Analyse Databasegebruik van het ChipSoft Framework

Patronen in SQL Server trace-logs

Daniël Vrancken
0594229

(15-08-2006)

Afstudeerdocent:	Jan van Eijck
Stagebegeleider:	Lars Truijens
Opdrachtgever:	ChipSoft
Publicatiestatus:	Openbaar (v1.1)

Master Software Engineering

Universiteit van Amsterdam

Adressenlijst

Afstudeerder:
Daniël Vrancken
Capucijnenweg 9a
6286 BA Wahlwiller
Tel: 043-4552051
E-mail: dvrancken@gmail.com

Stagebedrijf:
ChipSoft
TT Melissaweg 23-25
1033 SP Amsterdam
Tel: +31 (0)20 4939000
Fax: +31 (0)20 6331975
E-mail: info@chipsoft.nl

Postadres:
Postbus 37039
1033 AA Amsterdam

Bedrijfsbegeleider:
Lars Truijens
E-mail: larst@chipsoft.nl

Schoolbegeleider:
Jan van Eijck
E-mail: jan.van.eijck@cwi.nl

Onderwijsinstelling:
Universiteit van Amsterdam
Faculteit Wiskunde Natuurkunde Informatica
Opleiding Master Software Engineering
Spui 21
1012 WX Amsterdam
Tel: 020-52591111

Samenvatting

Deze scriptie beschrijft het onderzoek naar de analyse van het databasegebruik van het ChipSoft framework. Dit onderzoek is verricht bij ChipSoft te Amsterdam, in het kader van het afstuderen van de eenjarige Master Software Engineering aan de universiteit van Amsterdam.

ChipSoft is een bedrijf dat zich richt op de zorgsector, waarbij vooral ziekenhuizen de belangrijkste klanten vormen. Het bedrijf is opgericht in 1986 en is binnen de zorgsector uitgegroeid tot één van de marktleiders voor ziekenhuisinformatiesystemen. Dit komt mede doordat ChipSoft in 2001 een pioniersrol vervulde door als eerste een op Windows gebaseerd zorginformatiesysteem te lanceren, genaamd het ChipSoft Elektronisch Zorg Informatie Systeem (CS-EZIS).

Het CS-EZIS systeem is modulair opgebouwd, en bestaat uit een zogenaamde 3-tier architectuur. De drie lagen van deze architectuur zijn de presentatie laag, de business logica laag en de data-access laag. Door deze verdeling kan er gebruik worden gemaakt van verschillende soorten dataopslag zonder dat daarvoor de presentatie laag en de business logica laag aangepast hoeven te worden.
Door de 3-tier architectuur is de directe toegang tot de database voor de programmeurs afgeschermd. Zij kunnen gegevens opvragen middels de functies die aangeboden worden vanuit de data-access laag. Door het gebruik van deze functies zien de programmeurs niet welke bewerkingen er onder water op de database worden uitgevoerd. Er bestaat dus de mogelijkheid dat programmeurs functies in bepaalde volgorde aanroepen die onder water inefficiënte bewerkingen uitvoeren op de database.

Omdat er performance problemen zijn, is het van belang om het databasegebruik van het EZIS onder controle te hebben. Er is daarom onderzoek verricht naar de queries die worden uitgevoerd door het EZIS. Dit onderzoek is gericht op het analyseren van SQL Server trace-logs. Deze trace-logs bevatten gegevens over de uitgevoerde queries van het EZIS. De specifieke vraag die daarbij gesteld werd is de volgende:

In hoeverre is het mogelijk om in de bestaande SQL-traces patronen te herkennen die duiden op onnodig of verkeert databasegebruik?

Tijdens het onderzoek is er naar vijf mogelijke patronen onderzoek verricht. Dit zijn onderstaande patronen. Bij dit onderzoek is er gebruik gemaakt van een trace-log van een ziekenhuis. De resultaten hiervan blijken een goede blik op de werkelijkheid.

1. SQL queries die veel tijd kosten
SQL queries die veel tijd kosten hebben slechte invloed op de performance. Dit kan komen doordat een query op een inefficiënte manier is geïmplementeerd. Bijvoorbeeld door verkeerd gebruik te maken van sub-queries.

2. SQL queries die vaak voorkomen
Dit zijn queries die vaak worden uitgevoerd maar wel noodzakelijk zijn. Als deze queries in verhouding veel performance kosten dan heeft dit een grote invloed op de performance van het hele systeem. Een geringe performance verbetering van deze queries kan een grote performance verbetering betekenen voor het hele systeem.

3. SQL queries die dubbel worden uitgevoerd
Dit zijn queries die meermaals achter elkaar uitgevoerd worden, terwijl één keer voldoende is. Dit kan komen doordat programmeurs een functie meerdere malen aanroepen om een resultaat te verkrijgen, en niet zien wat er “onder water” voor queries uitgevoerd worden door de data-acces laag.

4. SQL queries die ongebruikte data ophalen
Dit zijn queries die data ophalen uit de database die niet gebruikt wordt in de applicatie. Denk hierbij aan *select* * terwijl maar twee van de zes kollommen uit de tabel gebruikt worden. Het ophalen van deze gegevens uit de tabel en versturen naar de applicatie kost performance die gespaard kan worden. Het betreft hier vooral I/O performance. Hieronder vallen ook queries die data ophalen die helemaal niet gebruikt wordt.

5. SQL queries die semantische errors bevatten
Dit zijn queries die worden uitgevoerd, maar die in geen enkele situatie een resultaat geven. Ook kan een gedeelte van het SQL statement overbodig zijn, en geen toegevoegde waarde hebben op het resultaat. Dit kost resources die geen resultaat geven.

Van patroon 4 bleek het niet mogelijk te zijn om dit te herkennen in een trace-log. Om dit wel te kunnen herkennen is een voorstel gedaan hoe dit gerealiseerd kan worden binnen het EZIS. De overige patronen zijn wel herkend. De performancewinst die gemaakt kan worden bij de gevonden patronen is verdeeld in performancewinst per query en performancewinst voor het gehele EZIS. Voor de herkende patronen kon er een performancewinst gemaakt worden bij het optimaliseren van de losse queries.
De optimalisaties zijn echter nog niet geïmplementeerd in het EZIS. Er kon daarom niet getest worden hoe groot het performanceverschil daadwerkelijk zal zijn als dit in de praktijk wordt uitgevoerd. Verwacht wordt echter dat dit niet groot zal zijn, daar de gevonden patronen slechts in kleine mate voorkwamen. Er zal daarom verder onderzoek verricht moeten worden om de bottleneck te vinden van het performanceprobleem.

Inhoudsopgave

Lijst met afkortingen

CPU	:	Central Processing Unit
CS	:	ChipSoft
DBA	:	Database Administrator
DBMS	:	Database Management System
EZIS	:	Elektronisch Zorg Informatie Systeem
I&S	:	Implementatie & Support
MSSQL	:	Microsoft SQL
R&D	:	Research & Development
SEQUEL	:	Structured English Query Language
SQL	:	Structured Query Language

Lijst van figuren en tabellen

Voorwoord

De afstudeerperiode van drie maanden is zeer snel verlopen. Ik ben elke dag met plezier naar ChipSoft toe gegaan. Er was een zeer prettige informele sfeer en er waren leuke collega's. Dit zorgde ervoor dat ik me op mijn gemak voelde en me goed kon concentreren op het onderzoek.

Tijdens de afstudeerperiode heb ik veel vrijheid gehad voor het uitvoeren van het onderzoek. Toch was er een intensieve begeleiding, mede door een wekelijkse voortgangsbespreking met mijn bedrijfsbegeleider. Dit vond ik zeer prettig. Bij deze wil ik mijn bedrijfsbegeleider dan ook bedanken voor de goede begeleiding tijdens de afstudeerperiode.

Het afgelopen studiejaar was een druk en intensief jaar waarin hard gewerkt is, met mee- en tegenvallers. Ondanks dat het af en toe moeilijk was om de motivatie op te brengen om de studie af te maken, is het toch gelukt om door te gaan tot het einde. Ik ben blij dit gedaan te hebben. Ik heb het afgelopen jaar zeer veel geleerd. Bij deze wil ik dan ook de opleiding en de leraren hiervoor bedanken.

1. De organisatie

1.1 Historie

ChipSoft is een bedrijf dat zich richt op de zorgsector, waarbij vooral ziekenhuizen de belangrijkste klanten vormen. Het bedrijf is opgericht in 1986 en is binnen de zorgsector uitgegroeid tot één van de marktleiders voor ziekenhuis informatiesystemen. Dit komt mede doordat ChipSoft in 2001 een pioniersrol vervulde door als eerste een op Windows gebaseerd zorginformatiesysteem te lanceren, genaamd het ChipSoft Elektronisch Zorg Informatie Systeem (CS-EZIS).

Het systeem van ChipSoft bestaat uit een paar groepen kernmodules en kan worden uitgebreid met verschillende andere modules die een totaalpakket vormen. Klanten hebben hierdoor de mogelijkheid alleen modules te kopen die voor hen interessant zijn. Ook kunnen producten van andere producenten gebruikt worden binnen de ChipSoft applicatie. Dit levert een heel sterk en dynamisch pakket op.

Binnen het bedrijf zijn op het moment van schrijven in totaal 136 medewerkers werkzaam, verdeeld over twee vestigingen. Ongeveer 50 hiervan werken aan de ontwikkeling van het pakket en rond de 60 aan Implementatie & Support. Verder zijn er ook nog mensen werkzaam op de afdelingen Marketing & Sales, Administratie, Secretariaat en Hardware.

De afstudeeropdracht werd uitgevoerd binnen de vestiging Amsterdam.

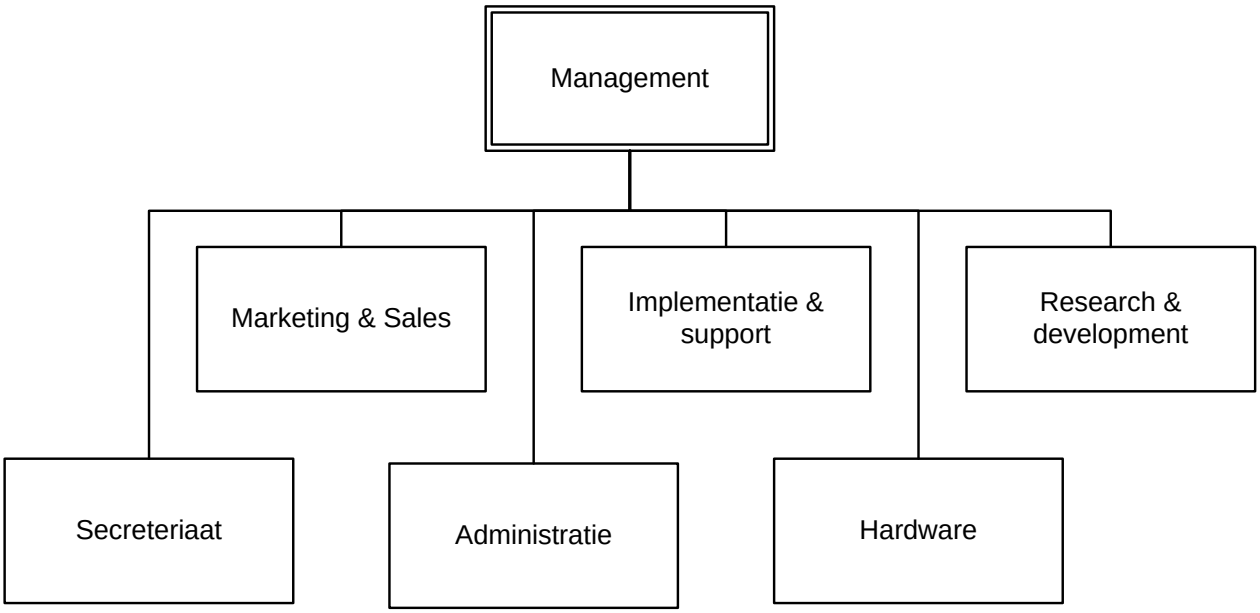
1.2 Doelstelling

Het automatiseren van de zorginstellingen ziet ChipSoft als de voornaamste taak voor het bedrijf. ChipSoft wil dit in een zo totaal mogelijk pakket van diensten aanbieden, wat betekent dat ze naast software ontwikkelen en implementeren ook hardware consultancy leveren en projectmanagement en kantoorautomatisering aanbieden. Hierbij vormt begeleiding van de klanten en hun gebruikers een belangrijk onderdeel. Getracht wordt om een zo goed mogelijke relatie op te bouwen met de klanten, zodat de uiteindelijk op te leveren software ook echt voldoet aan de wensen en eisen van de klant. Om deze reden probeert ChipSoft ook echt een totale oplossing aan te bieden.

1.3 Organisatiestructuur

Het personeel van ChipSoft is verdeeld in een aantal afdelingen: Research & Development, Implementatie & Support, Management, Marketing, Hardware, Documentatie en Administratie. De management afdeling staat aan het hoofd van de organisatie. Alle overige afdelingen vallen hier onder.

De vestiging Drachten wordt gecoördineerd door Richard Storm. Deze vestiging valt op zijn beurt weer onder de afdeling Research & Development in Amsterdam. Hieronder is in Afbeelding 1 het organogram van ChipSoft te zien.



Afbeelding 1 : Organogram van ChipSoft

1.4 Bedrijfscultuur

Vanaf het ontstaan van het bedrijf tot de dag van vandaag heeft de gehele opzet van het bedrijf zich geconcentreerd rond de afdeling Research & Development. De andere afdelingen (Support, Management, Marketing, Hardware, Documentatie en Administratie) dienen ter ondersteuning van deze afdeling. Dit betekent dat de programmeurs veel vrijheid hebben in de werkwijze die ze willen gebruiken om hun doel te bereiken.

De informele sfeer en de vrijheid die de programmeurs krijgen zorgen ervoor dat iedereen met plezier aan het werk is. Lunchtijden kunnen zelf worden ingevuld en drinken is vrij verkrijgbaar. Bovendien kan er elke vrijdag op kosten van de baas eten worden besteld.

1.5 De Software

Deze paragraaf beschrijft het CS-EZIS pakket. Hiervan worden eerst de functionaliteiten besproken, gevolgd door enkele modules. De interne werking en architectuur van het CS-EZIS wordt in het volgende hoofdstuk besproken.

1.5.1 Functionaliteiten

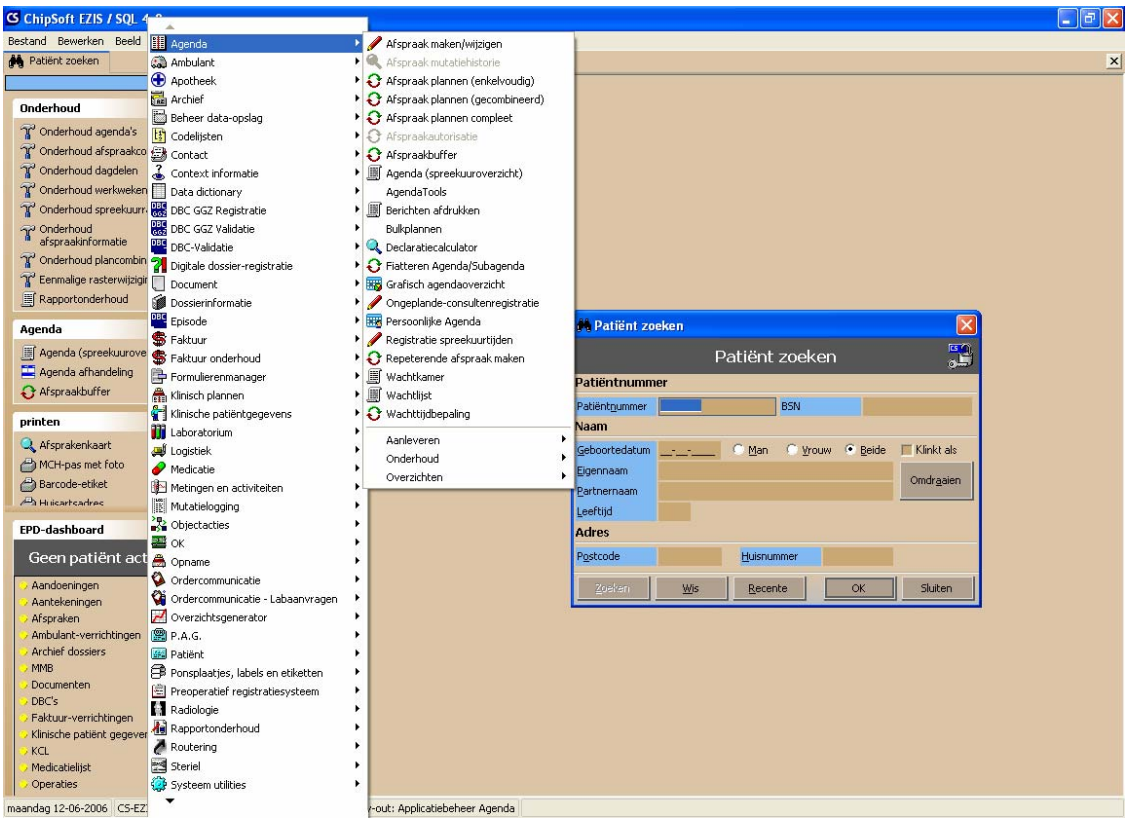
Het bestaande software pakket CS-EZIS (ChipSoft Elektronisch Zorg Informatie Systeem) is een volledig pakket voor ziekenhuizen. Het programma bestaat uit een programmaschil (de userinterface) en een groot aantal modules. Iedere module kan los in deze schil geladen worden en zo een compleet stuk functionaliteit voor zijn rekening nemen. Afhankelijk van de rechten die een gebruiker heeft, zijn bepaalde modules en functionaliteiten wel of niet beschikbaar. Dit wordt geregeld aan de hand van een gebruikersnaam en wachtwoord, en moet zorgen voor de privacy van de gegevens. Als voorbeeld nemen we een baliemedewerkster en een chirurg. De baliemedewerkster kan patiënten opzoeken in het systeem en ponskaartjes uitprinten. Ze kan echter niet zien welke operaties een patiënt heeft ondergaan. De chirurg kan wel precies zien welke operaties de patiënt heeft ondergaan, maar kan bijvoorbeeld geen ponskaartjes printen. Het voordeel van dit systeem, ten opzichte van twee van elkaar onafhankelijk werkende programma's, is dat alle medewerkers van een ziekenhuis met hetzelfde systeem werken, en alle gegevens in dezelfde database staan. Op deze manier wordt er zo efficiënt mogelijk met de gegevens van patiënten omgegaan.

1.5.2 Modules

Binnen het CS-EZIS zijn een groot aantal modules die ieder hun eigen functionaliteit hebben (Afbeelding 2). Om de functionaliteit van het CS-EZIS pakket te verduidelijken worden hieronder een paar modules toegelicht:

- **Agenda**
  CS-Agenda ondersteunt de poliklinische patiëntlogistiek. Het eenvoudig en snel kunnen plannen van enkelvoudige afspraken of van combinatieafspraken gebaseerd op een behandelprotocol behoort tot de standaardfunctionaliteit. Ook wordt binnen CS-Agenda onder andere de afhandeling van verwijskaarten geregeld.
- **Document**
  CS-Document wordt ingezet als correspondentie- en verslagleggingsysteem. Alle binnen het CS-EZIS vastgelegde informatie kan zonodig automatisch in een verslag of brief worden opgenomen. CS-Document werkt op basis van sjablonen waarin de opmaak van de brief of het verslag is vastgelegd, aangevuld met de gegevens uit het CS-EZIS. Daarnaast kunnen veel gebruikte stukken tekst als tekstblokken worden ingevoegd.

- **Formulieren manager**
  Om ervoor te zorgen dat mensen van verschillende specialismen op een zo gemakkelijk mogelijke manier met hetzelfde systeem kunnen werken, is er een formulierenmanager in CS-EZIS opgenomen. Hiermee is het mogelijk formulieren in elkaar te zetten die alleen de gegevens laten zien die van toepassing zijn.
- **Patiënt**
  CS-Patiënt vormt de basis van het CS-EZIS. In CS-Patiënt vindt de registratie en vastlegging plaats van de patiëntgegevens, zoals naam, adres en woonplaats, maar ook de verzekeringsgegevens, e-mail en vrij te definiëren gegevens.
- **What's new**
  In deze module kan de gebruiker belangrijke berichten ontvangen. Wanneer een arts bijvoorbeeld urine heeft opgestuurd naar het laboratorium voor onderzoek, kan deze in What's new de uitslag ontvangen zodra deze bekend is.
- **Ziscon (Zorg Informatie Systeem CONfiguratie)**
  Met Ziscon kunnen de systeembeheerders van een ziekenhuis het hele systeem instellen naar eigen wens. Ze kunnen bijvoorbeeld de lay-outs regelen en gebruikers rechten geven op onderdelen die ze nodig hebben.



Afbeelding 2 : CS-EZIS en de verschillende modules

2. Onderzoeksvraag

In dit hoofdstuk wordt besproken wat het onderwerp is van de afstudeerstage is, waarom dit onderwerp interessant is, en wat er precies onderzocht werd. Verder wordt een beeld gegeven van de situatie waarbinnen het probleem zich afspeelt. Ten slotte zal de doelstelling van het afstudeerproject besproken worden.

2.1 Wat te doen

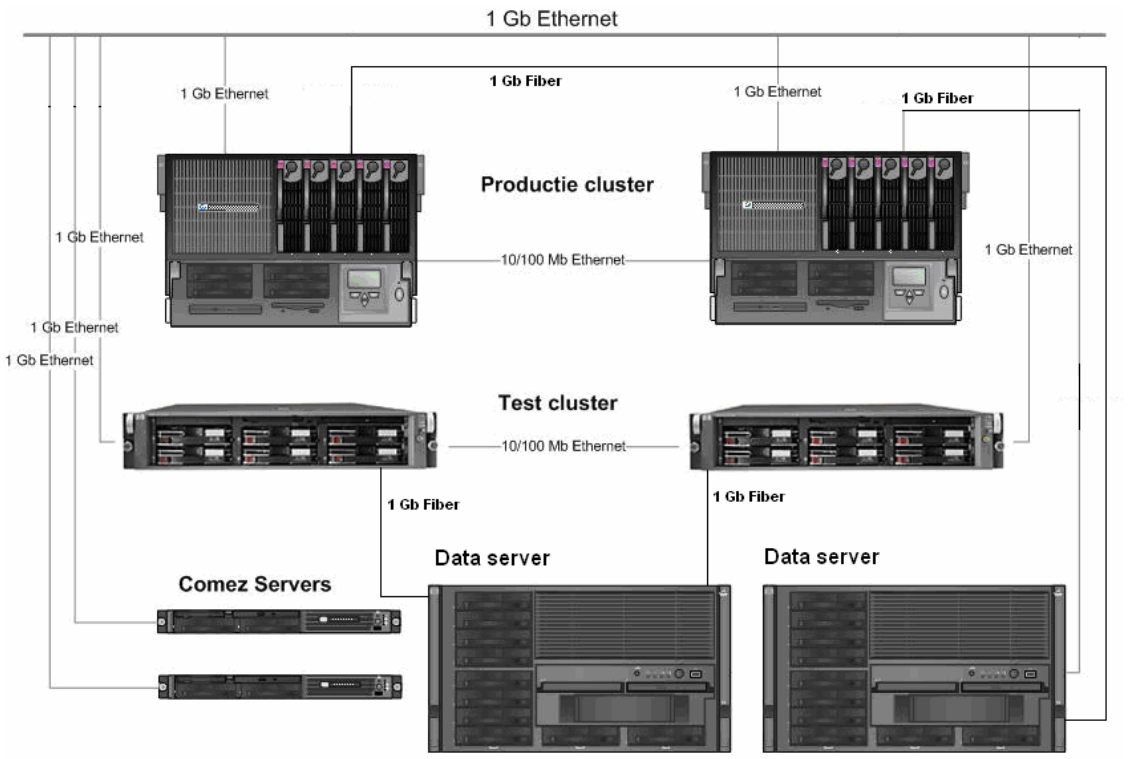
Tijdens de afstudeerperiode is er onderzoek gedaan naar het databasegebruik van het CS-EZIS systeem. Er worden door het systeem vele queries uitgevoerd waardoor er problemen zijn met de performance. Door inzicht te krijgen in het databasegebruik moet duidelijk worden waar optimalisaties gemaakt kunnen worden in het databasegebruik van het EZIS. Op deze manier moet de performance verbeterd worden. Om een idee te krijgen van de omvang van het aantal queries die worden uitgevoerd door het systeem, zijn hieronder enkele statistieken te zien uit een groot ziekenhuis. Hierbij is ook de langst durende query weergegeven. Zulke queries zorgen voor slechte performance van het systeem. In hoofdstuk 5 wordt hier dieper op in gegaan.

Tijdsduur : 30 minuten
Van : 14:11:09
Tot : 14:41:14
Aantal verschillende gebruikers : 613
Aantal uitgevoerde queries : 1.302.726
Langst durende query : 16626 ms
Langst durende query :

```
sp_executesql N'
SELECT A.[NAAM],A.[OWNER],A.[VALUE],A.[INSTTYPE],A.[SPECCODE],A.[ETD_STATUS]
FROM [dbo].[CONFIG_INSTVARS] AS A
WHERE (((A.[INSTTYPE] = @SV1 AND A.[OWNER] LIKE @SV2+"%") OR  (A.[INSTTYPE] =
@SV3 AND A.[OWNER] LIKE @SV2+"%")) AND NOT ((A.[INSTTYPE] = @SV1 AND A.[OWNER]
= @SV2 AND (A.[NAAM] LIKE @SV4+"%" OR  A.[NAAM] LIKE @SV5+"%" OR  A.[NAAM] LIKE
@SV6+"%" OR  A.[NAAM] LIKE @SV7+"%" OR  A.[NAAM] LIKE @SV8+"%" OR  A.[NAAM] LIKE
@SV9+"%" OR  A.[NAAM] LIKE @SV10+"%"))))
ORDER BY A.[INSTTYPE] ASC,A.[OWNER] ASC,A.[NAAM] ASC,A.[SPECCODE] ASC
OPTION (KEEP PLAN, KEEPFIXED PLAN, LOOP JOIN)
', N'@SV1 nvarchar(1),@SV2 nvarchar(8),@SV3 nvarchar(1),@SV4 nvarchar(16),@SV5
nvarchar(16),@SV6 nvarchar(16),@SV7 nvarchar(16),@SV8 nvarchar(16),@SV9 nvarchar(16),@SV10
nvarchar(16)',
@SV1='D',@SV2='CHIPSOFT',@SV3='G',@SV4='OPMAAK_BROWSE',@SV5='OPMAAK_INVOER',
@SV6='OVZGENRESULTATEN',@SV7='OPN_GRFBEDBEZ',@SV8='OPN_OPMOPNSCRN',@SV9=
'AGN_OP',@SV10='AG_OP'
```

2.2 Waarom dit onderwerp

Als er door de applicatie gretig gebruik wordt gemaakt van de database server dan leidt dit tot performance problemen. Dit vooral als daarbij ook nog meerdere mensen tegelijk met de applicatie werken.
Om de performance problemen te verhelpen, is het belangrijk om het databasegebruik van de applicatie onder controle te hebben. Een snelle applicatie geeft meer vertrouwen aan een gebruiker, en levert een bijdrage aan de kwaliteit van het product. Dit in tegenstelling tot een langzame applicatie die leidt tot frustraties van gebruikers bij het dagelijks gebruik.
Niet alleen het voorkomen van performance problemen is van belang. Denk ook aan kosten die bespaard kunnen worden doordat er minder krachtige hardware nodig is om de database server zijn werk te laten verrichten. Ook de schaalbaarheid van het product kan belangrijk zijn. Deze kan vergroot worden indien een database server meer gebruikers tegelijk kan afhandelen.
De hardwarearchitectuur van het ziekenhuis uit de voorbeeld log is te vinden in Afbeelding 3. Op het productiecluster worden de dagelijkse werkzaamheden verricht. Het testcluster wordt gebruikt voor het testen van nieuwe releases van het EZIS voordat deze live worden gebracht in de productieomgeving. Beide clusters hebben een bijbehorende data server. De comez servers zorgen voor de koppeling van andere systemen met het EZIS. Er wordt hiervoor gebruik gemaakt van het HL7 protocol, dat een standaard is binnen de zorgsector. In Tabel 1 is de configuratie van de servers te zien. Elke server in het productiecluster en het testcluster beschikt over deze capaciteit.



Afbeelding 3 : De hardwarearchitectuur van het betreffende ziekenhuis

Hardware	Configuratie
Cpu	4x Intel Xeon 3.0 Ghz
Ram	8 Gb
Netwerk	1 Gbit Ethernet + 1 Gbit Fiber Ethernet
Harddisk(s) (per data server)	15k SCSI Schijven voor: SQL transactie log (130 GB) / SQL Database (600 GB) / Archief (1 TB)

Tabel 1 : Hardwareconfiguratie per server uit het productie- en testcluster

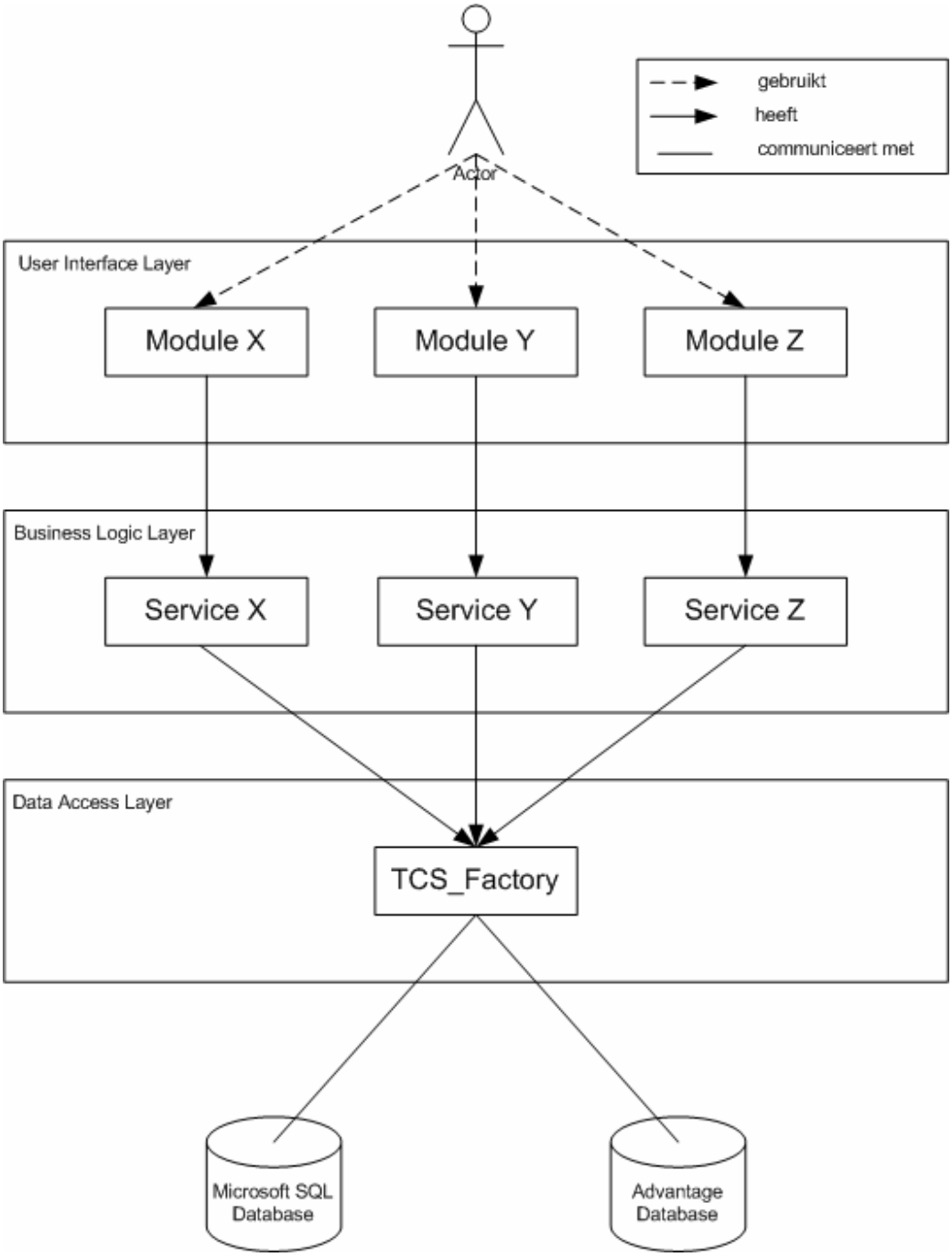
2.3 Probleem situatie

Binnen een groot ziekenhuis kent het CS-EZIS systeem ongeveer 3500 verschillende gebruikers. Op het piekmoment tussen 8.00 en 16.00 zijn hiervan ongeveer 1200 gebruikers tegelijkertijd actief.
De performance van het CS-EZIS is een belangrijk issue. Een requirement is dat een scherm binnen 1,5 seconden getoond wordt. Enkele uitzonderingen hierop zijn acceptabel bij het genereren van bepaalde overzichten. Uit een interview met de afdeling implementatie & support blijkt dat er toch regelmatig klachten binnenkomen van verschillende klanten over de performance van het CS-EZIS systeem. Deze klachten komen vooral uit bepaalde modules van het systeem. Dit zijn de modules Agenda, Opname en OK. Dit zijn ook de meest gebruikte modules.
Klachten komen vooral binnen bij nieuwe releases van bijvoorbeeld hotfixes. Er wordt dan niet op de performance getest. Alleen bij hoofdreleases wordt er getest op performance. Er worden ook geen stresstesten uitgevoerd bij ChipSoft. De klant is zelf verantwoordelijk voor het stresstesten van de applicatie op hun eigen test servers. Dit is omdat ChipSoft niet de beschikking heeft over dezelfde capaciteit servers en hoeveelheden data die sommige klanten hebben.

2.4 Probleem oorzaak

Het CS-EZIS systeem is modulair opgebouwd, en bestaat uit een zogenaamde 3-tier architectuur. De drie lagen van deze architectuur zijn de presentatie laag, de business logica laag en de data-acces laag (Afbeelding 4). Door deze verdeling kan er gebruik worden gemaakt van verschillende soorten dataopslag zonder dat daarvoor de presentatie laag en de business logica laag aangepast hoeven te worden. Voor de dataopslag wordt er bij ChipSoft gebruik gemaakt van Microsoft SQL Server, en Advantage Database Server.

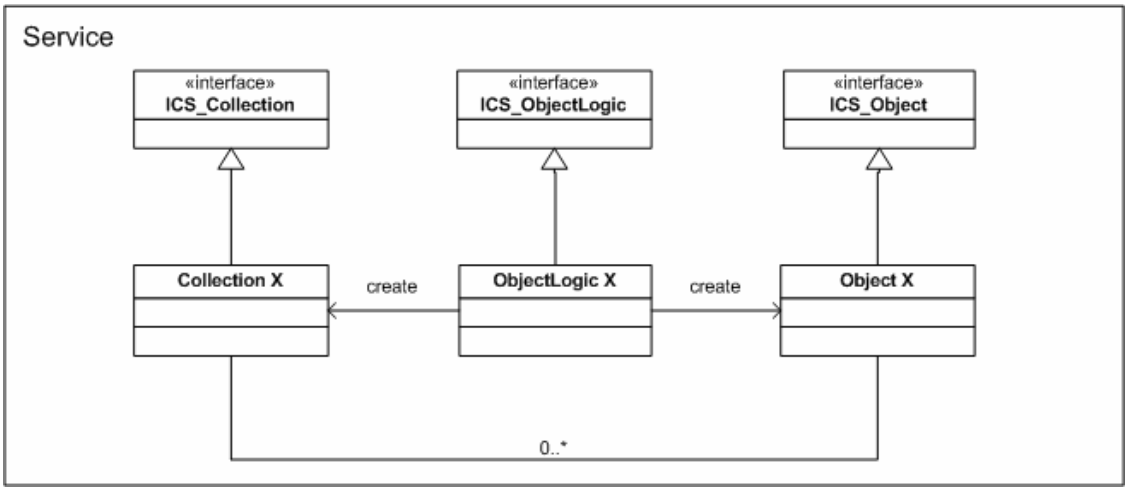
Door de 3-tier architectuur is de directe toegang tot de database voor de programmeurs afgeschermd. Zij kunnen gegevens opvragen middels de functies die aangeboden worden vanuit de data-acces laag. Door het gebruik van deze functies zien de programmeurs niet welke bewerkingen er onder water op de database worden uitgevoerd. Er bestaat dus de mogelijkheid dat programmeurs functies in bepaalde volgorde aanroepen die onder water inefficiënte bewerkingen uitvoeren op de database. Dit heeft ten gevolge dat de performance van het systeem afneemt. De interne werking van het EZIS zal in de volgende subparagrafen worden toegelicht.



Afbeelding 4 : EZIS Layered view

2.4.1 Service

In het vorige hoofdstuk hebben we enkele modules van het CS-EZIS gezien. Om deze modules te laten werken hebben alle modules een bijbehorende service. De agenda module heeft een agenda service, de document module een document service, de formulieren manager een formulieren manager service, de patiënt module een patiënt service, enz...
Deze services zorgen ervoor dat er gecommuniceerd wordt met de juiste tabellen uit de database, dat de modules aan hun gegevens komen, en dat de business rules worden nageleefd. Een service bestaat uit verschillende objecten welke zijn afgeleid van verschillende interfaces (Afbeelding 5). Deze interfaces vormen de objectstructuur van het CS-EZIS, en worden besproken in de volgende subparagraaf.



Afbeelding 5 : EZIS Service

2.4.2 De interfaces

Om een duidelijk beeld te geven van de structuur die in elke service terug te vinden is, worden de belangrijkste interfaces hier omschreven en hun onderlinge relaties toegelicht in een schematisch overzicht (Afbeelding 6).

- **ICS_ObjectLogic**
  Een *ObjectLogic* levert *Objecten* aan de hand van een unieke sleutel (ID) en *Collecties* aan de hand van een *Scope* en/of *Filter*. Daarnaast zorgt de *ObjectLogic* ervoor dat de *Objecten* gelezen, geschreven en gewist kunnen worden. Voordat *Objecten* weggeschreven worden, wordt gecontroleerd of deze voldoen aan de business rules. Een *ObjectLogic* is vergelijkbaar met een tabel, maar kan ook nul of meerdere tabellen omvatten. Een *ObjectLogic* weet welke *Velden* het bevat en welke daarvan het ID vormen.

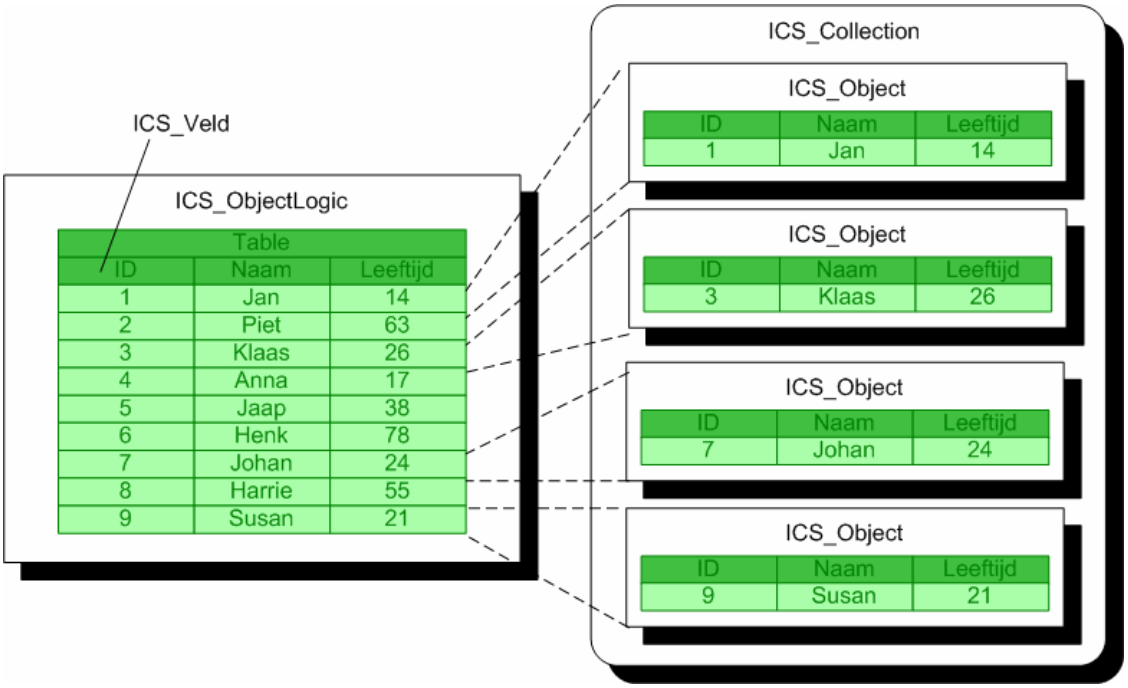
- **ICS_Collection**
  Een *Collection* is een verzameling van *Objecten*. De verzameling kan verkregen worden uit een *ObjectLogic*, maar ook uit een al bestaande *Collection*. De *Collection* wordt in eerste instantie samengesteld aan de hand van criteria die gegeven zijn via een *Scope* en/of *Filter*. Als er geen criteria gegeven worden zal de collectie alle *Objecten* bevatten. Er kan ook een lege collectie aangemaakt worden.
  Als een *Collection* dan uiteindelijk aangemaakt is kunnen er nog steeds *Objecten* van hetzelfde type worden toegevoegd of verwijderd. Om een *Object* uit de collectie op te halen kan het ID van het *Object* gebruikt worden, maar mocht dat niet bekend zijn kan er object voor object doorheen gelopen worden.

- **ICS_Object**
  Een *Object* is vergelijkbaar met een record uit een tabel. Het bevat dan ook *Velden*. Deze *Velden* bevatten de waarden van de velden in een record dat het *Object* representeert. Alle *Objecten* worden gemaakt, ingelezen, opgeslagen en verwijderd door een *ObjectLogic* en hebben daarom een unieke sleutel nodig (ID).

- **ICS_Veld**
  Een V*eld* stelt een enkel veld uit een tabel voor en heeft een groot aantal properties. Zo wordt bijgehouden wat de inhoud is van het veld, en kan deze natuurlijk aangepast worden, mits het niet een read-only veld is.

- **ICS_Scope**
  Een *Scope* geeft een sortering aan van de gegevens en daarnaast kunnen er ook onder- en bovengrenzen aangegeven worden waaraan bepaalde velden moeten voldoen, zogenaamde High en Low Bounds. Een *Scope* kan alleen gecreëerd worden op velden waar in de database een index op gelegd is. Een *Scope* kan gecreëerd worden vanuit een *ObjectLogic* of een *Collectie*.

- **ICS_FilterExpression**
  Een *Filter*, ook wel *Filterexpression* genoemd, geeft in tegenstelling tot een *Scope* geen sortering aan en heeft veel meer vrijheden in het stellen van criteria voor het samenstellen van een *Collection*. Zo kunnen er filters op velden aangelegd worden met verschillende operatoren zoals ‘groter dan’, ‘kleiner dan’, ‘bevat’, enz. Op deze manier is elke gewenste selectie te realiseren. Velden die gebruikt worden in een *Filter* hoeven geen index te hebben, maar zijn daarom wel langzamer dan een *Scope*. Net zoals *Scopes* kunnen *Filters* gecreëerd worden vanuit een *ObjectLogic* of een *Collectie*.



Afbeelding 6 : Objectstructuur EZIS

2.5 Aanpak

Om inzicht te krijgen in het databasegebruik is er onderzoek gedaan naar de queries die uitgevoerd worden op de database server. Voor de uitgevoerde queries is onderzocht of er queries waren die zorgen voor onnodig databasegebruik en dus onnodige performance kosten.
Als eerste is gekeken wat voor soort queries kunnen zorgen voor onnodig databasegebruik, en is hiervan een omschrijving gemaakt. Deze verschillende soorten queries worden genoemd in hoofdstuk 4, en zijn daar verder uitgewerkt.
Daarna is er onderzoek gedaan, met behulp van een eigen gemaakt tool, of deze queries te herkennen waren in de trace-log bestanden van SQL Server. Bij het herkennen van deze queries is vervolgens onderzocht wat de oorzaak was van het optreden van deze queries, en of dit verholpen of geoptimaliseerd kon worden. Ook dit is deels gebeurd door het eigen geschreven tool.

2.6 Doelstelling

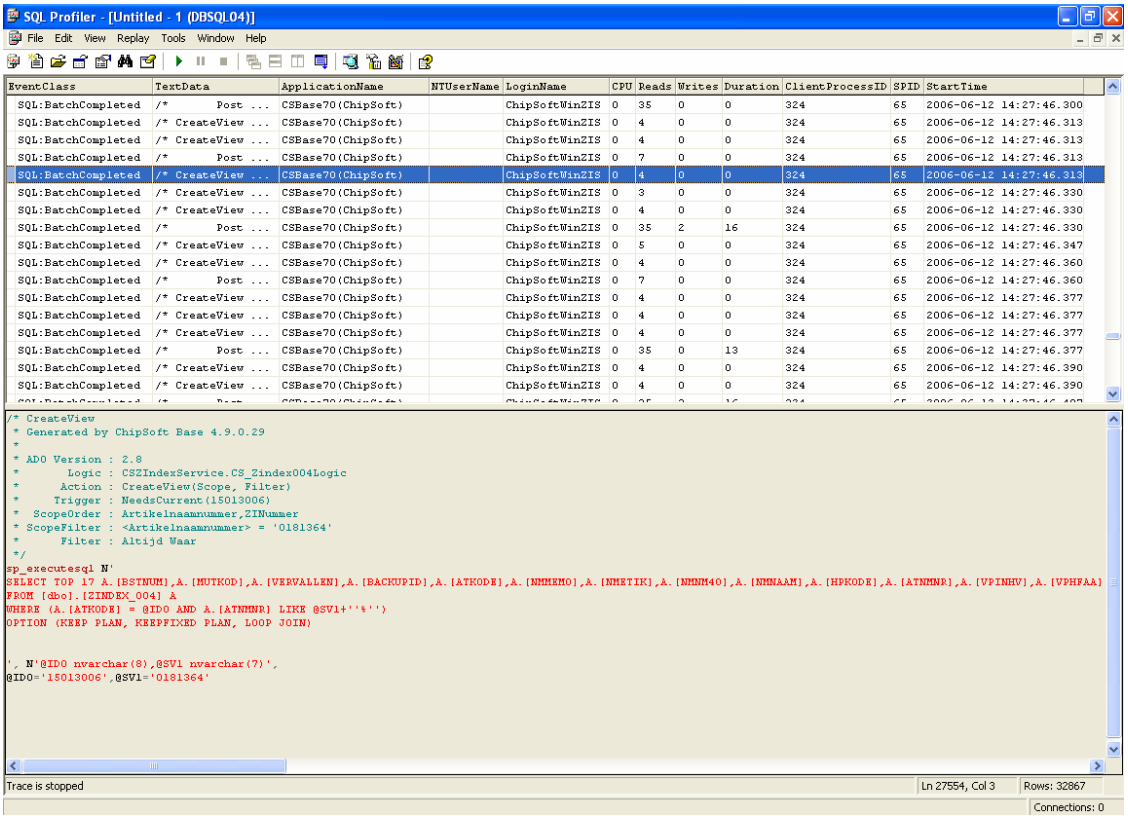
De doelstelling van het onderzoek was om inzicht te krijgen in het databasegebruik van het CS-EZIS systeem. Met het verkregen inzicht moet de performance van het systeem geoptimaliseerd worden.

Het CS-EZIS systeem maakt veelal gebruik van Microsoft SQL Server 2000. Deze biedt de mogelijkheid om logs te maken van de SQL transacties. Uit deze logs kunnen allerhande gegevens gehaald worden over de betreffende SQL transacties, zoals de tijd die een transactie nodig heeft om te executeren.

Om grip te krijgen op de gevolgen die de software implementatie heeft op de database belasting en het netwerk verkeer, dienden de volgende onderzoeksvragen beantwoord te worden.

- In hoeverre is het mogelijk om in de bestaande SQL-traces patronen te herkennen die duiden op onnodig of verkeerd databasegebruik?

- Welke patronen in de software leiden tot de problemen van onnodig of verkeerd databasegebruik? Als dit bekend is kunnen programmeurs getraind worden over deze specifieke problemen.



Afbeelding 7 : SQL Profiler voor het maken van trace-logs

3. Achtergrond en Context

Voor het uitvoeren van het onderzoek was enige kennis nodig over MSSQL Server en SQL. Ook over performance verbetering was enige kennis noodzakelijk. Over deze zaken volgt meer in dit hoofdstuk.

3.1 SQL Historie

SQL staat voor Structured Query Language. In 1974 definieerde Donald Chamberlin et al. bij IBM Research de taal SEQUEL, dat staat voor Structured English Query Language. SEQUEL is gebaseerd op het relationeel data model van Codd [P05]. Dit model vormt de basis voor relationele databases.
In 1976 vond een revisie plaats van SEQUEL, die SEQUEL 2 werd genoemd. Deze naam is daarna veranderd naar SQL, zoals we die de dag van vandaag kennen. De taal SEQUEL 2 (SQL) en enkele voorbeelden in het gebruik hiervan staan beschreven in [P06].

3.2 Database Management System

Een eerste implementatie van een DBMS gebaseerd op SQL [P06] was het experimentele System R, dat in 1975 is ontwikkeld door IBM Research. Het verwerken van een SQL statement in dit systeem bestaat uit vier stappen. Dit is parsing, optimaliseren, code generatie en executie. Ook huidige DBMS zoals Microsoft SQL Server voeren deze stappen uit [B1]. De belangrijkste stap is de optimalisatie stap. Hierbij wordt het optimale acces path voor een query gekozen. Hoe beter het acces path, hoe sneller de query wordt uitgevoerd. Zo wordt er onder andere gekeken of er bij het ophalen van gegevens gebruik gemaakt kan worden van een index. Een beschrijving van het selecteren van acces paths in System R is te vinden in [P08]. Indien er voor een SQL statement geen goed acces path geselecteerd kan worden, dan kan een SQL statement herschreven worden, zodat een acces path wordt geselecteerd dat een betere performance heeft.

3.3 Query tuning

Bij query tuning gaat het er om dat de response tijd voor een gegeven query minimaal wordt. Query tuning is dus alleen redelijk indien de response tijd naar de gebruiker de bottleneck is, en geoptimaliseerd moet worden [P09]. Bij Query tuning is het van belang dat het SQL statement zo herschreven wordt, zodat er door het DBMS een ander acces path wordt geselecteerd [P08]. In [P07] is dit gedaan door SQL statements met subqueries te herschrijven naar een SQL statement zonder subquery. Zo zijn de twee onderstaande queries equivalent aan elkaar.

```
SELECT A.a
FROM A, B
WHERE A.b = B.b
```

```
SELECT A.a
FROM A
WHERE A.b IS IN (       SELECT B.b
                        FROM B          )
```

3.4 Database tuning

Bij het tunen van een database, kan er onderscheid gemaakt worden tussen het tunen van de database server, en uitgevoerde SQL statements op deze server [P04]. Bij het tunen van de database server, gaat het om instellingen die gemaakt kunnen worden in de server software of de hardware van de server. Zo wordt er in [P04] naar verschillende parameter instellingen van een Oracle database server gekeken. Zo kan bepaald worden of bijvoorbeeld het cache van de server de juiste grootte heeft.
Als er wordt gekeken naar uitgevoerde SQL statements, dan kan er onderscheid worden gemaakt tussen queries die een hoge response tijd hebben en queries die vaak voorkomen [P04]. Door het optimaliseren van deze queries kan de response tijd verkort worden.

Een ander manier van database tuning is zorgen dat gegevens worden gecached. In [P01] wil men dit bereiken met behulp van Language Modeling. Language Modeling is afkomstig vanuit de spraakherkenning, en wordt gebruikt om de kans te voorspellen dat een bepaald woord volgt na een reeks van al bekende woorden. Met Language Modeling wil men vanuit een trace-log sessies identificeren. Een sessie is een verzameling van queries uitgevoerd door een gebruiker of applicatie om een bepaalde taak te verrichten. Stel dat het volgende voorbeeld van queries met behulp van Language Modeling wordt herkend als een sessie.

Q1 : select authority from employee where employee id ='1025'
Q2 : select count(*) as num from customer where cust num = '1074'
Q3 : select card name from customer t1,member card t2 where 1.cust num = '1074' and t1.card id = t2.card id
Q4 : select contact last,contact _rst from customer where cust num = '1074'
Q5 select t1.branch ,t2.* from record t1, treatment t2 where t1.contract no = t2.contract no and t1.cust id ='1074' and
check in date = '2002/03/04' and t1.branch = 'scar'
Q6: select top 10 contract no from treatment schedule where cust id = '1074' order by checkin date desc

Te zien is dat bij Q2, Q3 en Q4 gegevens uit de customer tabel worden opgehaald. Q2 zou herschreven kunnen worden om alle benodigde data op te halen uit de customer tabel, en dit in het cache te zetten. Indien Q3 en Q4 worden uitgevoerd, dan kunnen deze direct vanuit het cache beantwoord worden.

3.5 Performance tuning

Bij performance tuning kan er een onderscheid gemaakt worden tussen het vergroten van de output voor een gegeven resource, en het minimaliseren van de resource voor een gegeven output [P09].
Voor het performance tunen van een applicatie is het van belang om eerst de huidige performance van de applicatie te kennen. Dit is nodig om bij het proberen te tunen te kunnen meten hoe sterk de performance toeneemt, en of deze ook wel daadwerkelijk toeneemt, en niet afneemt. Het meten van de performance kan gebeuren met behulp van een stresstest tool [P04].

3.6 Semantisch errors in SQL statements

Een semantisch error in een SQL statement wil zeggen dat het statement syntactisch correct is, maar dat bepaalde code van het statement geen effect heeft. Een DBMS geeft geen errors of warnings voor semantische errors. Bij semantische errors kan er onderscheid gemaakt worden tussen queries waarbij je moet weten wat het doel is van de query om te bepalen of het om een semantisch error gaat, en queries waarbij je niet hoeft te weten wat het doel is. Bij deze laatste maakt het niet uit wat het doel is, de query zal altijd een semantische error bevatten. Zo zal onderstaande query nooit een resultaat terug geven. Een overzicht van verschillende semantische errors die voor kunnen komen in SQL statements wordt gegeven in [P03].

```
SELECT *
FROM Persoon
WHERE geslacht = 'M' AND geslacht = 'V'
```

3.7 Microsoft SQL Server

Voor het monitoren en optimaliseren van de performance van SQL Server zijn een aantal tools beschikbaar die zowel aanwezig zijn in Windows als in SQL Server zelf. Een overzicht en bespreking van deze tools wordt gegeven in [WP1] [P02] [B1].
Voor het onderzoek zijn twee tools van SQL Server van belang. Dit zijn SQL Profiler en SQL Query Analyser.

3.7.1 SQL Profiler

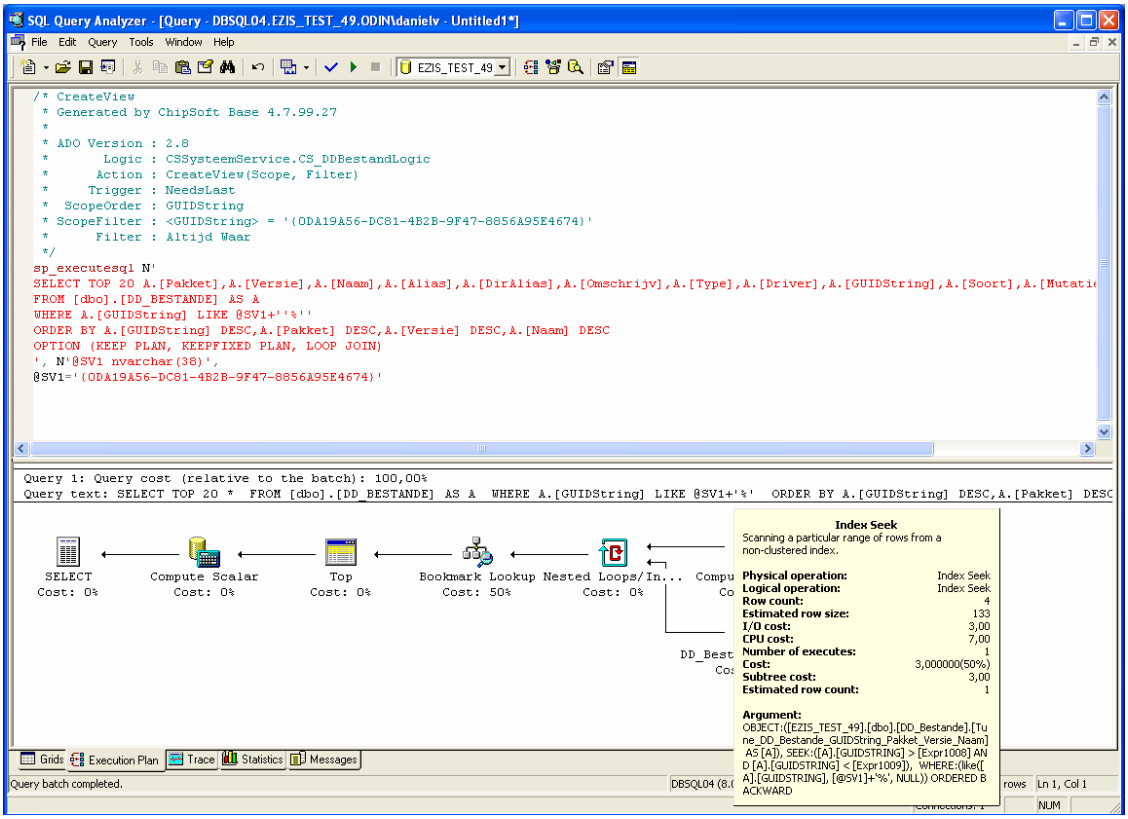
SQL Profiler (Afbeelding 7) is het tool dat de trace-logs maakt van de uitgevoerde queries op de server. De informatie die per query gelogd wordt is weergegeven in Tabel 2. Hierbij is het mogelijk om voor elke data kolom een filter in de stellen zodat maar een beperkt aantal queries worden weergegeven. Bijvoorbeeld queries die langer duren dan een bepaald aantal (milli)seconden, of queries van een bepaalde gebruiker.

Data Kolom	**Omschrijving**
Eventclass	Type event dat ontvangen is
TextData	De uitgevoerde query
ApplicationName	Naam van de client applicatie die de verbinding met SQL Server heeft gemaakt
NTUserName	Windows gebruikersnaam
LoginName	Gebruikersnaam (SQL Server security login of Windows login)
CPU	Hoeveelheid CPU tijd die het event heeft gebruikt (in milliseconden)
Reads	Het aantal logische disk reads
Writes	Het aantal fysieke disk writes
Duration	Hoeveelheid tijd die het event in beslag neemt (in milliseconden)
ClientProcessID	Het process ID van de applicatie
SPID	Process ID van een process van SQL Server dat geassocieerd is met een client applicatie
StartTime	Tijd wanneer het event is gestart

Tabel 2 : SQL Profiler log data

3.7.2 SQL Query Analyzer

Met SQL Query Analyzer kunnen SQL statements uitgevoerd worden en kan bekeken worden welk gedeelte van het statement hoeveel resources in beslag neemt. Dit is te zien in het executie plan van het SQL statement. In Afbeelding 8 is SQL Query Analyzer te zien met een query en bijbehorend executie plan. Indien een statement op een bepaalt punt veel resource in beslag neemt, bijvoorbeeld bij een subquery, dan kan het statement herschreven worden. Bij het herschrijven van het SQL statement is het de bedoeling dat het executie plan efficiënter wordt. Het SQL statement kan vervolgens opnieuw uitgevoerd worden en kan er gekeken worden of er daadwerkelijk minder resources nodig zijn om hetzelfde resultaat te bereiken.



Afbeelding 8 : SQL Query Analyzer

4. Plan van Aanpak

In dit hoofdstuk wordt beschreven welke aanpak is gevolgd om inzicht te krijgen in het databasegebruik. Als eerste wordt besproken wat wordt verstaan onder queries die zorgen voor verkeerd of onnodig databasegebruik. Vervolgens wordt besproken hoe deze queries zijn geprobeerd te vinden. Ten slotte wordt besproken hoe het onderzoek is geprobeerd te valideren.

4.1 Verkeerd of onnodig databasegebruik

Er zijn verschillende soorten queries die kunnen zorgen voor verkeerd of onnodig databasegebruik. Voor het onderzoek is er onderscheid gemaakt in vijf verschillende categorieën.

SQL queries die veel tijd kosten
SQL queries die veel tijd kosten hebben slechte invloed op de performance [P04]. Dit kan komen doordat een query op een inefficiënte manier is geïmplementeerd. Bijvoorbeeld door verkeerd gebruik te maken van subqueries. Deze queries hoeven echter niet altijd te duiden op een verkeerde implementatie. Het is mogelijk dat dit de beste implementatie was voor het probleem, maar dat het gewoon veel tijd kost om uit te voeren door bijvoorbeeld de grootte van de tabellen.

SQL queries die vaak voorkomen
Dit zijn queries die vaak worden uitgevoerd maar wel noodzakelijk zijn [P04]. Als deze queries in verhouding veel performance kosten dan heeft dit een grote invloed op de performance van het hele systeem. Een geringe performance verbetering van deze queries kan een grote performance verbetering betekenen voor het hele systeem.

SQL queries die dubbel worden uitgevoerd
Dit zijn queries die meermaals achter elkaar uitgevoerd worden, terwijl één keer voldoende is. Dit kan komen doordat programmeurs een functie meerdere malen aanroepen om een resultaat te verkrijgen, en niet zien wat er “onder water” voor queries uitgevoerd worden door de data-acces laag.

SQL queries die ongebruikte data ophalen
Dit zijn queries die data ophalen uit de database die niet gebruikt wordt in de applicatie. Denk hierbij aan *select* * terwijl maar twee van de zes kollommen uit de tabel gebruikt worden. Het ophalen van deze gegevens uit de tabel en versturen naar de applicatie kost performance die gespaard kan worden. Het betreft hier vooral I/O performance. Hieronder vallen ook queries die data ophalen die helemaal niet gebruikt wordt.

SQL queries die semantische errors bevatten
Dit zijn queries die worden uitgevoerd, maar die in geen enkele situatie een resultaat geven [P03]. Ook kan een gedeelte van het SQL statement overbodig zijn, en geen toegevoegde waarde hebben op het resultaat. Dit kost resources die geen resultaat geven. De performance zal toenemen als zulke queries (gedeeltelijk) niet meer uitgevoerd worden.

4.2 Queries vinden

Deze paragraaf beschrijft hoe de verschillende queries die kunnen zorgen voor onnodig of inefficiënt databasegebruik zijn geprobeerd te vinden. Hierbij komen verschillende tools aan de orde. Een beschrijving van deze tools is te vinden in Bijlage 2.

SQL queries die veel tijd kosten
Bij deze queries is zowel gezocht naar queries die veel CPU tijd vragen als queries die lang duren. Om deze queries te vinden kon er gebruik gemaakt worden van SQL Profiler, Quest Speed Coefficient, of Quest Performance Analysis.
Het voordeel van de tools van Quest is dat op een gekozen kolom gesorteerd kan worden, en zo snel gezien kan worden welke queries de meeste tijd kosten. Ook is hierbij in een keer te zien hoe vaak deze queries voorkomen, waardoor duidelijk is of de optimalisatie een grote impact zal hebben. Bij SQL Profiler zijn zelf queries geschreven om analyse uit te voeren op de trace-logs.
Het verschil tussen Speed Coefficient en Performance Analysis is dat Speed Coefficient statistieken gegeneerd uit trace-logs, terwijl Performance Analysis actief moet zijn op een computer om performance gegevens van queries op de database server te verzamelen en op te slaan in een repository. Performance Analysis heeft als voordeel dat deze meer gedetailleerdere informatie kan geven over de uitgevoerde queries. Er konden echter geen bestaande trace-logs mee geanalyseerd worden. Voor het analyseren van de trace-logs is er daarom gebruik gemaakt van Speed Coefficient.

Om te kijken of queries efficiënter kunnen, is er gebruik gemaakt van SQL Query Analyzer. Bij het gebruik hiervan moesten de queries handmatig herschreven en getest worden.
Dit ging beter bij het gebruik van Quest Central SQL Tuning. Dit tool probeert optimalisaties te genereren voor een bepaalde query, en test daarbij hoeveel tijd het kost voor het uitvoeren van de gegenereerde queries. Indien een betere query gegenereerd is, kan gekeken worden wat het verschil is met de originele query. Vervolgens kan, indien mogelijk, dit verschil geïmplementeerd worden in de software.

De tools van Quest bieden een aantal voordelen tegenover de tools van SQL Server:

- Er hoeven geen eigen queries geschreven te worden om statistieken uit de trace-logs te halen.
- Er is geen uitgebreide kennis nodig van SQL om inefficiënte queries te herschrijven.
- Het is sneller omdat een herschreven query niet handmatig getest hoeft te worden of deze daadwerkelijk sneller is.
- Er is geen kans op fouten die gemaakt konden worden bij het handmatig herschrijven van queries. Het tool genereert semantisch equivalente queries.

SQL queries die vaak voorkomen
Hiervoor is gebruik gemaakt van SQL Profiler en Quest Speed Coefficient. Ook hiervoor gelde dat voor SQL Profiler zelf queries geschreven moesten worden om de trace-logs te analyseren op veel voorkomende queries.
Queries die het vaakste voorkwamen zijn (eventueel met behulp van Quest Central SQL Tuning) geprobeerd te optimaliseren.

SQL queries die dubbel worden uitgevoerd
Deze queries konden gevonden worden door te kijken of eenzelfde query meermaals achter elkaar werd uitgevoerd. Een trace-log diende hiervoor wel gefilterd te worden op een enkele gebruiker, want twee verschillende gebruikers zouden dezelfde query achter elkaar kunnen uitvoeren.
Het hoefde niet om exact dezelfde queries te gaan. Denk aan *Select* met daarna een *Select Top 20* die beide dezelfde *Where* conditie hebben en dezelfde kolommen aan gegevens ophalen. Een van de queries is overbodig en kan gezien worden als een query die dubbel wordt uitgevoerd.

Om deze queries te is een eigen tool geschreven waarmee een trace-log doorlopen wordt om zo dubbele queries er uit te filteren. Om te herkennen of een query hetzelfde was, is gekeken naar de TextData (Afbeelding 9) kolom uit de trace-log. Hierin staat een stuk commentaar en het SQL statement dat is uitgevoerd. Uit het commentaar en het SQL statement kon informatie gehaald worden om te bepalen of het om een dubbele query ging.

```
/* CreateView
 * Generated by ChipSoft Base 4.8.0.28
 *
 * ADO Version : 2.8
 *       Logic : CSMetingenService.CS_MonitorProtocolLogic
 *      Action : CreateView(Scope, Filter)
 *     Trigger : NeedsCurrent(DR_MEDIBS2)
 *  ScopeOrder : ProtocolID
 * ScopeFilter : Altijd Waar
 *      Filter : Altijd Waar
 */
sp_executesql N'
SELECT A.[PROTOCOLID],A.[VERSIE],A.[OMSCHRIJVI],A.[COMM_TYPE],A.[COMM_SET],A.[SAMPLERATE],A.[EXTRACODES]
FROM [dbo].[METINGEN_MONIPROT] AS A
WHERE (A.[PROTOCOLID] = @IDO AND (1=1))
OPTION (KEEP PLAN, KEEPFIXED PLAN, LOOP JOIN)
', N'@IDO nvarchar(10)',
@IDO='DR_MEDIBS2'
```

Afbeelding 9 : TextData kolom bestaande uit commentaar en het SQL statement

SQL queries die ongebruikte data ophalen
Bij deze queries kon onderscheid gemaakt worden tussen gegevenskolommen die worden opgehaald en niet worden gebruikt, en tussen records die worden opgehaald en niet worden gebruikt.
Uit een trace-log was echter geen informatie te halen of data daadwerkelijk gebruikt wordt in de applicatie. Dit kan wel in de applicatie getest worden. Er is daarom een voorstel gedaan hoe bepaald kan worden of gegevenskolommen die opgehaald zijn uit de database ook daadwerkelijk gebruikt worden. Ditzelfde is ook gedaan voor de records die opgehaald worden.

SQL queries die semantische errors bevatten
Semantische errors zijn gezocht door een eigen gemaakt tool dat de queries uit een trace-log op deze errors analyseert. In dit tool zijn niet op alle semantische errors gezocht die staan beschreven in [P03], dit omdat er door het EZIS maar een beperkt aantal mogelijke queries gegeneerd wordt.

4.3 Hypothesen

De verschillende queries die zijn geprobeerd te vinden hadden de volgende verwachtingen.

SQL queries die veel tijd kosten
Er werd verwacht dat queries gevonden zouden worden die veel tijd kosten. Deze queries waren namelijk al te herkennen bij het verdiepen in de trace-logs en SQL Profiler. Ook waren op de werkvloer regelmatig geluiden te horen over queries die lang duurde. Er werd verwacht dat de performance van sommige queries wel, en van sommige queries niet verbeterd kon worden. Van de queries waarvan de performance verbeterd kon worden werd verwacht dat dit gedaan kon worden door de query te herschrijven waardoor het executieplan van de query werd veranderd.

SQL queries die vaak voorkomen
Er werd verwacht dat er queries gevonden zouden worden die uitzonderlijk vaak voorkomen tegenover andere queries. Dit omdat er modules zijn in het EZIS die meer gebruikt worden dan andere modules, zoals bleek uit het interview met de afdeling I&S. De verwachting was dat deze queries niet veel tijd zouden kosten en dat het moeilijk zou zijn om deze te optimaliseren, omdat ze al een effectief executieplan hebben. Indien dit het geval zou zijn, dan zou alleen een optimalisatie gemaakt kunnen worden door te kijken of ook wel alle data wordt gebruikt. Door geen data op te halen die niet gebruikt wordt zou een performance winst behaald kunnen worden.

SQL queries die dubbel worden uitgevoerd
Ook deze queries waren bij het verdiepen in de trace-logs en het gebruik van SQL Profiler al te herkennen. Van deze queries werd dan ook verwacht dat deze gevonden zouden worden tijdens de analyse van de trace-logs. Indien deze queries gevonden zouden worden kon gekeken worden welk gedeelte van de applicatie hier verantwoordelijk voor is. Als de applicatie wordt aangepast en dubbele queries niet meer worden uitgevoerd dan is er een performance winst.

SQL queries die ongebruikte data ophalen
Aangezien door het EZIS bij elke query alle datakolommen opgehaald worden, werd verwacht dat er zeer zeker queries gevonden zouden worden waarbij data opgehaald wordt die niet gebruikt wordt.

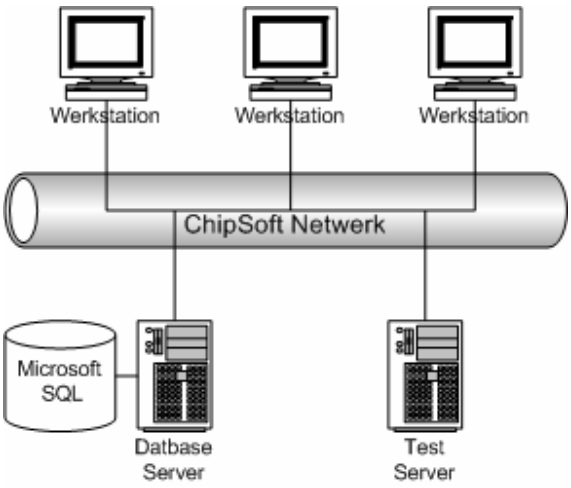
SQL queries die semantische errors bevatten
Deze queries zouden waarschijnlijk ook gevonden worden. Aangezien de queries gegenereerd worden door de data-acces laag, is het mogelijk dat semantisch errors worden gegenereerd als programmeurs bepaalde filters instellen voor een collectie van records die ze opvragen.

4.4 Valideren onderzoek

Voor het valideren van het onderzoek is er getest of de performance wordt verbeterd. Hierbij werd onderscheid gemaakt tussen de performance van een enkele query en de performance van het hele EZIS. Tijdens het onderzoek is hierbij alleen op de performance van enkele queries getest.
Voor dit testen was het van groot belang dat er valide vergelijkingen gemaakt konden worden om te kijken of de performance was toegenomen. Er diende dan ook gebruik gemaakt te worden van de juiste testopstelling.

4.4.1 Testopstelling

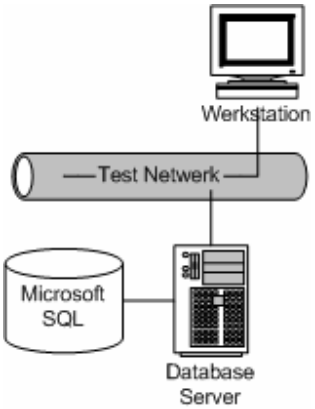
Huidige testopstelling
In de huidige testopstelling (Afbeelding 10) is er een test server waar de laatste versie van het EZIS pakket op getest wordt. De server wordt vaak (meermaals per dag) geüpdate met een nieuwe versie van de software. De database van de applicatie staat op een aparte database server.
De applicatie op de test server wordt door meerdere mensen gebruikt. Zo kunnen zowel ontwikkelaars als testers gebruik maken hiervan. Het aantal mensen dat gelijktijdig gebruik maakt van de applicatie en database server kan verschillen van uur tot uur. Ook het aantal mensen die aanwezig zijn op kantoor kan elke dag anders zijn. De belasting van het netwerk is dan ook variërend.



Afbeelding 10 : ChipSoft Netwerk

Ideale testopstelling

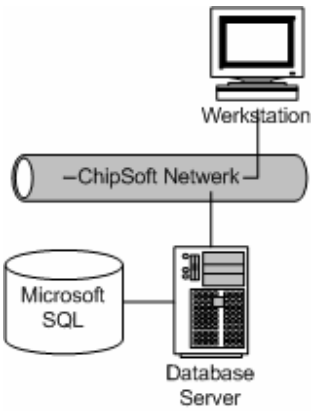
De ideale testopstelling (Afbeelding 11) voor het valideren van het onderzoek zou zijn geweest dat er geen factoren van invloed konden zijn op de performance buiten de gemaakte verandering. Om dit te kunnen realiseren diende er een aparte testopstelling gemaakt te worden.
Deze opstelling zou dan bestaan uit een database server en een werkstation met één versie van het EZIS pakket. In deze versie kunnen dan veranderingen worden doorgevoerd en worden getest hoe deze de performance beïnvloeden. De database server en het werkstation dienen daarvoor in een eigen netwerk aangesloten te worden waardoor er geen invloed zou zijn van andere gebruikers op het netwerkverkeer.



Afbeelding 11 : Test Netwerk

Conclusie testopstelling
De huidige opstelling voor het valideren van de onderzoeksresultaten was niet ideaal. De performance van het systeem kon beïnvloed worden door meerdere factoren, zoals het updaten van de versie van de applicatie of het aantal mensen die gelijktijdig werken met de applicatie. Je weet zo bij het maken van een aanpassing niet of de verandering in performance komt door de aanpassing of door het aantal andere gebruikers die actief zijn. Als er gebruik gemaakt zou worden van een aparte testopstelling, dan waren deze factoren uitgesloten en konden er valide tests worden uitgevoerd.

Gebruikte testopstelling
Het was echter niet mogelijk om de ideale testopstelling te realiseren. Daarom is er gebruik gemaakt van een alternatieve testopstelling (Afbeelding 12). In deze testopstelling is er gebruik gemaakt van een aparte database server en een apart werkstation. Het verschil met de ideale testopstelling is dat er geen gebruik wordt gemaakt van een apart netwerk, maar de configuratie is aangesloten op het interne netwerk. Deze opstelling was goed genoeg voor het testen van de performance. Het netwerk werd niet zo zwaar belast dat dit de resultaten zal beïnvloeden. Dit bleek uit verschillende ping tests vanaf het werkstation naar de database server. Deze gaven een reactietijd van minder dan 1 milliseconde.



Afbeelding 12 : Alternatieve testopstelling

4.4.2 Query performance

Voor het testen van de performance van een enkele query kon gekeken worden of het executie plan van een query verbeterd werd, en wat de waarden waren van CPU en Duration. Ook kon hierbij gekeken worden naar de hoeveelheid data die werd verstuurd. Voor het meten van de performance van een enkele query is gebruik gemaakt van SQL Query Analyzer en van SQL Profiler.

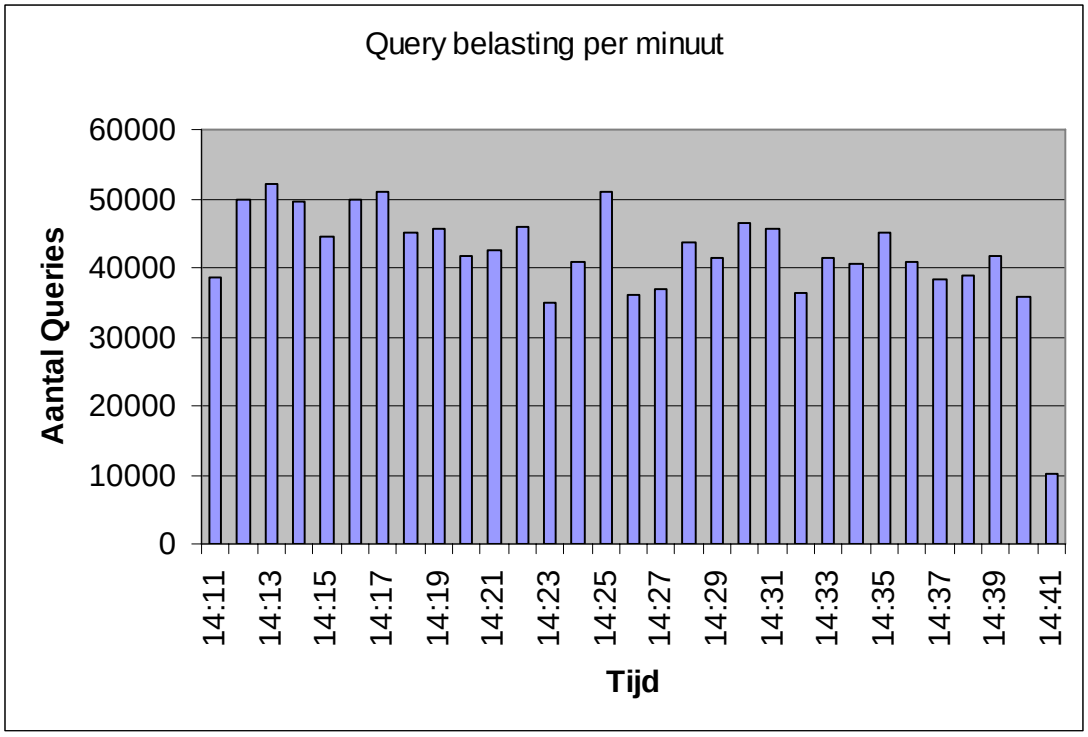
5. Uitvoering

In dit hoofdstuk wordt de uitvoering van het onderzoek besproken. Bij het testen van de performance van queries is gebruikt gemaakt van de testopstelling zoals deze eerder is beschreven in hoofdstuk 4.

5.1 Onderzoeksdata

Voor het onderzoek kon gebruik gemaakt worden van verschillende trace-logs. Dit waren trace-logs gemaakt in de testomgeving bij ChipSoft zelf, en een trace-log van een ziekenhuis. Trace-logs vanuit de testomgeving bij ChipSoft zijn niet geheel valide om het onderzoek op uit te voeren. Deze geven geen beeld van de queries die de applicatie uitvoert in een productieomgeving. De trace-log van het ziekenhuis doet dit wel. Tijdens het onderzoek is er dan ook gebruik gemaakt van de trace-log van een ziekenhuis. Deze heeft onderstaande eigenschappen, en is gelijk aan het voorbeeld uit hoofdstuk 2.

Tijdsduur : 30 minuten
Van : 14:11:09
Tot : 14:41:14
Aantal verschillende gebruikers : 613
Aantal uitgevoerde queries : 1.302.726
Query Duration > 10 seconden : 77



Afbeelding 13 : De belasting van de server in het aantal uitgevoerde queries per minuut

5.2 Queries die lang duren

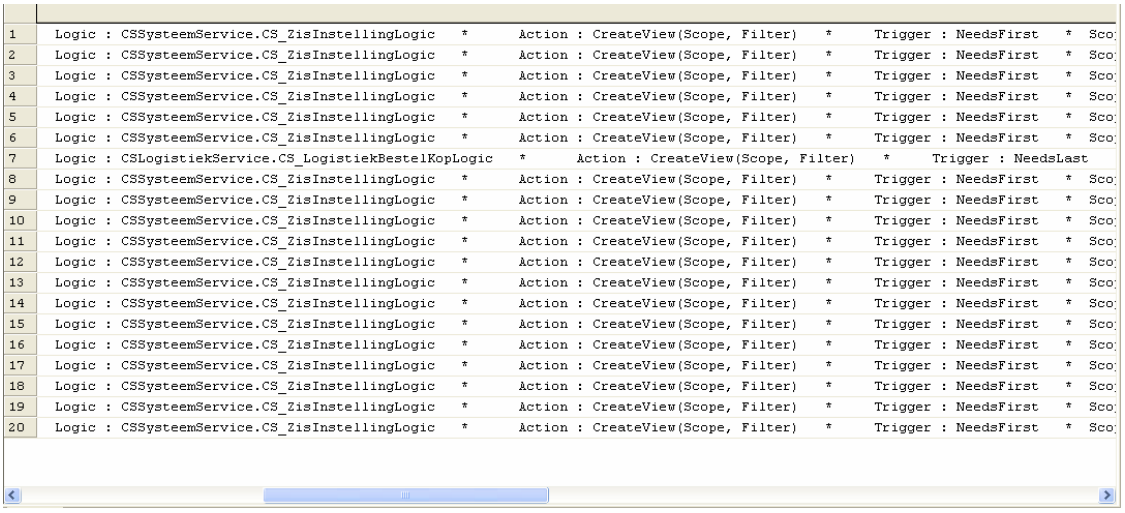
Als eerste is er onderzoek gedaan om queries te vinden die lang duren. Hiervoor is onderzoek gedaan of er tools waren die hierbij konden helpen, zoals te lezen was in hoofdstuk 4. Uit dit onderzoek is als resultaat gekomen dat er gebruik gemaakt kon worden van tools van Quest. Het betrof hier Quest Speed Coefficient. Met dit tool kunnen statistieken uit trace logs gehaald worden.

Gebruik maken van het tool Quest Speed Coefficient
Bij het willen gebruiken van Speed Coefficient bleek dat dit niet werkte met bestaande trace-logs. Het tool werkt alleen met trace-logs die het ook zelf gemaakt heeft, en niet met trace-logs die zijn gemaakt met SQL Profiler van Microsoft. Er is nog op verschillende manieren geprobeerd om een huidige trace-log te laten analyseren door Speed Coefficient, maar deze pogingen waren tevergeefs.

Gebruik maken van SQL Statement
Om toch de lang durende queries uit de trace-logs te halen, is er een eigen SQL statement geschreven.

```
SELECT TOP 20 *
FROM trace_file
ORDER BY Duration DESC
```

Dit SQL statement geeft de top durende queries terug. Als de 20 langst durende queries worden opgevraagd uit de trace-loge, dan is te zien dat 19 van deze queries exact hetzelfde zijn (Afbeelding 14). Op deze manier is niet te zien of er ook nog andere queries zijn die lang duren, maar net iets korter duren dan de nu 20 weergegeven queries. Er kan voor gekozen worden om in plaats van 20 meerdere queries weer te geven. Bij het weergeven van 40 queries is het echter nog steeds zo dat 39 queries hetzelfde zijn.
Op deze manier is niet duidelijk welke verschillende queries tot de top durende queries behoren. Er kan voor gekozen worden om bijvoorbeeld de top 100 weer te geven. Hierbij bestaat echter de mogelijkheid dat het probleem hetzelfde blijft, of dat er zoveel verschillende queries door elkaar worden gegeven dat er geen overzicht meer in gezien kan worden.



1	Logic : CSSysteemService.CS_ZisInstellingLogic * Action : CreateView(Scope, Filter) * Trigger : NeedsFirst * Sco
2	Logic : CSSysteemService.CS_ZisInstellingLogic * Action : CreateView(Scope, Filter) * Trigger : NeedsFirst * Sco
3	Logic : CSSysteemService.CS_ZisInstellingLogic * Action : CreateView(Scope, Filter) * Trigger : NeedsFirst * Sco
4	Logic : CSSysteemService.CS_ZisInstellingLogic * Action : CreateView(Scope, Filter) * Trigger : NeedsFirst * Sco
5	Logic : CSSysteemService.CS_ZisInstellingLogic * Action : CreateView(Scope, Filter) * Trigger : NeedsFirst * Sco
6	Logic : CSSysteemService.CS_ZisInstellingLogic * Action : CreateView(Scope, Filter) * Trigger : NeedsFirst * Sco
7	Logic : CSLogistiekService.CS_LogistiekBestelKopLogic * Action : CreateView(Scope, Filter) * Trigger : NeedsLast
8	Logic : CSSysteemService.CS_ZisInstellingLogic * Action : CreateView(Scope, Filter) * Trigger : NeedsFirst * Sco
9	Logic : CSSysteemService.CS_ZisInstellingLogic * Action : CreateView(Scope, Filter) * Trigger : NeedsFirst * Sco
10	Logic : CSSysteemService.CS_ZisInstellingLogic * Action : CreateView(Scope, Filter) * Trigger : NeedsFirst * Sco
11	Logic : CSSysteemService.CS_ZisInstellingLogic * Action : CreateView(Scope, Filter) * Trigger : NeedsFirst * Sco
12	Logic : CSSysteemService.CS_ZisInstellingLogic * Action : CreateView(Scope, Filter) * Trigger : NeedsFirst * Sco
13	Logic : CSSysteemService.CS_ZisInstellingLogic * Action : CreateView(Scope, Filter) * Trigger : NeedsFirst * Sco
14	Logic : CSSysteemService.CS_ZisInstellingLogic * Action : CreateView(Scope, Filter) * Trigger : NeedsFirst * Sco
15	Logic : CSSysteemService.CS_ZisInstellingLogic * Action : CreateView(Scope, Filter) * Trigger : NeedsFirst * Sco
16	Logic : CSSysteemService.CS_ZisInstellingLogic * Action : CreateView(Scope, Filter) * Trigger : NeedsFirst * Sco
17	Logic : CSSysteemService.CS_ZisInstellingLogic * Action : CreateView(Scope, Filter) * Trigger : NeedsFirst * Sco
18	Logic : CSSysteemService.CS_ZisInstellingLogic * Action : CreateView(Scope, Filter) * Trigger : NeedsFirst * Sco
19	Logic : CSSysteemService.CS_ZisInstellingLogic * Action : CreateView(Scope, Filter) * Trigger : NeedsFirst * Sco
20	Logic : CSSysteemService.CS_ZisInstellingLogic * Action : CreateView(Scope, Filter) * Trigger : NeedsFirst * Sco

Afbeelding 14 : Top 20 langst durende queries, waarvan 19 keer dezelfde query

Gebruik maken van een eigen tool

Om het probleem met het SQL statement te voorkomen, is er een eigen tool geschreven. Dit tool (Afbeelding 15) doorloopt een trace-log en slaat per SQL statement de Duration en CPU tijden op. Als met behulp van het tool de langst durende queries opgevraagd worden, dan worden verschillende queries gegeven, en niet dezelfde. Op deze manier is snel duidelijk welke verschillende queries lang duren. Indien een query vaker voorkomt, dan worden de overige Duration en CPU tijden in een aparte lijst in het tool weergegeven.

Omschrijving	**Waarde**
Top Count	96
Highest CPU	437
Lowest CPU	188
Highest Duration	*16626*
Lowest Duration	1233

Tabel 3 : Analyse gegevens van de query met de hoogste Duration

Omschrijving	**Waarde**
Top Count	1
Highest CPU	*1469*
Lowest CPU	-
Highest Duration	1670
Lowest Duration	-

Tabel 4 : Analyse gegevens van de query met de hoogste CPU

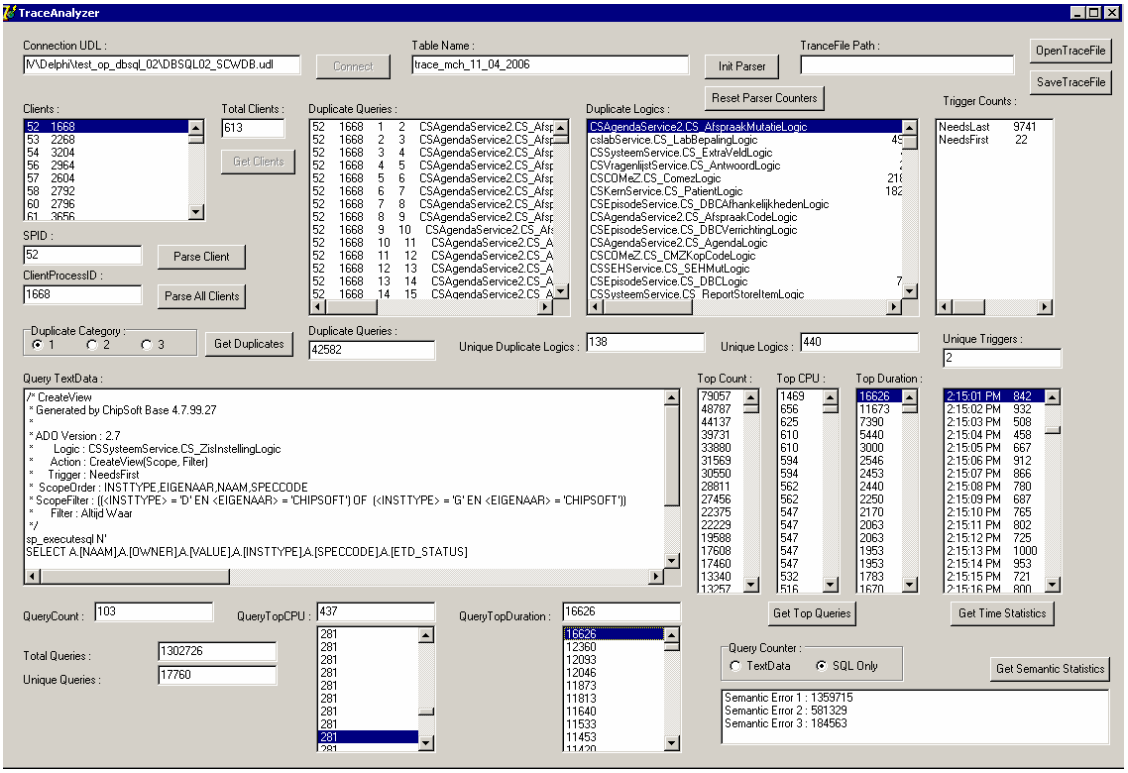
In Tabel 3 zijn de analyse uitkomsten van het tool weergegeven voor de query met de hoogste Duration. In Tabel 4 staan de analyse gegevens van de query met de hoogste CPU tijd.

5.2.1 Hoogste Duration

```
/* CreateView
 * Generated by ChipSoft Base 4.7.99.27
 *
 * ADO Version : 2.8
 *       Logic : CSSysteemService.CS_ZisInstellingLogic
 *      Action : CreateView(Scope, Filter)
 *     Trigger : NeedsFirst
 *  ScopeOrder : INSTTYPE,EIGENAAR,NAAM,SPECCODE
 * ScopeFilter : ((<INSTTYPE> = 'D' EN <EIGENAAR> = 'CHIPSOFT') OF  (<INSTTYPE> = 'G' EN
<EIGENAAR> = 'CHIPSOFT'))
 *      Filter : Altijd Waar
 */
sp_executesql N'
SELECT A.[NAAM],A.[OWNER],A.[VALUE],A.[INSTTYPE],A.[SPECCODE],A.[ETD_STATUS]
FROM [dbo].[CONFIG_INSTVARS] AS A
WHERE (((A.[INSTTYPE] = @SV1 AND A.[OWNER] LIKE @SV2+''%'') OR  (A.[INSTTYPE] = @SV3
AND A.[OWNER] LIKE @SV2+''%'')) AND NOT ((A.[INSTTYPE] = @SV1 AND A.[OWNER] = @SV2
AND (A.[NAAM] LIKE @SV4+''%'' OR  A.[NAAM] LIKE @SV5+''%'' OR  A.[NAAM] LIKE @SV6+''%''
OR  A.[NAAM] LIKE @SV7+''%'' OR  A.[NAAM] LIKE @SV8+''%'' OR  A.[NAAM] LIKE @SV9+''%''
OR  A.[NAAM] LIKE @SV10+''%''))))
ORDER BY A.[INSTTYPE] ASC,A.[OWNER] ASC,A.[NAAM] ASC,A.[SPECCODE] ASC
OPTION (KEEP PLAN, KEEPFIXED PLAN, LOOP JOIN)
', N'@SV1 nvarchar(1),@SV2 nvarchar(8),@SV3 nvarchar(1),@SV4 nvarchar(16),@SV5
nvarchar(16),@SV6 nvarchar(16),@SV7 nvarchar(16),@SV8 nvarchar(16),@SV9 nvarchar(16),@SV10
nvarchar(16)',
@SV1='D',@SV2='CHIPSOFT',@SV3='G',@SV4='OPMAAK_BROWSE',@SV5='OPMAAK_INVOER',@
SV6='OVZGENRESULTATEN',@SV7='OPN_GRFBEDBEZ',@SV8='OPN_OPMOPNSCRN',@SV9='AG
N_OP',@SV10='AG_OP'
```

De query met de hoogste duration is bovenstaande query. De query is verantwoordelijk voor het ophalen van de instellingen van een gebruiker bij het opstarten van het EZIS. Deze query komt 103 keer voor. Als wordt gekeken naar het totaal aantal uitgevoerde queries, dan zal, als over het gehele systeem gekeken wordt, zo goed als geen performance winst geboekt worden indien deze query geoptimaliseerd wordt. Daarbij komt ook nog dat deze query niet elke keer zo een hoge duration tijd heeft.

Als we in de trace-log kijken naar het aantal queries die langer duren dan 10 sec, dan zijn dit er 77. Als we dit getal uitzetten tegen de 613 gebruikers, dan betekent dit dat op tijd van een half uur 12,56% van alle gebruikers een moment heeft gehad waarop deze 10 seconden of langer moest wachten. Dit is een aardig percentage, en het niet meer voorkomen van deze queries zou dan ook ten goede komen van het systeem.



Afbeelding 15 : TraceAnalyzer tool – Highest Duration query

Probleem oorzaak

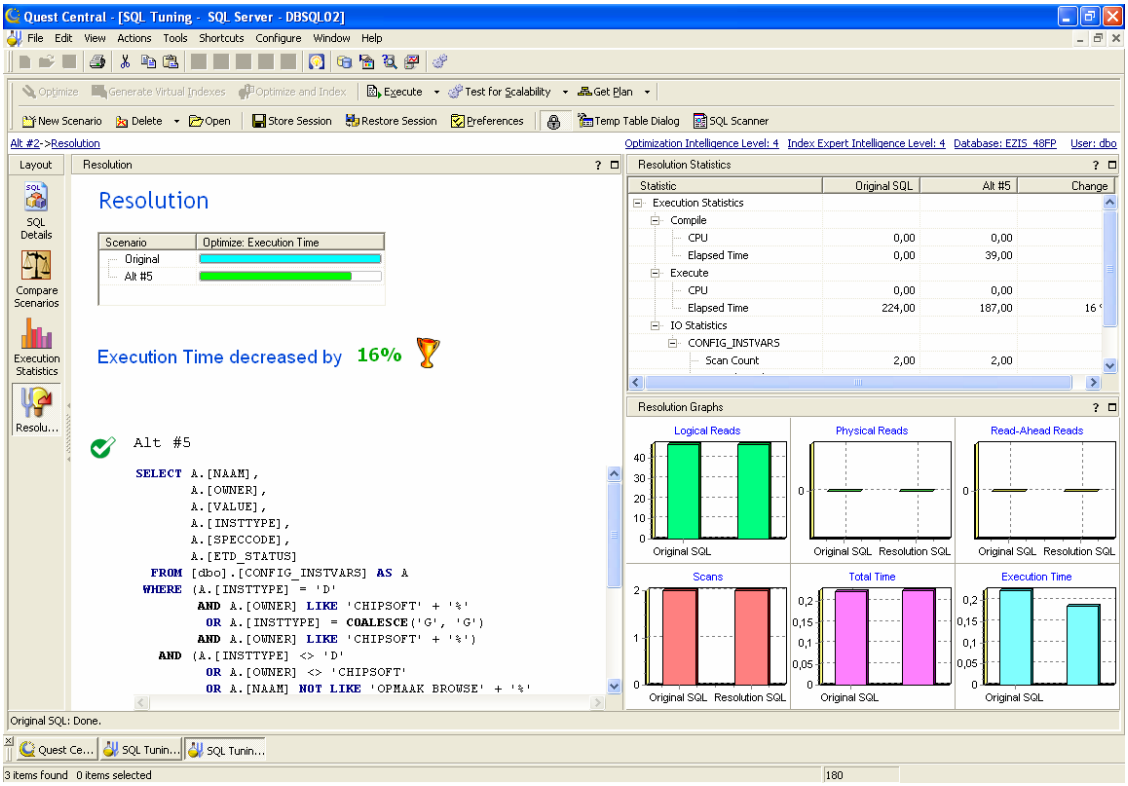
In Tabel 3 is te zien dat er een groot verschil zit tussen de hoogste duration en de laagste duration van deze query. Dit zijn respectievelijk 16626 en 1233 milliseconden. Dit wil zeggen dat de oorzaak van het zo lang duren van de query niet aan het executieplan van de query zelf ligt, maar dat er een andere oorzaak is.
Ook is te zien dat de CPU tijd die wordt gebruikt door deze query beneden de 500 milliseconden is. De query gebruikt dus ook niet veel CPU waardoor deze ook langer duurt. Er is dus een andere oorzaak voor het zo lang duren van deze query. Dit zou kunnen liggen aan de drukte op de server op het moment dat de query werd uitgevoerd, of dat er een geval van blocking is opgetreden.

Om te kijken of de oorzaak aan de drukte op de server ligt, is het tool uitgebreid zodat een lijst kan worden weergegeven met het aantal queries die per seconde worden uitgevoerd. Indien op de verschillende duration tijden wordt geklikt, wordt er in de lijst naar het tijdstip gesprongen waarop de query is uitgevoerd. Zo kan worden gezien hoeveel queries in die betreffende seconde, en eventueel ervoor en erna zijn uitgevoerd. Bij het nalopen van de verschillende duration tijden en het daarbij horende aantal queries voor dat tijdstip was echter geen patroon te herkennen. Queries met een hoge duration kwamen zowel voor op tijdstippen met een zware query belasting als op tijdstippen met een lage belasting. De drukte van de server is dus niet de oorzaak van de lang durende queries. Er zal in dit geval dus blocking zijn opgetreden.

Optimaliseren Query

Ondanks dat het probleem van het zo lang duren van de query niet aan de query zelf ligt, is de laagste duration toch nog steeds 1233 milliseconden, en duurt dus vrij lang. Er is daarom geprobeerd om de query met behulp van Quest Central SQL Tuning te optimaliseren. Hierbij werden door SQL Tuning nieuwe executieplannen gegenereerd voor de query. Deze werden vervolgens meermaals achter elkaar uitgevoerd om te testen of er een optimaler executieplan gegenereerd was. Het meermaals achter elkaar laten uitvoeren is van belang om een valide resultaat te verkrijgen. Bij het eerste keer uitvoeren van een query moet deze eerst gecompileerd worden en kost dus meer tijd. Dit is dus niet valide om te vergelijken. Nadat een query een keer is uitgevoerd, is deze in het cache van de server gezet, en hoeft niet opnieuw gecompileerd te worden bij een tweede en derde keer van uitvoeren.

Quest Central SQL Tuning kon een alternatief SQL statement genereren. Met dit alternatief SQL statement werd een optimalisatie behaald van 16% (Afbeelding 16), tegenover het originele SQL statement.



Afbeelding 16 : Quest Central SQL Tuning

5.2.2 Hoogste CPU

```
/* CreateView
 * Generated by ChipSoft Base 4.7.99.27
 *
 * ADO Version : 2.8
 *      Logic : CSVragenlijstService.CS_VragenBeantwoordingLogic
 *     Action : CreateView(Scope, Filter)
     *     Trigger : NeedsFirst
 *  ScopeOrder : KoppelObjectID,PatientNummer,Datum
 * ScopeFilter : ((<KoppelObjectID> = '0000001135' EN <PatientNummer> = '6050359')
OF  (<KoppelObjectID> = '0000001135' EN <PatientNummer> = '9097319') OF  (<KoppelObjectID> =
'0000001135' EN <PatientNummer> = '4861659') OF  (<KoppelObjectID> = '0000001135' EN
<PatientNummer> = '1088804') OF  (<KoppelObjectID> = '0000001136' EN <PatientNummer> =
'6050359') OF  (<KoppelObjectID> = '0000001136' EN <PatientNummer> = '9097319')
OF  (<KoppelObjectID> = '0000001136' EN <PatientNummer> = '4861659') OF  (<KoppelObjectID> =
'0000001136' EN <PatientNummer> = '1088804') OF  (<KoppelObjectID> = '0000001154' EN
<PatientNummer> = '6050359') OF  (<KoppelObjectID> = '0000001154' EN <PatientNummer> =
'9097319') OF  (<KoppelObjectID> = '0000001154' EN <PatientNummer> = '4861659')
OF  (<KoppelObjectID> = '0000001154' EN <PatientNummer> = '1088804') OF  (<KoppelObjectID> =
'' EN <PatientNummer> = '6050359') OF  (<KoppelObjectID> = '' EN <PatientNummer> = '9097319')
OF  (<KoppelObjectID> = '' EN <PatientNummer> = '4861659') OF  (<KoppelObjectID> = '' EN
<PatientNummer> = '1088804') OF  (<KoppelObjectID> = '0000001153' EN <PatientNummer> =
'6050359') OF  (<KoppelObjectID> = '0000001153' EN <PatientNummer> = '9097319')
OF  (<KoppelObjectID> = '0000001153' EN <PatientNummer> = '4861659') OF  (<KoppelObjectID> =
'0000001153' EN <PatientNummer> = '1088804') OF  (<KoppelObjectID> = '0000001177' EN
<PatientNummer> = '6050359') OF  (<KoppelObjectID> = '0000001177' EN <PatientNummer> =
'9097319') OF  (<KoppelObjectID> = '0000001177' EN <PatientNummer> = '4861659')
OF  (<KoppelObjectID> = '0000001177' EN <PatientNummer> = '1088804') OF  (<KoppelObjectID> =
'0000001151' EN <PatientNummer> = '6050359') OF  (<KoppelObjectID> = '0000001151' EN
<PatientNummer> = '9097319') OF  (<KoppelObjectID> = '0000001151' EN <PatientNummer> =
'4861659') OF  (<KoppelObjectID> = '0000001151' EN <PatientNummer> = '1088804'))
 *     Filter : Altijd Waar
 */
sp_executesql N'
SELECT TOP 68
A.[OPSLAGID],A.[COMMENTAAR],A.[LIJSTID],A.[CATEGORIE],A.[GEBRUIKER],A.[DATUM],A.[TI
JD],A.[DEFINITIEF],A.[COMPLEET],A.[KOPPELID],A.[OBJECTID],A.[PATNR],A.[LAYOUTTYPE],A.
[MUTDAT],A.[MUTTIJD]
FROM [dbo].[VRLIJST_LSTOPSLG] AS A
WHERE ((A.[KOPPELID] = @SV1 AND A.[PATNR] LIKE @SV2+''%'') OR  (A.[KOPPELID] = @SV1
AND A.[PATNR] LIKE @SV3+''%'') OR  (A.[KOPPELID] = @SV1 AND A.[PATNR] LIKE @SV4+''%'')
OR  (A.[KOPPELID] = @SV1 AND A.[PATNR] LIKE @SV5+''%'') OR  (A.[KOPPELID] = @SV6 AND
A.[PATNR] LIKE @SV2+''%'') OR  (A.[KOPPELID] = @SV6 AND A.[PATNR] LIKE @SV3+''%'')
OR  (A.[KOPPELID] = @SV6 AND A.[PATNR] LIKE @SV4+''%'') OR  (A.[KOPPELID] = @SV6 AND
A.[PATNR] LIKE @SV5+''%'') OR  (A.[KOPPELID] = @SV7 AND A.[PATNR] LIKE @SV2+''%'')
OR  (A.[KOPPELID] = @SV7 AND A.[PATNR] LIKE @SV3+''%'') OR  (A.[KOPPELID] = @SV7 AND
A.[PATNR] LIKE @SV4+''%'') OR  (A.[KOPPELID] = @SV7 AND A.[PATNR] LIKE @SV5+''%'')
OR  ((A.[KOPPELID] IS NULL OR A.[KOPPELID] LIKE '''') AND A.[PATNR] LIKE @SV2+''%'')
OR  ((A.[KOPPELID] IS NULL OR A.[KOPPELID] LIKE '''') AND A.[PATNR] LIKE @SV3+''%'')
OR  ((A.[KOPPELID] IS NULL OR A.[KOPPELID] LIKE '''') AND A.[PATNR] LIKE @SV4+''%'')
OR  ((A.[KOPPELID] IS NULL OR A.[KOPPELID] LIKE '''') AND A.[PATNR] LIKE @SV5+''%'')
OR  (A.[KOPPELID] = @SV8 AND A.[PATNR] LIKE @SV2+''%'') OR  (A.[KOPPELID] = @SV8 AND
A.[PATNR] LIKE @SV3+''%'') OR  (A.[KOPPELID] = @SV8 AND A.[PATNR] LIKE @SV4+''%'')
OR  (A.[KOPPELID] = @SV8 AND A.[PATNR] LIKE @SV5+''%'') OR  (A.[KOPPELID] = @SV9 AND
```

```
A.[PATNR] LIKE @SV2+"%") OR  (A.[KOPPELID] = @SV9 AND A.[PATNR] LIKE @SV3+"%")
OR  (A.[KOPPELID] = @SV9 AND A.[PATNR] LIKE @SV4+"%") OR  (A.[KOPPELID] = @SV9 AND
A.[PATNR] LIKE @SV5+"%") OR  (A.[KOPPELID] = @SV10 AND A.[PATNR] LIKE @SV2+"%")
OR  (A.[KOPPELID] = @SV10 AND A.[PATNR] LIKE @SV3+"%") OR  (A.[KOPPELID] = @SV10
AND A.[PATNR] LIKE @SV4+"%") OR  (A.[KOPPELID] = @SV10 AND A.[PATNR] LIKE
@SV5+"%"))
ORDER BY A.[KOPPELID] ASC,A.[PATNR] ASC,A.[DATUM] ASC,A.[OPSLAGID] ASC
OPTION (KEEP PLAN, KEEPFIXED PLAN, LOOP JOIN)
', N'@SV1 nvarchar(10),@SV2 nvarchar(13),@SV3 nvarchar(13),@SV4 nvarchar(13),@SV5
nvarchar(13),@SV6 nvarchar(10),@SV7 nvarchar(10),@SV8 nvarchar(10),@SV9 nvarchar(10),@SV10
nvarchar(10)',
@SV1='0000001135',@SV2='6050359',@SV3='9097319',@SV4='4861659',@SV5='1088804',@SV6='00
00001136',@SV7='0000001154',@SV8='0000001153',@SV9='0000001177',@SV10='0000001151'
```

De query met de hoogste CPU tijd is bovenstaande query, en zij komt maar één keer voor. De query haalt antwoorden op bij een vragenlijst van een patiënt. De hoge CPU tijd kan verklaard worden doordat de query maar één keer voorkomt, en daardoor dus eerst nog gecompileerd moet worden omdat deze niet in het cache staat. Toch ziet de query er vrij complex uit, en is er geprobeerd deze te optimaliseren.

Optimaliseren Query

Ook bij het proberen te optimaliseren van deze query is er gebruik gemaakt van Quest Central SQL Tuning. De alternatieven die werden gegenereerd konden echter niet worden uitgevoerd door SQL Tuning. Er werd de volgende foutmelding gegeven : “Query Cancelled, exceeded benchmark”. Er is vervolgens geprobeerd om de query handmatig te optimaliseren.

In de query is te zien dat er vaak dezelfde vergelijking worden gemaakt van een veld. Deze dubbele vergelijkingen zijn er uitgehaald, en leverde onderstaande query op. Bij het testen van beide queries bleek dat de aangepaste versie een stuk efficiënter was dan de originele query. In de testomgeving had de originele query een duration van 1280 milliseconden. Voor de aangepaste versie was dit 186. Dit is een optimalisatie van 85%.

```
sp_executesql N'
SELECT TOP 68
A.[OPSLAGID],A.[COMMENTAAR],A.[LIJSTID],A.[CATEGORIE],A.[GEBRUIKER],A.[DATUM],A.[TI
JD],A.[DEFINITIEF],A.[COMPLEET],A.[KOPPELID],A.[OBJECTID],A.[PATNR],A.[LAYOUTTYPE],A.
[MUTDAT],A.[MUTTIJD]
FROM [dbo].[VRLIJST_LSTOPSLG] AS A
WHERE ((A.[KOPPELID] = @SV1 OR A.[KOPPELID] = @SV6 OR A.[KOPPELID] = @SV7 OR
(A.[KOPPELID] IS NULL OR A.[KOPPELID] LIKE "") OR A.[KOPPELID] = @SV8 OR A.[KOPPELID]
= @SV9 OR A.[KOPPELID] = @SV10) AND ((A.[PATNR] LIKE @SV2+"%") OR  (A.[PATNR] LIKE
@SV3+"%") OR  (A.[PATNR] LIKE @SV4+"%") OR  (A.[PATNR] LIKE @SV5+"%")))
ORDER BY A.[KOPPELID] ASC,A.[PATNR] ASC,A.[DATUM] ASC,A.[OPSLAGID] ASC
OPTION (KEEP PLAN, KEEPFIXED PLAN, LOOP JOIN)
', N'@SV1 nvarchar(10),@SV2 nvarchar(13),@SV3 nvarchar(13),@SV4 nvarchar(13),@SV5
nvarchar(13),@SV6 nvarchar(10),@SV7 nvarchar(10),@SV8 nvarchar(10),@SV9 nvarchar(10),@SV10
nvarchar(10)',
@SV1='0000001135',@SV2='6050359',@SV3='9097319',@SV4='4861659',@SV5='1088804',@SV6='00
00001136',@SV7='0000001154',@SV8='0000001153',@SV9='0000001177',@SV10='0000001151'
```

5.3 Queries die vaak voorkomen

Als tweede is er onderzoek gedaan naar queries die vaak voorkomen. Ook hierbij was het de bedoeling om gebruik te maken van Quest Speed Coefficient, maar zoals uit eerder onderzoek bleek is dit niet mogelijk.

Gebruik maken van SQL Statement
Een query wordt geïdentificeerd doormiddel van de TextData kolom. Bij het willen kijken hoeveel verschillende queries er zijn en welke er vaak voorkomen, blijkt het niet mogelijk te zijn om een *Distinct* of *Group By* uit te voeren op de TextData kolom. SQL Server staat dit niet toe op kolommen van het type ntext, wat het type is van de TextData kolom.

```
SELECT COUNT(TextData)
FROM trace_mch_11_04_2006
GROUP BY TextData
```

```
Server: Msg 306, Level 16, State 2, Line 1
The text, ntext, and image data types cannot be compared or sorted, except when using IS NULL or LIKE operator.
```

Door het werken met SQL en het meer inzicht krijgen hierin, bleek later dat het toch mogelijk was om met behulp van een SQL statement te kijken welke query het vaakste voorkomt. Dit is mogelijk met behulp van de functie SUBSTRING, waarmee een gedeelte of het hele TextData veld als resultaat gegeven kan worden. Dit resultaat kan wel gebruikt worden in *Group By*.

```
SELECT COUNT(*), SUBSTRING(TextData,CHARINDEX('SELECT',TextData),
(CHARINDEX('OPTION',TextData) - CHARINDEX('SELECT',TextData)))
FROM trace_file
WHERE  CHARINDEX('OPTION',TextData) <> 0
GROUP BY SUBSTRING(TextData,CHARINDEX('SELECT',TextData),
(CHARINDEX('OPTION',TextData) - CHARINDEX('SELECT',TextData)))
ORDER BY COUNT(*)DESC
```

Op het moment dat dit het geval was, was echter al het eigen tool aangepast om deze queries te vinden. Er kon toen met behulp van het SQL statement de resultaten van het eigen tool, zoals hieronder beschreven staat, gevalideerd worden (Afbeelding 17). De resultaten van het tool en het SQL statement waren beide gelijk aan elkaar. Er van uitgaande dat het SQL statement correct is, kan geconcludeerd worden dat het eigen tool ook correct werkt.

	(No column name)	(No column name)
1	79057	SELECT A.[PATIENTNR],A.[ACHTERNAAM],A.[VOORVOEGA],A.[MEISJESNAA],A.[VOORVOEGM],A.[VOORLETTER],A.[ROEPNAAM],A....
2	48787	SELECT A.[ARTSCODE],A.[ZOEKCODE],A.[ZORGVSOORT],A.[SIGCODE],A.[LISZCODE],A.[ARTSTYPE],A.[SPECIALISM],A.[ARTSG...
3	44137	SELECT A.[VRAAGSTELL],A.[OBJECTID01],A.[ANTWOORD],A.[UITBEHANDE],A.[VERPLACODE],A.[LAATSTEAFS],A.[VERPLAATST]...
4	39731	SELECT TOP 20 A.[Pakket],A.[Versie],A.[Bestand],A.[Naam],A.[VeldNaam],A.[VolgNr],A.[Omschrijv],A.[VeldDef],A....
5	33880	SELECT A.[SUBAGENDA],A.[AGENDA],A.[NAAM],A.[VOLGNUMMER],A.[OMSCHR],A.[LOCATIE],A.[ARTSCODE],A.[LENER],A.[UNIT...
6	31569	SELECT A.[VRAAGID],A.[STELLING],A.[CONTROLID],A.[KORTEOMS],A.[EENHEIDCD],A.[KEUZELIJST],A.[CATEGORIE],A.[VERV...
7	30550	SELECT A.[KAMERCODE],A.[BEDNR],A.[BEZET],A.[GEBLOKKEER],A.[GERESERVEE],A.[USERCODE],A.[BLOKOMSCHR],A.[SEHNUMM...
8	28811	SELECT A.[SEHID],A.[PATIENTNR],A.[BEGSTATUS],A.[EXTRAINFO],A.[DOORWIE],A.[KLACHT],A.[URGENTIE],A.[CONCLUSIE],...
9	27456	SELECT TOP 20 A.[Pakket],A.[Versie],A.[Naam],A.[Alias],A.[DirAlias],A.[Omschrijv],A.[Type],A.[Driver],A.[GUID...
10	22375	SELECT A.[TREELAYID],A.[CHILDID],A.[VOLGNR],A.[TYPE],A.[LIJSTID],A.[CTRLCFG],A.[NEWPAGE],A.[ONZZANTW],A.[LSTO...
11	22229	SELECT TOP 128 A.[FROM_TYPE],A.[FROM_ID],A.[TO_TYPE],A.[TO_ID],A.[DELETE],A.[POST],A.[FROM_SUBTP],A.[AutoID] ...
12	19588	SELECT TOP 102 A.[Pakket],A.[Versie],A.[Bestand],A.[Volgorde],A.[Naam],A.[Tagname],A.[Omschrijv],A.[Key],A.[D...
13	17608	SELECT A.[OPSLAGID],A.[BEANTWID],A.[ITEMID],A.[ANTWOORD],A.[PARENTANTW],A.[GEBRUIKER],A.[DATUM],A.[TIJD],A.[R...

Afbeelding 17 : Top Count Queries bij vergelijking van alleen het SQL Statement

Gebruik maken van een eigen tool

Om toch te zien welke queries vaak voorkomen is in eerste instantie het eigen tool uitgebreid. Als het tool een trace-log doorloopt, wordt er per query opgeslagen hoe vaak deze voorkomt. Zo kan met het tool gezien worden hoe vaak welke query voorkomt. Er kan onderscheid gemaakt worden tussen de vergelijking van het hele TextData veld (Afbeelding 9), of alleen het SQL statement zonder de vergelijking van het commentaar en de variable waarden.

5.3.1 Vergelijking van de TextData

Bij de vergelijking van het hele TextData veld wordt er gekeken naar hoe vaak een exact dezelfde query voorkomt. Waarbij het commentaar en de variable waarden van de query dus ook gelijk zijn. Dit levert 330.265 unieke queries op. De query die hierbij het vaakst voorkomt is onderstaande query. Omdat de variabelen steeds gelijk zijn, zal de query altijd hetzelfde resultaat geven. Als deze resultaten gecached worden in het EZIS, dan hoeft de query maar één keer uitgevoerd te worden. Indien de resultaten een volgende keer benodigd zijn, dan kunnen deze uit de cache gehaald worden, en hoeft de query niet opnieuw uitgevoerd te worden. Dit wordt dan ook in de volgende versie van het EZIS gerealiseerd. Zoals te zien is in Tabel 5 zal dit +-1.45% minder queries opleveren.

```
/* CreateView
 * Generated by ChipSoft Base 4.7.99.27
 *
 * ADO Version : 2.8
 *       Logic : CSSysteemService.CS_DDBestandLogic
 *      Action : CreateView(Scope, Filter)
 *     Trigger : NeedsLast
 *  ScopeOrder : GUIDString
 * ScopeFilter : <GUIDString> = '{0DA19A56-DC81-4B2B-9F47-8856A95E4674}'
 *      Filter : Altijd Waar
 */
sp_executesql N'
SELECT TOP 20
A.[Pakket],A.[Versie],A.[Naam],A.[Alias],A.[DirAlias],A.[Omschrijv],A.[Type],A.[Driver],A.[GUIDStrin
g],A.[Soort],A.[MutatieLog],A.[OBJOMSEXPR]
FROM [dbo].[DD_BESTANDE] AS A
WHERE A.[GUIDString] LIKE @SV1+"%"
ORDER BY A.[GUIDString] DESC,A.[Pakket] DESC,A.[Versie] DESC,A.[Naam] DESC
```

OPTION (KEEP PLAN, KEEPFIXED PLAN, LOOP JOIN)
', N'@SV1 nvarchar(38)',
@SV1='{0DA19A56-DC81-4B2B-9F47-8856A95E4674}'

Omschrijving	**Waarde**
Top Count	*18968 (1.45%)*
Highest CPU	16
Lowest CPU	0
Highest Duration	16
Lowest Duration	0

Tabel 5 : Analyse gegevens van de query met de hoogste Count (TextData)

5.3.2 Vergelijking van het SQL Statement

Bij het vergelijken van alleen het SQL Statement worden 17760 unieke queries geïdentificeerd. De query die het vaakste voorkomt is onderstaande query, en deze komt 79057 keer voor. Dit is slechts 6% van het totaal aantal queries. Deze query haalt de gegevens van een patiënt op aan de hand van het gegeven patiëntnummer.

SELECT
A.[PATIENTNR],A.[ACHTERNAAM],A.[VOORVOEGA],A.[MEISJESNAA],A.[VOORVOEGM],A.[VOOR
LETTER],A.[ROEPNAAM],A.[VOORNAAM],A.[TITELS],A.[GESLACHT],A.[GEBDAT],A.[GEBPLAATS]
,A.[GEBLAND],A.[INSTANTIE],A.[MERGED],A.[BSN],A.[TELEFOON1],A.[TYPETEL1],A.[TELEFOON
2],A.[TYPETEL2],A.[HUISARTS],A.[TANDARTS],A.[APOTHEEK],A.[VERZVORM],A.[IDDATUM],A.[I
DTYPE],A.[IDNUMMER],A.[HOOFDVERZ],A.[INGANGSDAT],A.[REKENINGNR],A.[DEBITEURNR],
A.[PATTYPE],A.[PATCODES],A.[OVERLEDEN],A.[OVERLDAT],A.[MEERLING],A.[MEMO],A.[BURG
STAAT],A.[TAAL],A.[RELIGIE],A.[GEESTVERZ],A.[PONSDAT],A.[PONSAANT],A.[TMPREG],A.[MUT
DAT],A.[MUTTIJD],A.[MUTWIE],A.[MUTBRON],A.[KOPPELNR],A.[KOPPELSTAT],A.[ADRES],A.[H
UISNR],A.[POSTCODE],A.[WOONPLAATS],A.[LAND],A.[PARTVERZ],A.[AANVVERZ],A.[NRZIEKENF
O],A.[NRPARTVERZ],A.[NRAANVVERZ],A.[KLASSEZIEK],A.[KLASSEPART],A.[KLASSEAANV],A.[ZIE
KENFOND],A.[NAAMGEBR]
FROM [dbo].[PATIENT_PATIENT] AS A
WHERE (A.[PATIENTNR] = @ID0 AND (1=1))
OPTION (KEEP PLAN, KEEPFIXED PLAN, LOOP JOIN)

Optimaliseren Query

Aan de analyse gegevens van deze query in Tabel 6 is te zien dat deze geen hoge CPU tijd heeft. De query neemt dus zo goed als geen CPU resource in beslag. De Duration van 250 kwam slechts één keer voor, en heeft daarom niets met de query zelf te maken. Zoals ook aan de query zelf te zien is, is het een simpele query met maar één vergelijking in de *Where* clausule. Het is dus niet mogelijk om de *Where* clausule aan te passen zodat de query geoptimaliseerd kan worden. De enige optimalisatie die gemaakt kan worden, is dat het patiëntnummer uit *Select* wordt gehaald, omdat dit toch al bekend is. Op deze manier hoeft er minder data terug gestuurd te worden naar de client. Indien dit wordt gedaan, dan levert dit het volgende resultaat op:

Ontvangen bytes op client met patiëntnummer : 12645
Ontvangen bytes op client zonder patiëntnummer : 12468

Dit is een verschil van 177 bytes wat overeenkomt met 1.4% winst aan data die niet verzonden hoeft te worden. Dit is slechts een heel klein percentage, en het zal de performance van het systeem niet of nauwelijks beïnvloeden.

Omschrijving	Waarde
Top Count	*79057 (6%)*
Highest CPU	16
Lowest CPU	0
Highest Duration	250
Lowest Duration	0

Tabel 6 : Analyse gegevens van de query met de hoogste Count (SQL)

5.4 Queries die dubbel worden uitgevoerd

Om dubbele queries te vinden was het van belang dat naar de queries van een enkele user werd gekeken. Om een unieke gebruiker te identificeren kon naar de volgende velden uit een trace-log gekeken worden:

- ClientProcessID : Het ProcessID van de applicatie op de client
- SPID : Het ProcessID van SQL Server voor een client
- HostName : De naam van het werkstation van de client

Meerdere clients kunnen tegelijk actief zijn op de SQL Server met hetzelfde ClientProcessID. Hierdoor was het niet mogelijk om voor een gegeven tijdstip een unieke client te identificeren aan het ClientProcessID.
Een SPID is op een gegeven tijdstip uniek. Als een client disconnect, kan een nieuwe client echter weer hetzelfde SPID krijgen.
Vanaf eenzelfde HostName kunnen meerdere clients connecten naar SQL Server. Dit is als verschillende gebruikers via terminal server naar deze host connecten en vanaf deze host gebruik maken van het EZIS.

Trace-log Veld	Aantal Unieke Gebruikers
ClientProcessID :	426
SPID :	540
HostName :	565
SPID / ClientProcessID :	613
SPID / HostName :	609
Hostname / ClientProcessID :	613
Hostname / SPID / ClientProcessID :	613

Tabel 7 : Trace MCH 11-04-2006 – Verschillend aantal unieke gebruikers

In Tabel 7 is te zien hoeveel unieke gebruikers er worden geïdentificeerd door te kijken naar (combinaties van) de verschillende velden uit de trace-log. Uit de test bleek dat de meest identieke gebruikers zijn te identificeren door een combinatie van de velden die als uitkomst 613 hebben. Er is daarom gekozen voor de combinatie van SPID en ClientProcessID. Deze zitten standaard in een trace-log template. HostName moet er extra in worden gezet. Mocht dit vergeten worden dan kan de trace-log niet geanalyseerd worden.

De combinatie van SPID en ClientProcessID is niet geheel waterdicht, om een unieke user te identificeren. De kans bestaat dat twee fysiek andere gebruikers eenzelfde ClientProcessID en eenzelfde SPID kunnen hebben in een trace-log. Er mag echter van uit worden gegaan dat de kans minimaal is dat de combinatie van twee gegenereerde ID's twee keer hetzelfde is.

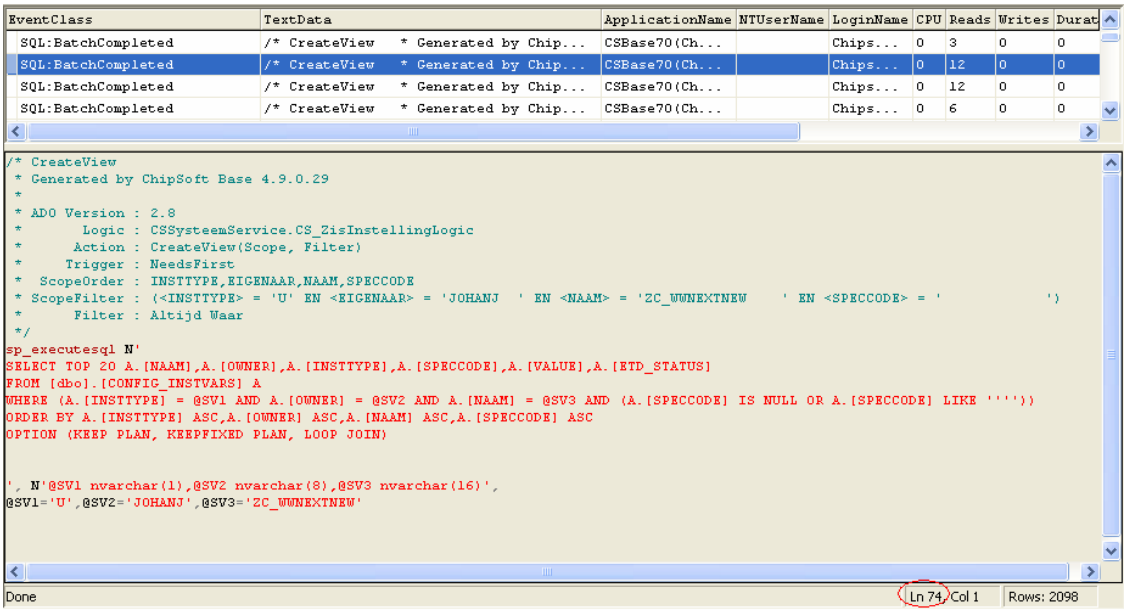
Om deze queries te vinden is er weer gebruik gemaakt van het eigen tool. Dit tool haalt eerst de unieke gebruikers uit een trace-log, zoals hierboven beschreven is. Vervolgens wordt er per gebruiker gekeken of er dubbele queries zijn uitgevoerd. Als dit het geval is, dan geeft het tool als resultaat een lijst met de gebruikers en de regelnummers waar deze dubbele queries zijn uitgevoerd.

In Tabel 8 is een voorbeeld te zien van de gegevens van dubbele queries, dat het tool als output geeft.

SPID	**ClientProcessID**	**LINE_1**	**LINE_2**
215	4320	73	74
215	4320	314	315
215	4320	510	511
215	4320	528	529
215	4320	937	938

Tabel 8 : Dubbele Queries voor gebruiker met SPID : 215 en ClientProcessID : 4320

Met behulp van de regelnummers kunnen de queries terug gevonden worden in SQL Profiler. Hiervoor moet in SQL Profiler in de eigenschappen van de trace-log de filter van het SPID en het ClientProcessID worden ingesteld voor de gegeven gebruiker. Dit om alleen de queries van deze gebruiker weer te geven. Omdat SQL Profiler als eerste regel een regel geeft met daarin 'StartTrace' moet er bij de output van het tool 1 worden opgeteld bij de gegeven regelnummers. In Afbeelding 18 en Afbeelding 19 is de dubbele query terug te zien die dan respectievelijk op regel 74 en regel 75 te vinden is, zoals in de onderstaande afbeeldingen is omcirkeld rechtsonder in de hoek.



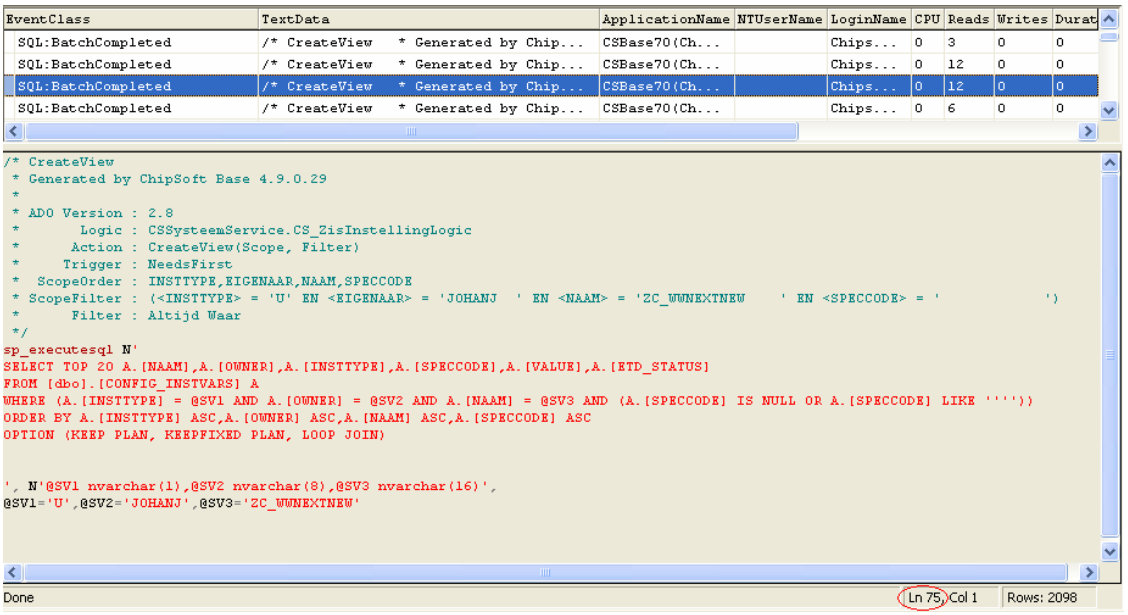
EventClass	TextData	ApplicationName	NTUserName	LoginName	CPU	Reads	Writes	Durat
SQL:BatchCompleted	/* CreateView * Generated by Chip...	CSBase70(Ch...		Chips...	0	3	0	0
SQL:BatchCompleted	/* CreateView * Generated by Chip...	CSBase70(Ch...		Chips...	0	12	0	0
SQL:BatchCompleted	/* CreateView * Generated by Chip...	CSBase70(Ch...		Chips...	0	12	0	0
SQL:BatchCompleted	/* CreateView * Generated by Chip...	CSBase70(Ch...		Chips...	0	6	0	0

```
/* CreateView
 * Generated by ChipSoft Base 4.9.0.29
 *
 * ADO Version : 2.8
 *       Logic : CSSysteemService.CS_ZisInstellingLogic
 *      Action : CreateView(Scope, Filter)
 *     Trigger : NeedsFirst
 *  ScopeOrder : INSTTYPE,EIGENAAR,NAAM,SPECCODE
 * ScopeFilter : (<INSTTYPE> = 'U' EN <EIGENAAR> = 'JOHANJ   ' EN <NAAM> = 'ZC_WWNEXTNEW     ' EN <SPECCODE> = '             ')
 *      Filter : Altijd Waar
 */
sp_executesql N'
SELECT TOP 20 A.[NAAM],A.[OWNER],A.[INSTTYPE],A.[SPECCODE],A.[VALUE],A.[ETD_STATUS]
FROM [dbo].[CONFIG_INSTVARS] A
WHERE (A.[INSTTYPE] = @SV1 AND A.[OWNER] = @SV2 AND A.[NAAM] = @SV3 AND (A.[SPECCODE] IS NULL OR A.[SPECCODE] LIKE ''''))
ORDER BY A.[INSTTYPE] ASC,A.[OWNER] ASC,A.[NAAM] ASC,A.[SPECCODE] ASC
OPTION (KEEP PLAN, KEEPFIXED PLAN, LOOP JOIN)


', N'@SV1 nvarchar(1),@SV2 nvarchar(8),@SV3 nvarchar(16)',
@SV1='U',@SV2='JOHANJ',@SV3='ZC_WWNEXTNEW'
```

Done Ln 74, Col 1 Rows: 2098

Afbeelding 18 : Dubbele Query 1



EventClass	TextData	ApplicationName	NTUserName	LoginName	CPU	Reads	Writes	Durat
SQL:BatchCompleted	/* CreateView * Generated by Chip...	CSBase70(Ch...		Chips...	0	3	0	0
SQL:BatchCompleted	/* CreateView * Generated by Chip...	CSBase70(Ch...		Chips...	0	12	0	0
SQL:BatchCompleted	/* CreateView * Generated by Chip...	CSBase70(Ch...		Chips...	0	12	0	0
SQL:BatchCompleted	/* CreateView * Generated by Chip...	CSBase70(Ch...		Chips...	0	6	0	0

```
/* CreateView
 * Generated by ChipSoft Base 4.9.0.29
 *
 * ADO Version : 2.8
 *       Logic : CSSysteemService.CS_ZisInstellingLogic
 *      Action : CreateView(Scope, Filter)
 *     Trigger : NeedsFirst
 *  ScopeOrder : INSTTYPE,EIGENAAR,NAAM,SPECCODE
 * ScopeFilter : (<INSTTYPE> = 'U' EN <EIGENAAR> = 'JOHANJ   ' EN <NAAM> = 'ZC_WWNEXTNEW     ' EN <SPECCODE> = '             ')
 *      Filter : Altijd Waar
 */
sp_executesql N'
SELECT TOP 20 A.[NAAM],A.[OWNER],A.[INSTTYPE],A.[SPECCODE],A.[VALUE],A.[ETD_STATUS]
FROM [dbo].[CONFIG_INSTVARS] A
WHERE (A.[INSTTYPE] = @SV1 AND A.[OWNER] = @SV2 AND A.[NAAM] = @SV3 AND (A.[SPECCODE] IS NULL OR A.[SPECCODE] LIKE ''''))
ORDER BY A.[INSTTYPE] ASC,A.[OWNER] ASC,A.[NAAM] ASC,A.[SPECCODE] ASC
OPTION (KEEP PLAN, KEEPFIXED PLAN, LOOP JOIN)


', N'@SV1 nvarchar(1),@SV2 nvarchar(8),@SV3 nvarchar(16)',
@SV1='U',@SV2='JOHANJ',@SV3='ZC_WWNEXTNEW'
```

Done Ln 75, Col 1 Rows: 2098

Afbeelding 19 : Dubbele Query 2

5.4.1 Exact dezelfde queries

Hierbij wordt gekeken naar exact dezelfde queries. In het tool worden deze queries aangegeven als dubbele query van categorie 1.

Resultaat:
Bij het uitvoeren van het tool beek dat er inderdaad exact dezelfde queries achter elkaar worden uitgevoerd. Deze queries zullen overbodig zijn en hebben geen toegevoegde waarde.
Uit de trace-log van het ziekenhuis komt als resultaat dat 41755 van de 1302726 queries dubbel zijn uitgevoerd. Dit is 3,2% van het totaal aantal queries. Dit is dus slechts een klein percentage van het totaal aantal uitgevoerde queries. Er zal dan ook geen grote performance winst geboekt worden indien deze queries niet meer worden uitgevoerd.

Probleem oorzaak:
Om te achterhalen waar de oorzaak ligt van de dubbele queries moet duidelijk zijn uit welk deel van het EZIS deze queries afkomstig zijn. Daarom wordt er door het tool bijgehouden hoeveel dubbele queries uit welke logic van het systeem afkomstig zijn. De meeste dubbele queries komen uit de logic CSAgendaService2.CS_AfspraakMutatieLogic. Deze logic behoord tot de module Agenda, waarvan uit het interview met de afdeling implementatie en support al is gebleken dat hiermee performance problemen zijn.

Door deze logic worden 9767 *Select* queries dubbel uitgevoerd. Dit is 23.37% van het totaal aantal dubbele queries. In totaal worden er 16948 *Select* queries uitgevoerd door deze logic. Dit wil zeggen dat 57.63% van de *Select* queries uit deze logic dubbel wordt uitgevoerd. Dit zijn in verhouding vrij hoge percentages.

Om binnen de source code te vinden waar deze queries worden gegenereerd, is er in de map van de agenda module gezocht naar source files die ‘CS_AfspraakMutatieLogic’ bevatten. Dit leverde de volgende vijf bestanden op:

- fmAfspraakMutatieHistory.pas
- fmBaseGrafischAgena.pas
- fmEnkelvoudigPlannenCompleet.pas
- fmPlanMogelijkheden.pas
- fmSpreekUurOverzicht.pas

Er is vervolgens in ieder bestand gezocht naar variabelen van het type CS_AfspraakMutatieLogic. De functies waar deze variabelen in voorkomen zijn weergegeven in Tabel 9. Ook zijn daarbij de triggers weergegeven die door de variable wordt aangeroepen. Dit leverde het volgende resultaat op.

Bestand	Functie	Trigger
fmAfspraakMutatieHistory.pas	RefreshData()	-
	GetLastMutation()	NeedsLast
	GetScopeAfspraakMutations()	NeedsLast
fmBaseGrafischAgena.pas	GetMutatieLogic()	-
fmEnkelvoudigPlannenCompleet.pas	SetMutatieChecked()	NeedsLast
	NeedRefresh()	NeedsFirst NeedsNext NeedsLast
fmPlanMogelijkheden.pas	SetMutatieChecked()	NeedsLast
	NeedRefresh()	NeedsFirst NeedsNext NeedsLast
fmSpreekUurOverzicht.pas	FormCreate()	-
	GetNeedRefresh()	-

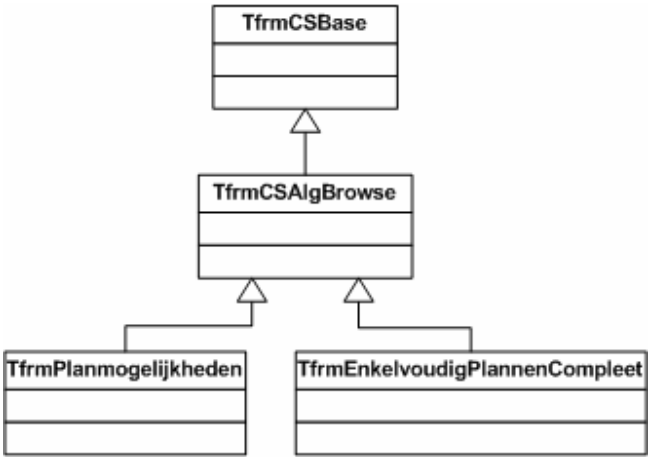
Tabel 9 : Database verzoeken van CS_AfspraakMutatieLogic

Te zien is dat er verschillende triggers de oorzaak kunnen zijn van de dubbel uitgevoerde queries. Er is toen in SQL Profiler gekeken welke triggers voorkomen bij de dubbele queries van CSAgendaService2.CS_AfspraakMutatieLogic. Hier was te zien dat dit steeds de NeedsLast trigger was. Wat ook opviel was dat er steeds een interval van 30 seconden zat tussen twee opvolgende dubbele queries.
Het tool is vervolgens aangepast om bij te houden welke trigger de oorzaak is van de dubbele queries. Voor de CSAgendaService2.CS_AfspraakMutatieLogic bleek dat van de 9767 dubbele queries 9746 keer de NeedsLast trigger verantwoordelijk is, en 21 keer de NeedsFirst trigger. Zo was duidelijk dat de oorzaak van de dubbele queries in een bepaald gedeelte van de source code zit, namelijk daar waar NeedsLast wordt aangeroepen door een *Collection* variable van het type CS_AfspraakMutatielogic.

Als er in Tabel 9 wordt gekeken, dan is te zien dat NeedsLast echter in verschillende functies wordt aangeroepen. Er kunnen dus verschillende plaatsen in de source code verantwoordelijk zijn voor de dubbele queries. Het tool geeft echter als uitkomst dat ook NeedsFirst enkele keren verantwoordelijk is voor een dubbele query. Om deze reden is er als eerste gekeken naar de functie NeedRefresh uit fmEnkelvoudigPlannenCompleet.pas en fmPlanMogelijkheden.pas. Deze functie roept beide triggers aan.

In beide files is gezocht waar aanroepen worden gedaan naar NeedRefresh. In alle twee de files is dit in de procedure trmUpdateTimer. Deze procedure wordt in de formCreate functie toegekend aan het event onRefresh. Beide source files bevatten namelijk de implementatie van een form, namelijk TfrmEnkelvoudigPlannenCompleet en TfrmPlanmogelijkheden. In beide classes was verder niet te vinden waar het onRefresh event gegenereerd werd. Er is daarom in de base classes (Afbeelding 20) gekeken of hier een onRefresh event wordt gegenereerd. Dit was inderdaad het geval in TfrmCSBase. TfrmCSBase bevat een timer die voor een ingesteld interval een Refresh event

gegenereerd. Dit geeft dan ook een verklaring van de intervallen van 30 seconden tussen twee dubbele queries.



Afbeelding 20 : Class diagram TrfmPlanmogelijkheden en TfrmEnkelvoudigPlannenCompleet

Validatie:
Om te valideren of de getraceerde source code daadwerkelijk de oorzaak is van de dubbele queries, is er met een ontwikkelaar van de Agenda module gesproken. De getraceerde code bleek inderdaad de oorzaak te zijn van de dubbele queries. Echter zijn de dubbele queries in dit geval noodzakelijk. Beide forms tonen een overzicht waarin vrije plaatsen staan waarin afspraken gepland kunnen worden. Om de 30 seconden wordt dit overzicht ververst, om zo te zien wat nog daadwerkelijk vrij is om in te plannen. Er wordt daardoor elke 30 seconden een query gestuurd die de vrije plaatsen ophaalt uit de database. Dit zorgt ervoor dat de verantwoordelijke query steeds achter elkaar wordt uitgevoerd.
Het verversen van het scherm gebeurt op basis van polling omdat het EZIS niet de mogelijkheid bezit om events te versturen tussen werkstations, om zo door te geven wanneer een vrije plaats wordt ingepland en niet meer beschikbaar is.
Om dit probleem op te lossen, wordt er gewerkt aan een notificationserver [D1] voor het EZIS. Deze server stuurt de updates van een tabel naar de clients die hierop een abonnement hebben. Op deze manier hoeven de clients niet meer zelf te pollen of er gegevens zijn veranderd in de database.

5.4.2 Niet exact dezelfde queries

Bij het kijken naar queries die niet exact hetzelfde zijn, maar die net op een enkel puntje verschillen, is er onderscheid gemaakt tussen twee categorieën. Dit om verschil aan te geven in hoe groot de kans is dat het daadwerkelijk om een dubbele query gaat. Om te bepalen binnen welke categorie een dubbele query combinatie valt, wordt er gekeken of de grijs gekleurde velden in onderstaande voorbeeld queries gelijk zijn aan elkaar. De velden uit categorie 3 komen hierbij overeen met die uit categorie 2, maar controleren nog extra op het ScopeOrder en ScopeFilter veld uit het commentaar blok.

Categorie 2 :

```
/* CreateView
 * Generated by ChipSoft Base 4.8.0.28
 *
 * ADO Version : 2.8
 *       Logic : CSKernService.CS_EPDDiagnoserelatieLogic
 *      Action : CreateView(Scope, Filter)
 *     Trigger : NeedsFirst
 *  ScopeOrder : CODE
 * ScopeFilter : Altijd Waar
 *      Filter : Altijd Waar
 */
sp_executesql N'
SELECT TOP 20
A.[CODE],A.[OMSCHR],A.[ISSUBDIAG],A.[ISCOMPL],A.[KOPPELOBJ],A.[XTRA],A.[DIAGREG],A.[
HEEFTEIND],A.[KOPPELVELD],A.[KLEUR],A.[INV_FUNCN],A.[INV_FUNCO],A.[OVZ_FUNCN],A.[
OVZ_FUNCO],A.[CLCODE],A.[ALTCLCODES],A.[XMLSTREAM]
FROM [dbo].[DOSSIER_DIAGRELS] AS A
WHERE ((1=1) AND A.[ISSUBDIAG] = @SV1)
ORDER BY A.[CODE] ASC
OPTION (KEEP PLAN, KEEPFIXED PLAN, LOOP JOIN)
', N'@SV1 bit',
@SV1=0
```

Categorie 3 :

```
/* CreateView
 * Generated by ChipSoft Base 4.8.0.28
 *
 * ADO Version : 2.8
 *       Logic : CSKernService.CS_EPDDiagnoserelatieLogic
 *      Action : CreateView(Scope, Filter)
 *     Trigger : NeedsFirst
 *  ScopeOrder : CODE
 * ScopeFilter : Altijd Waar
 *      Filter : Altijd Waar
 */
sp_executesql N'
SELECT TOP 20
A.[CODE],A.[OMSCHR],A.[ISSUBDIAG],A.[ISCOMPL],A.[KOPPELOBJ],A.[XTRA],A.[DIAGREG],A.[
HEEFTEIND],A.[KOPPELVELD],A.[KLEUR],A.[INV_FUNCN],A.[INV_FUNCO],A.[OVZ_FUNCN],A.[
OVZ_FUNCO],A.[CLCODE],A.[ALTCLCODES],A.[XMLSTREAM]
FROM [dbo].[DOSSIER_DIAGRELS] AS A
WHERE ((1=1) AND A.[ISSUBDIAG] = @SV1)
ORDER BY A.[CODE] ASC
OPTION (KEEP PLAN, KEEPFIXED PLAN, LOOP JOIN)
', N'@SV1 bit',
@SV1=0
```

Resultaat:
In categorie 2 zijn 193.936 “dubbele” queries gedetecteerd in 152 verschillende logics, en in categorie 3 zijn dit er 239.157 in 232 verschillende logics. In Tabel 10 en Tabel 11 zijn voor respectievelijk categorie 2 en categorie 3 de drie logics weergegeven die de meeste “dubbele” queries bevatten. Te zien is dat het hier om behoorlijke aantallen queries gaat. Indien het daadwerkelijk overbodige queries zijn, dan zou dit een flinke daling kunnen betekenen in het aantal uit te voeren queries. Nader onderzoek zal dit moeten uitwijzen, daar dit door tijdsgebrek niet meer mogelijk was tijdens de onderzoeksperiode.

Logic	**Aantal**	**Top Trigger Combinatie**	**Aantal Top Trigger Combinatie**
CSSysteemService.CS_ZisInstellingLogic	41.461 *(21,4%)*	NeedsCurrent & NeedsCurrent	37.901
CSSysteemService.CS_DDVeldLogic	30.664 *(15,8%)*	NeedsFirst & NeedsFirst	26.192
CSArchiefService.CS_BehandelgangLogic	27.437 *(14,2%)*	NeedsFirst & NeedsLast NeedsLast & NeedsFirst	14.762 12.271

Tabel 10 : Top 3 “dubbele” queries categorie 2

Logic	**Aantal**	**Top Trigger Combinatie**	**Aantal Top Trigger Combinatie**
CSSysteemService.CS_ExtraVeldLogic	32.100 *(13,4%)*	NeedsCurrent & NeedsCurrent	31.630
CSKernService.CS_PatientLogic	20.008 *(8,3%)*	NeedsCurrent & NeedsCurrent	19.529
CSAgendaService2.CS_SubAgendaLogic	18.532 *(7,7%)*	NeedsCurrent & NeedsCurrent	18.478

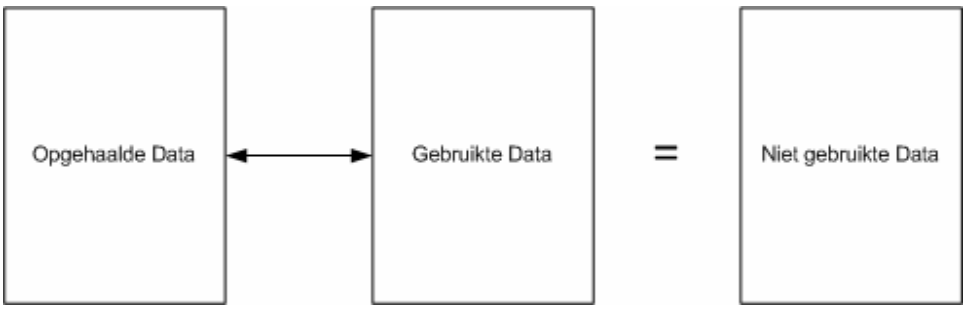
Tabel 11 : Top 3 “dubbele” queries categorie 3

5.5 Queries die ongebruikte data ophalen

Zoals al eerder gezegd, was het onmogelijk om uit een trace-log gegevens te halen om te zien of de data die wordt opgehaald ook daadwerkelijk gebruikt wordt binnen de applicatie. Daarom is er een voorstel gedaan hoe dit eventueel getest kan worden binnen de applicatie zelf.

Voorstel

Om te weten welke opgehaalde data uit de database ook daadwerkelijk gebruikt wordt, moet er een overzicht gegeven kunnen worden van het verschil tussen wat is opgehaald en wat daadwerkelijk gebruikt is. Dit verschil kan gemaakt worden door twee bestanden met elkaar te vergelijken, waarbij het eerste bestand gegevens bevat over de opgehaalde data, en het tweede bestand gegevens over de daadwerkelijk gebruikte data (Afbeelding 21). Bestanden kunnen hierbij textfiles, tabellen, of werkgeheugen zijn, afhankelijk hoe het voorstel het beste geïmplementeerd kan worden.



Afbeelding 21 : Detecteren ongebruikte data

Bij het ophalen van niet gebruikte data kan onderscheid gemaakt worden tussen twee categorieën:

Categorie 1: Losse *Velden* uit een *Object* die worden opgehaald en niet worden gebruikt.
Categorie 2: Records die worden opgehaald en in een *Object* worden gezet, terwijl dit *Object* nooit gebruikt word.

Realisatie

Om dit te realiseren dient er binnen de data acces laag van het EZIS debug code aangebracht te worden die de bestanden met opgehaalde en gebruikte data vult.

Opgehaalde data:
Het bestand met opgehaalde data kan gevuld worden in de functie SpawnCurrentFromRecordSet() in de class TCS_ADOFactory. Deze functie vult de *Velden* van een *Object* met de gegevens die zijn opgehaald uit de database. Elk *Object* dat wordt gevuld heeft een uniek ID. Dit ID kan gebruikt worden om een *Object* te identificeren, en hiermee het bestand van opgehaalde data te vullen.

Gebruikte data:
Het ophalen van data uit een *Object* gebeurd in de functie DoGetValue() van de class TimpCS_Veld. Hier wordt de data dus uit een *Veld* gehaald. Via de parent van het veld, het *Object*, kan het ID van het *Object* van dit veld achterhaald worden. Dit ID kan dan gebruikt worden om het bestand te vullen met gebruikte data.

Door het vergelijken van de twee gevulde bestanden met ID's kan gezien worden hoeveel *Objecten* worden gevuld met gegevens en hoeveel er daadwerkelijk gebruikt worden. Voor het controleren of een *Veld* van een record gebruikt word, dient naast het ID van het *Object* ook de naam van het *Veld* opgeslagen te worden in alle twee de bestanden. Zo kan er per object gezien worden welke velden wel en niet gebruikt worden.

Oorzaak:
Indien blijkt dat er data is die wordt opgehaald en niet wordt gebruikt, is het ook van belang om te weten waar de oorzaak daarvan ligt. Daarom zal er naast de ID's van de *Objecten* ook opgeslagen moet worden uit welke logic van het EZIS de query afkomstig is. Het opslaan van dit gegeven is alleen nodig bij het bestand van opgehaalde data. De gegevens over de betreffende logic en eventuele andere informatie zoals de verantwoordelijke trigger van de uitgevoerde query kan uit het recordset gehaald die de data heeft opgehaald. Deze recordset is aanwezig in de functie SpawnCurrentFromRecordSet().

Voor het bestand met niet gebruikte data dient een tool geschreven te worden dat de gegevens uit dit bestand analyseert. Dit kan op dezelfde manier gebeuren zoals het eigen gemaakte tool dit doet bij dubbel voorkomende queries. Zo kan met behulp van de logic en de trigger achterhaald worden waar in de source code de oorzaak zit van de niet gebruikte data.

5.6 Queries die semantische errors bevatten

Er zijn verschillende semantische errors die voor kunnen komen in een SQL Statement [P03]. Om te onderzoeken of er ook semantische errors worden gegenereerd door het EZIS, is het eigen tool aangepast om hier controle op uit te voeren. Bij het onderzoek is er naar drie verschillende semantische errors gezocht.

Constante output kolom
Hierbij wordt gekeken of een kolom in de *Where* clausule wordt vergeleken met slechts één variable. Indien dit het geval is, dan hoeft deze kolom niet opgehaald te worden middels *Select*. De waarde van deze kolom zal altijd gelijk zijn aan de variable, en heeft geen toegevoegde waarde.

Onnodige Order By kolom
Net als bij een constante output kolom, wordt gekeken of een kolom in de *Where* clausule slechts met één variable wordt vergeleken. Indien dit het geval is, dan is de waarde van deze kolom altijd gelijk. Het voorkomen van deze kolom in *Order By* heeft dan geen toegevoegde waarde, daar de volgorde altijd hetzelfde is.

Vergelijking met NULL en ‘’
Hierbij wordt gekeken of eenzelfde kolom in de *Where* clausule wordt vergeleken met IS NULL en LIKE ‘’. Beide hebben dus als doel om te kijken of er geen waarde in de kolom staat. Het verschil is echter dat een keer wordt gekeken of de kolom echt leeg (NULL) is, en de andere keer of er een lege waarde (‘’) in staat. Beide gevallen kunnen namelijk voorkomen in de tabellen van het EZIS.

Er is naar deze drie semantische errors gezocht omdat het EZIS slechts een beperkt aantal mogelijk queries genereert, en dus niet alle genoemde semantische errors uit [P03] voor kunnen komen. Bovendien is het bepalen of er een semantische error voorkomt een complexe taak, waar nog veel onderzoek in gedaan kan worden. Het controleren op de bovengenoemde semantische errors was redelijkerwijs uit te voeren binnen de onderzoeksperiode.

Resultaat
Bij het analyseren van de trace-log op semantische errors, zijn alledrie de “errors” waarop gecontroleerd is gevonden (Tabel 12). Te zien is dat het aantal van constante output kolom groter is dan het aantal uitgevoerde queries. Dit komt doordat een enkele query meerdere constante output kolommen kan bevatten.

Error	**Aantal**
Constante output kolom	1.359.715
Onnodige Order By	581.329
Vergelijking met NULL en ‘’	184.563

Tabel 12 : Resultaten van semantische errors

Optimalisatie
Om te testen of er ook daadwerkelijk performancewinst wordt geboekt indien de query wordt aangepast, is er een query genomen om dit te testen. Deze query bevatte alle drie de “errors” . Het ging hierbij om onderstaande query, en leverde het resultaat op uit Tabel 13**Error! Reference source not found.**. Elke query is vier keer uitgevoerd om een valide resultaat te verkrijgen.
De sterk afwijkende waarden bij (3) van reads en ontvangen bytes komt doordat deze query geen resultaat meer gaf.
Verder is te zien dat het aantal verzonden bytes afneemt, dit komt doordat het SQL statement kleiner wordt omdat overbodig SQL niet wordt mee verzonden. Ook het aantal ontvangen bytes neemt af. Bij het weg laten van een extra veld in *Select* is dit logisch. Voor de andere gevallen is geen logische verklaring te geven. Wat opvalt, is dat er geen verandering is in de duration van de query. Dit is mogelijk doordat de hardware te sterk is, en dat een verschil hierin niet goed detecteerbaar is. Dit zou ook verklaren waarom de duration per query verschilt. De queries testen in een stresstest of op langzamere hardware zou hier eventueel uitkomst kunnen bieden.

```
sp_executesql N'
SELECT TOP 20
A.[SPECIALISM],A.[CODE],A.[DATUM],A.[OMSCHRIJV],A.[AANLCODE],A.[EINDDATUM],A.[ZOE
KCODE],A.[GROEPCODE],A.[ISGROEP],A.[LANGEOMSCH],A.[MAGVERVOLG],A.[ISVERVOLG]
FROM [dbo].[EPISODE_DIAG] AS A
WHERE ((A.[SPECIALISM] = @SV1 AND A.[GROEPCODE] LIKE @SV2+"%") AND ((A.[DATUM] IS
NULL OR A.[DATUM] <= @SV3) AND (A.[EINDDATUM] >= @SV4 OR  (A.[EINDDATUM] IS NULL
OR A.[EINDDATUM] LIKE ""))))
ORDER BY A.[SPECIALISM] ASC,A.[GROEPCODE] ASC,A.[DATUM] ASC,A.[CODE] ASC
OPTION (KEEP PLAN, KEEPFIXED PLAN, LOOP JOIN)
', N'@SV1 nvarchar(5),@SV2 nvarchar(10),@SV3 datetime,@SV4 datetime',
@SV1='OOG',@SV2='1',@SV3='20060411',@SV4='20060411'
```

	Normaal	**Zonder Select (1)**	**Zonder Order (2)**	**Zonder NULL (3)**	**Zonder ‘’ (4)**	**Zonder 1 + 2 + 4 (5)**
Bytes send	1468	1438	1430	1418	1414	1346
Bytes received	10535	10408	10495	9121	10301	9954
Reads	64	64	64	39	64	64
CPU	0	0	0	0	0	0
Duration	0 / 16 / 0 / 16	0 / 16 / 0 / 16	0 / 16 / 0 / 13	16 / 16 / 0 / 0	0 / 0 / 16 / 0	16 / 0 / 16 / 0

Tabel 13 : Resultaat bij het optimaliseren van de semantische errors in een query

Conclusie

In een stresstest of op langzamere hardware zal getest moeten of er een duidelijker verschil is te zien in de performance van de verschillende queries. Het aantal ontvangen bytes zal sowieso afnemen indien een *Select* veld weg gelaten wordt. Bij deze query is het verschil niet zo groot, maar dit is per query verschillend afhankelijk van hoeveel records worden opgehaald en de grootte van het veld.

Indien deze optimalisaties van de queries worden doorgevoerd in het EZIS, dan dient getest te worden of dit ook daadwerkelijk performance snelheid oplevert. Een enkele query kan wel een betere performance hebben, maar de optimalisatie van de query voordat deze uitgevoerd wordt, kost ook tijd. Er dient daarom getest te worden of deze optimalisatie minder tijd kost dan het uitvoeren van de niet geoptimaliseerde query [P09].

De vergelijking van een veld met NULL en ‘’ hoeft niet per query geoptimaliseerd te worden. Het vergelijken met beide waarden komt door de historie van de velden. Deze kunnen zowel NULL als ‘’ zijn. Om te zorgen dat nog maar een van deze twee vergelijking gemaakt wordt, zullen eenmalig alle tabellen aangepast moeten worden naar één waarde.

5.7 TraceAnalyzer tool

In de vorige paragrafen was te lezen dat er tijdens het onderzoek een eigen tool is geschreven voor het analyseren van de trace-logs van SQL Server. Dit is het TraceAnalyzer tool. Deze paragraaf zal een verdere toelichting geven van het TraceAnalyzer tool en het verdere belang ervan. De handleiding van de TraceAnalyzer is opgenomen in bijlage 4.

5.7.1 Opzet TraceAnalyzer

De TraceAnalyzer is gemaakt met als doel de bottleneck te vinden in de uitgevoerde queries van het EZIS. Dit is mogelijk dankzij de functionaliteiten die dit tool biedt ten opzichte van SQL Profiler, het tool dat tot op heden werd gebruikt binnen ChipSoft voor het analyseren van trace-logs.

Detecteren en weergeven van verdachte queries
Het TraceAnalyzer tool geeft een overzicht van queries die lang duren, veel CPU tijd gebruiken, en dubbel worden uitgevoerd. Indien deze gegevens achterhaald moeten worden met SQL Profiler, dan moet een trace-log van boven tot beneden doorlopen worden. Dit kost veel tijd. Bovendien is het lastig om dit alles in een oogopslag te zien. Ook geeft het tool weer hoe vaak een bepaalde query voorkomt. Dit is een helemaal onmogelijke taak met SQL Profiler.

Achterhalen probleemoorzaak
Het TraceAnalyzer tool verzamelt gegevens per query. Hierdoor wordt er dus per query alle Durations en CPU tijden opgeslagen. Bij het weergeven van bijvoorbeeld de langst durende query worden dan ook de overige tijden van deze query weergegeven. Hierdoor is te zien of deze query altijd zo lang duurt of dat het maar een enkele keer is voorgekomen. Zo is te zien of het aan de query zelf ligt of dat er een andere oorzaak is. Indien het een andere oorzaak is, dan hoeft de query zelf niet geoptimaliseerd te worden, en kan er verder gegaan worden naar het zoeken van de echte oorzaak.
Indien er gebruik was gemaakt van SQL Profiler, dan was niet duidelijk geweest wat de overige tijden van deze query waren, en was alleen de hoogste Duration van deze query opgevallen. Er zou dan zijn geprobeerd om de query te optimaliseren, om er later achter te komen dat de query zelf niet de oorzaak is. Doormiddel van de TraceAnalyzer wordt deze tijd gespaard.

Dit is ook het geval bij queries die dubbel voorkomen. Hierbij worden aantallen gegeven over hoe vaak een bepaalde query dubbel voorkomt, en uit welk deel van het systeem deze afkomstig is. Zo is snel te zien welk deel van het EZIS de grootste boosdoener is. Op deze manier kan gericht het grootste probleem worden aangepakt. Dit in tegenstelling tot SQL Profiler waar de eerste en beste dubbele query zou zijn aangepakt, terwijl deze in verhouding tot andere dubbele queries misschien maar een enkele keer voorkomt.

5.7.2 Gebruik binnen ChipSoft

Het TraceAnalyzer tool is niet alleen ontwikkeld om er het onderzoek mee te verrichten, maar zal ook in de toekomst gebruikt worden bij ChipSoft.

Development
Omdat het EZIS een groeiend systeem is waar steeds meer functionaliteiten bij komen, zullen er ook steeds nieuwe queries worden gegenereerd. Deze nieuwe queries kunnen zorgen voor nieuwe en meer performance problemen. Het is daarom van belang dat ontwikkelaars tijdens het ontwikkelen en testen van de software de queries die worden gegenereerd analyseren. Dit om in een vroeg stadium nieuwe performanceproblemen te vinden en op te lossen voordat deze opduiken bij een klant. Tot op heden maakten de ontwikkelaars gebruik van SQL Profiler. Door gebruik te maken van het TraceAnalyzer tool worden de voordelen van dit tool, zoals beschreven in de vorige paragraaf, behaald ten opzichte van SQL Profiler. Hierdoor kan het ontwikkelproces sneller worden uitgevoerd, en wordt de productiviteit van de ontwikkelaars verhoogd.

Trace-logs van klanten
Het tool zal ook gebruikt worden om trace-logs van klanten te analyseren indien deze performanceproblemen hebben. Tot op heden werd voor het analyseren van logs van klanten SQL Profiler gebruikt. Hiermee werd een trace-log helemaal handmatig doorlopen om te kijken waar het probleem zit, zoals blijkt uit het interview met I&S (Bijlage 1). Dit is een tijdrovend proces. Door gebruikt te maken van het TraceAnalyzer tool kan sneller gezien worden waar het probleem zit, of waar dit juist niet zit. Dit heeft tot voordeel dat er sneller geantwoord kan worden richting de klanten. Dit zorgt voor een betere service richting de klanten, en zal ten goede komen van het vertrouwen in ChipSoft en haar software. Zo is er tijdens het ontwikkelen van het tool een opgestuurde trace-log van een klant geanalyseerd die last had van performanceproblemen. Deze kon binnen enkele minuten op de hoogte gesteld worden dat er inderdaad queries waren die lang duurde, maar dat het niet aan de query zelf lag. De query kwam namelijk ook voor met tijden die niet lang duurde. Het probleem lag aan de drukte op de server.

5.7.3 Ontwikkeling TraceAnayzer

Het TraceAnalyzer tool geeft een aantal functionaliteiten voor het analyseren van trace-logs. In de toekomst bestaat de kans dat deze functionaliteiten uitgebreid of aangepast moeten worden, omdat meer of andere gegevens uit de trace-logs opgehaald moeten worden voor het analyseren van het databasegebruik. Het is dus van belang dat het tool gemakkelijk aangepast kan worden. Er is tijdens de ontwikkeling van het tool dan ook een hoge prioriteit gegeven aan het design van het tool, en het constant refactoren van de source code. Zo was bij de uitbreiding van het tool gedurende het onderzoek te merken dat nieuwe functionaliteiten snel te implementeren waren. Een klassendiagram is opgenomen in bijlage 3.

Indien er verder ontwikkeld wordt aan het tool dan zal dit door andere ontwikkelaars worden gedaan dan de initiële ontwikkelaar. Het is dus van belang dat zij zich goed kunnen inwerken in de source code om zo efficiënt aanpassingen te maken. Er is daarom bij alle functies commentaar geschreven over de functionaliteit ervan [B2]. Ook is bij belangrijke code binnen een functie commentaar geschreven. Zo kan in een oogopslag gezien worden waar wat gebeurd.

Om de correctheid van het tool aan te tonen is er tijdens het ontwikkelen ook debug code geschreven die debug informatie op het scherm toont of in een log-file schrijft. Als het tool wordt aangepast kan de debug code worden “aangezet” om zo te controleren of het tool nog steeds op de juiste manier werkt.

6. Resultaten

Er zijn in de trace-logs van SQL Server patronen gevonden die duiden op onnodig of verkeerd databasegebruik. Dit hoofdstuk zal een opsomming geven van de resultaten van deze gevonden patronen. Verder wordt besproken hoe deze resultaten gezien moeten worden, en wat er in de toekomst aan verder onderzoek verricht kan worden.

6.1 Gevonden patronen

Tijdens het onderzoek zijn er geen sessies geïdentificeerd om zo caching toe te kunnen passen zoals is beschreven in [P01]. Het EZIS haalt bij elke query altijd alle gegevens op uit een tabel. Dit zorgt er al voor dat de gegevens gecached worden.
Ook is er geen onderzoek gedaan naar sub-query optimalisatie zoals dit is gedaan in [P07]. In de huidige versie van het EZIS wordt namelijk geen gebruik gemaakt van sub-queries.
Er is wel onderzoek gedaan naar lang durende queries en queries die vaak voorkomen, zoals dit ook het geval is in [P04]. De resultaten hiervan, en van enkele andere kritische SQL statements in het EZIS, worden hieronder besproken.

SQL queries die veel tijd kosten
Er zijn queries gevonden die veel tijd kosten. Dit is dus zoals werd verwacht. Indien dit aan de query zelf lag, dan kon deze ook geoptimaliseerd worden. Dit is gevalideerd zoals staat beschreven in hoofdstuk 4. Per query verschilt echter in hoeverre deze geoptimaliseerd kan worden, en hoe vaak deze voorkomt. Het is dus moeilijk in te schatten hoe de totale performance [P04] van het EZIS verbeterd zal worden indien deze optimalisaties doorgevoerd worden.

SQL queries die vaak voorkomen
Bij vergelijking van zowel het hele TextData veld als alleen het SQL statement was er niet duidelijk een query die de overmacht had in het aantal keer dat deze uitgevoerd werd. Verwacht werd dat dit wel het geval zou zijn. Hier is dus niet de bottleneck van het performance probleem in terug te vinden. Deze zal ergens anders gezocht moeten worden.
Bij vergelijking van alleen het SQL statement was het percentage van de query die het vaakste voorkwam 6%. Hierbij kon een optimalisatie gemaakt worden in de te ontvangen data. Dit was echter zo minmaal dat dit zo goed als niet merkbaar zal zijn. Ook deze optimalisatie is gevalideerd zoals beschreven in hoofdstuk 4.
Bij de vergelijking van het hele TextData veld was het percentage van de query die het vaakste voorkomt 1,45%. Als optimalisatie zal deze query gecached worden in het EZIS, omdat deze altijd hetzelfde resultaat geeft. Dit levert dan ook 1,45% minder queries op.

SQL queries die dubbel worden uitgevoerd
Het percentage van het aantal dubbel uitgevoerde queries was 3,2% indien gekeken werd naar exact dezelfde queries. Een deel hiervan zal niet meer voorkomen omdat er in de toekomst gebruik gemaakt gaat worden van een notificationserver [D1]. Het niet meer voorkomen van dubbele queries zal 3,2% minder queries opleveren.

Er zijn dus zoals verwacht dubbele queries gevonden, echter geeft het lage percentage aan dat ook hier niet de bottleneck zit.
Voor de overige dubbele queries (categorie 2 en 3) zijn wel grote aantallen aan dubbele queries gevonden. Echter is hier niet duidelijk of het ook daadwerkelijk om overbodige queries gaat. Verder onderzoek zal dit moeten uitwijzen.

SQL queries die ongebruikte data ophalen
Hiervoor zijn geen resultaten bekend daar dit niet uit een trace-log af te leiden is. Het voorstel dat gedaan is om hiervoor wel resultaten te verkrijgen zal in de toekomst uitgewerkt en getest moeten worden.

SQL queries die semantische errors bevatten
Zoals werd verwacht, zijn er queries gevonden die bepaalde semantische errors bevatten. Hierbij was bij een enkele query echter niet duidelijk te meten hoeveel performance dit op zal leveren indien deze errors niet meer voorkomen. Een stresstest hiervan en implementatie in het EZIS zal moeten uitwijzen of dit daadwerkelijk een betere performance oplevert.
In tegenstelling tot [P03] is er tijdens het onderzoek slechts op enkele semantische errors gecontroleerd, dit omdat het EZIS een beperkt aantal queries genereert. Wel is er op een extra “error” gecontroleerd die niet vermeld staat in [P03]. Dit is de vergelijking van een veld met NULL en ‘’.

6.2 Bruikbaarheid van het onderzoek

Het probleem van de slechte performance zal niet verholpen worden door de gevonden patronen in de trace-log, en zal dus op een andere manier verholpen moeten worden. In zoverre is voor de opdrachtgever van belang dat deze patronen niet de bottleneck vormen van het performance probleem. De oorzaak licht ergens anders, en hierna zal verder onderzoek verricht moeten worden. Hierbij moet echter wel rekening worden gehouden dat dit resultaat uit een valide trace-log is behaald, maar dat het hierbij om één enkele trace-log ging.

6.2.1 Bruikbaarheid van het tool

Het tool dat is ontwikkeld om het onderzoek mee te verrichten kan in de toekomst verder gebruikt worden, zoals ook staat beschreven in het vorige hoofdstuk. Het EZIS blijft in ontwikkeling en zal daardoor steeds nieuwe queries genereren die het databasegebruik negatief kunnen beïnvloeden. Ontwikkelaars gebruiken tot op heden SQL Profiler om te zien wat hun implementatie aan queries genereert. Dit is een tijdrovend proces, daar de trace-logs van voor tot achteren “handmatig” worden doorlopen. Door de ontwikkelaars gebruik te laten maken van het tool kan er in een oogopslag gezien worden of er door het tool ondersteunde patronen in de trace-log voorkomen. Dit gaat een stuk sneller. Ook indien er geen patronen gevonden worden door het tool is er winst behaald, al dan niet in performance van het EZIS maar in tijd van de ontwikkelaar. Er wordt immers geen tijd besteed door naar iets te zoeken dat er niet is.

6.3 Conclusie

Tijdens het onderzoek zijn er patronen gevonden die duiden op onnodig of verkeerd databasegebruik. Dit komt overeen met wat werd gesteld in de hypothesen. Het niet meer voorkomen van deze queries, of het optimaliseren ervan zal de performance van het EZIS systeem positief beïnvloeden. Het EZIS zal echter niet veel sneller worden door het niet meer voorkomen van deze queries. De optimalisaties die gemaakt kunnen worden met behulp van de gevonden patronen zijn slechts minimaal, en zullen de performance slechts met enkele procenten verbeteren. De bottleneck van het EZIS zit dus niet in de gevonden patronen. Hierbij dient wel rekening gehouden te worden dat deze resultaten zijn gebaseerd op slechts één trace-log.
Mogelijk kan de bottleneck gevonden worden in de dubbele queries van categorie 2 en 3, indien deze daadwerkelijk dubbel zijn. Een andere mogelijkheid is dat deze zit in gegevens die worden opgehaald, maar niet worden gebruikt.

6.4 Toekomstig werk

Tijdens het onderzoek is er getest of optimalisaties van enkele queries een betere performance opleveren. Dit geeft echter geen beeld hoe dit in de praktijk zal uitpakken. Er zal daarom getest moeten worden wat het resultaat is van de performance indien de resultaten van de onderzoekspunten geïmplementeerd worden in het EZIS. Ook zal getest moeten worden of er een daadwerkelijk performance verschil is indien de berekende percentages aan queries niet meer zullen voorkomen. Hoe deze test het beste uitgevoerd kan worden staat beschreven in de volgende subparagraaf.
Voor de dubbele queries van categorie 2 en 3 zal verder onderzoek verricht moeten worden om te bepalen of deze echt overbodig zijn. Als dit het geval is, dan kan dit een daling betekenen in het aantal queries die uitgevoerd moeten worden.
Bij de check op semantische errors, is er slechts gecontroleerd op enkele van deze errors. Deze check kan uitgebreid worden met nog meer mogelijke semantische errors die kunnen voorkomen in het EZIS.
Verder dient in het EZIS debug code ingebouwd te worden om te testen of er data uit de database wordt opgehaald en die niet wordt gebruikt, zoals dit is beschreven in hoofdstuk 5.6.

6.4.1 EZIS performance

Om te testen of de performance van het EZIS daadwerkelijk wordt verbeterd dienen de onderzoeksresultaten verwerkt te worden in het EZIS. Hierbij dient er vanuit een basisconfiguratie steeds maar één verandering doorgevoerd te worden om te kijken of deze verandering positieve gevolgen heeft voor de performance [P04]. Als een verandering getest is kan weer terug gegaan worden naar de basisconfiguratie om de volgende verandering te testen. Ook kan de aangepaste versie, indien deze positieve invloed heeft op de performance, als nieuwe basisconfiguratie genomen worden. Zo wordt er per aanpassing duidelijk hoe deze de performance beïnvloedt van het systeem. Hierbij kan gebruik worden gemaakt van een stresstest tool [P04].

Bibliografie

Papers

[P01]
Qingsong Yao, Xiangji Huang and Aijun An, Applying Language Modeling to Session Identification from Database Trace Logs.
Accepted by Knowledge and Information Systems: An International Journal (KAIS). Springer-Verlag Publisher. ISSN (Printed): 0219-1377 and ISSN (Online): 0219-3116. To appear in 2006
http://www.cs.yorku.ca/~qingsong/research/ngram.pdf

[P02]
Tim Foxon, Performance Management and Capacity Planning for Microsoft SQL Server.
Int. CMG Conference 2000, pages 581-592
http://www.metron-athene.com/reference/technical/documents/tech18.pdf

[P03]
Stefan Brass, Christion Goldberg, Semantic Errors in SQL Queries: A Quite Complete List.
Journal of Systems and Software, Volume 79, Issue 5, May 2006, pages 630-644
http://users.informatik.uni-halle.de/~brass/sem04_2/semerr3.pdf

[P04]
Georg Sonneck, Thomas Mueck, Performance Tuning of an Intranet Application – A Case Study.
Proceedings of the 16th International Parallel and Distributed Processing Symposium, 2002, page 266.
http://csdl.computer.org/dl/proceedings/ipdps/2002/1573/02/15730223.pdf

[P05]
E. F. Codd, A Relational Model of Data for Large Shared Data Banks.
ACM Communications, Vol. 13, No. 6, June 1970, pages 377-387
http://www.seas.upenn.edu/~zives/03f/cis550/codd.pdf

[P06]
D. D. Chamberlin et al, SEQUEL 2: A Unified Approach to Data Definition, Manipulation, and Control.
IBM Journal of Research and Development, Vol. 20, No. 6, November 1976, pages 560-575
http://www.research.ibm.com/journal/rd/206/ibmrd2006E.pdf

[P07] Won Kim, On Optimizing an SQL –like Nested Query.
ACM Transactions on Database Systems 7, September 1982, page 443-469

[P08]
P. Griffiths Selinger, M. M. Astrahan, D. D. Chamberlin, R. A. Lorie, T. G. Price, Access Path Selection in a Relational Database Management System.
Proceedings of the 1979 ACM SIGMOD international conference on Management of Data ACM, 1979, pages 23-34
http://edlab-www.cs.umass.edu/cs645/645-paper4.pdf

[P09]
Matthias Jarke, Jurgen Koch, Query Optimization in Database Systems.
ACM Computing Surveys, Vol. 16, No. 2, June 1984, pages 111-152
http://www.cs.cornell.edu/courses/cs632/2003sp/jarke.pdf

Boeken

[B1]
Microsoft SQL Server 2000 Performance Optimization and Tuning Handbook
2001, Digital Press, ISBN: 1555582419

[B2]
Steve McConnel, Code Complete 2, Microsoft Press, ISBN: 0735619670

White papers

[WP1]
Microsoft Training and Certification, Module 8: Monitoring SQL Server for Performance 2000, Microsoft

[WP2]
Johnny Graettinger, DSpace Performance for AIHT

[WP3]
Keven Kline, Finding and Fixing Bad SQL in Microsoft SQL Server 2000
2005, Quest Software Inc.
http://www.quest.com/documents/list.aspx?searchoff=true&contenttypeid=1&prod=104

[WP4]
John Theron , SQL Server Contention Diagnosing and Resolving Blocking Problems
2004, Quest Software Inc.
http://www.quest.com/documents/list.aspx?searchoff=true&contenttypeid=1&prod=104

[WP5]
Keven Kline, Claudia Fernandez, Tuning SQL Statements on Microsoft SQL Server 2000 2005, Quest Software Inc.
http://www.quest.com/documents/list.aspx?searchoff=true&contenttypeid=1&prod=104

Links

[L1] http://en.wikipedia.org/wiki/Microsoft_SQL_Server
[L2] http://en.wikipedia.org/wiki/Three_tier
[L3] http://en.wikipedia.org/wiki/Pattern_recognition
[L4] http://www.corba.ch/e/3tier.html#The%20advantages%20of%203-tier%20architecture
[L5] http://en.wikipedia.org/wiki/Performance_tuning
[L6] http://en.wikipedia.org/wiki/Performance_problem
[L7] http://en.wikipedia.org/wiki/Performance_testing

Documenten

[D1] S. Petrosian, System Wide Notification Service, ChipSoft, mei 2006
[D2] L. Truijnes, ChipSoft Base, ChipSoft, 2003

Bijlage 1 : Interview Implementatie & Support

1) Is performance een groot issue?
Performance is absoluut een groot issue. De applicatie waarmee het personeel werkt moet snel reageren.

2) Zijn er eisen opgesteld waaraan het systeem moet voldoen wat betreft performance?
Ja, een scherm moet binnen 1,5 seconden getoond zijn. Er zijn enkele uitzonderingen bij overzichten, deze mogen langer duren.

3) Komen klachten/opmerkingen binnen van klanten over performance?
Ja, er komen klachten binnen over de performance van het systeem. Vaak komen er ook geen klachten binnen over performance problemen. Mensen van de werkvloer geven de klachten dan niet door aan het systeembeheer, of het systeembeheer geeft de klachten niet door aan ChipSoft.
Ook komt het voor dat klanten denken dat het normaal is dat het systeem "traag" is. Er wordt dan ook zeer bij klanten op aangedrongen dat als er problemen zijn met performance, dat er een melding van gedaan wordt.

4) Zo ja, zijn deze klachten/opmerkingen over de snelheid van de applicatie, of de belasting van de server?
Deze klachten zijn over de snelheid van de applicatie. Klachten over de belasting van de server zijn er niet vaak. Applicatiebeheerders denken dat het normaal is als de CPU bijvoorbeeld voor langere tijd zwaar belast wordt.

5) Zo ja, met welke regelmaat worden deze klachten/opmerkingen ontvangen?
Klachten komen vooral binnen bij nieuwe releases van bijvoorbeeld hotfixes. Er wordt dan niet op de performance getest. Alleen bij hoofdreleases wordt er getest op performance. Er worden ook geen stresstesten uitgevoerd bij ChipSoft. De klant is zelf verantwoordelijk voor het stresstesten van de applicatie op hun eigen test servers. Dit is omdat ChipSoft niet de beschikking heeft over dezelfde capaciteit servers en hoeveelheden data die sommige klanten hebben.

6) Zo ja, gaat dit over specifieke modules in het systeem, of over het systeem in het algemeen?
Dit komt vooral van de modules Agenda, Opname en Ok. Dit komt waarschijnlijk omdat deze modules veel gebruikt worden, en het daardoor snel vervelend wordt als dit traag is. Dingen die minder vaak gebruikt worden zullen niet snel tot ergernis lijden indien dit iets trager is, en er zal dan ook niet snel een melding van gemaakt worden.

7) Zo ja, wat wordt er gedaan als een klacht van een klant binnenkomt?
Als eerste moet duidelijk zijn wat precies het probleem is. Er wordt aan de klanten duidelijk gemaakt dat ze precies door moeten geven wat ze gedaan hebben en wat er fout gaat. Vervolgens wordt geprobeerd om met behulp van logs van code-site de oorzaak te vinden. Als dit niet lukt, dan worden er trace-logs gemaakt en wordt dit nagelopen om te kijken waar bijvoorbeeld veel queries voorkomen. Dit wordt handmatig doorgelopen door dingen te proberen en te kijken wat er aan queries wordt gegenereerd.

8) Zo ja, komen deze klachten van bepaalde klanten of van verschillende klanten?
De laatste tijd is er één klant specifiek gevolgd, maar over het algemeen komen de klachten van verschillende klanten.

9) Hoeveel gebruikers heeft het systeem?
Het grootste aantal is ongeveer 3500. De rest is moeilijk wat van te zeggen.

10) Hoeveel gebruikers zijn gelijktijdig actief op het systeem?
Bij het grootste systeem van ongeveer 3500 gebruikers zijn er ongeveer 1200 tegelijk actief. Dit is op piekmomenten tussen 8.00 en 16.00.

11) Valt er iets te zeggen over welke modules het meest gebruikt worden?
Dit zijn dezelfde als bij vraag 6: Agenda, Opname en OK.

Bijlage 2 : Onderzoek tool support

Tools
Voor het onderzoek naar het databasegebruik van het EZIS systeem kan er gebruik gemaakt worden van tools. Naast de native tools van SQL Server zijn er ook tools verkrijgbaar van derden. De native tools van SQL Server (SQL Profiler / SQL Query Analyzer) bieden een beperkte mogelijkheid tot het analyseren van de queries. Tools van derden geven hier meer mogelijkheden en kunnen daardoor meer inzicht geven in de uitgevoerde queries. Tools waarna onderzoek is gedaan zijn afkomstig van Quest Software en Embarcadero.

Microsoft SQL Server

SQL Profiler:
SQL Profiler biedt de mogelijkheid om trace-logs te maken van de queries die uitgevoerd worden op SQL Server. Hierbij is het mogelijk om een filter in de stellen zodat maar een beperkt aantal queries worden weergegeven. Indien de trace-log gesorteerd dient te worden op bijvoorbeeld langst durende query, dan is dit niet mogelijk. Hiervoor moeten dan zelf SQL statements geschreven worden die uitgevoerd kunnen worden op de trace-log.

SQL Query Analyzer:
Met SQL Query Analyzer kunnen SQL statements uitgevoerd worden en kan bekeken worden welk gedeelte van het statement hoeveel resource in beslag neemt. Indien een statement op een bepaalt punt veel resource in beslag neemt, bijvoorbeeld bij een join, dan kan het statement herschreven worden. Het statement kan vervolgens opnieuw uitgevoerd worden en kan er gekeken worden of er nu minder resource nodig is om hetzelfde resultaat te bereiken.

Enterprise Manager:
Dit tool geeft informatie van een database server, zoals de databases die erop staan en bijbehorende tabellen. Ook is te zien wie er is ingelogd op de database server, kunnen er logs worden bekeken van de server, enz.

Quest Software
www.quest.com

Quest Spotlite:
http://www.quest.com/spotlight_on_sql_server_enterprise/

Spotlite kan gebruikt worden in plaats van Windows Performance Monitor. Er is dan geen kennis nodig over performance counters die gebruikt kunnen worden en welke waarden hiervan aangeven dat er slechte performance is. Met Spotlite kunnen zowel de resources van Windows als van SQL Server bekeken worden. Zo kan bekeken worden hoe de CPU en het ram geheugen belast worden. Voor SQL Server kan er bijvoorbeeld gekeken worden hoeveel CPU of I/O resource een bepaalde gebruiker in beslag neemt. Ook kan er bekeken worden hoe het cache van de SQL Server gebruikt wordt.

Spotlite kan zichzelf kalibreren om zo te weten wat de normale toestand is, en wanneer waarden binnen een kritisch bereik komen. Als dit het geval is kan er een melding worden gegeven.

Quest Central:
http://www.quest.com/Quest_Central_for_SQL_Server/

Quest Central is hetzelfde als de Enterprise Manager van SQL Server. Deze is niet van belang voor het onderzoek en zal verder niet besproken worden.

Quest Central - Performance Analysis:
http://www.quest.com/Quest_Central_for_SQL_Server/performance_analysis/performance_analysis.asp

Performance Analysis slaat performance data op over uitgevoerde SQL statements. De performance metrics die gelogd en geanalyseerd worden zijn allemaal gebaseerd op SQL Server wachttoestanden, zoals CPU wait, I/O wait, Network wait, enz. Er kan op ieder van deze kolommen gesorteerd worden. Zo kan makkelijk gezien worden wat de bottleneck is van een bepaald SQL statement. Ook is het mogelijk om het resource gebruik van de verschillende users in te zien.
Een andere mogelijkheid van Performance Analysis is het bijhouden van configuratie opties van bijvoorbeeld SQL Server parameters. Het veranderen van deze parameters kan de invloed hebben op de performance van SQL Server.

Quest Central - SQL Tuning:
http://www.quest.com/quest_central_for_sql_server/sql_tuning/sql--tuning.asp

SQL Tuning kan voor een gegeven SQL statement nieuwe SQL statements genereren die semantisch gelijk zijn. De manier waarop het SQL statement wordt uitgevoerd is hierbij verschillend. Het originele SQL statement en de gegenereerde statements kunnen vervolgens allemaal uitgevoerd worden, en kan er per statement gekeken worden hoeveel van een bepaalde resource dit in beslag neemt. Ook wordt er aangeven welk statement het minste tijd kost om uitgevoerd te worden. Er kan dan voor gekozen worden om een efficiënter gegenereerd SQL statement te gebruiken.

SQL Scanner:
SQL Scanner is een onderdeel SQL Tuning en kan uit bijvoorbeeld trace-logs SQL statements halen en deze indelen in verschillende categorieën van performance. SQL statements die kritisch kunnen zijn voor performance kunnen vervolgens geprobeerd worden te optimaliseren met SQL Tuning.

Quest Central - Database Analysis:
http://www.quest.com/Quest_Central_for_SQL_Server/database_analysis/database_analysis.asp

Database Analysis analyseert SQL Server en geeft een algemeen beeld van de status ervan. Het tool kan aanbevelingen en advies geven over hoe de SQL Server getuned kan worden. Dit afhankelijk van de workload, configuratie en applicaties.

Quest Benchmark Factory:
http://www.quest.com/benchmark_factory/default.aspx

Met Benchmark Factory kan de load van een database server getest worden. Zo is te zien bij welk aantal users er een acceptabele response tijd is en hoe de performance van de database server varieert bij verschillende loads.
Met Benmark Factory kunnen meerdere gebruikers gesimuleerd worden. Ook kunnen er taken ingesteld worden die de taken van een gebruiker simuleren. Per taak is ook aan te geven hoe vaak deze uitgevoerd word in tegenstelling tot een andere taak. Voor een betreffende taak kunnen de SQL statements opgegeven worden die daarbij uitgevoerd moeten worden. Ook is het mogelijk om deze SQL statements te genereren uit verschillende soorten log-files.

Quest Speed Coefficient:
http://www.quest.com/coefficient/

Met Speed Coefficient kunnen statistieken gegenereerd worden uit trace-logs van Microsoft SQL Server. Zo kan er informatie uit de trace-logs gehaald worden over hoe vaak bepaalde SQL queries voor komen. Ook is het mogelijk om een in te stellen top aan queries weer te geven die bijvoorbeeld de meeste CPU tijd gebruiken of de meeste reads of writes hebben. Met Speed Coefficient kan zo dus belangrijke informatie uit grote hoeveelheden trace-log data gehaald worden.

Embarcadero Software
www.embarcadero.com

DBArtisan:
http://www.embarcadero.com/products/dbartisan/dbdatasheet.html

DBAtrisan bevat verschillende pakketten voor het managen van databases, zoals availability, performance en security. Het kan vergeleken worden met Enterprise Manager van SQL Server. Voor het onderzoek naar het databasegebruik zijn slechts de pakketten voor performance van belang.

Embarcadero Performance Analyst:
http://www.embarcadero.com/products/dbartisan/padatasheet.html

Performance Analyst is geïntegreerd in DBArtisan en geeft statistieken van verschillende performance categorieën zoals CPU, ram en I/O. Ook kan bekeken worden waar hierin de bottleneck zit. Statistieken van resource gebruik zijn ook te zien per SQL statement. Al deze statistieken kunnen worden weergegeven worden op zeer gedetailleerd niveau. Performance Analyst biedt ook de mogelijkheid om thresholds op te geven, zodat weergegeven kan worden wanneer een threshold overschreden wordt.

Performance Center:
http://www.embarcadero.com/products/performancecenter/index.html

Met Performance Center kan er gekeken worden naar de healt index van een database server. De healt index wordt berekend aan de hand van de status van geheugen, CPU, I/O, netwerk, enz. Zo kan snel gezien worden of een database server performance problemen heeft.

Test
Van beide tools is een trial versie geïnstalleerd en geprobeerd. De trial versies bezitten de volledige functionaliteiten maar zijn maar een beperkte tijd te gebruiken. Zowel de tools van Quest als van Embarcadero maken in de database stored procedures aan die gebruikt worden om analyses uit te voeren. Voor de tools van Quest worden er ook extra tabellen in de database aangemaakt waar informatie opgeslagen wordt om hier analyses op uit te voeren.
De tools kunnen ook de status van de server zelf weergeven, hiervoor zijn echter de juiste rechten nodig op de server. Voor enkele tools van Quest zijn ook speciale rechten nodig op de server om deze uit te kunnen voeren.

Conclusie
Quest en Embarcadero hebben beide krachtige tools voor database administrators. De verschillende tools van beide hebben veel overeenkomsten qua informatie en analyse die zij kunnen geven over de database server en SQL statements.
Op internet is geen duidelijke voorkeur te vinden voor tools van een van beide. Alle twee hebben ze hun voor en nadelen, en moet het juiste tool voor de juiste situatie gekozen worden. Aangezien het onderzoek gericht is op het analyseren van de trace-logs en het verbeteren van de performance zal er gekozen worden voor de tools van Quest. Quest beidt hiervoor de betere tools aan. Deze tools kunnen dan ook gebruikt worden tijdens het onderzoek indien de juiste rechten op de database server worden verleent om de tools uit te voeren.

De tools van Quest die gebruikt kunnen worden:

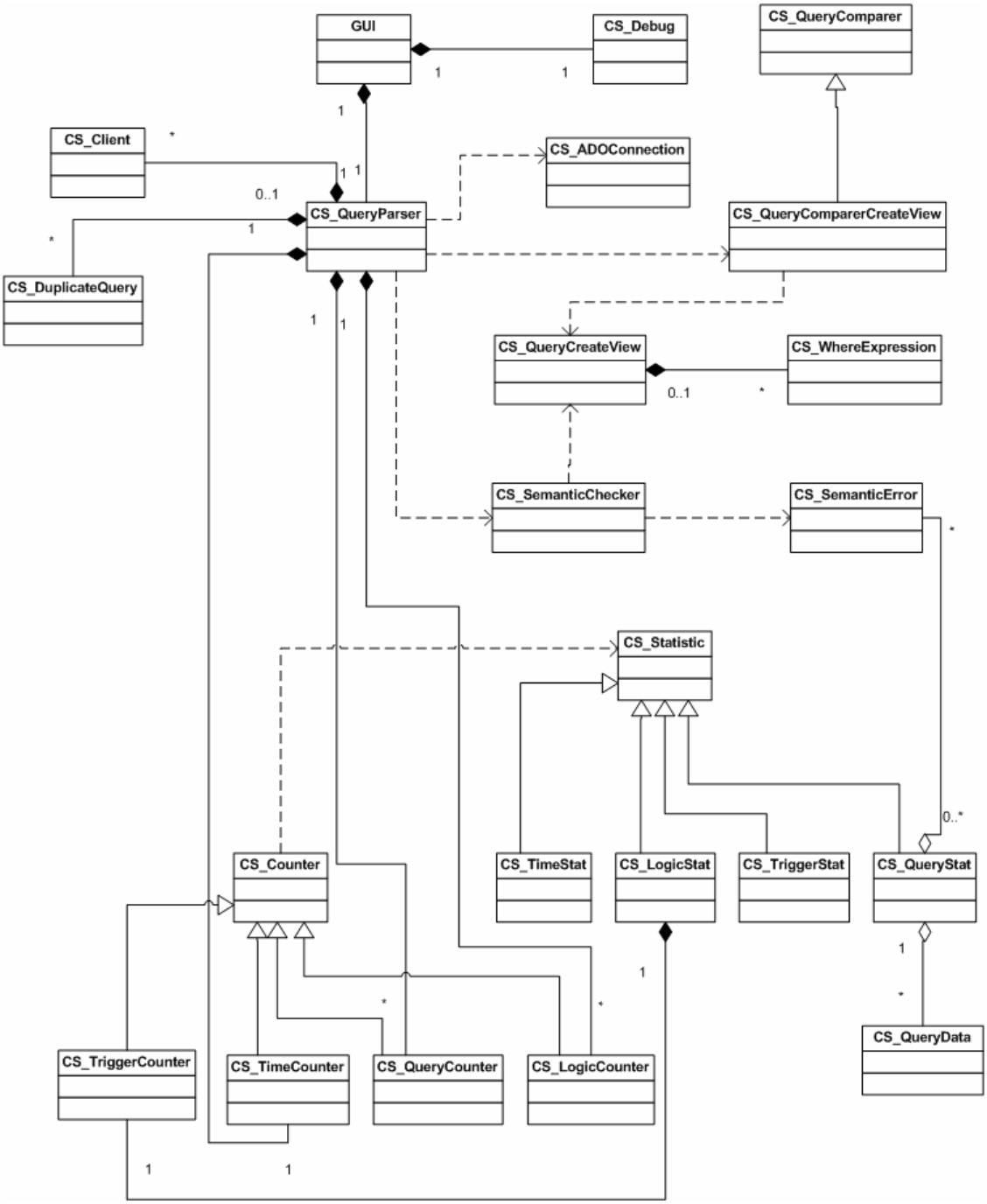
Speed Coefficient:
Voor het genereren van statistieken uit trace-logs.

Quest Central – SQL Tuning:
Voor het proberen te genereren van geoptimaliseerde SQL statements.

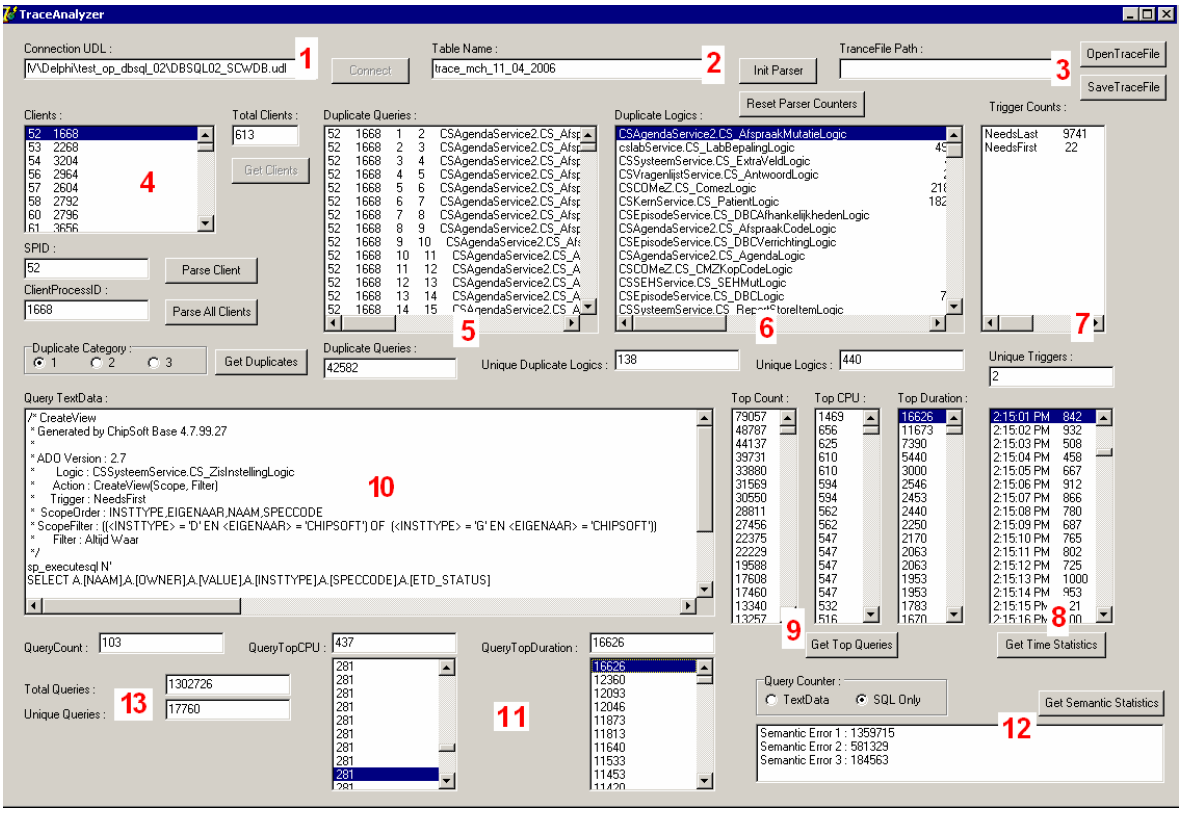
Quest Benchmark Factory:
Voor het testen van de load (performance) van de database server bij een aangepast SQL statement.

Quest Spotlite & Quest Database Analysis:
Voor analyse van de database server, en het maken van verdere aanbevelingen over het databasegebruik.

Bijlage 3 : TraceAnalyzer class diagram


GUI
CS_Debug
CS_QueryComparer
CS_Client
CS_ADOConnection
CS_QueryParser
CS_QueryComparerCreateView
CS_DuplicateQuery
CS_QueryCreateView
CS_WhereExpression
CS_SemanticChecker
CS_SemanticError
CS_Statistic
CS_Counter
CS_TimeStat
CS_LogicStat
CS_TriggerStat
CS_QueryStat
CS_TriggerCounter
CS_TimeCounter
CS_QueryCounter
CS_LogicCounter
CS_QueryData

Bijlage 4 : TraceAnalyzer handleiding



1. Connection

In het veld “Connection UDL” dient het pad en bestandsnaam van het UDL bestand ingevuld te worden van de database server waarmee gewerkt wordt. Standaard wordt hier het pad ingevuld waarin de applicatie staat, en het default UDL bestand DBSQL04_MCH_TRACE.udl. Met de knop “Connect” kan vervolgens de verbinding worden gemaakt met de database server.

2. Trace Table

Bij “Table Name” moet de tabelnaam worden ingevuld van de trace tabel die moet worden geanalyseerd. Door op “Init Parser” te klikken wordt deze ingesteld, en kan gestart worden met het ophalen van gegevens.

Met “Reset Parser Counters” kunnen de analysegegevens gewist worden. Dit kan gebruikt worden als gebruikers (4) stuk voor stuk geanalyseerd willen worden. Bij het niet resetten worden de resultaten van de losse analyses per gebruiker samengevoegd. De opgehaalde gebruikers (4) en tijd statistieken (8) worden niet gereset. Deze blijven hetzelfde zolang de applicatie niet wordt geinitialiseerd met een nieuwe tabel.

3. Save Trace File

Indien geen trace tabel voor handen is, maar een trace bestand, dan kan dit bestand met "OpenTraceFile" geopend worden. Het pad van het trace bestand wordt hierbij omgeschreven naar een netwerkpad (\\Server\Map\...). Indien het trace bestand van de lokale C:\ schijf wordt geopend dan wordt het pad vervangen door de standaard C$ share. Bij het openen van een trace bestand wordt het veld "Table Name" (2) gevuld met de bestandsnaam van het bestand.

Met "SaveTraceFile" wordt het bestand opgeslagen in de database MCH_Trace welke op DBSQL04 aanwezig is. Het is dus van belang dat voor deze optie een verbinding is gemaakt met DBSQL04.

4. Clients

Met de knop "Get Clients" worden alle gebruikers uit een trace-log opgehaald. Een unieke gebruiker wordt geïdentificeerd aan het SPID (1^e getal) en ClientProcessID(2^e getal). Bij het selecteren van een gebruiker uit de lijst, wordt het "SPID" veld en "ClientProcessID" gevuld, waarna met de knop "Parse Client" alle queries voor deze gebruiker geanalyseerd worden. Met "Parse All Clients" worden alle queries van alle gebruikers uit de lijst geanalyseerd.

5. Duplicate Queries

Na het analyseren van de queries wordt in deze lijst de dubbele queries weergegeven. Dit zijn standaard de dubbele queries uit categorie 1. Dit zijn exact dezelfde queries die na elkaar zijn uitgevoerd.
In de "Duplicate Category" box kan ook voor dubbele queries uit categorie 2 en 3 gekozen worden. Als op "Get Duplicate" wordt geklikt dan worden de betreffende queries uit deze categorie weergegeven. Bij dubbele queries uit categorie 2 zijn de Base versie, de ADO Version, de Logic, de Select velden, en het From veld gelijk aan elkaar. Bij queries uit categorie 3 komt hierbij dat de ScopeOrder en ScopeFilter ook gelijk zijn aan elkaar.
In de lijst wordt het SPID, het ClientProcessID, het regelnummer van de eerste query, het regelnummer van de tweede query, de Logic, en de triggers van de twee queries weergegeven. De regelnummers kunnen worden gebruikt om de betreffende queries te bekijken in SQL Profiler. Tel hierbij wel 1 op bij de regelnummers, dit omdat SQL Profiler als eerste regel "TraceStart" weergeeft.
In het veld "Duplicate Queries" wordt het aantal dubbele queries uit de betreffende categorie weergegeven.

Let op: *Er worden alleen dubbele queries gedetecteerd van het type CreateView.*

6. Duplicate Logics

In deze lijst worden de logics weergegeven van de dubbele queries, gesorteerd op de logic die de meeste dubbele queries bevat. In de lijst staat de logic, het aantal dubbele queries van de logic, het totaal aantal queries van de logic, en het percentage van het aantal dubbele ten opzichte van het totaal van de betreffende logic. Bij het selecteren van een logic uit deze lijst worden in (7) de trigger combinaties weergegeven van de dubbele queries.

Het veld “Unique Duplicate Logics” geeft het aantal logics weer waarvan een dubbele query is voorgekomen. Het veld “Unique Logics” geeft het aantal verschillende logics weer die zijn voorgekomen in de trace-log.

7. Trigger Counts

Hier worden voor een betreffende logic de trigger combinaties weergegeven die voorkomen bij de dubbele queries en het aantal keer dat deze combinatie voorkomt. Bij dubbele queries van categorie 1 zijn de triggers van een combinatie uiteraard altijd gelijk aan elkaar. Het veld “Unique Triggers” geeft het aantal verschillende trigger combinaties voor de betreffende logic.

8. Time Statistics

Met “Get Time Statistics” worden de queries geteld die per seconde zijn uitgevoerd, en wordt dit weergeven in de lijst.

9. Top Queries

Met de knop “Get Top Queries” worden de lijsten “Top Count”, “Top CPU” en “Top Duration” gevuld. De data in deze lijsten is aflopend gesorteerd. De “Top Count” lijst bevat de waarden van hoe vaak een query voorkomt. De “Top CPU” lijst bevat de hoogste CPU waarde van een query, en “Top Duration” de hoogste Duration waarde van een query.
Bij het klikken op een item in een van de lijsten wordt de query van het betreffende item weergegeven in (10). Daarbij worden ook de lijsten en velden bij (11) gevuld.

Bij het ophalen van de top waarden, wordt er een vergelijking gemaakt om te bepalen of het om dezelfde query gaat. Hierbij kan onderscheid gemaakt worden tussen de vergelijking van het hele TextData veld, of alleen het SQL statement uit het TextData veld.

Bij vergelijking van alleen het SQL statement wordt, als er een item uit een van de lijsten wordt gekozen, in (12) de semantische errors van de query weergegeven.

Let op: *Bij het vergelijken van alleen het SQL statement worden alleen queries vergeleken van het type CreateView.*

10. Query
Dit veld geeft de betreffende query weer als een item wordt gekozen uit een van de lijsten bij (9).

11. Top Times
Deze velden worden gevuld bij het selecteren van een item uit een van de lijsten bij (9). Het veld “Top Count” geeft weer hoe vaak de betreffende query in (10) voorkomt. Het veld “QueryTopCPU” geeft de hoogste CPU waarde voor de query weer, en het veld “QueryTopDuration” de hoogste Duration waarde. In de lijsten onder “QueryTopCPU” en “QueryTopDuration” worden de overige waarden weergegeven voor deze velden, gesorteerd op aflopende volgorde.

Als in de lijst van “QueryTopDuration” op een item wordt geklikt, dan wordt in de lijst van “QueryTopCPU” ook de bijbehorende waarde geselecteerd. Indien de tijd statistieken (8) zijn opgehaald, wordt ook de seconde geselecteerd waarin de betreffende query is uitgevoerd.
Ook wordt in het geval er is vergeleken op alleen het SQL statement, het hele TextData veld weergegeven in (10) van een van de queries uit de lijst.

12. Semantic Errors
In deze lijst worden de semantische errors weergegeven van een query bij het klikken op een item uit een van de lijsten bij (9). Semantische errors worden alleen weergegeven indien bij (9) “SQL Only” is geselecteerd. Met “Get Semantic Statistics” worden de totalen van het aantal semantische errors weergegeven in de lijst. Er worden drie verschillende semantische errors geïdentificeerd.

Error 1 : Constante output kolom
Hierbij wordt gekeken of een kolom in de *Where* clausule wordt vergeleken met slechts één variable. Indien dit het geval is, dan hoeft deze kolom niet worden opgehaald middels *Select*. De waarde van deze kolom zal altijd gelijk zijn aan de variable, en heeft geen toegevoegde waarde.

Error 2 : Onnodige Order By kolom
Net als bij een constante output kolom, wordt gekeken of een kolom in de *Where* clausule slechts met één variable wordt vergeleken. Indien dit het geval is, dan is de waarde van deze kolom altijd gelijk. Het voorkomen van deze kolom in *Order By* heeft dan geen toegevoegde waarde, daar de volgorde altijd hetzelfde is.

Error 3 : Vergelijking met NULL en ‘’
Hierbij wordt gekeken of eenzelfde kolom in de *Where* clausule wordt vergeleken met IS NULL en LIKE ‘’. Beide hebben dus als doel om te kijken of er geen waarde in de kolom staat. Het verschil is echter dat een keer wordt gekeken of de kolom echt leeg (NULL) is, en de andere keer of er een lege waarde (‘’) in staat.

13. Parse Statistics

Na het analyseren van de queries van een of meerdere gebruikers, wordt in “Total Queries” het aantal queries die zijn geanalyseerd weergegeven. Het veld “Unique Queries” geeft weer hoeveel verschillende queries er zijn voorgekomen tijdens de analyse. Het veld wordt gevuld na het klikken op “Get Top Statistics” (9), en is afhankelijk van of het hele TextData veld is vergeleken, of alleen het SQL statement.