CustomerTypeImporter]: total:0,0262808
81  Starting importer [Monitor.DatabaseConverter.Importer.
        DeliveryMethodImporter] on thread [18]
82  Timings for [Monitor.DatabaseConverter.Importer.
        DeliveryMethodImporter]: total:0,0533396
83  Starting importer [Monitor.DatabaseConverter.Importer.
        DeliveryTermImporter] on thread [15]
84  Timings for [Monitor.DatabaseConverter.Importer.
        DeliveryTermImporter]: total:0,0335065
85  Starting importer [Monitor.DatabaseConverter.Importer.
        PartCodeImporter] on thread [18]
86  Timings for [Monitor.DatabaseConverter.Importer.PartCodeImporter
        ]: total:0,0677158
87  Starting importer [Monitor.DatabaseConverter.Importer.
        PartStatusImporter] on thread [18]
88  Timings for [Monitor.DatabaseConverter.Importer.
        PartStatusImporter]: total:0,028055
89  Starting importer [Monitor.DatabaseConverter.Importer.
        PaymentTermImporter] on thread [18]
90  Timings for [Monitor.DatabaseConverter.Importer.
        CentralBankCodeImporter]: total:0,6939295
91  Starting importer [Monitor.DatabaseConverter.Importer.
        PrinterImporter] on thread [19]
92  Timings for [Monitor.DatabaseConverter.Importer.
        PaymentTermImporter]: total:0,0376307
93  Starting importer [Monitor.DatabaseConverter.Importer.
        ProbabilityImporter] on thread [18]
94  Timings for [Monitor.DatabaseConverter.Importer.PrinterImporter
        ]: total:0,0356477
95  Starting importer [Monitor.DatabaseConverter.Importer.
        ProductGroupImporter] on thread [15]
96  Timings for [Monitor.DatabaseConverter.Importer.
        ProbabilityImporter]: total:0,0664129
97  Starting importer [Monitor.DatabaseConverter.Importer.
        ProfitMarkupImporter] on thread [18]
98  Timings for [Monitor.DatabaseConverter.Importer.
        ProductGroupImporter]: total:0,0408566
99  Starting importer [Monitor.DatabaseConverter.Importer.
        SalesOverheadMarkupImporter] on thread [15]
100 Timings for [Monitor.DatabaseConverter.Importer.
        ProfitMarkupImporter]: total:0,023924
101 Timings for [Monitor.DatabaseConverter.Importer.
        SalesOverheadMarkupImporter]: total:0,0203186
```

```
102 Starting importer [Monitor.DatabaseConverter.Importer.
        StatisticalGoodsCodeImporter] on thread [15]
103 Starting importer [Monitor.DatabaseConverter.Importer.
        StorageOverheadMarkupImporter] on thread [19]
104 Timings for [Monitor.DatabaseConverter.Importer.
        StatisticalGoodsCodeImporter]: total:0,0342135
105 Starting importer [Monitor.DatabaseConverter.Importer.
        SupplierStatusImporter] on thread [18]
106 Timings for [Monitor.DatabaseConverter.Importer.
        StorageOverheadMarkupImporter]: total:0,0314486
107 Starting importer [Monitor.DatabaseConverter.Importer.
        UnitImporter] on thread [15]
108 Timings for [Monitor.DatabaseConverter.Importer.
        SupplierStatusImporter]: total:0,0376414
109 Starting importer [Monitor.DatabaseConverter.Importer.
        ValidityTimeImporter] on thread [19]
110 Timings for [Monitor.DatabaseConverter.Importer.CurrencyImporter
        ]: total:0,8321405
111 Starting importer [Monitor.DatabaseConverter.Importer.
        AccountImporter] on thread [20]
112 Timings for [Monitor.DatabaseConverter.Importer.
        ValidityTimeImporter]: total:0,561889
113 Starting importer [Monitor.DatabaseConverter.Importer.
        DiscountCategoryImporter] on thread [18]
114 Timings for [Monitor.DatabaseConverter.Importer.UnitImporter]:
        total:0,6292006
115 Starting importer [Monitor.DatabaseConverter.Importer.
        AbcCodeImporter] on thread [19]
116 Timings for [Monitor.DatabaseConverter.Importer.
        DiscountCategoryImporter]: total:0,0198417
117 Starting importer [Monitor.DatabaseConverter.Importer.
        CompanyImporter] on thread [15]
118 Timings for [Monitor.DatabaseConverter.Importer.AbcCodeImporter
        ]: total:0,1744571
119 Starting importer [Monitor.DatabaseConverter.Importer.
        PriceListImporter] on thread [19]
120 Timings for [Monitor.DatabaseConverter.Importer.
        PriceListImporter]: total:0,048653
121 Starting importer [Monitor.DatabaseConverter.Importer.
        ToolImporter] on thread [19]
122 Timings for [Monitor.DatabaseConverter.Importer.ToolImporter]:
        total:0,0729196
123 Timings for [Monitor.DatabaseConverter.Importer.CompanyImporter
        ]: total:0,3177283
124 Starting importer [Monitor.DatabaseConverter.Importer.
        ApplicationUserImporter] on thread [15]
125 Starting importer [Monitor.DatabaseConverter.Importer.
        ContactImporter] on thread [18]
126 Timings for [Monitor.DatabaseConverter.Importer.ContactImporter
        ]: total:0,0743794
127 Starting importer [Monitor.DatabaseConverter.Importer.
        CompanyCalendarRowImporter] on thread [18]
128 Timings for [Monitor.DatabaseConverter.Importer.
        ApplicationUserImporter]: total:0,3396757
129 Starting importer [Monitor.DatabaseConverter.Importer.
        ImageImporter] on thread [19]
```

```
130 Timings for [Monitor.DatabaseConverter.Importer.ImageImporter]:
        total:0,0235411
131 Starting importer [Monitor.DatabaseConverter.Importer.
        DepartmentImporter] on thread [15]
132 Timings for [Monitor.DatabaseConverter.Importer.
        DepartmentImporter]: total:0,0141387
133 Starting importer [Monitor.DatabaseConverter.Importer.
        PersonImporter] on thread [15]
134 Timings for [Monitor.DatabaseConverter.Importer.
        PackageTypeImporter]: total:2,2662809
135 Starting importer [Monitor.DatabaseConverter.Importer.
        AccountsReceivableLedgerImporter] on thread [14]
136 Timings for [Monitor.DatabaseConverter.Importer.
        AccountsReceivableLedgerImporter]: total:0,0917103
137 Timings for [Monitor.DatabaseConverter.Importer.PersonImporter]:
         total:0,2213971
138 Timings for [Monitor.DatabaseConverter.Importer.
        CompanyCalendarRowImporter]: total:1,1730148
139 Timings for [Monitor.DatabaseConverter.Importer.AccountImporter
        ]: total:31,636336
140 Starting importer [Monitor.DatabaseConverter.Importer.
        BankAccountImporter] on thread [15]
141 Starting importer [Monitor.DatabaseConverter.Importer.
        CustomerAccountGroupImporter] on thread [20]
142 Starting importer [Monitor.DatabaseConverter.Importer.
        PartImporter] on thread [21]
143 Starting importer [Monitor.DatabaseConverter.Importer.
        PaymentMethodImporter] on thread [6]
144 Timings for [Monitor.DatabaseConverter.Importer.
        CustomerAccountGroupImporter]: total:0,319988
145 Starting importer [Monitor.DatabaseConverter.Importer.
        SupplierAccountGroupImporter] on thread [20]
146 Timings for [Monitor.DatabaseConverter.Importer.
        BankAccountImporter]: total:0,352317
147 Starting importer [Monitor.DatabaseConverter.Importer.
        ProductGroupSalesCodingImporter] on thread [15]
148 Timings for [Monitor.DatabaseConverter.Importer.
        SupplierAccountGroupImporter]: total:0,1315311
149 Starting importer [Monitor.DatabaseConverter.Importer.
        ProductGroupPurchaseCodingImporter] on thread [20]
150 Timings for [Monitor.DatabaseConverter.Importer.
        PaymentMethodImporter]: total:0,6084076
151 Starting importer [Monitor.DatabaseConverter.Importer.
        CustomerImporter] on thread [6]
152 Timings for [Monitor.DatabaseConverter.Importer.
        ProductGroupSalesCodingImporter]: total:2,0949029
153 Timings for [Monitor.DatabaseConverter.Importer.
        ProductGroupPurchaseCodingImporter]: total:2,0276837
154 Timings for [Monitor.DatabaseConverter.Importer.CustomerImporter
        ]: total:4,2399098
155 Starting importer [Monitor.DatabaseConverter.Importer.
        AccountsReceivableImporter] on thread [6]
156 Starting importer [Monitor.DatabaseConverter.Importer.
        CustomerLatestImporter] on thread [20]
157 Starting importer [Monitor.DatabaseConverter.Importer.
        CustomerPaymentHistoryImporter] on thread [15]
```

```
158 Timings for [Monitor.DatabaseConverter.Importer.
        CustomerLatestImporter]: total:0,1119832
159 Timings for [Monitor.DatabaseConverter.Importer.
        CustomerPaymentHistoryImporter]: total:0,3498481
160 Timings for [Monitor.DatabaseConverter.Importer.
        AccountsReceivableImporter]: total:8,4967558
161 Timings for [Monitor.DatabaseConverter.Importer.PartImporter]:
        total:33,561146
162 Starting importer [Monitor.DatabaseConverter.Importer.
        CustomerPartLinkImporter] on thread [21]
163 Starting importer [Monitor.DatabaseConverter.Importer.
        InvoicingPlanImporter] on thread [6]
164 Starting importer [Monitor.DatabaseConverter.Importer.
        PartCalculationImporter] on thread [22]
165 Starting importer [Monitor.DatabaseConverter.Importer.
        PartLocationImporter] on thread [15]
166 Timings for [Monitor.DatabaseConverter.Importer.
        InvoicingPlanImporter]: total:0,6221507
167 Starting importer [Monitor.DatabaseConverter.Importer.
        PaymentPlanImporter] on thread [6]
168 Timings for [Monitor.DatabaseConverter.Importer.
        PaymentPlanImporter]: total:0,018348
169 Starting importer [Monitor.DatabaseConverter.Importer.
        SalesPriceImporter] on thread [6]
170 Timings for [Monitor.DatabaseConverter.Importer.
        CustomerPartLinkImporter]: total:1,2188653
171 Starting importer [Monitor.DatabaseConverter.Importer.
        SupplierImporter] on thread [21]
172 Timings for [Monitor.DatabaseConverter.Importer.
        SalesPriceImporter]: total:3,0102343
173 Timings for [Monitor.DatabaseConverter.Importer.SupplierImporter
        ]: total:2,8612688
174 Starting importer [Monitor.DatabaseConverter.Importer.
        AccountsPayableImporter] on thread [21]
175 Starting importer [Monitor.DatabaseConverter.Importer.
        PurchaseOrderImporter] on thread [14]
176 Timings for [Monitor.DatabaseConverter.Importer.
        AccountsPayableImporter]: total:2,3931493
177 Starting importer [Monitor.DatabaseConverter.Importer.
        SupplierPartLinkImporter] on thread [21]
178 Timings for [Monitor.DatabaseConverter.Importer.
        SupplierPartLinkImporter]: total:0,3298216
179 Starting importer [Monitor.DatabaseConverter.Importer.
        WorkCenterImporter] on thread [6]
180 Timings for [Monitor.DatabaseConverter.Importer.
        WorkCenterImporter]: total:0,6298892
181 Starting importer [Monitor.DatabaseConverter.Importer.
        PartPreparationImporter] on thread [21]
182 Timings for [Monitor.DatabaseConverter.Importer.
        PurchaseOrderImporter]: total:6,0691893
183 Timings for [Monitor.DatabaseConverter.Importer.
        PartLocationImporter]: total:16,3536446
184 Starting importer [Monitor.DatabaseConverter.Importer.
        InventoryLogImporter] on thread [15]
185 Timings for [Monitor.DatabaseConverter.Importer.
        InventoryLogImporter]: total:3,9148805
```

```
186 Starting importer [Monitor.DatabaseConverter.Importer.
        CustomerOrderImporter] on thread [15]
187 Timings for [Monitor.DatabaseConverter.Importer.
        CustomerOrderImporter]: total:17,2395779
188 Starting importer [Monitor.DatabaseConverter.Importer.
        ProductRecordImporter] on thread [15]
189 Starting importer [Monitor.DatabaseConverter.Importer.
        PickingListImporter] on thread [6]
190 Timings for [Monitor.DatabaseConverter.Importer.
        ProductRecordImporter]: total:0,0928669
191 Timings for [Monitor.DatabaseConverter.Importer.
        PartCalculationImporter]: total:42,5521217
192 Timings for [Monitor.DatabaseConverter.Importer.
        PickingListImporter]: total:15,5950144
193 Starting importer [Monitor.DatabaseConverter.Importer.
        CustomerOrderInvoiceImporter] on thread [14]
194 Starting importer [Monitor.DatabaseConverter.Importer.
        NumberSeriesImporter] on thread [6]
195 Timings for [Monitor.DatabaseConverter.Importer.
        NumberSeriesImporter]: total:0,0605773
196 Timings for [Monitor.DatabaseConverter.Importer.
        CustomerOrderInvoiceImporter]: total:4,316318
197 Starting importer [Monitor.DatabaseConverter.Importer.
        CustomerOrderInvoiceImporterPostProcessing] on thread [22]
198 Timings for [Monitor.DatabaseConverter.Importer.
        CustomerOrderInvoiceImporterPostProcessing]: total:0,6373027
199 Timings for [Monitor.DatabaseConverter.Importer.
        PartPreparationImporter]: total:51,4276375
200 Starting importer [Monitor.DatabaseConverter.Importer.
        ManufacturingOrderImporter] on thread [21]
201 Timings for [Monitor.DatabaseConverter.Importer.
        ManufacturingOrderImporter]: total:17,7671186
202 Total time: 210.7496748
```

Listing B.12: Körtider för konvertering till PostgreSQL

```
1  Test Name:  ConvertDemo3ToPgSql
2  Test Outcome: Passed
3  Result StandardOutput:
4  Starting importer [Monitor.Server.Pil.LanguageImporter] on
       thread [18]
5  Starting importer [Monitor.Server.Pil.MenuImporter] on thread
       [19]
6  Starting importer [Monitor.Server.Pil.
       FormReportBuildingBlockImporter] on thread [15]
7  Starting importer [Monitor.Server.Pil.
       FormReportConfigurationImporter] on thread [14]
8  Timings for [Monitor.Server.Pil.LanguageImporter]: total
       :0,0782744
9  Starting importer [Monitor.Server.Pil.NumberSeriesImporter] on
       thread [18]
10 Timings for [Monitor.Server.Pil.FormReportConfigurationImporter
       ]: total:0,1441496
11 Timings for [Monitor.Server.Pil.FormReportBuildingBlockImporter
       ]: total:0,1445752
12 Starting importer [Monitor.Server.Pil.PostalCodeImporter] on
       thread [14]
```

```
13 Timings for [Monitor.Server.Pil.NumberSeriesImporter]: total
       :0,1051617
14 Starting importer [Monitor.Server.Pil.CalendarImporter] on
       thread [15]
15 Starting importer [Monitor.Server.Pil.CodingDimesionImporter] on
        thread [18]
16 Timings for [Monitor.Server.Pil.CalendarImporter]: total
       :0,0756186
17 Starting importer [Monitor.Server.Pil.
       CurrencyExchangeTypeImporter] on thread [20]
18 Timings for [Monitor.Server.Pil.CodingDimesionImporter]: total
       :0,0769741
19 Starting importer [Monitor.Server.Pil.FormReportSettingImporter]
        on thread [18]
20 Timings for [Monitor.Server.Pil.CurrencyExchangeTypeImporter]:
       total:0,034823
21 Starting importer [Monitor.Server.Pil.CustomerOrderTypeImporter]
        on thread [20]
22 Timings for [Monitor.Server.Pil.CustomerOrderTypeImporter]:
       total:0,0303507
23 Starting importer [Monitor.Server.Pil.
       FormReportTranslationGroupImporter] on thread [20]
24 Timings for [Monitor.Server.Pil.FormReportSettingImporter]:
       total:0,278257
25 Starting importer [Monitor.Server.Pil.PaymentFormatImporter] on
       thread [15]
26 Timings for [Monitor.Server.Pil.PaymentFormatImporter]: total
       :0,0145523
27 Starting importer [Monitor.Server.Pil.PurchaseOrderTypeImporter]
        on thread [18]
28 Timings for [Monitor.Server.Pil.PurchaseOrderTypeImporter]:
       total:0,0237854
29 Starting importer [Monitor.Server.Pil.RoleImporter] on thread
       [18]
30 Timings for [Monitor.Server.Pil.MenuImporter]: total:0,8114207
31 Timings for [Monitor.Server.Pil.RoleImporter]: total:0,2716348
32 Starting importer [Monitor.Server.Pil.ScheduledServiceImporter]
       on thread [15]
33 Starting importer [Monitor.Server.Pil.CurrencyImporter] on
       thread [18]
34 Timings for [Monitor.Server.Pil.CurrencyImporter]: total
       :0,0790982
35 Starting importer [Monitor.Server.Pil.ApplicationUserImporter]
       on thread [19]
36 Timings for [Monitor.Server.Pil.
       FormReportTranslationGroupImporter]: total:0,694539
37 Timings for [Monitor.Server.Pil.ApplicationUserImporter]: total
       :0,0303045
38 Starting importer [Monitor.Server.Pil.CountryImporter] on thread
        [19]
39 Starting importer [Monitor.Server.Pil.SystemParameterImporter]
       on thread [20]
40 Timings for [Monitor.Server.Pil.SystemParameterImporter]: total
       :0,1390917
41 Timings for [Monitor.Server.Pil.ScheduledServiceImporter]: total
       :0,5768421
```

```
42 Timings for [Monitor.Server.Pil.CountryImporter]: total
       :1,1141724
43 Timings for [Monitor.Server.Pil.PostalCodeImporter]: total
       :13,602927
44 Starting importer [Monitor.DatabaseConverter.Importer.
       CategoryImporter] on thread [18]
45 Starting importer [Monitor.DatabaseConverter.Importer.
       AccountingYearImporter] on thread [20]
46 Starting importer [Monitor.DatabaseConverter.Importer.
       CodingElementImporter] on thread [19]
47 Starting importer [Monitor.DatabaseConverter.Importer.
       BudgetChartImporter] on thread [15]
48 Timings for [Monitor.DatabaseConverter.Importer.
       BudgetChartImporter]: total:0,1888414
49 Starting importer [Monitor.DatabaseConverter.Importer.
       CentralBankCodeImporter] on thread [15]
50 Timings for [Monitor.DatabaseConverter.Importer.CategoryImporter
       ]: total:0,210995
51 Starting importer [Monitor.DatabaseConverter.Importer.
       GoodsTypeImporter] on thread [6]
52 Timings for [Monitor.DatabaseConverter.Importer.
       CodingElementImporter]: total:0,2373545
53 Timings for [Monitor.DatabaseConverter.Importer.
       AccountingYearImporter]: total:0,266721
54 Timings for [Monitor.DatabaseConverter.Importer.
       GoodsTypeImporter]: total:0,0548277
55 Starting importer [Monitor.DatabaseConverter.Importer.
       PackageTypeImporter] on thread [14]
56 Starting importer [Monitor.DatabaseConverter.Importer.
       DeliveryTimeImporter] on thread [6]
57 Starting importer [Monitor.DatabaseConverter.Importer.
       InterruptionCodeImporter] on thread [19]
58 Timings for [Monitor.DatabaseConverter.Importer.
       DeliveryTimeImporter]: total:0,0883719
59 Starting importer [Monitor.DatabaseConverter.Importer.
       LocationTypeImporter] on thread [6]
60 Timings for [Monitor.DatabaseConverter.Importer.
       InterruptionCodeImporter]: total:0,0931073
61 Starting importer [Monitor.DatabaseConverter.Importer.
       PackingTermImporter] on thread [19]
62 Timings for [Monitor.DatabaseConverter.Importer.
       LocationTypeImporter]: total:0,0383283
63 Starting importer [Monitor.DatabaseConverter.Importer.
       ProjectImporter] on thread [6]
64 Timings for [Monitor.DatabaseConverter.Importer.
       PackingTermImporter]: total:0,0412104
65 Starting importer [Monitor.DatabaseConverter.Importer.
       RejectionCodeItemImporter] on thread [20]
66 Timings for [Monitor.DatabaseConverter.Importer.ProjectImporter
       ]: total:0,0673796
67 Starting importer [Monitor.DatabaseConverter.Importer.
       ResponseTimeImporter] on thread [6]
68 Timings for [Monitor.DatabaseConverter.Importer.
       ResponseTimeImporter]: total:0,0156901
69 Starting importer [Monitor.DatabaseConverter.Importer.
       LanguageSettingImporter] on thread [19]
```

```
70 Timings for [Monitor.DatabaseConverter.Importer.
       RejectionCodeItemImporter]: total:0,0984172
71 Starting importer [Monitor.DatabaseConverter.Importer.
       CurrencyImporter] on thread [6]
72 Timings for [Monitor.DatabaseConverter.Importer.
       LanguageSettingImporter]: total:0,0321497
73 Starting importer [Monitor.DatabaseConverter.Importer.
       CustomerDistrictImporter] on thread [19]
74 Timings for [Monitor.DatabaseConverter.Importer.
       CustomerDistrictImporter]: total:0,0527772
75 Starting importer [Monitor.DatabaseConverter.Importer.
       CustomerRelationshipActivityImporter] on thread [20]
76 Timings for [Monitor.DatabaseConverter.Importer.
       CustomerRelationshipActivityImporter]: total:0,061319
77 Starting importer [Monitor.DatabaseConverter.Importer.
       CustomerStatusImporter] on thread [20]
78 Timings for [Monitor.DatabaseConverter.Importer.
       CustomerStatusImporter]: total:0,0494481
79 Starting importer [Monitor.DatabaseConverter.Importer.
       CustomerTypeImporter] on thread [19]
80 Timings for [Monitor.DatabaseConverter.Importer.
       CustomerTypeImporter]: total:0,0486363
81 Starting importer [Monitor.DatabaseConverter.Importer.
       DeliveryMethodImporter] on thread [19]
82 Timings for [Monitor.DatabaseConverter.Importer.
       DeliveryMethodImporter]: total:0,1119772
83 Starting importer [Monitor.DatabaseConverter.Importer.
       DeliveryTermImporter] on thread [20]
84 Timings for [Monitor.DatabaseConverter.Importer.
       DeliveryTermImporter]: total:0,0751776
85 Starting importer [Monitor.DatabaseConverter.Importer.
       PartCodeImporter] on thread [20]
86 Timings for [Monitor.DatabaseConverter.Importer.PartCodeImporter
       ]: total:0,155355
87 Starting importer [Monitor.DatabaseConverter.Importer.
       PartStatusImporter] on thread [19]
88 Timings for [Monitor.DatabaseConverter.Importer.
       PartStatusImporter]: total:0,0557148
89 Starting importer [Monitor.DatabaseConverter.Importer.
       PaymentTermImporter] on thread [19]
90 Timings for [Monitor.DatabaseConverter.Importer.
       PaymentTermImporter]: total:0,0825098
91 Starting importer [Monitor.DatabaseConverter.Importer.
       PrinterImporter] on thread [19]
92 Timings for [Monitor.DatabaseConverter.Importer.PrinterImporter
       ]: total:0,0460262
93 Starting importer [Monitor.DatabaseConverter.Importer.
       ProbabilityImporter] on thread [20]
94 Timings for [Monitor.DatabaseConverter.Importer.CurrencyImporter
       ]: total:0,92523
95 Starting importer [Monitor.DatabaseConverter.Importer.
       ProductGroupImporter] on thread [19]
96 Timings for [Monitor.DatabaseConverter.Importer.
       CentralBankCodeImporter]: total:1,307532
97 Timings for [Monitor.DatabaseConverter.Importer.
       ProbabilityImporter]: total:0,1439212
```

```
 98 Starting importer [Monitor.DatabaseConverter.Importer.
        ProfitMarkupImporter] on thread [20]
 99 Timings for [Monitor.DatabaseConverter.Importer.
        ProductGroupImporter]: total:0,0784497
100 Starting importer [Monitor.DatabaseConverter.Importer.
        SalesOverheadMarkupImporter] on thread [6]
101 Timings for [Monitor.DatabaseConverter.Importer.
        ProfitMarkupImporter]: total:0,0523229
102 Timings for [Monitor.DatabaseConverter.Importer.
        SalesOverheadMarkupImporter]: total:0,0312202
103 Starting importer [Monitor.DatabaseConverter.Importer.
        StatisticalGoodsCodeImporter] on thread [6]
104 Starting importer [Monitor.DatabaseConverter.Importer.
        StorageOverheadMarkupImporter] on thread [20]
105 Timings for [Monitor.DatabaseConverter.Importer.
        StatisticalGoodsCodeImporter]: total:0,046528
106 Starting importer [Monitor.DatabaseConverter.Importer.
        SupplierStatusImporter] on thread [6]
107 Timings for [Monitor.DatabaseConverter.Importer.
        StorageOverheadMarkupImporter]: total:0,0515508
108 Starting importer [Monitor.DatabaseConverter.Importer.
        UnitImporter] on thread [20]
109 Starting importer [Monitor.DatabaseConverter.Importer.
        ValidityTimeImporter] on thread [15]
110 Timings for [Monitor.DatabaseConverter.Importer.
        SupplierStatusImporter]: total:0,0527793
111 Starting importer [Monitor.DatabaseConverter.Importer.
        AccountImporter] on thread [6]
112 Timings for [Monitor.DatabaseConverter.Importer.
        ValidityTimeImporter]: total:0,0339432
113 Starting importer [Monitor.DatabaseConverter.Importer.
        AbcCodeImporter] on thread [15]
114 Timings for [Monitor.DatabaseConverter.Importer.AbcCodeImporter
        ]: total:0,3362855
115 Starting importer [Monitor.DatabaseConverter.Importer.
        CompanyImporter] on thread [15]
116 Timings for [Monitor.DatabaseConverter.Importer.UnitImporter]:
        total:0,4449842
117 Starting importer [Monitor.DatabaseConverter.Importer.
        PriceListImporter] on thread [19]
118 Timings for [Monitor.DatabaseConverter.Importer.
        PriceListImporter]: total:0,1048345
119 Starting importer [Monitor.DatabaseConverter.Importer.
        DiscountCategoryImporter] on thread [19]
120 Timings for [Monitor.DatabaseConverter.Importer.
        DiscountCategoryImporter]: total:0,0365887
121 Starting importer [Monitor.DatabaseConverter.Importer.
        ToolImporter] on thread [20]
122 Timings for [Monitor.DatabaseConverter.Importer.ToolImporter]:
        total:0,1078629
123 Timings for [Monitor.DatabaseConverter.Importer.CompanyImporter
        ]: total:0,3678249
124 Starting importer [Monitor.DatabaseConverter.Importer.
        ApplicationUserImporter] on thread [15]
125 Starting importer [Monitor.DatabaseConverter.Importer.
        ContactImporter] on thread [19]
```

```
126 Timings for [Monitor.DatabaseConverter.Importer.ContactImporter
        ]: total:0,0950535
127 Starting importer [Monitor.DatabaseConverter.Importer.
        CompanyCalendarRowImporter] on thread [19]
128 Timings for [Monitor.DatabaseConverter.Importer.
        ApplicationUserImporter]: total:0,6234287
129 Starting importer [Monitor.DatabaseConverter.Importer.
        ImageImporter] on thread [20]
130 Timings for [Monitor.DatabaseConverter.Importer.ImageImporter]:
        total:0,0271867
131 Starting importer [Monitor.DatabaseConverter.Importer.
        DepartmentImporter] on thread [15]
132 Timings for [Monitor.DatabaseConverter.Importer.
        DepartmentImporter]: total:0,0371136
133 Starting importer [Monitor.DatabaseConverter.Importer.
        PersonImporter] on thread [20]
134 Timings for [Monitor.DatabaseConverter.Importer.PersonImporter]:
         total:0,3679348
135 Starting importer [Monitor.DatabaseConverter.Importer.
        AccountsReceivableLedgerImporter] on thread [20]
136 Timings for [Monitor.DatabaseConverter.Importer.
        AccountsReceivableLedgerImporter]: total:0,1090194
137 Timings for [Monitor.DatabaseConverter.Importer.
        PackageTypeImporter]: total:4,6466376
138 Timings for [Monitor.DatabaseConverter.Importer.
        CompanyCalendarRowImporter]: total:3,2938924
139 Timings for [Monitor.DatabaseConverter.Importer.AccountImporter
        ]: total:67,9108273
140 Starting importer [Monitor.DatabaseConverter.Importer.
        BankAccountImporter] on thread [20]
141 Starting importer [Monitor.DatabaseConverter.Importer.
        CustomerAccountGroupImporter] on thread [6]
142 Starting importer [Monitor.DatabaseConverter.Importer.
        PartImporter] on thread [14]
143 Starting importer [Monitor.DatabaseConverter.Importer.
        PaymentMethodImporter] on thread [19]
144 Timings for [Monitor.DatabaseConverter.Importer.
        CustomerAccountGroupImporter]: total:0,1132694
145 Starting importer [Monitor.DatabaseConverter.Importer.
        SupplierAccountGroupImporter] on thread [6]
146 Timings for [Monitor.DatabaseConverter.Importer.
        SupplierAccountGroupImporter]: total:0,2848035
147 Starting importer [Monitor.DatabaseConverter.Importer.
        ProductGroupSalesCodingImporter] on thread [23]
148 Timings for [Monitor.DatabaseConverter.Importer.
        BankAccountImporter]: total:0,4582871
149 Starting importer [Monitor.DatabaseConverter.Importer.
        ProductGroupPurchaseCodingImporter] on thread [20]
150 Timings for [Monitor.DatabaseConverter.Importer.
        PaymentMethodImporter]: total:0,9794585
151 Starting importer [Monitor.DatabaseConverter.Importer.
        CustomerImporter] on thread [21]
152 Timings for [Monitor.DatabaseConverter.Importer.
        ProductGroupPurchaseCodingImporter]: total:0,7584211
153 Timings for [Monitor.DatabaseConverter.Importer.
        ProductGroupSalesCodingImporter]: total:2,707078
```

```
154 Timings for [Monitor.DatabaseConverter.Importer.CustomerImporter
        ]: total:12,2371531
155 Starting importer [Monitor.DatabaseConverter.Importer.
        CustomerLatestImporter] on thread [21]
156 Starting importer [Monitor.DatabaseConverter.Importer.
        AccountsReceivableImporter] on thread [9]
157 Starting importer [Monitor.DatabaseConverter.Importer.
        CustomerPaymentHistoryImporter] on thread [20]
158 Timings for [Monitor.DatabaseConverter.Importer.
        CustomerLatestImporter]: total:0,166651
159 Timings for [Monitor.DatabaseConverter.Importer.
        CustomerPaymentHistoryImporter]: total:0,4765095
160 Timings for [Monitor.DatabaseConverter.Importer.
        AccountsReceivableImporter]: total:24,6218337
161 Timings for [Monitor.DatabaseConverter.Importer.PartImporter]:
        total:112,1555676
162 Starting importer [Monitor.DatabaseConverter.Importer.
        CustomerPartLinkImporter] on thread [9]
163 Starting importer [Monitor.DatabaseConverter.Importer.
        InvoicingPlanImporter] on thread [6]
164 Starting importer [Monitor.DatabaseConverter.Importer.
        PartLocationImporter] on thread [14]
165 Starting importer [Monitor.DatabaseConverter.Importer.
        PartCalculationImporter] on thread [20]
166 Timings for [Monitor.DatabaseConverter.Importer.
        InvoicingPlanImporter]: total:0,6258343
167 Starting importer [Monitor.DatabaseConverter.Importer.
        PaymentPlanImporter] on thread [17]
168 Timings for [Monitor.DatabaseConverter.Importer.
        PaymentPlanImporter]: total:0,0355613
169 Starting importer [Monitor.DatabaseConverter.Importer.
        SalesPriceImporter] on thread [6]
170 Timings for [Monitor.DatabaseConverter.Importer.
        CustomerPartLinkImporter]: total:1,8109425
171 Starting importer [Monitor.DatabaseConverter.Importer.
        SupplierImporter] on thread [9]
172 Timings for [Monitor.DatabaseConverter.Importer.
        SalesPriceImporter]: total:6,463849
173 Timings for [Monitor.DatabaseConverter.Importer.SupplierImporter
        ]: total:9,1110225
174 Starting importer [Monitor.DatabaseConverter.Importer.
        AccountsPayableImporter] on thread [9]
175 Starting importer [Monitor.DatabaseConverter.Importer.
        PurchaseOrderImporter] on thread [22]
176 Timings for [Monitor.DatabaseConverter.Importer.
        AccountsPayableImporter]: total:6,3791218
177 Starting importer [Monitor.DatabaseConverter.Importer.
        SupplierPartLinkImporter] on thread [9]
178 Timings for [Monitor.DatabaseConverter.Importer.
        SupplierPartLinkImporter]: total:0,6114817
179 Starting importer [Monitor.DatabaseConverter.Importer.
        WorkCenterImporter] on thread [9]
180 Timings for [Monitor.DatabaseConverter.Importer.
        WorkCenterImporter]: total:1,9965661
181 Starting importer [Monitor.DatabaseConverter.Importer.
        PartPreparationImporter] on thread [9]
```

```
182 Timings for [Monitor.DatabaseConverter.Importer.
        PurchaseOrderImporter]: total:12,7557653
183 Timings for [Monitor.DatabaseConverter.Importer.
        PartLocationImporter]: total:49,5602606
184 Starting importer [Monitor.DatabaseConverter.Importer.
        InventoryLogImporter] on thread [14]
185 Timings for [Monitor.DatabaseConverter.Importer.
        InventoryLogImporter]: total:6,3601248
186 Starting importer [Monitor.DatabaseConverter.Importer.
        CustomerOrderImporter] on thread [14]
187 Timings for [Monitor.DatabaseConverter.Importer.
        CustomerOrderImporter]: total:57,7306038
188 Timings for [Monitor.DatabaseConverter.Importer.
        PartPreparationImporter]: total:163,4664906
189 Starting importer [Monitor.DatabaseConverter.Importer.
        ProductRecordImporter] on thread [14]
190 Starting importer [Monitor.DatabaseConverter.Importer.
        PickingListImporter] on thread [6]
191 Timings for [Monitor.DatabaseConverter.Importer.
        ProductRecordImporter]: total:0,203659
192 Timings for [Monitor.DatabaseConverter.Importer.
        PickingListImporter]: total:39,0780618
193 Starting importer [Monitor.DatabaseConverter.Importer.
        CustomerOrderInvoiceImporter] on thread [6]
194 Starting importer [Monitor.DatabaseConverter.Importer.
        NumberSeriesImporter] on thread [21]
195 Timings for [Monitor.DatabaseConverter.Importer.
        NumberSeriesImporter]: total:0,0846895
196 Timings for [Monitor.DatabaseConverter.Importer.
        CustomerOrderInvoiceImporter]: total:14,0208113
197 Starting importer [Monitor.DatabaseConverter.Importer.
        CustomerOrderInvoiceImporterPostProcessing] on thread [21]
198 Timings for [Monitor.DatabaseConverter.Importer.
        CustomerOrderInvoiceImporterPostProcessing]: total:1,4406701
199 Timings for [Monitor.DatabaseConverter.Importer.
        PartCalculationImporter]: total:461,5757133
200 Starting importer [Monitor.DatabaseConverter.Importer.
        ManufacturingOrderImporter] on thread [20]
201 Timings for [Monitor.DatabaseConverter.Importer.
        ManufacturingOrderImporter]: total:109,4361079
202 Total time: 2188.2353878
```

Bilaga C

Tidstest

C.1 Kod

Listing C.1: Person.cs

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6  using System.Runtime.Serialization;
7  using NHibernate.Mapping.ByCode.Conformist;
8  using NHibernate.Mapping.ByCode;
9
10 namespace NHibernateSpeedTest
11 {
12   public class Person
13   {
14     public virtual int Id { get; set; }
15
16     public virtual string name { get; set; }
17   }
18
19   public class PersonMap : ClassMapping<Person>
20   {
21     public PersonMap()
22     {
23       Id(x => x.Id, map => { map.Column("id"); });
24       Id(x => x.Id, map => { map.Generator(Generators.Assigned);
           });
25       Property(x => x.name, map => { map.Length(30); });
26     }
27   }
28 }
```

Listing C.2: Program.cs

```
1  using NHibernate;
```

```
2  using NHibernate.Cfg;
3  using NHibernate.Cfg.MappingSchema;
4  using NHibernate.Mapping.ByCode;
5  using NHibernate.Tool.hbm2ddl;
6  using Npgsql;
7  using NpgsqlTypes;
8  using System;
9  using System.Collections.Generic;
10 using System.Diagnostics;
11 using System.Linq;
12 using System.Reflection;
13 using System.Text;
14 using System.Threading.Tasks;
15 using System.Xml.Serialization;
16
17 namespace NHibernateSpeedTest
18 {
19   class Program
20   {
21     private static Configuration initPostgreSQL()
22     {
23       var config = new Configuration();
24
25       config.SessionFactoryName("buildit");
26
27       config.DataBaseIntegration(db =>
28       {
29         db.Dialect<NHibernate.Dialect.PostgreSQLDialect>();
30         db.Driver<NHibernate.Driver.NpgsqlDriver>();
31         db.KeywordsAutoImport = Hbm2DDLKeyWords.AutoQuote;
32
33         db.ConnectionString = "Server=localhost;database=test;
               User ID=postgres;Password=H3j";
34         db.Timeout = 10;
35
36       });
37
38       var mapping = GetMappings();
39       config.AddDeserializedMapping(mapping, null);
40       SchemaMetadataUpdater.QuoteTableAndColumns(config);
41       return config;
42     }
43
44     private static Configuration initSQLAnywhere()
45     {
46       var config = new Configuration();
47
48       config.SessionFactoryName("buildit");
49
50       config.DataBaseIntegration(db =>
51       {
52         db.Dialect<NHibernate.Dialect.SybaseSQLAnywhere12Dialect
               >();
53         db.KeywordsAutoImport = Hbm2DDLKeyWords.AutoQuote;
54
55         db.ConnectionString = @"ENG=MonitorInstall;UID=DBA;PWD=
               sql;START=dbeng12 -n MonitorInstall;DBN=monitor;DBF=
```

```
                c:\temp\speedTest\monitor.db;UID=DBA;PWD=sql";
56            db.Timeout = 10;
57
58          });
59
60          var mapping = GetMappings();
61          config.AddDeserializedMapping(mapping, null);
62          SchemaMetadataUpdater.QuoteTableAndColumns(config);
63          return config;
64        }
65
66        private static HbmMapping GetMappings()
67        {
68          var mapper = new ModelMapper();
69
70          mapper.AddMappings(Assembly.GetAssembly(typeof(PersonMap))
                .GetExportedTypes());
71          var mapping = mapper.
                CompileMappingForAllExplicitlyAddedEntities();
72
73          return mapping;
74        }
75
76        private static double CreatePersons(ISession session, int
              start, int end)
77        {
78          var sw = new Stopwatch();
79
80          sw.Start();
81          using (var transaction = session.BeginTransaction())
82          {
83            Person person;
84            for (int i = start; i < end; i++)
85            {
86              person = new Person { Id = i, name = "test" + i };
87              session.Save(person);
88            }
89            transaction.Commit();
90          }
91          sw.Stop();
92          return sw.Elapsed.TotalSeconds;
93        }
94
95        private static void AddPersons(ISessionFactory anywhere,
              ISessionFactory pgsql)
96        {
97          double[,] times = new double[4, 2];
98
99          int amount = 1000;
100
101         times[0, 0] = CreatePersons(anywhere.OpenSession(), 1,
                amount);
102         times[1, 0] = CreatePersons(anywhere.OpenSession(), amount
                , amount * 10);
103         times[2, 0] = CreatePersons(anywhere.OpenSession(), amount
                 * 10, amount * 100);
```

```
104         times[3, 0] = CreatePersons(anywhere.OpenSession(), amount
                * 100, amount * 1000);
105
106         times[0, 1] = CreatePersons(pgsql.OpenSession(), 1, amount
                );
107         times[1, 1] = CreatePersons(pgsql.OpenSession(), amount,
                amount * 10);
108         times[2, 1] = CreatePersons(pgsql.OpenSession(), amount *
                10, amount * 100);
109         times[3, 1] = CreatePersons(pgsql.OpenSession(), amount *
                100, amount * 1000);
110
111         Console.WriteLine("#Inserts\tSQL Anywhere\tPostgreSQL");
112         for (int i = 0; i < 4; i++)
113         {
114           Console.WriteLine("<{2}\t{0}\t{1}", times[i, 0], times[i
                , 1], "100" + new String('0', i + 1));
115         }
116       }
117
118       private static void InsertTest()
119       {
120         var sw = new Stopwatch();
121
122         // Create databases and table
123         sw.Start();
124         var anyCreator = new SqlAnywhereCreator();
125         anyCreator.Create();
126         var anyConfig = initSQLAnywhere();
127         anyConfig.AddAssembly(typeof(Person).Assembly);
128         new SchemaExport(anyConfig).Execute(false, true, false);
129         sw.Stop();
130         Console.WriteLine("SQLAnywhere database creation time: {0}
                ", sw.Elapsed.TotalSeconds);
131
132         sw.Reset();
133         sw.Start();
134         var pgCreator = new PostgreSQLCreator();
135         pgCreator.Create();
136         var pgConfig = initPostgreSQL();
137         pgConfig.AddAssembly(typeof(Person).Assembly);
138         new SchemaExport(pgConfig).Execute(false, true, false);
139         sw.Stop();
140         Console.WriteLine("PostgreSQL database creation time: {0}"
                , sw.Elapsed.TotalSeconds);
141
142         var anySession = anyConfig.BuildSessionFactory();
143         var pgSession = pgConfig.BuildSessionFactory();
144
145         AddPersons(anySession, pgSession);
146       }
147
148       private const string connectString = @"Server=localhost;Port
              =5432;Database=test;User Id=postgres;Password=H3j;";
149
150       private static double CreatePersonsConcat(int start, int end
              )
```

```
151     {
152       var sw = new Stopwatch();
153       sw.Start();
154       using (var connection = new NpgsqlConnection(connectString
              ))
155       {
156         connection.Open();
157         var cmd = connection.CreateCommand();
158         var transaction = connection.BeginTransaction();
159         cmd.Transaction = transaction;
160
161         for (int i = start; i < end; i++)
162         {
163           cmd.CommandText = String.Format("insert into person (
                id, name) values ({0}, '{1}');", i, "test" + i);
164           cmd.ExecuteNonQuery();
165         }
166
167         transaction.Commit();
168       }
169       sw.Stop();
170       return sw.Elapsed.TotalSeconds;
171     }
172
173     private static double CreatePersonsPrepared(int start, int
          end)
174     {
175       var sw = new Stopwatch();
176       sw.Start();
177       using (var connection = new NpgsqlConnection(connectString
              ))
178       {
179         connection.Open();
180         var cmd = connection.CreateCommand();
181         var transaction = connection.BeginTransaction();
182         cmd.Transaction = transaction;
183         cmd.CommandText = "insert into person (id, name) values
              (@id, @name);";
184         cmd.Parameters.Add("id", NpgsqlDbType.Integer);
185         cmd.Parameters.Add("name", NpgsqlDbType.Varchar);
186         cmd.Prepare();
187         for (int i = start; i < end; i++)
188         {
189           cmd.Parameters["id"].Value = i;
190           cmd.Parameters["name"].Value = "test" + i;
191           cmd.ExecuteNonQuery();
192         }
193
194         transaction.Commit();
195       }
196       sw.Stop();
197       return sw.Elapsed.TotalSeconds;
198     }
199
200     private static double CreatePersonsCopy(int start, int end)
201     {
202       var sw = new Stopwatch();
```

```
203         sw.Start();
204         using (var connection = new NpgsqlConnection(connectString
                ))
205         {
206           connection.Open();
207           var cmd = connection.CreateCommand();
208           var transaction = connection.BeginTransaction();
209           cmd.Transaction = transaction;
210           cmd.CommandText = "COPY person(id, name) FROM STDIN;";
211           NpgsqlCopySerializer serializer = new
                  NpgsqlCopySerializer(connection);
212           NpgsqlCopyIn copyIn = new NpgsqlCopyIn(cmd, connection,
                  serializer.ToStream);
213           try
214           {
215             copyIn.Start();
216             for (int i = start; i < end; i++)
217             {
218               serializer.AddInt32(i);
219               serializer.AddString("text" + i);
220               serializer.EndRow();
221               serializer.Flush();
222             }
223
224             copyIn.End();
225             serializer.Close();
226           }
227           catch (Exception e)
228           {
229             connection.Close();
230             throw e;
231           }
232
233           transaction.Commit();
234         }
235         sw.Stop();
236         return sw.Elapsed.TotalSeconds;
237       }
238
239       private static double[,] AddPersonsNpgsql(int amount)
240       {
241         double[,] times = new double[1, 3];
242         times[0, 0] = CreatePersonsConcat(amount * 1, amount * 2);
243         times[0, 1] = CreatePersonsPrepared(amount * 2, amount *
                3);
244         times[0, 2] = CreatePersonsCopy(amount * 3, amount * 4);
245         return times;
246       }
247
248       private static void NpgsqlInsertTests()
249       {
250         var pgCreator = new PostgreSQLCreator();
251         pgCreator.Create();
252         var pgConfig = initPostgreSQL();
253         pgConfig.AddAssembly(typeof(Person).Assembly);
254         new SchemaExport(pgConfig).Execute(false, true, false);
255
```

```
256         double[][,] tests = new double[4][,];
257         tests[0] = AddPersonsNpgsql(1000);
258         tests[1] = AddPersonsNpgsql(10000);
259         tests[2] = AddPersonsNpgsql(100000);
260         tests[3] = AddPersonsNpgsql(1000000);
261         Console.WriteLine("#Inserts\tConcat\tPrepared\tCopy");
262         for (int i = 0; i < 4; i++)
263         {
264           Console.WriteLine("{3}\t{0}\t{1}\t{2}", tests[i][0, 0],
                  tests[i][0, 1], tests[i][0, 2], "100" + new String
                  ('0', i + 1));
265         }
266       }
267
268       static void Main(string[] args)
269       {
270         NpgsqlInsertTests();
271
272         //InsertTest();
273       }
274     }
275   }
```

Listing C.3: PostgreSQLCreator.cs

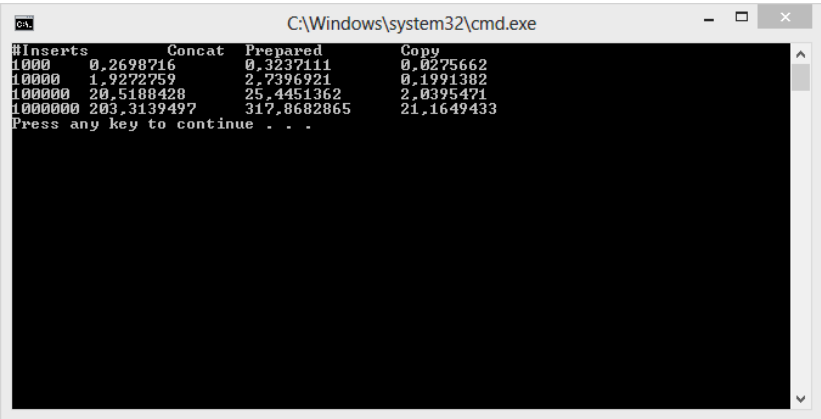
```
1   using System;
2   using System.Collections.Generic;
3   using System.Linq;
4   using System.Text;
5   using System.Threading.Tasks;
6   using Npgsql;
7
8   namespace NHibernateSpeedTest
9   {
10    public class PostgreSQLCreator
11    {
12      public void Create()
13      {
14        const string baseConnectionString = @"Server=localhost;
              Port=5432;Database=postgres;User Id=postgres;Password=
              H3j;";
15
16        using (var connection = new NpgsqlConnection(
              baseConnectionString))
17        {
18          connection.Open();
19          var command = connection.CreateCommand();
20
21          command.CommandText = String.Format(@"DROP DATABASE IF
                EXISTS test");
22          command.ExecuteNonQuery();
23
24          command.CommandText = "create database test";
25          command.ExecuteNonQuery();
26
27          connection.Close();
28        }
```

```
29      }
30    }
31  }
```

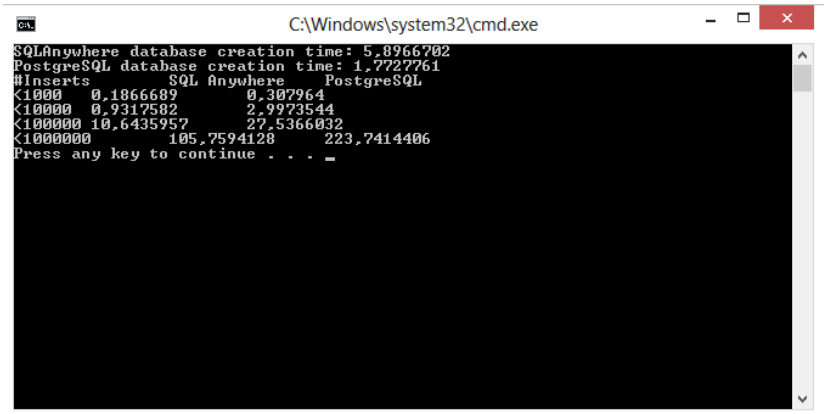
Listing C.4: SqlAnywhereCreator.cs

```
1  using System;
2  using System.Collections.Generic;
3  using System.IO;
4  using System.Linq;
5  using System.Text;
6  using System.Threading.Tasks;
7  using iAnywhere.Data.SQLAnywhere;
8
9  namespace NHibernateSpeedTest
10 {
11   public class SqlAnywhereCreator
12   {
13     private const string CreateDb = @"create database '{0}'";
14
15     public void Create()
16     {
17       var path = @"C:\temp\speedTest";
18       string databaseFullName = string.Format(@"{0}\monitor.db",
             path);
19
20       if (!Directory.Exists(path))
21       {
22         Directory.CreateDirectory(path);
23       }
24
25       const string baseConnectionString = "ENG=MonitorInstall;
             UID=DBA;PWD=sql;START=dbeng12 -n MonitorInstall;";
26
27       using (var myConnection = new SAConnection(string.Format("
             {0}DBN=utility_db", baseConnectionString)))
28       {
29         myConnection.Open();
30         var createCommand = myConnection.CreateCommand();
31         createCommand.CommandText = string.Format(CreateDb,
               databaseFullName);
32         createCommand.ExecuteNonQuery();
33         myConnection.Close();
34       }
35     }
36   }
37 }
```

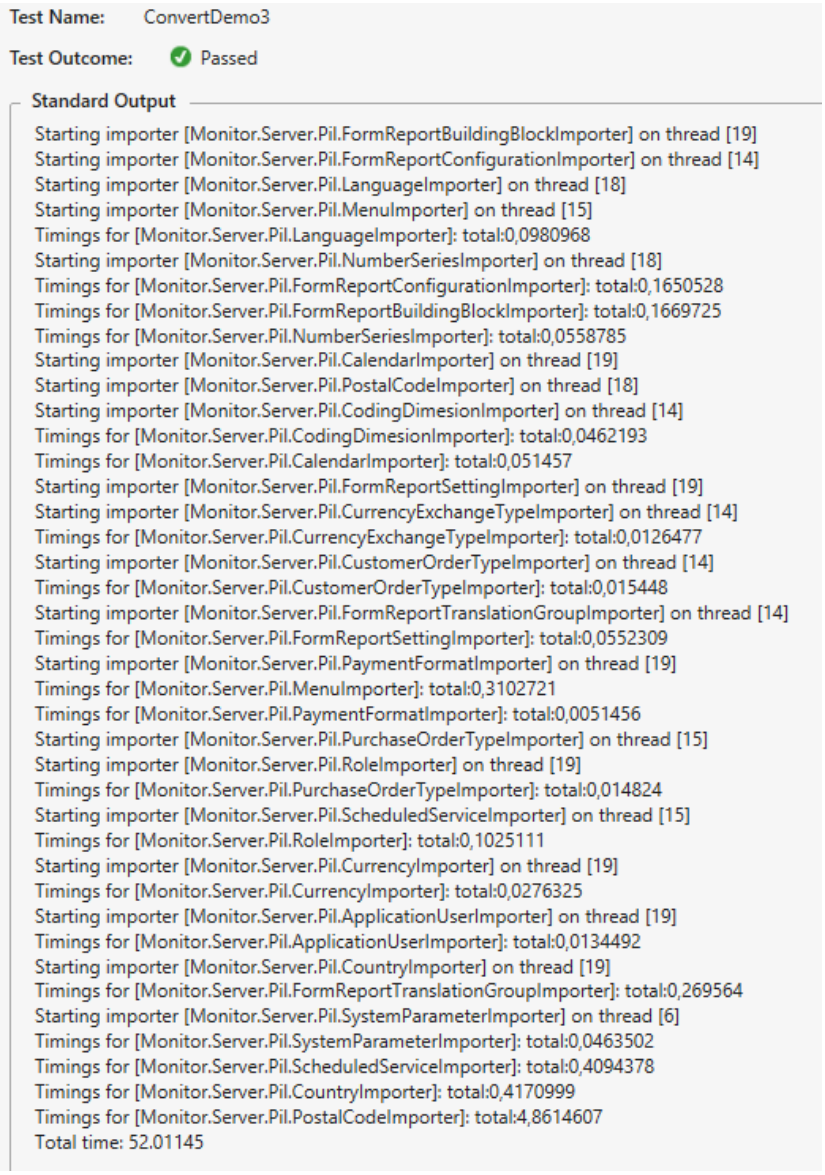
C.2 Utskrifter



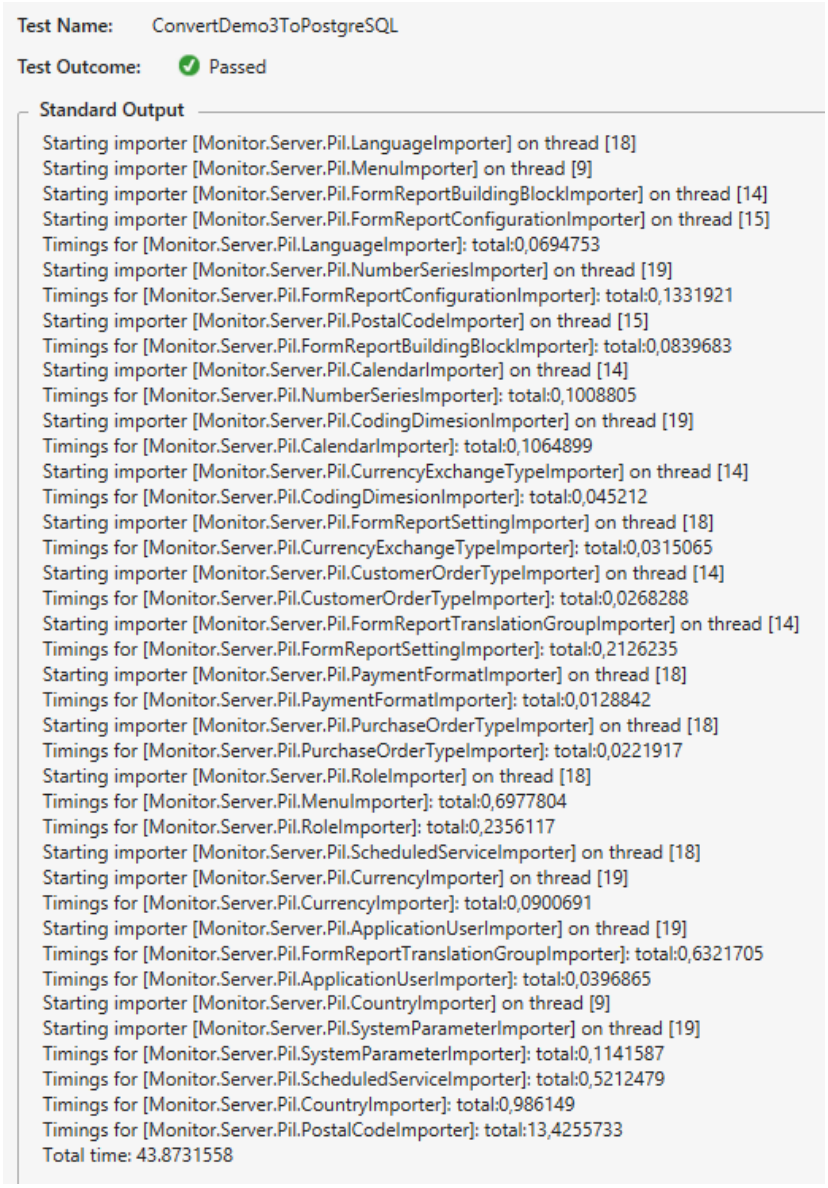
Figur C.1: Körtider för olika insättningsmetoder i PostgreSQL (*NpgsqlInsertTests*)



Figur C.2: Jämförelse konkatenerade insättningar (*InsertTest*)



Figur C.3: Tider för skapande av databas och tabeller i SQL Anywhere



Figur C.4: Tider för skapande av databas och tabeller i PostgreSQL