Afstudeerstageverslag

Pim van Dongen

Verbeteren performance rapportagegeneratie

Stageperiode: 2013-1.1
Studentnummer: 09033580
Opleiding Informatica

Inhoudsopgave

1: Inleiding

In dit document beschrijf ik hoe mijn afstudeerstage bij Panteia in Zoetermeer is verlopen. Het doel van dit document is het verloop van mijn afstudeerstage te beschrijven zodat de examinatoren het proces en het eindproduct kunnen beoordelen.

Ik heb gekozen voor Panteia als afstudeerbedrijf omdat de opdracht mij de mogelijkheid zou geven om tonen hoe ver mijn kennis gaat in een voor mij bekende softwareomgeving.

2: Over Panteia

Op de website van Panteia[1] staat een beschrijving van het bedrijf.

Over Panteia
Panteia is een allround onderzoeksbureau voor economisch en sociaal beleidsonderzoek, transportonderzoek en marktonderzoek. Ons werk omvat de volledige breedte van de maatschappij. Met meer dan 250 medewerkers is Panteia het grootste onderzoeksbureau van Nederland.

Panteia heeft als missie: bijdragen aan een betere samenleving door de randvoorwaarden te scheppen voor beter beleid van onze klanten. Bij Panteia werken hoogopgeleide professionals uit alle relevante vakdisciplines. Voor uw probleem stellen wij het sterkst mogelijke team samen. Zo helpen wij onze klanten vooruit met wat wij noemen: Research to Progress. Voor consultancywerkzaamheden heeft Panteia een strategische samenwerkingsrelatie met de Van Spaendonck Groep (VSG).

Full service op beleids- en marktonderzoek
Panteia komt voort uit samenvoeging van toonaangevende onderzoeksbureaus. De namen van deze bureaus leven voort in de door Panteia gehanteerde merken:

EIM, voor economisch beleidsonderzoek, vooral is gericht op het bedrijfsleven;
Research voor Beleid, dat vooral sociaal gericht beleidsonderzoek uitvoert;
NEA, dat gespecialiseerd is in onderzoek, training en advisering op het gebied van verkeer, transport en logistiek;
Stratus, gespecialiseerd in marktonderzoek, klant- en medewerkeronderzoek, dataverzameling en monitoring;
IOO, dat zich richt op doelmatigheidsonderzoek voor de publieke sector;
IPM, gespecialiseerd in kwalitatief marktonderzoek voor commerciële organisaties in de sectoren fmcg, food/non food, durables, retail, financiën en farmacie.

Het logo van Panteia:

Panteia
Research to Progress

[1] www.panteia.nl

De thema's

In het onderstaande organogram is de structuur van Panteia gemodelleerd. (Zie Figuur 1).

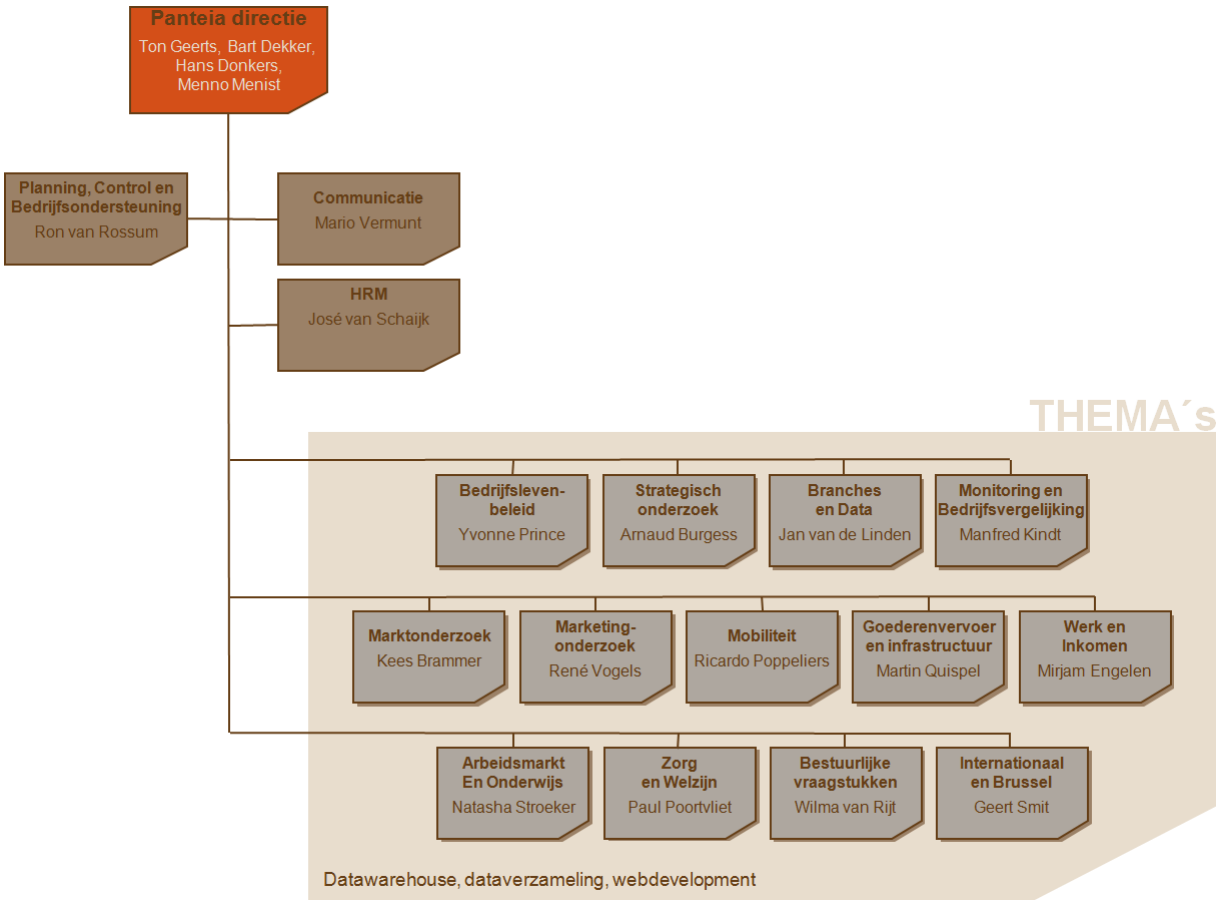


Fig1. Organogram Panteia.

Panteia is opgedeeld in thema's. Dit zijn afdelingen die bezig houden met een bepaald type onderzoek. De beschrijvingen van deze thema's zijn van de website van Panteia afkomstig.

Bedrijfslevenbeleid

De kernactiviteiten van het thema Bedrijfslevenbeleid betreffen onderzoek, informatie en advies over alle beleidsvraagstukken die betrekking hebben op de mutatiemomenten van bedrijven, van start en groei, innovatie en internationalisering, bedrijfsprestaties en financiering, tot krimp en overdracht, alsmede die betrekking hebben op de bedrijfsomgeving zoals het ondernemingsklimaat, de concurrentieverhoudingen op de markt, toegang tot openbare aanbestedingen, arbeidsmarkttekorten, kennisoverdracht, ICT en samenwerking.

Strategisch onderzoek

Het thema Strategisch Onderzoek identificeert en investeert in de toekomstige markten van Panteia. De investeringen stellen wij vast in samenspraak met de andere thema's. De valorisatie van de innovatieve producten die Panteia ontwikkelt in de programma's – het EL&I Programmaonderzoek en het EC Kaderonderzoek (FP7) - staat centraal.

Het thema Strategisch Onderzoek zet zich in voor het behoud en voor versterking van het EL&I Programmaonderzoek en van het EC Kaderonderzoek in alle themagroepen. Het thema Strategisch Onderzoek wil synergie creëren tussen de beide programma's. Het thema Strategisch Onderzoek wil

het terrein waarop Panteia actief is in binnen het EC Kaderprogramma verbreden en daarbij alle thema's, zover mogelijk, betrekken.

Kernbegrippen voor het werk van het thema Strategisch Onderzoek zijn: identificeren van toekomstige markten voor onderzoek; het creëren van draagvlak; 'value for money'; valorisatie; ontwikkelen van Panteia als kennisinstituut naast de Planbureaus en de Kennisinstituten; up-to-date datacollectie op het gebied van MKB en Ondernemerschap.

Branches en Data

Het thema 'Branches en Data' richt zich volledig op de analyse en beschrijving van de eigen markt waarop ondernemingen opereren, gekoppeld aan een diepgaande SWOT-analyse voor de branche-eigen zaken. Het beschrijft de huidige aanbod- en vraagstructuur en de te verwachten veranderingen daarin gegeven de trends die zich op de markt voordoen. De concurrentieverhoudingen op de markt worden beschreven en de kracht en zwaktes van de branche-eigen zaken worden nauwgezet in beeld gebracht. Het onderzoek wordt gecompleteerd met duidelijke adviezen aan de ondernemingen binnen een branche en aan de branche-organisaties. Hierin worden de mogelijkheden die de branche-eigen zaken hebben om in de toekomst sterker te staan ten opzichte van allerlei concurrenten op de eigen en aanpalende markten helder in beeld gebracht en wordt tevens aangegeven wat de brancheorganisaties kunnen doen om het verbeterproces bij de bedrijven te versterken.

Monitoring en Bedrijfsvergelijking

In het thema Monitoring en Bedrijfsvergelijking houdt men zich bezig met monitoren van het bedrijf. Dit betekent sturen op cijfers, dataverzameling, analyse en rapportage.

Sturen op cijfers

Panteia maakt producten die worden gebruikt bij planning, benchmarking, financieringsaanvragen, verkoop-analyses en lobbyactiviteiten. Daarbij leveren ze de juiste informatie op het juiste moment, waarbij de focus ligt op snelheid van opleveren, accuratesse en flexibiliteit. Panteia voert projecten zowel jaarlijks, per kwartaal, maandelijks of wekelijks uit. Deze projecten bestaan uit drie fases: dataverzameling, analyse en rapportage.

Dataverzameling

Een van de belangrijkste onderdelen van de projecten is de dataverzameling. Zonder goede data geen informatie. De data is onder andere afkomstig van groothandels, grootwinkelbedrijven, fabrikanten, kassasystemen en jaarrekeningen. Afhankelijk van het soort project wordt de data geaggregeerd of op transactieniveau, wie heeft welk product gekregen, aangeleverd.

Analyse

De eerste stap van de analyse bestaat uit het controleren van de aangeleverde data. Zijn er afwijkende patronen? Is alle data wel aangeleverd? Indien nodig neemt Panteia contact op met de leverancier van de data. Vervolgens wordt de data geüniformiseerd, zodat er geen appels met peren worden vergeleken.

Rapportage

De rapportage kan op verschillende manieren plaatsvinden, zowel op papier als via internet. Afhankelijk van de doelgroep van de informatie wordt deze op maat aangeboden. Fabrikanten hebben bijvoorbeeld een andere informatiebehoefte dan brancheverenigingen en de

informatiebehoefte van Account Managers kan weer verschillen van die van ondernemers. De resultaten kunnen ook als interactieve online dashboards worden aangeboden. De gebruikers kunnen dan zelf aan de knoppen zitten .

Het thema verkoopt ook een product bedrijfsvergelijking waarbij kennis wordt gedeeld van het eigen bedrijf en vergelijken referentiebedrijven. Vanuit de eigen administratie en de balans en resultatenrekening wordt informatie over de ontwikkeling van de opbrengsten, de kosten omgezet naar relevante kengetallen. Bij de interpretatie van deze gegevens wordt referentiemateriaal gebruikt van vergelijkbare bedrijven. Vaak is dat soort referentiemateriaal afwezig, te gedateerd of niet toegesneden op uw specifieke situatie.
In dit soort gevallen biedt bedrijfsvergelijking uitkomst. Door het inschakelen van een onafhankelijke partij voor het delen van vertrouwelijke bedrijfsinformatie, wordt meerwaarde gecreëerd. Deze ontstaat door op een uniforme wijze (definities) bedrijfseconomische gegevens van gelijksoortige bedrijven te verzamelen, kengetallen te berekenen en op basis hiervan ijkpunten op te stellen.

Deze ijkpunten zijn zeer bruikbaar bij het beoordelen van de eigen bedrijfsprestaties. De eigen prestaties op bepaalde kengetallen kunnen worden vergelijke met die van gelijksoortige bedrijven. Hiermee ontstaat inzicht in sterke punten, maar ook in de verbetermogelijkheden.

Marktonderzoek

Marktonderzoek staat in dienst van de gehele gedrags-, koop-, product- en beleidscyclus van organisaties: van de voorbereiding tot en met de evaluatie. Het is een belangrijk middel om zicht te krijgen in de beweegredenen of gedragingen van betrokkenen. Meten is weten. De essentie van marktonderzoek is het verzamelen en analyseren van de gewenste gegevens op een (statistisch) betrouwbare manier.

Marketingonderzoek

Panteia biedt praktische en effectieve marketingoplossingen vanuit de psychologie van de consument. Door onze roots in de consumentenpsychologie en marketing is Panteia al meer dan vijftig jaar voor veel bedrijven en organisaties de basis voor succesvolle marketingintroducties en communicatiecampagnes. Wij bieden inzicht in zowel het bewuste als onbewuste gedrag van de doelgroep (met behulp van kwalitatief onderzoek (face to face/online) en kwantitatief onderzoek). Hiermee maken we irrationeel gedrag grijpbaar. Op basis van onze ervaring en creativiteit zijn wij in staat de opgedane inzichten te vertalen naar effectieve en praktische marketingoplossingen.

Mobiliteit

Bereikbaarheid is van vitaal belang voor de economie. Transport en mobiliteit spelen daarin een belangrijke rol, of het nu over de weg, spoor of water of door de lucht gaat. Ook vanuit sociaal oogpunt is mobiliteit voor iedereen natuurlijk erg belangrijk. Door onder meer congestie en stijgende olieprijzen is slim en optimaal gebruik van transportsystemen steeds belangrijker. Daarbij is er een toenemende druk vanuit de maatschappij om duurzaam te transporteren. Geluidshinder en emissies van schadelijke stoffen worden steeds minder geaccepteerd.

Panteia geeft u grip op deze ontwikkelingen met gedegen (internationaal en nationaal) onderzoek, advies en begeleiding op basis van de nieuwste inzichten. Verbeteringen in de infrastructuur, het inzetten van intermodale vervoerconcepten, het ontwikkelen en benutten van intelligente transportsystemen en intensievere - op innovatie toegeruste - samenwerking in de gehele

vervoerketen, zal broodnodig zijn om de uitdagingen van morgen aan te kunnen. Soepel lopende vervoerstromen zijn immers van levensbelang voor de economische slagkracht en sociale cohesie van Nederland en Europa.

Panteia levert overheden en organisaties betrokken in transport– en mobiliteitsvraagstukken, inzichten en advies gebaseerd op feiten. Wij beschikken over wereldwijde ervaring en expertise waarop beleidsmakers en beslissers verder kunnen bouwen. Binnenlandse en buitenlandse klanten contracteren Panteia om beleidsonderzoek uit te voeren, advies uit te brengen, en partners te begeleiden in haalbaarheid en economische effecten van: infrastructuur, logistieke en fysieke distributie, duurzame mobiliteitsvisies en plannen (inclusief openbaar vervoer), visies op zorgvervoer en vertaling van beleidsvisies in aanbestedingen, implementatie en evaluatie van beleid.

Goederenvervoer en infrastructuur

Voor verschillende partijen maakt Panteia prognoses en scenariostudies over ontwikkelingen in de vervoermarkten. Wij beschikken over de nodige data, modellen en panels die ons in staat stellen om trends te duiden en van economische betekenis te voorzien. Zo publiceren we op eigen initiatief ieder kwartaal de Korte Termijn Voorspeller (KTV) Goederenvervoer met het doel voor de volgende vier kwartalen inzicht te geven in de ontwikkeling van het goederenvervoer voor zowel over de weg, het spoor als via de binnenvaart. Daarnaast brengen we diverse andere ontwikkelingen in beeld zoals trends in de vervoerskosten en trends in het openbaar vervoer.

Panteia heeft als onafhankelijke onderzoek- en adviesorganisatie een brugfunctie tussen beleidsontwikkeling en bedrijfsvoering. Hierdoor zijn we zeer goed op de hoogte van allerlei ontwikkelingen op beleidsniveau en in de markt rond de organisatie van verkeer en vervoer. Denk hierbij bijvoorbeeld aan Intelligente Transport Systemen (ITS) door ICT-toepassingen in de transportinfrastructuur en voertuigen om het verkeer veiliger, efficiënter, betrouwbaarder en milieuvriendelijker te maken. Daarbij kunnen we ook de kwaliteit van dit soort ontwikkelingen toetsen door monitoring, klanttevredenheidsonderzoeken en evaluatie. We geven trainingen op maat, én wanneer nodig, is het mogelijk om onze experts op detacheringbasis in te huren. Hierdoor kunnen we de capaciteit in uw organisatie aanmerkelijk versterken voor een betere aansluiting op de veranderende behoeften uit de samenleving.

Werk en Inkomen

Nederland staat voor grote uitdagingen op de arbeidsmarkt. De beroepsbevolking vergrijst en krimpt. Werkgelegenheid is in onze open economie gevoelig voor outsourcing, offshoring en conjunctuurschokken. En minder kansrijke groepen zoals werklozen, etnische minderheden en arbeidsgehandicapten staan nog vaak langs de kant. Dit soort ontwikkelingen vraagt om een adequate reactie van beleidsmakers. Dit thema ondersteunt beleidsmakers bij het vinden van antwoorden op deze complexe vraagstukken.

Arbeidsmarkt en Onderwijs

Dit thema verbindt kennis van relevante databestanden met inzicht in de werking van de arbeidsmarkt en onderwijs. Ook hebben we ons gespecialiseerd in het maken van middellange en lange termijn prognoses van vraag en aanbod. Daarbij richten we ons wat betreft de arbeidsmarkt zowel op het nationale niveau als op het regionale niveau, evenals op de verschillende sectoren en branches. Relevante thema's zijn: de arbeidsmarktgerichtheid van het beroepsonderwijs, imago van

bedrijfstakken als werkgevers, employability en oudere werknemers, re-integratie van arbeidsgehandicapten en werklozen, zelfstandigen zonder personeel, het Europees Sociaal Fonds, etc.

Zorg en Welzijn

De beleidsterreinen zorg, welzijn en jeugd zijn in beweging. In de komende jaren vindt decentralisatie van de jeugdzorg en de extramurale begeleiding plaats en zorgverzekeraars gaan de AWBZ uitvoeren. Tegelijkertijd groeit de vraag naar zorg door de vergrijzing, is er behoefte aan meer personeel in de zorg. Het motto 'Meer kwaliteit met minder geld' stelt gemeenten, zorgverzekeraars en zorgaanbieders voor een uitdaging.

Deze ontwikkelingen vragen om nieuwe samenwerkingsverbanden, het ontwikkelen van vernieuwende arrangementen en een versterkte aandacht voor preventie. Daarvoor is inzicht nodig in de behoeften van cliënten, (de toereikendheid van) het huidige aanbod en mogelijkheden tot verbetering van de organisatie van zorg en ondersteuning. Dit thema levert aan overheden, zorgverzekeraars, zorgaanbieders, brancheorganisaties en kennisinstituten dergelijke inzichten door professioneel, op maat gesneden onderzoek met een hoge gebruikswaarde.

Bestuurlijke vraagstukken

Goed bestuur: wat is dat? Deze vraag is nu zeer actueel. Gemeenten krijgen nieuwe taken, moeten misschien fuseren en kampen met tekorten op hun begrotingen. Ministeries hebben flinke taakstellingen. Provincies manoeuvreren zich tussen deze bestuurslagen door. Ondertussen liggen de plannen om landsdelen te vormen weer op tafel en zijn nieuwe regionale uitvoeringsdiensten gevormd. Hoewel de onzekerheden zich opstapelen, is één ding wel duidelijk: het speelveld verschuift zich steeds verder naar de regionale en lokale bestuurders. Panteia beweegt daarin mee en kan beleidsmakers en bestuurders vooruit helpen met het formuleren, monitoren en evalueren van beleid.

Organisaties ontwikkelen voortdurend beleid. Zij doen dat om problemen op te lossen, te verminderen of te voorkomen. Kortom voor goed bestuur. Beleidsadviezen spelen daarbij een belangrijke rol. Een goed beargumenteerd beleidsadvies biedt een beleidsmaker een gedegen basis om te komen tot een weloverwogen besluit. De voorbereiding is vaak een stevige klus, die vraagt om een stap-voor-stap-benadering. Panteia ondersteunt dit proces met eigen unieke data en gericht onderzoek.

Effectief beleid vereist kennis van zaken. Kennis van de effecten en kosten van bestaand beleid, knelpunten in de uitvoeringspraktijk, behoeften aan nieuw beleid en de daarvan te verwachten resultaten. Als beleidsmaker verlangt u hoogwaardige informatie die kan dienen als solide basis voor beslissingen. Beleidsonderzoek is een kernactiviteit van Panteia. Wij genereren kennis en informatie op grond waarvan beleidsmakers en beslissers nieuw beleid kunnen ontwikkelen, of bestaand beleid kunnen aanpassen of afschaffen. Praktische toepasbaarheid is het primaire en ultieme doel van het beleidsonderzoek van Panteia.

Internationaal en Brussel

Dit thema richt zich op internationaal aanleveren van de diensten en producten van Panteia.

Mijn thema: Monitoring, Bedrijfsvergelijking en Webdevelopment

Mijn afdeling, het thema “Monitoring, Bedrijfsvergelijking en Webdevelopment”, is onder andere verantwoordelijk voor het uitvoeren van marktonderzoeken. Deze marktonderzoeken bestaan uit het vergelijken van bijvoorbeeld klanttevredenheid en verkoopcijfers tussen verschillende bedrijven in een marktbranch. Voor klanttevredenheidsonderzoeken worden enquêtes gebruikt. Deze enquêtes worden vaak online gehouden. Om deze online enquêtes te maken en daarna online te ontsluiten omvat het thema “Monitoring en Bedrijfsvergelijking” een team van ontwikkelaars.

Op basis van mijn observaties binnen het bedrijf kan ik concluderen dat de werkwijze die gehanteerd wordt door de ontwikkelaars anders is dan dat ik had verwacht. Men moet met korte deadlines werken. Dit leidt tot veel hergebruik van bestaande code via knip-en-plakwerk. Daarnaast wordt weinig gedocumenteerd en bestaat het testen uit controleren of het eindproduct werkt.

Een van de producten die mijn thema levert is Blik op Werk. In het marketingplan van mijn thema wordt Blik op Werk beschreven:

Blik op Werk (270.000 euro per jaar)
Sinds 2009 verzorgen we de klanttevredenheidsonderzoeken voor de deelnemers aan het Blik op Werk keurmerk. Het onderzoek is continu van aard, wat inhoudt dat we het hele jaar door voor ongeveer 600 bedrijven die actief zijn in de re-integratie en inburgering hun klanten benaderen. Dit gaat voornamelijk via het web, maar ook telefonisch en schriftelijk. Om dit allemaal behapbaar te houden is in 2010 een portal gebouwd waarin de dienstverleners zelf hun onderzoek kunnen inrichten. Zij leveren cliënten aan, maken keuzes voor de dataverzameling en aan het eind van het jaar halen zij in de portal ook hun rapport op.

Mijn opdracht is gerelateerd aan Blik op Werk. In het volgende hoofdstuk wordt de opdracht in detail beschreven.

3: Opdrachtomschrijving

In dit hoofdstuk wordt de afstudeeropdracht beschreven aan de hand van een beschrijving van de huidige situatie en een voorbeeld waarmee de werkwijze geïllustreerd wordt.

Samenvatting

De opdracht is: het verbeteren van de performance van het genereren van de rapportages voor de klanten van Panteia. Dit is een ruime opdrachtomschrijving, wat betekent dat ik gedurende het afstuderen de opdracht zal moeten specificeren. Dit kan door bijvoorbeeld te specificeren hoe de performance verbeterd gaat worden.

Gedurende het afstuderen heb ik in overleg met mijn leidinggevende besloten om mij te richten op de situatie waar de meeste last ondervonden wordt, een project genaamd Blik op Werk. Een bredere oplossing is welkom, maar de rapportgeneratie bij Blik op Werk is het belangrijkst.

De situatie van Blik op Werk

Blik op Werk[2] is een stichting die zich bezig houdt met kwaliteitscontrole van diensten om mensen weer aan het werk te krijgen. Bedrijven die deze diensten leveren kunnen zich aanmelden voor een tevredenheidsonderzoek. Zij moeten dan een lijst e-mailadressen aanleveren van klanten waar zij mee gewerkt hebben. Deze klanten kunnen dan een enquête invullen over hoe tevreden zij zijn over bepaalde aspecten van het bedrijf. Panteia voert deze tevredenheidsonderzoeken in opdracht van Blik op Werk uit. Het bedrijf kan inloggen op de website van Blik op Werk om een rapport te downloaden over de antwoorden die gegeven zijn op de enquête. Dit rapport wordt direct gegenereerd, wat zo'n 30-35 seconden duurt. Wanneer hetzelfde rapport voor de tweede keer wordt gegenereerd duurt dat zo'n 5-10 seconden. Voor de klanten is deze 30-35 seconden te lang.

Panteia gebruikt van een softwarepakket van IBM genaamd IBM Dimensions. Hiermee kan met een druk op de knop aan de hand van een vragenlijst een database en webpagina's gegenereerd worden voor het afnemen van een enquête. Voor een organisatie als Panteia, waarbij klanten niet lang willen wachten op onderzoeksresultaten, is het belangrijk dat een enquête snel uitgezet kan worden. De tijdsbesparing van een pakket als IBM Dimensions is daarmee zeer belangrijk.

Procesbeschrijving

Panteia voert tevredenheidsonderzoeken in opdracht van Blik op Werk uit.

Bij deze tevredenheidsonderzoeken wordt gebruik gemaakt van een online enquête, omdat de automatisering een enorme besparing van kosten is. Ook werkt dit makkelijker voor de klanten van de bedrijven.

Met behulp van IBM Dimensions[3] wordt een project aangemaakt. Hierbij wordt aan de hand van een vragenlijst automatisch een database opgezet en, als dat nodig is, een applicatie aangemaakt om deze vragenlijst mee te presenteren aan de klanten. Wanneer dit klaar is kunnen de klanten de enquête online invullen.

[2] http://www.blikopwerk.nl/

[3] Een softwarepakket die aan de hand van een vragenlijst een database en applicatie opzet. IBM Dimensions wordt in latere hoofdstukken verder uitgelegd.

Om de enquête in te vullen, gaat een klant met zijn webbrowser naar de website, waar hij de vragenlijst kan invullen. Zijn webbrowser vraagt dan aan de webserver om de pagina met de vragenlijst te tonen. Met behulp van de enquêteringsapplicatie wordt deze pagina gegenereerd.

De klant vult de enquête in. Tijdens het invullen worden de resultaten opgeslagen in de database.

Op een gegeven moment zijn de klanten klaar met invullen van de enquête en wil het bedrijf een overzicht van de resultaten hebben. Hiervoor gaat (bijvoorbeeld) een manager naar een overzichtspagina via dezelfde portal die de enquête gebruikt. Nadat hij daar heeft ingelogd kan hij een overzicht opvragen.

Voor het opbouwen van een rapportage moet men de database raadplegen. De database is geabstraheerd tot slechts 5 tabellen. Om de benodigde gegevens te verkrijgen uit de database moeten er vrij ingewikkelde queries geschreven worden. Om dit werk te versimpelen is door Panteia een module “MrIFunctions” ontwikkeld, die aan de hand van een gestructureerde set restricties queries genereert.

Met behulp van MrIFunctions wordt het opgevraagde rapport live gegenereerd, wat voor Blik op Werk zo’n 30-35 seconden duurt. De klant kan dit rapport dan als een PDF downloaden. Wanneer hetzelfde rapport voor de tweede keer wordt gegenereerd, duurt dat zo’n 5-10 seconden. Voor de klanten is deze eerste 30-35 seconden te lang.

Mijn opdracht is het vinden of maken van een oplossing, die zowel de klant als de ontwikkelaars behartigt: Het genereren van de rapportages moet sneller, zonder dat de ontwikkelaars daar veel extra inspanning voor moeten leveren.

4: Aanpak

In dit gedeelte van het afstudeerverslag wordt op een objectieve wijze beschreven hoe ik de opdracht heb aangepakt. Ik beschrijf hierbij de stappen die ik genomen heb en waarom deze stappen genomen moeten worden.

Tevens beschrijf ik enkele belangrijke beslissingen die ik heb genomen gedurende het project.

Één van deze beslissingen is het niet gebruiken van een detailplanning. Aan het begin van het project, bij het schrijven van het plan van aanpak, had ik namelijk niet genoeg kennis van de situatie om een detailplanning te maken. Voor het plan van aanpak heb ik een globale planning gemaakt en gebruikt. Dit omdat wel bekend was dat ik een aantal stappen moet ondernemen. Ik ga namelijk de huidige situatie analyseren en daarover rapporteren aan de opdrachtgever. In overleg met de opdrachtgever kom ik dan tot een oplossing voor het performanceprobleem. Deze rapportages omvatten de probleemanalyse, architectuuranalyse, requirementsanalyse en technische analyse. In de komende hoofdstukken worden deze rapportages uitgewerkt.

Na het vaststellen van de oplossing voor het performanceprobleem heb ik ook geen gebruik gemaakt van een detailplanning. Het implementeren van de oplossing bestaat uit een cyclus van ontwerpen, implementeren (programmeren) en vervolgens aanpassen van het ontwerp aan de hand van de problemen die ondervonden zijn tijdens het implementeren. Deze aanpassingen heb ik gepresenteerd aan de opdrachtgever. Aan het begin van de cyclus is het ongeveer duidelijk hoe lang het duurt om de huidige aanpassingen in het ontwerp te implementeren. Er is echter niet duidelijk hoeveel iteraties de cyclus nog moet doorlopen, of waar de iteraties van de cyclus uit bestaan. Daardoor kan er geen detailplanning worden gemaakt. Wel is in overleg met de opdrachtgever aan het begin van iedere iteratie het doel en de duur van de iteratie vastgesteld. De volgorde van de cycli werd bepaald door de mate waarin onderdelen van het systeem afhankelijk zijn van elkaar. De eerste iteratie bestaat uit het opstellen van de structuur van het gehele systeem. De volgende iteraties werken een onderdeel van deze structuur uit. Deze iteraties staan beschreven in het hoofdstuk Technisch Ontwerp.

Een tweede beslissing van mij is het niet gebruiken van een systeemontwikkelingsmethode. Het gebruik van een systeemontwikkelingsmethode zorgt ervoor dat het werk op een gestructureerde manier uitgevoerd wordt. Dit komt doordat het maken van documentatie onderdeel is van een systeemontwikkelingsmethode. Deze structuur helpt om op een gecoördineerde manier samen te werken. Aangezien deze opdracht een solo-opdracht is, levert het gebruik van een systeemontwikkelingsmethode niet veel voordeel op. Gedurende het project is technische documentatie gemaakt.

De verdere voordelen van een systeemontwikkelingsmethode bestaan uit controle over de planning van werk en overlegmomenten. In de eerste weken van het project is duidelijk geworden dat ik op bijna ieder moment bij mijn opdrachtgever langs kan voor overleg. Het inplannen van overlegmomenten is daardoor niet nodig.

Daarnaast wordt een systeemontwikkelingsmethode gebruikt om het werk in te plannen en om opdrachtgevers meer controle te geven over de uitvoering van het werk. Ik heb geen systeemontwikkelingsmethode gebruikt omdat er vrijwel geen keuzes te maken zijn over de

uitvoering van het werk. Dit komt doordat de oplossing voor het performanceprobleem bestaat uit een oplosser en niet een oplossing. Dit betekent dat een groot gedeelte van de functionaliteit van de oplossing niet weggelaten kan worden. Pas in de eindfases van de implementatie kunnen keuzes gemaakt worden. In de eerdere fases is het werk echter niet te verdelen in segmenten en zijn er geen keuzes die gemaakt kunnen worden. De voordelen van het gebruik van een systeemontwikkelingsmethode, planning en sturing van het werk en planning van overlegmomenten, zijn hierbij niet van toepassing op dit project. Alhoewel er geen gebruik gemaakt is van een systeemontwikkelingsmethode voor het vastleggen van documentatie, is er wel technische documentatie aan de opdrachtgever opgeleverd. In het afstudeerdossier zijn de rapportages en de technische documentatie opgenomen. Deze technische documentatie bestaat uit het technisch ontwerp en de beschrijving daarvan. Iedere wijziging in deze documentatie is met een advies gepresenteerd aan de opdrachtgever.

Een derde beslissing van mij is het schrijven van een adviesrapport. Gedurende de probleemanalyse is gesteld dat een enkel onderdeel binnen het gehele systeem de grootste bijdrage levert aan het performanceprobleem. Het aanpassen van dit onderdeel zou het meeste effect hebben op het performanceprobleem. Gedurende het project is dit als argument gebruikt om niet naar optimalisaties te kijken in andere systeemonderdelen. Tijdens het werken met deze andere systeemonderdelen zijn echter wel ideeën ontstaan over mogelijke optimalisaties. Het adviesrapport was aanvankelijk niet onderdeel van de opdracht. Het adviesrapport behandelt de mogelijke verdere optimalisaties die Panteia kan toepassen voor het verbeteren van de performance bij het genereren van rapportages.

4.1: Oriëntatie

De oriëntatie is de eerste periode in de opdracht waarin je jezelf oriënteert op het bedrijf, de opdracht en het probleemdomein.

Op mijn eerste dag word ik voorgesteld aan de mensen die bij het thema Monitoring, Bedrijfsvergelijking en Webdevelopment werken. Dit is het thema waar ik voor de periode van mijn afstudeerstage ook onder zal vallen. Dhr. Van der Storm is mijn stagebegeleider. Hij is onderdeel van het webteam van het thema. Zijn collega, dhr. Van Velzen, is ook onderdeel van het webteam. Dhr. Leen is de databaseadministrator voor het thema. Dhr. Kindt is de themamanager van het thema en daarmee mijn baas. Ik zal echter mijn beslissingen moeten voorleggen aan dhr.Van der Storm.[4]

De afstudeeropdracht komt in het kort op het volgende neer: “De performance van het huidige systeem moet verbeterd worden”. Het huidige systeem is geschreven in ColdFusion.

ColdFusion is een scripttaal die gebaseerd is op Java. Dit betekent dat je een Java-applicatie zou kunnen ontwikkelen, die je zonder problemen binnen ColdFusion kan aanroepen. ColdFusion lijkt op een soort XML/HTML/PHP combinatie – programmeren gaat via “tags”, zoals deze in XML/HTML aanwezig zijn. Met ColdFusion kunnen mensen die XML/HTML/PHP gewend zijn gebruik maken van de performance die Java levert.

Aangezien deze opdracht werken met ColdFusion omvat, zal ik gedurende de opdracht ColdFusion moeten leren. De opdracht omvat alleen het lezen van ColdFusion-code.

ColdFusion is per definitie trager dan Java, omdat ColdFusion op Java gebaseerd is. Dhr. Van der Storm is van mening dat de oplossing voor het probleem bestaat uit het herprogrammeren van de ColdFusion-code in Java.

Gedurende mijn eerste dag bij Panteia vraag ik mij iets af:
Dit is niet de eerste keer dat Panteia een tevredenheidsonderzoek uitvoert waar een rapport voor gegenereerd moet worden. Rapportages voor andere klanten hadden geen last van performance problemen – waarom nu wel? Dhr. Van der Storm geeft aan dat ik na een presentatie over het systeem vragen mag stellen.

In de presentatie wordt uitgelegd hoe het huidige systeem werkt. Er wordt gebruikgemaakt van een softwarepakket van IBM genaamd IBM Dimensions. Tijdens de presentatie wordt gedemonstreerd hoe dit pakket werkt – een testproject wordt aangemaakt, met database en al. Deze database wordt vervolgens in een databasebrowser getoond.

Het is opvallend dat de naam van de gegenereerde database erg veel lijkt op een tal van andere databases. Dat zijn databases die ook gegenereerd zijn door IBM Dimensions – andere projecten. Op dat moment worden er twee dingen duidelijk: Ten eerste – het gebruik van dit pakket, IBM Dimensions, is de standaard procedure en om dat aan te passen zou er een hele goede reden moeten zijn. Ten tweede – presentaties waarbij een testproject wordt aangemaakt maken geen gebruik van een testomgeving – er wordt gewoon op een productieserver gewerkt.

[4] In de bijlagen is een personenlijst opgenomen.

Na de presentatie kan ik de code van de rapportageapplicatie bekijken. Het is moeilijk om te zien hoe de code werkt. Bepaalde dingen die in andere programmeertalen als (stijl-)fout worden beschouwd zijn eigenlijk “trucjes” in ColdFusion. Een voorbeeld. Variabelen hebben geen datatype, dus controle of een lijst niet leeg is ziet er zo uit:

```
<cfif listLen(list)>
```

Dat is zeer verwarrend – wat er gebeurt is het volgende: “False” en “0” zijn hetzelfde. Ieder ander getal dan “0” is gelijk aan “True”. Bij een waarde van “True” gaat de code door met uitvoeren van de code in het if-statement; anders wordt het if-statement overgeslagen. Wanneer een lijst leeg is geeft `listLen(list)` (lijstLengte(lijst)) “0” terug. “0” is gelijk aan “False”, dus alleen wanneer de lijst niet leeg is wordt de code in het if-statement uitgevoerd. Dit is tegenstrijdig met wat ik gewend ben: een functie geeft “True” of “False” terug – als het een getal is dan moet je dit eerst vergelijken. Ik ben dus gewend aan dit:

```
if(! list.isEmpty()) //Als(NIET lijst.isLeeg())
if(list.size() != 0) //Als(lijst.grootte() niet gelijk aan 0)
```

In beide gevallen hier is er duidelijk een negatief aangegeven – het is altijd duidelijk dat dit wordt uitgevoerd als iets “niet” zo is. Dit is slechts 1 van de voorbeelden – er zijn nog vele andere situaties die niet helemaal duidelijk zijn.

Om de code beter te begrijpen heb ik de queries bestudeerd. Opmerkelijk is dat veel van de queries er ongeveer hetzelfde uitzien. Om het resultaat van de queries te bekijken, voer ik de queries uit in een databasebrowser. Het duurt 30 seconden om alle queries uit te voeren. Dit zou mogelijk de oorzaak van het probleem kunnen zijn: Als de queries 30 seconden duren, kan het genereren van het rapport niet korter duren dan die 30 seconden.

De database die gebruikt wordt voor dit project is MSSQL Server 2005. Een van de functies van MSSQL Server is het genereren van een Query Execution Plan. Daarmee kan je zien wat de database precies doet bij het uitvoeren van een query en hoe lang iedere stap duurt. Ook kan de database aangeven of het toevoegen van een index de queries zou verbeteren.

Voor een aantal van de queries kwam er een verassend resultaat uit: 98% van de tijd was de database bezig met een Index Scan (lezen vanaf de hardeschijf, op zoek naar alle gegevens). Het toevoegen van een index op een bepaalde set kolommen zou dit aanzienlijk verbeteren, gaf de database aan.

In een overleg met dhr. Van der Storm geef ik aan dat het toevoegen van een index zou kunnen helpen. Nadat dit goedgekeurd is door de databaseadmin, dhr. Leer, worden de voorgestelde indexen toegepast op de database. Dit heeft echter geen merkbare verbetering tot gevolg. Een verdere analyse van de nieuwe queryexecutieplannen liet zien dat de database wel gebruik maakte van de nieuwe indexen.

Tijdens verdere bestudering van de queries blijft het opmerkelijk dat de queries er veelal hetzelfde uitzagen. Veel van de queries bestonden uit een statistische functie over een complexe set JOINs op dezelfde tabel. Het uitvoeren van zo’n query duurde slechts 0.1 seconde, wat weinig ruimte liet voor

optimalisatie. Tijdens het bestuderen van de queries doet een mogelijke oplossing zich voor; Zijn de queries te combineren?[5]

Gedurende de blokken op school bestonden “huiswerk”-opdrachten voor databases vaak uit het “vertalen” van queries. Queries zijn in principe gewone taal – tenminste, het principe erachter. De queries worden gebruikt om een rapport over enquêteresultaten te genereren. Het is niet zo moeilijk om voor te stellen dat de queries iets als “Hoeveel mensen hebben op vraag 17 antwoord A gegeven?” waren.

Dus wat nou als de query “Hoeveel mensen hebben op vraag 17 antwoord A gegeven?” en “Hoeveel mensen hebben op vraag 17 antwoord B gegeven?” gecombineerd worden tot “Hoe vaak is er iedere antwoordcategorie van vraag 17 gekozen, waarbij de antwoordcategorie A of B is”?

Dan krijg je het resultaat van beide queries. Uit het resultaat blijkt verder dat de nieuwe query maar 0.12 seconden duurde om uit te voeren. Dat was iets langer dan 1 query, maar sneller dan de 2 queries samen! Aan de hand van dit resultaat ben ik gaan kijken of er nog andere queries waren die gecombineerd konden worden.

Door het combineren van de queries was een lijst van 268 queries teruggebracht tot ongeveer 70 queries. Deze lijst queries, die eerst nog 30 seconden had geduurd was nu in 14 seconden klaar met uitvoeren. Dit lijkt het definitieve bewijs dat dit de juiste oplossing was voor het performanceprobleem.

Voordat het implementeren van een oplossing kon beginnen, moest ik mijn bevindingen op een gecontroleerde en gestructureerde wijze documenteren. Een bepaalde denkwijze had tot deze oplossing geleid. Als deze denkwijze gedocumenteerd is, is het later mogelijk om een eventuele fout te traceren.

Om te voorkomen dat ik iets over het hoofd had gezien ben ik daarom een Probleemanalyse gaan maken.

[5] Twee van de queries zijn toegevoegd als bijlage. Zie tekstvak1.

4.2: Probleemanalyse

Om mijn denkwijze te documenteren en opgedane kennis te toetsen heb ik een probleemanalyse uitgevoerd.

De probleemanalyse tracht te verantwoorden waarom de gekozen oplossingsrichting de juiste is. Dit wordt gedaan door aan de hand van een set hypotheses de mogelijke oorzaken van het probleem te specificeren. Iedere hypothese bestaat uit een aanname, beschrijving en een mogelijke oplossing. Wanneer de aanname of de beschrijving van een hypothese niet overeen komt met de huidige situatie, is deze ontkracht. Op deze manier blijven er een of meerdere hypotheses over die binnen de huidige situatie passen. Vervolgens worden per hypothese de oplossingen op kleine schaal getest. Als de oplossing een duidelijk effect heeft, is de oplossingsrichting duidelijk.

Binnen de probleemanalyse zijn drie systeemonderdelen geïdentificeerd die mogelijk invloed hebben op het performanceprobleem. Dit zijn de database, de applicatieserver en het IBM Dimensions softwarepakket.

De probleemanalyse maakt gebruik van een hoofdaanname:
Alhoewel alle onderdelen een zekere mate van invloed hebben, oefent 1 component veel meer invloed uit op het eindresultaat dan alle andere componenten. Aanpassen van dat ene component lost het probleem op.

Hypotheses

Hypothese 1: De database is te traag. Dit is te zien doordat het uitvoeren van meerdere taken op de database tegelijk een enorme vertraging oplevert. De oplossing zou hierbij liggen in het kijken naar de inrichting van de database, zowel op configuratie- als op hardwareniveau.
Hypothese 2: De database is verkeerd geconfigureerd. De meeste handelingen die uitgevoerd worden op de database worden binnen een normale tijd uitgevoerd. Wanneer men echter een handeling onderneemt die niet gezien wordt als standaard, ondervindt de database problemen. Hiervoor zou naar de configuratie van de database gekeken moeten worden.
Hypothese 3: De database kan de hoeveelheid werk niet aan die vanuit het applicatieonderdeel binnenkomt. Dit is te zien doordat andere systemen die intensief gebruikmaken van de databaseserver geen last ondervinden van deze problemen. Daarnaast is dit zichtbaar doordat de database gedurende het genereren druk bezig is, terwijl de applicatie aan het wachten is.
Hypothese 4: De applicatieserver is te traag. Tijdens normaal gebruik van de applicaties op de applicatieserver zijn er al aanwijzingen van problemen. Wanneer men dan een complexe taak uitvoert, kan de applicatieserver de hoeveelheid werk niet goed verwerken. Om dit op te lossen zou er meer hardware ingezet moeten worden. Daarnaast zou er gekeken moeten worden of de applicatieserver wel de recentste versie is en of nieuwere versies het probleem zouden verhelpen.
Hypothese 5: De applicaties zijn niet goed ontwikkeld en zijn daardoor ruim trager dan zou moeten. Dit is te zien doordat taken die niet met andere componenten van het systeem communiceren ook dezelfde problemen ondervinden. Om dit te verhelpen zouden de applicaties aangepast moeten worden.
Hypothese 6: Het applicatieonderdeel wat met de databaseserver communiceert is inefficiënt. Dit is te zien doordat terwijl het genereren van het rapport bezig is, de database niet tot licht belast is, terwijl de applicatie druk bezig is. Dit is op te lossen door het applicatieonderdeel wat met de

databaseserver communiceert te vervangen of aan te passen.
Hypothese 7: De verwachtingen van de opdrachtgever zijn onrealistisch. Alles lijkt in orde te zijn en er is geen sprake van enige opvallende pieken binnen het gehele systeem. Dit betekent dat het gehele systeem aangepast zou moeten worden (opschalen), of dat de opdrachtgever genoegen moet nemen met het bestaan van het probleem.

Voor het systeemonderdeel IBM Dimensions zijn geen hypotheses opgesteld. Dit omdat ik na slechts twee dagen stage nog geen voldoende kennis heb vergaard over de werking van IBM Dimensions.

Uitgevoerde tests

Hypothese 1: De database is te traag. Hiervoor heb ik getest wat er gebeurt als de database meerdere taken tegelijk moet uitvoeren. Het laten genereren van meerdere rapportages (2-3) duurt niet merkbaar langer dan het laten genereren van een enkele rapportage. Dit geeft aan dat de database niet te traag is en dat het probleem elders ligt. Het uitvoeren en voltooien van meerdere taken tegelijk zou significant langer moeten duren dan het uitvoeren van een enkele taak, als de database verantwoordelijk is. Dit is echter niet het geval.

Hypothese 2: De database is verkeerd geconfigureerd. Hiervoor heb ik gekeken naar de Query Execution Plans die MSSQL Server biedt. De Query Execution Plans tonen een probleem aan. De databaseserver was namelijk voor het grootste gedeelte van de tijd bezig met het doorzoeken van een enkele index. Daarna was de rest van de query in een keer klaar.
Het toepassen van andere indexen zoals deze aangeraden worden door MSSQL Server zorgt echter niet voor een aanzienlijke verbetering. Wel is de workload beter verdeeld voor de query volgens de Execution Plans.
Deze hypothese stelt dat het aanpassen van de configuratie van de database de performance zou moeten verbeteren. Wanneer deze handeling wordt uitgevoerd is er echter geen merkbaar resultaat. Hierdoor lijkt deze hypothese niet van toepassing te zijn.

Hypothese 3: De database kan de hoeveelheid werk niet aan die vanuit het applicatieonderdeel binnenkomt. Hiervoor heb ik getest hoe lang de database erover doet om de SQL-queries uit te voeren die het applicatieonderdeel verstuurt bij het genereren van het rapport. Tijdens deze test was de applicatieserver niet actief. De database had maar liefst 30 seconden nodig om alle queries uit te voeren. Dit lijkt het probleem te zijn. Wanneer ik naar de afzonderlijke tijd kijk van iedere query, is deze best redelijk. Onder de 0,2 seconden voor een drie- of zelfs vierdubbele join. Er worden echter zoveel queries uitgevoerd dat de database heel lang bezig is. Deze hypothese lijkt dus een stap in de juiste richting van een mogelijke oplossing te zijn.

Hypothese 4: De applicatieserver is te traag. Dit lijkt niet het geval te zijn. Wanneer er namelijk een runtimefout aanwezig is in de code van de applicatie, wordt dit heel snel gerapporteerd. Het laten genereren van een complex rapport leidt bovendien tot een time-out op een databaseverbinding in plaats van een time-out op de uitvoering van de code. Bij een test voor een andere hypothese is geconstateerd dat het uitvoeren van alle queries, zonder de applicatieserver, 30 seconden duurt. Het genereren van het rapport duurt 30 tot 40 seconden. Dit betekent dat de overhead van de applicatieserver maximaal 10 seconden bedraagt. Hiermee is de applicatieserver niet de hoofdoorzaak van het performanceprobleem.

Hypothese 5: De applicaties zijn niet goed ontwikkeld en zijn daardoor ruim trager dan zou moeten. Hier heb ik nog geen tests uitgevoerd. Ik heb echter wel gezien dat het genereren van een overzicht van 5 enquêtevragen zo'n anderhalve seconde duurt. Per enquêtevraag zijn twee queries vereist. Elk van deze queries duurt 0.1 tot 0.2 seconden. Dit laat weinig tijd over voor de applicaties om traag te zijn. Hierdoor denk ik dat deze hypothese niet datgene beschrijft wat de hoofdoorzaak van het performanceprobleem is.

Hypothese 6: Het applicatieonderdeel wat met de databaseserver communiceert is inefficiënt. Ook dit is niet het geval. Bij het gebruik van logging in het applicatieonderdeel is de log veel eerder compleet dan het uiteindelijke resultaat. Zodra de log compleet is, is de applicatie in principe op de database aan het wachten. Zolang de applicatie ruimschoots klaar is voordat de database dat is, is de interne werking van het applicatieonderdeel niet de hoofdoorzaak van het performanceprobleem.

Hypothese 7: De verwachtingen van de opdrachtgever zijn onrealistisch. Ik heb al enkele problemen geconstateerd met het uitvoeren van queries op de database. Dit betekent dat er niet niets aan de hand is en dat er wel degelijk een performanceprobleem is.

Aan de hand van deze uitgevoerde tests ben ik verder gaan kijken naar de queries die naar de database gestuurd worden door het applicatieonderdeel. Veel van deze queries lijken op elkaar. Ik vind tussen de lijst queries enkele queries die wel heel erg op hetzelfde lijken en ik slaag erin om deze te combineren. Het uitvoeren van deze 12 queries op de oude wijze kost 1.2 seconden, terwijl mijn gecombineerde query klaar is in 0.19 seconden. Het valt me op dat er nog meer queries zijn die op hetzelfde lijken. Hierom lijkt het mij dat de hoeveelheid queries die naar de database gestuurd worden door het applicatieonderdeel de hoofdoorzaak van het performanceprobleem is.

Oplossingsrichting

Aan de hand van de uitgevoerde probleemanalyse blijkt dat de performanceproblemen niet te wijten zijn aan de database. Ook de applicatieserver is niet verantwoordelijk. Wat wel blijkt is dat de hoeveelheid werk die de database moet verrichten te veel is voor de database.

Dit betekent dat de oplossing slechts twee kanten op kan. De eerste optie bestaat uit het verminderen van de hoeveelheid werk die de database moet verrichten. De tweede optie bestaat uit het verbeteren van de hoeveelheid werk die de database tegelijkertijd kan verrichten, door bijvoorbeeld de databaseservers op te schalen.

Tijdens het testen van de hypotheses is de eerste optie op kleine schaal toegepast. Daarbij is een performanceverbetering van 1 seconde behaald. Eén seconde lijkt weinig, maar deze verbetering wordt behaald met combineren van 12 queries uit een lijst van 268 queries. Combineren lijkt daarmee de beste keuze als oplossing van het performanceprobleem.

Tijdens een overleg met dhr. Van der Storm worden beide opties en mijn advies gepresenteerd. Dhr. Van der Storm accepteert mijn advies voor het vinden van een oplossing die de hoeveelheid queries vermindert. Daarmee is de volgende stap het maken van een plan van aanpak om een oplossing te specificeren en te implementeren.

4.3: Plan van Aanpak

Het plan van aanpak helpt beide partijen in een project beter te begrijpen wat moet en kan gebeuren om het performanceprobleem op te lossen. Alhoewel de oplossingsrichting op dit moment duidelijk is voor mij als uitvoerende, is het voor mijn opdrachtgever nog niet duidelijk wat dit inhoudt. Daarnaast zijn er misschien aspecten van de opdracht die bij mij nog niet in zijn geheel duidelijk zijn. Het plan van aanpak helpt bij het op een lijn krijgen van beide partijen.

Om de huidige situatie duidelijk te maken is in het plan van aanpak een opdrachtomschrijving en opdrachtafbakening opgenomen. De afbakening bestaat uit het beperken van het project tot de oplossingsrichting die gekozen is aan de hand van de probleemanalyse.

Om de toekomstige situatie duidelijk te maken is er in het plan van aanpak een globale planning voor de eerste weken van het project opgenomen. Ook wordt er een lijst van producten beschreven die gedurende het project worden opgeleverd.

Aan de hand van het plan van aanpak zal eerst een verdere analyse gedaan worden van de huidige situatie. Dit wordt gedaan in een architectuuranalyse. De architectuuranalyse heeft daarnaast als doel om te lokaliseren waar een eventuele oplossing geplaatst kan worden binnen de huidige situatie. Wanneer de plaats van de oplossing bekend is, moet er een requirementsanalyse uitgevoerd worden om enige extra eisen aan de oplossing boven water te halen. Vervolgens kan er een functioneel ontwerp gemaakt worden, waarin de werking van de oplossing wordt beschreven. Daarna wordt er middels een technische analyse gekeken hoe de oplossing geïmplementeerd kan worden en welke problemen er kunnen voorkomen. Uiteindelijk kan dan in het technisch ontwerp de implementatie van de oplossing ontworpen worden, waarna deze wordt geïmplementeerd.

In de probleemanalyse is een aanname gemaakt dat 1 onderdeel van het gehele systeem het grootste effect heeft op het performanceprobleem. Het project zal volgens deze aanname werken. Dat betekent dat alhoewel het mogelijk is om performanceverbeteringen in andere systeemonderdelen te behalen zijn, dit niet relevant is voor dit project. Dit project limiteert zichzelf tot het verminderen van het grootste effect op het performanceprobleem.

Al deze vervolgstappen van het plan van aanpak, namelijk de architectuuranalyse, requirementsanalyse, functioneel ontwerp, technische analyse en technisch ontwerp worden in de volgende hoofdstukken uitgewerkt.

Het eerstvolgende is het maken van de architectuuranalyse.

4.4: Architectuuranalyse

De architectuuranalyse heeft als doel inzicht te geven in de architectuur van de huidige situatie. Daarnaast wordt er binnen de architectuuranalyse gekeken hoe het ondervonden performanceprobleem verholpen zou kunnen worden. Dit houdt in dat er wordt gekeken naar welke systemen er momenteel worden gebruikt binnen de huidige situatie. Vervolgens wordt per systeem gekeken wat de impact is op het performanceprobleem en hoe dit aangepast zou kunnen worden. Aan de hand van deze analyse kan een oplossing worden geformuleerd die een of meerdere systemen aanpast om het performanceprobleem te verhelpen.

Voor het in kaart brengen van de huidige situatie is een deploymentdiagram gemaakt (zie Figuur 2).

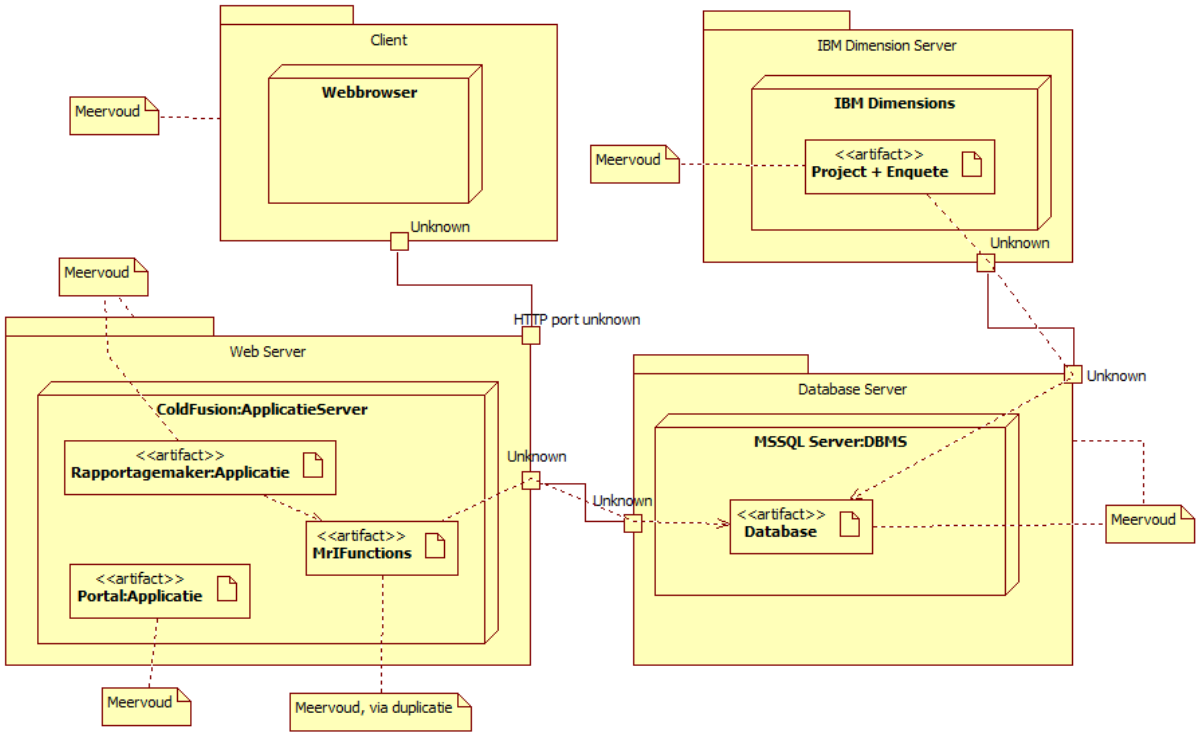


Fig2. Deploymentdiagram huidige situatie.

In de probleemanalyse is geconcludeerd dat de oorzaak van het performanceprobleem lag bij de hoeveelheid queries die de database moet verwerken bij het genereren van een rapport. Dit beperkt de mogelijke onderdelen van aanpassing tot de rapportageapplicatie, de database en de laag daar tussen (MrIFunctions). Zie Figuur 3 voor een uitvergroting van dit gedeelte van het deploymentdiagram.

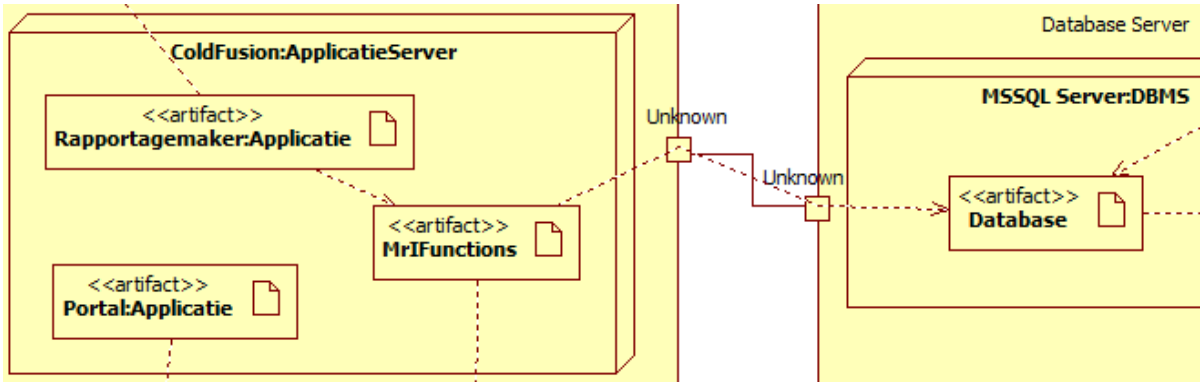


Fig3. Een uitvergroting van het deploymentdiagram.

Het aanpassen van de database leidt niet tot merkbare resultaten, zoals eerder geconstateerd in de probleemanalyse. Hierdoor verschuift de focus naar het MrIFunctions applicatieonderdeel. De drie geformuleerde oplossingen voor het performanceprobleem komen in principe op het volgende neer:

- Samenvoegen van de queries voordat ze omgezet worden in SQL.
- Samenvoegen van de queries tijdens het omzetten naar SQL.
- Samenvoegen van de queries na het omzetten in SQL.

Samenvoegen van de queries voordat ze omgezet worden in SQL vereist dat de rapportageapplicaties aangepast worden. Dit is tijdrovend werk en de oplossing is niet gemakkelijk over te brengen naar een volgende applicatie.

Samenvoegen van de queries nadat ze omgezet zijn in SQL vereist tekstanalyse van de SQL-queries. Dit is technisch zeer complex maar mogelijk en niet gemakkelijk uit te breiden voor het toevoegen van andere functionaliteit.

Samenvoegen van de queries tijdens het omzetten naar SQL is daarmee de makkelijkste oplossing. Een oplossing op die plaats binnen het systeem heeft de mogelijkheid om zowel de SQL-queries als het resultaat aan te passen. Dit is nodig, omdat het aanpassen van de SQL-queries tot een iets ander resultaat leidt (namelijk een lijst van resultaten in plaats van per SQL-query een enkel resultaat).

Dit leidt tot de volgende conclusie:
Binnen de huidige architectuur is het meeste resultaat te bereiken door het applicatieonderdeel MrIFunctions aan te passen. De oplossing bestaat uit het aanpassen of vervangen van het MrIFunctions applicatieonderdeel zodat deze queries voor de database combineert en het resultaat weer uitsplitst voor de rapportageapplicaties. Wel moet er rekening gehouden worden met de mogelijkheid dat de rapportageapplicaties aangepast moeten worden om te kunnen werken met de aangepaste MrIFunctions.

Op basis van deze conclusie is in overleg met dhr. Van der Storm besloten om te gaan voor een vervanging van MrIFunctions. Om de benodigde aanpassingen binnen de rapportageapplicaties te verminderen zal het huidige applicatieonderdeel MrIFunctions worden aangepast. Na deze aanpassing dient het applicatieonderdeel MrIFunctions als koppeling tussen de rapportageapplicaties en als de vervanging van MrIFunctions.

Om alle andere eisen en wensen betreffende het project in kaart te brengen wordt er een requirementsanalyse uitgevoerd. Met de requirementsanalyse worden alle belangen van de stakeholders[6] vastgelegd.

[6] Zie termenlijst

4.5: Requirementsanalyse

Bij de start van dit project werd het volgende gesteld: “Het genereren van de rapportages moet sneller, zonder dat de ontwikkelaars daar veel extra inspanning voor moeten leveren.” In de requirementsanalyse worden eisen en wensen als deze geïdentificeerd, gedocumenteerd en uiteengezet.

Om de eisen en wensen te identificeren zijn er interviews met ontwikkelaars en gebruikers van het huidige systeem. Uit deze interviews blijkt dat de ontwikkelaars zich voornamelijk zorgen maken over de performance van het huidige systeem. De ontwikkelaars hebben daarnaast voorkeur voor behoud van oude systemen en bestaande werkwijze, waar mogelijk. Zij zijn echter bereid nieuwe systemen en een nieuwe werkwijze te accepteren, als dit het performanceprobleem verhelpt.

De gebruikers leveren een ander perspectief. Zij maken zich zorgen over de flexibiliteit van het systeem. Klanten willen steeds meer controle over de vorm van de gegenereerde rapportages. Wanneer de gebruikers dit aandragen bij de ontwikkelaars ondervinden zij weerstand; het kost tijd om nieuwe functionaliteit te implementeren. Een nieuw systeem zou een extra afhankelijkheid kunnen opleveren, waardoor het voor de gebruikers nog lastig wordt om nieuwe functionaliteit aan te vragen bij de ontwikkelaars.

Er zijn dus drie hoofdeisen geïdentificeerd:

1. De performance van het systeem moet verbeterd worden.
2. De oude systemen en werkwijzen moeten zoveel mogelijk intact blijven.
3. Er moet rekening gehouden worden met latere toevoeging van functionaliteit aan het systeem.

De eisen en wensen die geïdentificeerd zijn gedurende de requirementsanalyse dienen als input voor het functioneel ontwerp. In het functioneel ontwerp wordt beschreven 'wat' de oplossing gaat doen om het performanceprobleem te verhelpen. Tevens wordt beschreven hoe de oplossing binnen de huidige architectuur wordt geplaatst.

4.6: Functioneel Ontwerp

Het functioneel ontwerp wordt opgesteld aan de hand van de architectuuranalyse en de requirementsanalyse. Het functioneel ontwerp dient antwoord te geven op de vraag “Wat gaat de oplossing doen?”. Hierbij wordt de gekozen oplossingsrichting vanuit de architectuuranalyse gehanteerd. Ook wordt er rekening gehouden met de geïdentificeerde eisen uit de requirementsanalyse.

In de architectuuranalyse is geconcludeerd dat het vervangen en aanpassen van MrIFunctions de beste manier is om de oplossing te implementeren. Deze oplossing combineert de queries tijdens het genereren daarvan. De oplossing krijgt de naam Batcher. “Batch” is het Engelse woord voor een aantal dingen die tegelijk worden uitgevoerd of geproduceerd. “Batcher” combineert dus de queries een geeft die door aan de database. De Batcher genereert en combineert de queries aan de hand van de gegevens die een rapportageapplicatie aanlevert via MrIFunctions. In principe is de Batcher dus een “oplosser”, in plaats van een oplossing.

Om de positie van de Batcher in het gehele systeem vast te stellen is een deploymentdiagram gemaakt van de nieuwe situatie (Zie Figuur 4). De aangepaste onderdelen zijn met oranje gemarkeerd.

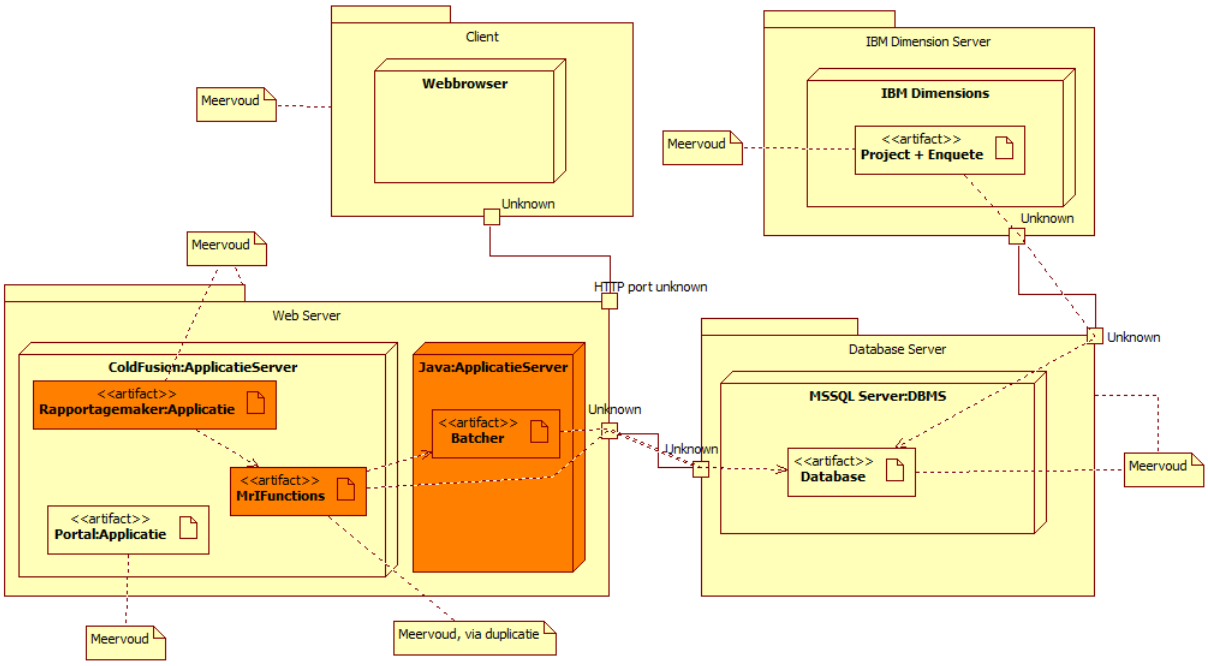


Fig4. Deploymentdiagram nieuwe situatie

De Batcher is toegevoegd tussen MrIFunctions en de Database. MrIFunctions wordt aangepast om de data-aanvragen van de rapportageapplicaties door te geven. Wel blijft het mogelijk om op de oude manier via MrIFunctions de database te benaderen. Hierdoor is de nieuwe versie van MrIFunctions compatibel met rapportageapplicaties die van het oude systeem gebruikmaken.

De rapportageapplicaties die “ernstige performanceproblemen” ondervinden worden aangepast, om met de Batcher om te kunnen gaan. Deze aanpassing bestaat uit het opsplitsen van de

programmaonderdelen waarin er om data voor rapportages gevraagd wordt en de programmaonderdelen waar de daadwerkelijke rapportage wordt gegenereerd. Door middel van deze opsplitsing is het mogelijk om andere methoden en technieken (zoals batching) toe te passen bij het ophalen van de data. Deze aanpassing is noodzakelijk, omdat de rapportageapplicaties momenteel direct per query gegevens terug willen hebben, wat combineren van de queries onmogelijk maakt.

Ter beschrijving van de interne werking en als input voor de technische analyse is een activity diagram gemaakt van de Batcher. Hierin zijn lagen gemarkeerd die onderdelen van de Batcher representeren. (Zie Figuur 5)

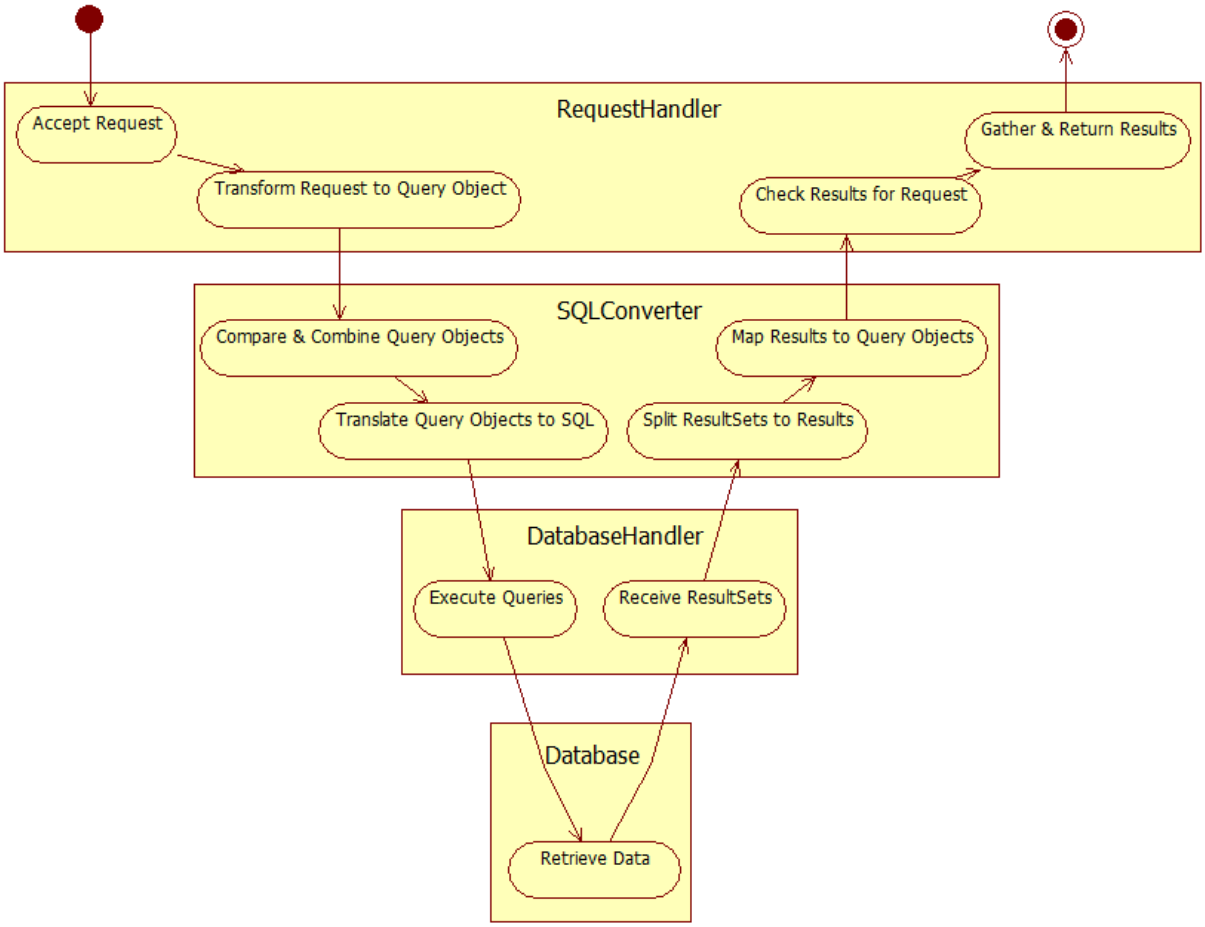


Fig5. Activity Diagram interne werking Batcher.

De RequestHandler is verantwoordelijk voor het accepteren van requests voor data uit de database. De RequestHandler vertaalt deze requests naar Query-objecten voor de SQLConverter en houdt in de gaten of een request vervuld is.

De SQLConverter is verantwoordelijk voor het combineren (batchen) van Query-objecten en het vertalen van de gecombineerde Query-objecten naar SQL. Vertaalde queries kunnen uitgevoerd worden op de database en daarmee kan de data opgehaald worden. De SQLConverter is daarnaast verantwoordelijk voor het uitsplitsen van de opgehaalde data en het vervolgens koppelen van een set resultaten aan het Query-object wat om die resultaten vroeg.

De DatabaseHandler is verantwoordelijk voor het beheren van de databaseconnectie. Dit houdt in dat de DatabaseHandler omgaat met het maken van de verbinding en foutafhandeling voor deze verbinding.

In het diagram wordt de Database gerepresenteerd door de databaselaag. De database is geen onderdeel van de Batcher, maar de toevoeging van de databaselaag helpt om het diagram te begrijpen. De database ontvangt queries, voert deze uit en stuurt resultsets terug.

Om aan de requirements te voldoen is bij een aantal onderdelen rekening gehouden met toekomstige aanpassingen. Voor het implementeren van nieuwe functionaliteiten is het mogelijk om nieuwe conversiealgoritmen en batchingsalgoritmen toe te voegen.

Tijdens een overleg met dhr. Van der Storm is dit functioneel ontwerp besproken en is er een alternatief gepresenteerd. Zo is het mogelijk om iedere rapportageapplicatie los aan te passen, wat het probleem ook zal verhelpen. In de requirementsanalyse is echter geconstateerd dat er voorkeur is voor het behoud van oude systemen en werkwijze, waar mogelijk. Het aanpassen omvat een drastische verandering in beide.

In het huidige systeem ligt de verantwoordelijkheid voor het opbouwen van de queries in het applicatieonderdeel MrIFunctions. Dit is een taak waar de ontwikkelaars zich momenteel niet mee bezig dienen te houden. Om een rapportageapplicatie in staat te stellen om complexere queries te stellen, moet de rapportageapplicatie zelf deze verantwoordelijkheid dragen. Dit geeft de ontwikkelaars een extra taak om uit te voeren. Hiermee wordt dus het ontwikkelproces trager.

Daarnaast is het zo dat het opbouwen van een query voor de database die momenteel gebruikt wordt een complexe taak is. Bij een hogere complexiteit van een taak wordt de kans op fouten bij de uitvoering van die taak groter. Opsporing van enige fouten wordt verder bemoeilijkt door de grote datasets waar de rapportageapplicaties mee werken.

Het aanpassen van de rapportageapplicaties wordt hierdoor gezien als een mogelijke oplossing, maar niet de juiste oplossing.

Ter controle van het activity diagram wordt in de technische analyse een proof of concept gemaakt. Daarmee wordt aangetoond of de gekozen oplossing werkt en met welke problemen een implementatie rekening moet houden.

4.7: Technische Analyse

Het doel van de technische analyse bestaat uit twee delen: als eerste deel is het doel om kennis te vergaren over de mogelijke vorm van de implementatie van de Batcher. Als tweede deel is het doel om enige problemen die bij de implementatie voor kunnen komen te identificeren, zodat hier in het technisch ontwerp rekening mee kan worden gehouden.

Om dit tweedelige doel te behalen wordt er gebruik gemaakt van een proof of concept. Dit proof of concept bestaat uit een versimpelde implementatie van de architectuur zoals deze in het functioneel ontwerp is beschreven.

Voor het behalen van het eerste deel van het doel, dient het proof of concept antwoord te geven op de volgende vragen:

- Hoe kan er verbinding gemaakt worden met de database?
- Hoe kan een Java-applicatie een ResultSet omvormen tot een datastructuur binnen ColdFusion?
- Hoe zou het algoritme van query-batching eruit zien?
- Is het mogelijk om met proxy-objecten te werken?

In onderstaande tekst zal per vraag worden uitgelegd met behulp van een korte toelichting waarom deze gesteld moet worden. Daarnaast wordt uitgewerkt hoe het proof of concept deze vragen kan beantwoorden en wat het uiteindelijke antwoord op deze vragen is.

Verbinding met de database

ColdFusion regelt databaseverbindingen via serverconfiguratie. Dit is in Java niet mogelijk, waardoor er naar een alternatief gekeken moet worden. Het proof of concept zal met de databaseverbinding moeten maken om de queries uit te kunnen voeren.

Met behulp van JDBC-drivers van Microsoft is het mogelijk om verbinding met de database te maken. Het is echter niet mogelijk om een JDBC-driver voor MSSQL Server te vinden die Windows-authenticatie ondersteunt, dus er zal gebruik gemaakt moeten worden van “normale” authenticatie voor de database.

Binnen Java een ResultSet omvormen tot een ColdFusion-datastructuur

Alhoewel ColdFusion gebaseerd is op Java maakt ColdFusion gebruik van andere typen datastructuren. Hierdoor moet er een conversie tussen datatypes gebeuren wanneer er gecommuniceerd wordt tussen ColdFusion en Java. In het proof of concept moeten, nadat de queries zijn uitgevoerd op de database, de resultaten worden teruggestuurd naar het ColdFusion-gedeelte.

Het blijkt dat het converteren van Java-objecten naar ColdFusion-objecten veel problemen oplevert. Java heeft echter geen problemen met ColdFusion-objecten. Als er vanuit ColdFusion een datastructuur naar Java gestuurd wordt, kan Java deze zonder enige conversie gebruiken. Wanneer deze datastructuur later weer naar ColdFusion terug gestuurd wordt, is er geen conversie nodig.

Algoritme query-batching

De Batcher werkt door “queries met elkaar te vergelijken en deze dan te combineren”. De manier waarop dit gebeurt is echter nog niet eerder exact beschreven, waardoor het nog onzeker is of het ook echt daadwerkelijk mogelijk is.

Theoretisch gezien zijn er twee vormen van query-batching:
Semantisch batchen en syntactisch batchen. Semantisch batchen is het combineren van queries aan de hand van de semantische betekenis. Syntactisch batching is het combineren van queries aan de hand van de querystructuur.

Om verschillende queries met elkaar te kunnen vergelijken worden deze queries eerst geabstraheerd tot querystructuren. Dit brengt het aantal mogelijke queries terug tot een overzichtelijk aantal. Met behulp van waarheidstabellen kunnen vervolgens regels vastgesteld worden. Deze regels kunnen vervolgens gebruikt worden om te bepalen of het mogelijk is om twee querystructuren te combineren.

Omdat dit onderwerp een zeer belangrijk onderdeel is van het project, wordt dit onderwerp verder uitgewerkt in een apart hoofdstuk. Zie hiervoor het hoofdstuk “Batchingsalgoritmes”.

Proxy-objecten

De eigenlijke vraag is “hoe kan men binnen ColdFusion a-synchroon resultaat ontvangen van queries”? Een van deze manieren is via proxy-objecten. Proxy-objecten dienen als “verpakking” om het daadwerkelijke resultaat. Wanneer de applicatie gegevens nodig heeft vraagt hij het proxy-object om deze gegevens. Als deze gegevens beschikbaar zijn worden deze direct geretourneerd, zo niet, dan moet er door de applicatie gewacht worden tot dat de gegevens beschikbaar zijn.

Uit het proof of concept is gebleken dat het mogelijk is om proxy-objecten te gebruiken. Wel betekent dit dat mogelijk eerst het resultaat uit de “verpakking” gehaald moet worden voordat hier mee gewerkt kan worden – er is dus een extra stap vereist binnen de rapportageapplicaties.

Hiermee is het eerste deel van het doel van de technische analyse behaald. Er is vastgesteld dat het technisch mogelijk is om de Batcher als oplossing voor het performanceprobleem te implementeren. Het tweede deel van het doel van de technische analyse is het identificeren van technische problemen die zich bij een implementatie kunnen voordoen.

Tijdens het maken van het proof of concept is er een enkel technisch probleem geïdentificeerd. ColdFusion is op Java gebaseerd. De Java-versie (JRE 1.6.0_29) die gebruikt wordt op de ColdFusion-productieservers bevat een bug waardoor verbinding maken met de database via JDBC ervoor zorgt dat de applicatie blijft hangen.

Om dit probleem te verhelpen is het mogelijk om de Java-versie te vervangen met een nieuwere versie. Dit zou echter mogelijk gevolgen hebben voor andere applicaties die op de applicatieservers draaien. Na een kort experiment blijkt dat het probleem ook te verhelpen valt door de Java-library die verbindingen beheert te vervangen met een nieuwere versie. Met deze oplossing blijft het huidige systeem zo veel mogelijk behouden en zijn er minder gevolgen voor andere applicaties.

Advies

Op basis van deze technische analyse heb ik een advies gedaan aan dhr. Van der Storm. Dit advies omvat het gebruik van zowel syntactisch als semantisch batchen. Daarnaast heb ik geadviseerd het verbindingsprobleem te verhelpen door middel van het vervangen van de Java-library voor verbindingen.

In overleg met dhr. Van der Storm is dit advies geaccepteerd. De exacte implementatie van de Batcher wordt verder besproken in het technisch ontwerp.

4.8: Technisch Ontwerp

Het technisch ontwerp beschrijft de structuur van de code van de Batcher. Door een technisch ontwerp te maken wordt een ontwikkelaar geforceerd na te denken over de structuur van de implementatie. Op deze manier voorkomt men dubbel werk.

In dit hoofdstuk wordt de eerste opzet van het technisch ontwerp beschreven. Gedurende het maken van het technisch ontwerp en het implementeren daarvan komen problemen aan het licht. Deze problemen en de oplossingen hiervoor worden in latere hoofdstukken uitgewerkt.

De eerste opzet van het technisch ontwerp is gebaseerd op de structuur die in het proof of concept is gebruikt. In het onderstaande klassendiagram staat deze structuur gemodelleerd (zie Figuur 6). Dit klassendiagram dient als analyseklassediagram. Daarom zijn de attributen en functies van de klassen weggelaten tenzij deze nodig lijken.

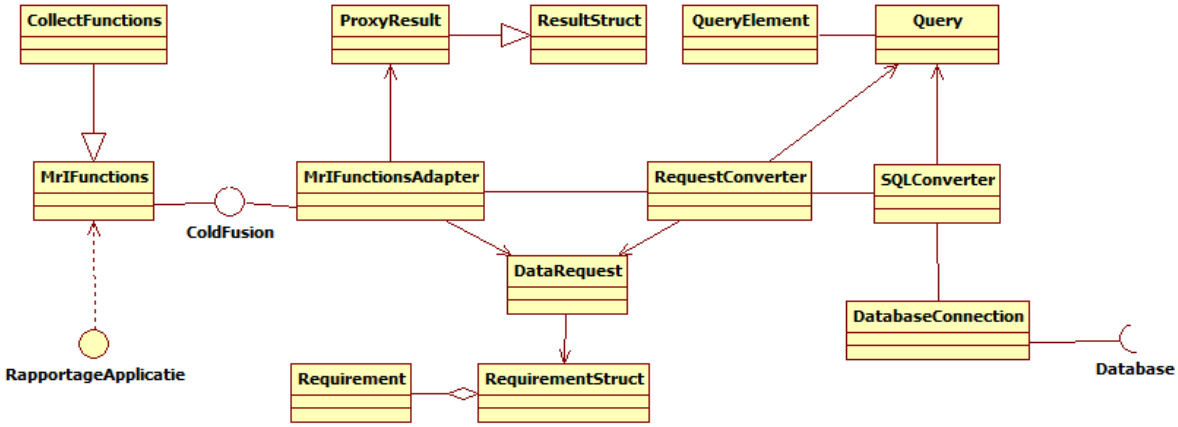


Fig6. Eerste opzet Technisch Ontwerp.

Het ColdFusion-element van het diagram representeert de scheiding tussen het ColdFusion-gedeelte van het diagram en het Java-gedeelte van het diagram. Hierbij representeert het linkergedeelte van het diagram (RapportageApplicatie, MrIFunctions en CollectFunctions) het ColdFusion-gedeelte van de Batcher.

Beschrijving onderlinge werking

Deze paragraaf beschrijft de onderlinge werking van de klassen van het technisch ontwerp.

Vanuit de rapportageapplicatie (gerepresenteerd als een externe interface in het diagram) komen requests voor gegevens binnen. Deze worden in de klassen MrIFunctions en CollectFunctions opgevangen. Vervolgens worden deze requests doorgestuurd naar de MrIFunctionsAdapter.

Om op semantisch niveau te batchen worden de requests aan de huidige batch toegevoegd. Dit wordt door de MrIFunctionsAdapter gedaan. De MrIFunctionsAdapter maakt een nieuw ProxyResult aan en koppelt deze aan het binnenkomende request. Vervolgens wordt het request als DataRequest toegevoegd aan de huidige batch. De MrIFunctionsAdapter stuurt via MrIFunctions het ProxyResult terug aan de rapportageapplicatie.

Wanneer de batch uitgevoerd moet worden begint het batchproces. De requests worden door de RequestConverter geconverteerd in Query-objecten. Tijdens het converteren worden de queries door de RequestConverter op semantisch niveau gecombineerd.

Om de Query-objecten als queries te kunnen uitvoeren moeten ze nog tot SQL geconverteerd worden. Dit wordt gedaan door de SQLConverter. Tijdens het converteren worden de Query-objecten met elkaar op een syntactisch niveau vergeleken en waar mogelijk gecombineerd.

Vervolgens worden de queries uitgevoerd op de database via de DatabaseConnection. De DatabaseConnection ontvangt het resultaat terug in de vorm van ResultSets.

Van de gecombineerde queries worden de ResultSets opgesplitst en aan de originele Query-objecten gekoppeld. Dit wordt gedaan door de SQLConverter, aan de hand van de informatie die is opgeslagen tijdens het batchproces.

Nadat de ResultSets gekoppeld zijn aan de queries, moeten de queries op de DataRequests gekoppeld worden. Dit wordt gedaan door de RequestConverter, aan de hand van de informatie die opgeslagen is tijdens het converteren.

Wanneer alle resultaten vervolgens bekend zijn, worden de resultaten in de ResultStructs geplaatst. Daarna wordt het batchproces beëindigd.

Binnen de structuur die deze onderlinge werking beschrijft is echter niet vermeld op welke wijze de informatie van het combineren wordt opgeslagen. Ook is niet duidelijk hoe querystructuren kunnen worden vergeleken, of wat een QueryElement precies betekent.

In het volgende hoofdstuk wordt beschreven hoe de uitbreiding van het Query-object samen met enkele andere aanpassingen deze problemen oplost.

4.9: Querystructuur

In het technisch ontwerp is niet duidelijk genoeg beschreven hoe querystructuren met elkaar vergeleken worden. Door middel van het specificeren van de opbouw van de querystructuur wordt duidelijk hoe querystructuren gebruikt kunnen worden in de Batcher.

Aanpassingen in het Technisch Ontwerp

Het aangepaste technisch ontwerp is te groot om zo in het document te plaatsen. Zie hiervoor Figuur 7 in de bijlagen.

Binnen de Batcher is een querystructuur in principe “SQL in objecten”. Queries in de vorm van een querystructuur worden gepresenteerd door StructuredQuery. Queries in de vorm van SQL worden gepresenteerd door SQLQuery.

Een StructuredQuery bestaat uit een aantal QueryParts. Ieder QueryPart (QueryOnderdeel op z’n Nederlands) representeert een SQL-statement binnen een query. Dit zijn SELECT, FROM, WHERE, ORDER BY, GROUP BY en WITH. Ieder van deze SQL-statements wordt gerepresenteerd door een QueryPart.

Naast statements bevat SQL verwijzingen naar andere objecten. Objecten zoals tabellen, kolommen, statistische functies en simpele getallen. Deze worden gerepresenteerd door objecten als ColumnReferences, TableReferences, AggregateFunctions en LiteralValues.

Om duidelijk te maken welke functionaliteit ieder van deze objecten representeert, worden de objecten aan interfaces gekoppeld. Deze interfaces (Selectable, Comparable, Groupable, Sortable, RowSource) geven aan waar een object geplaatst kan worden binnen een querystructuur. Ook specificeren deze interfaces een soort van “codecontract” waar de objecten aan moeten voldoen. Zo moeten alle objecten met de interface Selectable kunnen aangeven welke kolomnaam zij krijgen als de query wordt uitgevoerd. Daarnaast moet een object per interface in staat zijn om een stuk SQL te genereren wat dat object representeert.

SQL SELECT Queries volgen een bepaald patroon. Dit patroon (hieronder weergegeven), wordt door de nieuwe querystructuurklassen nagebootst.

```
<SELECT statement> ::=
    [WITH <common_table_expression> [,...n]]
    <query_expression>
    [ ORDER BY { order_by_expression | column_position [ ASC | DESC ] }
  [ ,...n ] ]
    [ <FOR Clause>]
    [ OPTION ( <query_hint> [ ,...n ] ) ]
<query_expression> ::=
    { <query_specification> | ( <query_expression> ) }
    [  { UNION [ ALL ] | EXCEPT | INTERSECT }
        <query_specification> | ( <query_expression> ) [...n ] ]
<query_specification> ::=
SELECT [ ALL | DISTINCT ]
    [TOP ( expression ) [PERCENT] [ WITH TIES ] ]
    < select_list >
    [ INTO new_table ]
    [ FROM { <table_source> } [ ,...n ] ]
    [ WHERE <search_condition> ]
    [ <GROUP BY> ]
    [ HAVING < search_condition > ]
```

Met deze aanpassingen kan de SQLConverter een querystructuur omzetten naar SQL. Dit gaat volgens een bepaald patroon, wat gebaseerd is op de opbouw van een SQL SELECT Query:

```
[WITH WithQueryPart.identifier (WithQueryPart.columnlist) AS]
SELECT {SelectQueryPart.selectables}
FROM FromQueryPart.rowSources[0]
[{JOIN FromQueryPart.rowSources[1+]}]
[WHERE {WhereQueryPart.conditions}]
[GROUP BY {GroupByQueryPart.groupables}]
[HAVING {GroupByQueryPart.conditions}]
[ORDER BY {OrderByQueryPart.sortables}]
```

Een invulling van dit patroon leidt tot een query.

```
SELECT COUNT(dig.diersoort) as "Aantal gebieden", dig.diersoort as
"Diersoort"
FROM diersoort_in_gebied dig
WHERE dig.type IN ("Bos","Gebergte","Rivier","Woestijn","Meer")
GROUP BY dig.diersoort
ORDER BY 1
```

In deze query is de ColumnReference “diersoort” gemarkeerd met onderlijning, de TableReference “diersoort_in_gebied” vetgedrukt en de AggregateFunction “COUNT” met cursief gemarkeerd. Hiermee wordt duidelijk dat een enkel object op meerdere plaatsen in een querystructuur kan bestaan. Vergelijken van querystructuren kan worden gedaan door een abstracte structuur op te

stellen van een querystructuur. De Batcher vergelijkt deze abstracte structuur met andere querystructuren op de volgende manier:

```
SELECT A(C.B), C.B
FROM C
WHERE C.D IN (Collection1)
GROUP BY C.B
ORDER BY A
```

Een Condition is een speciaal type object. In tegenstelling tot de andere onderdelen van de querystructuur representeert een Condition een stuk semantiek. Een Condition representeert filteringscriteria die toegepast moeten worden op het resultaat van een query. Alle types van een Condition-object (NullCondition, LikeCondition, etc.) representeren het vergelijken van een object met iets anders. Het type van het Condition-object bepaalt welke vergelijking gerepresenteerd wordt. Objecten die in een Condition-object gebruikt kunnen worden voor vergelijking, zijn gekoppeld aan de interface Comparable.

Aanpassingen in de verantwoordelijkheid van SQLConverter

De toevoeging van de querystructuur heeft enkele gevolgen voor het procesverloop van de Batcher. Zo is de SQLConverter niet meer volledig verantwoordelijk voor het converteren van queries naar SQL. Deze verantwoordelijkheid wordt nu gedeeld door de SQLConverter en de objecten waar de querystructuur uit bestaat.

De toevoeging van een uitgebreide querystructuur geeft beter inzicht in hoe het combineren van queries mogelijk zou kunnen zijn. In het huidige ontwerp is echter nog geen ruimte opgenomen om verantwoordelijkheid te nemen voor het combineren van queries.

In het volgende hoofdstuk wordt beschreven hoe verantwoordelijkheden gesplitst worden in het technisch ontwerp.

4.10: Splitsen verantwoordelijkheden naar nieuwe klassen

In de eerdere hoofdstukken is slechts op abstracte wijze beschreven wanneer het batchen plaatsvindt. In dit hoofdstuk wordt beschreven hoe de verantwoordelijkheid voor het combineren toegewezen kan worden aan een eigen klasse. Dit wordt gedaan door het aanpassen van het technisch ontwerp. Door deze aanpassing kan het batchproces op een gestructureerde wijze verlopen.

Aanpassing Technisch Ontwerp

In eerdere versies van het technisch ontwerp lag de verantwoordelijkheid van het batchen bij de RequestConverter en SQLConverter. Dit betekent dat deze klassen een dubbele verantwoordelijkheid hebben. Opsplitsen van deze verantwoordelijkheid verbetert de structuur van de code en maakt het makkelijker om later de Batcher uit te breiden.

Het aangepaste technisch ontwerp is te groot om zo in het document te plaatsen. Zie hiervoor Figuur 8 in de bijlagen.

Door het splitsen van MrIFunctionsAdapter in MrIFunctionsAdapter en Batch is de dubbele verantwoordelijkheid van de Adapter verdwenen. De Batch draagt nu de verantwoordelijkheid voor het batchproces. Daarmee is de MrIFunctionsAdapter de Adapter voor MrIFunctions om te communiceren met de Batch.

Ook verdwijnt de dubbele verantwoordelijkheid van de RequestConverter en SQLConverter door opsplitsing. De SemanticBatcher en SyntacticBatcher nemen de verantwoordelijkheid voor het batchen over. Beide batchers maken gebruik van “BatchingStrategies”, ofwel batchingsalgoritmen in objecten. Hierdoor is het later simpel om een batchingalgoritme toe te voegen of te verwijderen. Dit is gedaan vanwege de requirement “Er moet rekening gehouden worden met latere toevoeging van functionaliteit aan het systeem”.

De SemanticBatcher is verantwoordelijk voor het batchen op een semantisch niveau. Dit betekent dat bij deze vorm van batching de betekenis van de queries bekend is. Dit versimpelt het combineren van queries enorm, omdat er geen rekening gehouden hoeft te worden met mogelijke andere querystructuren. Een nadeel van semantisch batchen is dat per situatie een nieuw algoritme ontwikkeld moet worden; Een algoritme voor semantisch batchen is slechts toepasbaar bij een bepaalde semantiek (zoals het verkrijgen van het aantal diersoorten in een bepaald type gebied).

De SyntacticBatcher is verantwoordelijk voor het batchen op een syntactisch niveau. Dit betekent dat bij deze vorm van batching de structuur van de queries bekend is. Bij syntactisch batchen wordt de structuur van een aantal queries vergeleken. Aan de hand van deze structuur worden de queries gecombineerd. Hiermee is de toepasbaarheid van een syntactisch batchingsalgoritme zeer groot. Een semantisch batchingsalgoritme is gelimiteerd tot het combineren van “diersoorten in gebied A” en “diersoorten in gebied B” tot “diersoorten in gebieden, waarbij gebied A of B”. Een syntactisch batchingsalgoritme kan ook worden toegepast op “product in aanbieding”, “werknemers in project”, “studenten in klas” en dergelijke.

Een nadeel van syntactische batchingsalgoritmen is, dat het schrijven van een syntactisch batchingalgoritme technisch complex is. Daarnaast is de performance van het algoritme minder goed

dan een semantisch batchingsalgoritme. In het hoofdstuk “Batchingsalgoritmes” wordt in meer detail besproken hoe deze algoritmes geïmplementeerd zijn in de Batcher.

De andere aanpassingen in het diagram worden aan de hand van de veranderingen in het procesverloop uitgelegd.

Beschrijving aangepaste onderlinge werking

Door de opsplitsing van de dubbele verantwoordelijkheden is het procesverloop van de Batcher duidelijker geworden.

De MrIFunctionsAdapter voegt DataRequests toe aan een batch. Wanneer alle DataRequests zijn toegevoegd kan het batchproces beginnen.

Als eerste stap van het batchproces worden de requests door de SemanticBatcher gehaald. Wanneer het DataRequestType van een DataRequest overeen komt met het DataRequestType van een SemanticBatchingStrategy, wordt de DataRequest aan de SemanticBatchingStrategy toegevoegd. De overige lijst DataRequests gaat naar de RequestConverter toe.

Vervolgens worden in de SemanticBatchingStrategies de DataRequests gecombineerd op semantisch niveau. Dit wordt gedaan door middel van het aanmaken van een nieuw “BatchedDataRequest”-object, met een verwijzing naar de DataRequests waar deze BatchedDataRequest uit bestaat. Dit is nodig zodat het resultaat later op de juiste manier teruggekoppeld kan worden naar de originele DataRequests. De nieuwe BatchedDataRequests worden vervolgens ook naar de RequestConverter doorgestuurd.

Nadat de RequestConverter klaar is met het converteren van de DataRequests in querystructuren (StructuredQueries), kan het syntactische batchen beginnen. Er wordt een lijst van alle querystructuren gemaakt. Deze lijst wordt vervolgens gegeven aan de SyntacticBatcher. Deze geeft de lijst in seriële volgorde door aan iedere SyntacticBatchingStrategy.

Bij het combineren van StructuredQueries wordt een nieuwe StructuredQuery aangemaakt. Aan de nieuwe StructuredQuery wordt een QueryBatch-object toegevoegd. In dit object is een koppeling tussen Selectables van de nieuwe StructuredQuery en de Selectables van de oude StructuredQuery opgeslagen. Ook is per StructuredQuery een ConditionSet opgeslagen. Deze ConditionSet wordt samen met de koppeling van de Selectables gebruikt voor het filteren van de regels, zodat de juiste regels bij iedere originele query uitkomt.

Het opsplitsen van de ResultSets naar iedere StructuredQuery zoals deze door de RequestConverter is gemaakt, wordt nog steeds gedaan door de SQLConverter. Het opsplitsen van de ResultSets van een StructuredQuery voor een BatchedDataRequest wordt echter gedaan door de SemanticBatchingStrategy, die het combineren van de originele DataRequests verricht heeft. Dit biedt de meeste flexibiliteit voor het toevoegen van SemanticBatchingStrategies.

In de eerste opzet van het technisch ontwerp zijn problemen ontdekt, zoals het ontbreken van een locatie om informatie over het batchproces op te slaan. Deze problemen zijn met deze aanpassingen nu verholpen. Om deze reden is er na deze versie van het technisch ontwerp begonnen met het implementeren van het technisch ontwerp. Het implementatieproces wordt in het volgende hoofdstuk besproken.

4.11: Implementatie Querystructuur

In dit hoofdstuk wordt de implementatie van de querystructuur beschreven.
In het proof of concept is bewezen dat bepaalde onderdelen van het technisch ontwerp mogelijk zijn. De recente aanpassingen hebben het technisch ontwerp echter doen veranderen. Om deze reden is het nodig om te controleren of het genereren van een SQLQuery aan de hand van een StructuredQuery mogelijk is.

Het eerste deel van het technisch ontwerp dat geïmplementeerd moet worden is daarmee de querystructuur. Het doel is het opbouwen van een querystructuur, deze converteren naar SQL en uitvoeren op de database.

Ter kwaliteitscontrole van de code is gebruik gemaakt van Sonar. Sonar is een programma dat de bytecode analyseert. Aan de hand van deze analyse genereert Sonar rapporten met betrekking tot de mate waarin de code getest is. Ook wordt er gekeken of bepaalde patronen in de bytecode voorkomen, die ook bij bugs voorkomen. Deze rapporten worden per bestand gemaakt, waardoor je gemakkelijk kan zien op welke plekken het systeem het minst getest is.

Het implementeren van de querystructuur is vlekkeloos verlopen. Het converteren van deze querystructuur naar een echte query levert echter problemen op.

Het grootste probleem is dat het niet mogelijk is om een referentie te maken naar een tabel die in een Join wordt gebruikt. Dit probleem is te verhelpen door een aparte TableReference aan te maken. Maar dit geeft een probleem bij het gebruik van RowSources die geen tabellen zijn (SubSelectQueries). Daarnaast is een aparte TableReference niet herkenbaar in een abstracte structuur. Dit kan leiden tot een probleem bij syntactisch batchen.

Aanpassingen in het Technisch Ontwerp

Om het probleem met RowSources te verhelpen is het technisch ontwerp aangepast. De grootste aanpassing is dat TableReference verdwijnt als object.

Het aangepaste technisch ontwerp is te groot om zo in het document te plaatsen. Zie hiervoor Figuur 9 in de bijlagen.

TableReference representeerde een “verwijzing naar een Tabel” en bestond daarbij uit een tabelnaam en een tabelalias. Naast het representeren van een tabel was het doel van een TableReference om ambiguïteit te voorkomen bij ColumnReferences. Er kan namelijk ambiguïteit ontstaan wanneer er naar kolom “A” in tabel 1 en naar kolom “A” in tabel 2 verwezen wordt.

Het verwijderen van TableReference betekent dat er geen representatie is voor een tabel binnen het ontwerp. Om dit te verhelpen voegen we een Table-object toe. Een Table-object representeert een tabel in de database en is gekoppeld aan de RowSource-interface. Omdat alle RowSources een alias kunnen hebben in SQL, wordt het aanleveren van een TableAlias toegevoegd aan het “codecontract” van RowSource.

Het tweede doel van TableReference was het oplossen van de ambiguïteit die ontstaat, wanneer twee verschillende ColumnReference-objecten naar dezelfde kolomnaam verwijzen. Dit werd opgelost door het toevoegen van de tabelalias aan de kolomnaam (format “tabelalias.kolomnaam”). Deze oplossing wordt niet aangepast; Door het aanpassen van de verwijzing van ColumnReference

zodat deze naar een RowSource verwijst, blijft deze oplossing bestaan. Met deze aanpassing is het nu ook mogelijk dat er naar kolommen verwezen wordt die niet onderdeel van een tabel zijn, zoals bij SubSelectQueries.

De laatste aanpassing is het opschonen van de relatie tussen FromQueryPart en de RowSources die daar deel van uitmaken. Een query voor het ophalen van gegevens heeft altijd een basistabel. Via Joins kunnen andere (of dezelfde!) tabellen aan deze basistabel gekoppeld worden. Naast de tabelnaam en tabelalias die benodigd zijn voor de basistabel hebben JoinTables andere informatie nodig. Informatie als het JoinType en een ConditionSet, die de koppelrestricties bevat.

De relatie tussen de FromQueryPart en de RowSources is op te schonen door het aanmaken van een aparte variabele voor de basistabel. Omdat de lijst RowSources binnen FromQueryPart dan niet meer de basistabel mag bevatten, kunnen we het type van de lijst verder specificeren tot een lijst van Join-objecten. Dit maakt het omgaan met RowSources binnen batchingsalgoritmen makkelijker.

Verdere implementatie

Met de aanpassingen van het technisch ontwerp, compleet en geïmplementeerd in de code, is het programmeren van een conversie van een querystructuur naar een SQLQuery simpeler. Het algoritme functioneert naar behoren en het gestelde doel is behaald; Opbouwen van een querystructuur, deze converteren naar SQL en uitvoeren op de database.

De volgende stap is het implementeren van een gehele batch aan queries, zonder combinatiealgoritmes. Deze stap wordt in het volgende hoofdstuk beschreven.

4.12: Implementatie Batcher

In dit hoofdstuk wordt de implementatie van de Batcher zonder batchingsalgoritmes beschreven.

Het doel is het valideren van de correctheid van het converteren van requests naar SQLqueries. Dit wordt gedaan door de Batcher aan een rapportageapplicatie te koppelen. Het resultaat dat een rapportageapplicatie ontvangt zou niet moeten wijzigen door het gebruik van de Batcher.

Aanpassingen in het Technisch Ontwerp

Om te voorkomen dat de Batcher te afhankelijk van MrIFunctions wordt, wordt het technisch ontwerp aangepast. Een nieuwe interface, BatchManager, is verantwoordelijk voor het beheren van Batch-objecten. MrIFunctionsAdapter wordt gekoppeld aan deze nieuwe interface. Zo blijft de functionaliteit van het batchbeheer beperkt tot de BatchManager-functies van MrIFunctionsAdapter. De rest van de code heeft nu ook geen kennis meer van “MrIFunctionsAdapter”. Deze werkt nu met een BatchManager. De functionaliteit om met MrIFunctions te werken blijft hierdoor gelokaliseerd tot MrIFunctionsAdapter.

Andere aanpassingen aan het technisch ontwerp zijn het verwijderen van de RequirementStruct en Requirements. In plaats daarvan is het nu mogelijk om attributen aan een DataRequest toe te kennen. Op deze manier is een RequirementStruct als het ware geabstraheerd.

Één van de requirements die tijdens de requirementsanalyse zijn vastgesteld is: “De oude systemen en werkwijzen moeten zoveel mogelijk intact blijven”. Om dit te bewerkstelligen en toch MrIFunctions en de Batcher gescheiden te houden, maakt de RequestConverter nu gebruik van RequestConversionStrategies. Dit zijn “conversiealgoritmes” voor het converteren van een DataRequest naar een StructuredQuery. Daar waar eerst de RequestConverter aangepast moest worden, is het nu mogelijk om los van de Batcher deze conversiealgoritmes te maken. Deze conversiealgoritmes kunnen, wanneer deze nodig zijn, met een enkele regel code aan de RequestConverter toegevoegd worden.

Het aangepaste technisch ontwerp is te groot om zo in het document te plaatsen. Zie hiervoor Figuur 10 in de bijlagen.

Filtering

Niet alle queries hebben er baat bij om via de Batcher uitgevoerd te worden. Alleen voor queries die mogelijk gecombineerd kunnen worden, heeft het nut om deze via de Batcher uit te voeren.

Door deze filtering hoeven niet alle queries die door MrIFunctions gegenereerd worden ook door de Batcher gegenereerd te worden. Dit betekent dat er minder conversiealgoritmes gemaakt hoeven worden.

Een groot gedeelte van de rapporten die Blik op Werk genereert bestaat uit grafieken. Deze grafieken worden opgebouwd aan de hand van queries die gebruikmaken van een statistische functie. Verdere analyse van MrIFunctions toont dat het genereren van queries voor de verschillende statistische functies alleen maar verschilt in de gebruikte statistische functie. Het is dus mogelijk dat een enkel conversiealgoritme als basis zou kunnen dienen voor alle conversiealgoritmes voor queries met statistische functies.

Op basis van deze analyse heb ik besloten om te beginnen met het implementeren van een conversiealgoritme voor queries die gebruik maken van statistische functies. Dit combinatiealgoritme is echter afhankelijk van de structuur van de gegevens zoals deze door de rapportageapplicatie worden aangeleverd. Het is dus eerst nodig om de rapportageapplicatie te analyseren om erachter te komen hoe de data wordt aangeleverd aan MrIFunctions. Ook moet de rapportageapplicatie aangepast worden zodat deze gebruik kan maken van de Batcher.

In het volgende hoofdstuk worden deze analyse en de benodigde aanpassingen besproken.

4.13: Koppelen rapportageapplicatie

In dit hoofdstuk wordt beschreven hoe de rapportageapplicaties gekoppeld worden met de Batcher. Tijdens het koppelen van de rapportageapplicaties wordt een analyse uitgevoerd om vast te stellen wat de exacte implementatie van de conversiealgoritmes moet zijn.

Voordat de rapportageapplicaties met de Batcher kunnen werken moeten ze worden aangepast. Zo moet er bijvoorbeeld voor worden gezorgd dat alle informatie die nodig is voor het genereren van een query beschikbaar is voor de Batcher.

Aanpassingen aan de Rapportageapplicaties

MrIFunctions bouwt queries op aan de hand van een vragenlijst en een rapportenlijst. De vragenlijst wordt aangeleverd door IBM Dimensions als een bestand met de “mdd”-extensie. De rapportenlijst wordt aangeleverd door de rapportageapplicatie. Tijdens het genereren van de queries wordt er gebruik gemaakt van het mdd-bestand. Een van de aanpassingen die dus gedaan moet worden is dat de Batcher de benodigde informatie uit het mdd-bestand gebruikt. Zo is er nog een aantal aanpassingen die betrekking hebben op het mdd-bestand. MrIFunctions haalt bijvoorbeeld de variabelennamen op uit de database en gebruikt die informatie voor het genereren van de queries. Ook hier moet deze informatie meegestuurd worden naar de Batcher.

Naast deze aanpassing moet ook de structuur van de rapportageapplicaties aangepast worden. Momenteel bouwt een rapportageapplicatie eerst een rapportenlijst op. Vervolgens wordt per rapport om gegevens gevraagd. Wanneer deze gegevens binnen zijn, worden ze tot een rapportelement verwerkt en in een datastructuur gezet. Wanneer alle gegevens verwerkt zijn, wordt de datastructuur omgezet in een rapport.

Het probleem is hier dat de rapportageapplicaties direct antwoord verwachten. Het antwoord wat ze ontvangen, wordt vervolgens direct verwerkt. Het is dus niet mogelijk om binnen de Batcher meerdere DataRequests te ontvangen vanuit een rapportageapplicatie. Om dit wel mogelijk te maken, moeten het gedeelte waar de gegevens worden verwerkt en het gedeelte waar de gegevens worden aangevraagd losgekoppeld worden.

Om met de Batcher te kunnen werken moet de structuur van de rapportageapplicaties aangepast worden. Ook moet alle informatie die nodig is voor het genereren van de queries meegestuurd worden.

Aanpassingen aan de Batcher

De Batcher moet ook aangepast worden aan de hand van de uitgevoerde analyse. Nu de interne structuur van de rapportageapplicaties bekend is, kunnen de conversiealgoritmes voor de queries met statistische functies gemaakt worden. De conversiealgoritmes moeten ook toegevoegd worden aan de Batcher, zodat hier gebruik van gemaakt kan worden. Dit wordt gedaan door MrIFunctionsAdapter aan te passen. De conversiealgoritmes horen immers bij MrIFunctions.

Nadat deze aanpassingen toegepast zijn, is het mogelijk om te kijken of de implementatie correct is. Dit wordt gedaan door het rapport, wat op de oude manier wordt gegenereerd, te vergelijken met het rapport dat via de Batcher wordt gegenereerd. Deze twee rapporten blijken hetzelfde te zijn.
De volgende stap in het oplossen van het performanceprobleem is het ontwikkelen en implementeren van batchingsalgoritmes.

4.14: Batchingsalgoritmes

In dit hoofdstuk wordt beschreven hoe een batchingsalgoritme intern werkt en in welke situaties een batchingsalgoritme kan worden gebruikt. Ook beschrijft dit hoofdstuk hoe het gebruik van batchingsalgoritmes de performance kan verbeteren.
In het hoofdstuk technische analyse is eerder verteld dat er twee typen van batchingsalgoritmes zijn. Dit zijn semantisch batchen en syntactisch batchen. Deze twee typen batchingsalgoritmes hebben elk een eigen doel.

Syntactische batchingsalgoritmes hebben als doel om een zo breed mogelijke selectie van queries te kunnen combineren. Een syntactisch batchingsalgoritme wordt gebruikt om aan de hand van een veelvoorkomende structuur een set queries te optimaliseren. Zo dient een syntactisch batchingsalgoritme voor het optimaliseren van de queries in het algemeen.

Semantische batchingsalgoritmes hebben als doel een specifieke set queries te optimaliseren. Bij semantische batchingsalgoritmes worden assumpties gedaan aan de hand van domeinkennis. Door deze assumpties is het mogelijk om een batchingsalgoritme te versimpelen. Omdat het algoritme versimpeld kan worden is het mogelijk om meer complexere querystructuren te behandelen. Zo dient een semantisch batchingsalgoritme voor het optimaliseren van een enkele situatie.

Binnen de Batcher zijn deze twee typen batchingsalgoritmes geïmplementeerd. Wanneer het batchproces begint, worden eerst aan de hand van een semantisch batchingsalgoritme de DataRequests gecombineerd. De DataRequests worden vervolgens geconverteerd naar querystructuren. Daarna worden deze querystructuren door een syntactisch batchingsalgoritme gecombineerd. Daarna wordt de uiteindelijke lijst queries uitgevoerd.

Voor Blik op Werk

Voor Blik op Werk maakt de Batcher momenteel gebruik van twee batchingsalgoritmes. Een daarvan is een syntactisch batchingsalgoritme, de ander is een semantisch batchingsalgoritme.

Het semantische batchingsalgoritme voor Blik op Werk dient om de DataRequests van SortCat te combineren. De queries die voor de functie SortCat van MrIFunctions gemaakt maken gebruik van het WITH-gedeelte van een query. Geen van de andere functies van MrIFunctions levert queries op die gebruikmaken van het WITH-gedeelte.

MrIFunctions bouwt queries op aan de hand van een requirementstructure (kort genaamd reqstruct). In deze reqstruct staan de filteringscriteria die toegepast moeten worden op de dataset. Door middel van JOINS voegt MrIFunctions deze filteringscriteria toe aan een query.

Bij het gebruik van de Batcher is de Batcher verantwoordelijk voor het opbouwen van de queries. Voordat de queries worden opgebouwd, wordt er op semantisch niveau gebatcht. Het semantische batchingsalgoritme dat verantwoordelijk is voor de DataRequests van SortCat doet dit door de reqstructs aan te passen.

Werking semantisch batchingsalgoritme

Het semantische batchingsalgoritme krijgt een lijst van DataRequests binnen. Deze DataRequests komen van de SortCat functie van MrIFunctions. Als eerste stap gaat het semantische batchingsalgoritme de DataRequests opdelen in groepen. Deze verdeling wordt gemaakt aan de hand

van de gegevens die opgevraagd worden. Zo hoeft er geen rekening gehouden te worden met de gegevens die opgevraagd worden tussen verschillende DataRequests. Per groep zijn deze namelijk hetzelfde. De volgende stappen van het semantische batchingsalgoritme worden per groep uitgevoerd.

Als tweede stap gaat het semantische batchingsalgoritme per DataRequest de reqstruct analyseren. Voor het genereren van een rapport bouwt een rapportageapplicatie eerst een algemene reqstruct op. Deze algemene reqstruct bevat filteringscriteria, waaronder het bedrijf waar de rapportage over gaat en de periode waarvan de resultaten benodigd zijn (“alle resultaten van 2012”). Om de vervolgstappen van het algoritme te versimpelen gaat het semantische batchingsalgoritme op zoek naar deze algemene filteringscriteria.

Nadat de algemene filteringscriteria zijn vastgesteld worden per DataRequest de niet-algemene filteringscriteria vastgesteld. In deze derde stap worden er naar twee dingen gekeken. Als eerste wordt gekeken welke van de filteringscriteria direct met de opgevraagde gegevens heeft te maken. Deze filteringscriteria wordt gelabeld als “requestidentifier”. Als er nog andere filteringscriteria zijn (die niet algemene filteringscriteria noch een “requestidentifier” zijn) wordt de groep aan de hand van deze filteringscriteria opgesplitst.

Iedere groep zou op dit moment een set algemene filteringscriteria en een “requestidentifier” moeten hebben. Binnen iedere groep wordt gecontroleerd of alle DataRequests geheel hetzelfde zijn, op de requestidentifier na. Als dit zo is, begint het semantisch batchingsalgoritme met samenvoegen.

Het samenvoegen wordt gedaan door een nieuwe DataRequest aan te maken. Van een van de oude DataRequests wordt de reqstruct gedupliceerd. Vervolgens wordt de gedupliceerde reqstruct aangepast. De requestidentifiers van alle DataRequests uit de groep worden samengevoegd en toegevoegd als attribuut voor de DataRequest. Ook worden verwijzingen naar de oude DataRequests toegevoegd aan de nieuwe DataRequest.

Bij het converteren van de nieuwe DataRequest wordt gekeken of het attribuut voor een lijst van requestidentifiers een waarde heeft. Als dit zo is, voegt het conversiealgoritme enkele dingen toe aan de query. Aan het SELECT- en het GROUPBY-onderdeel wordt de kolom van de requestidentifier toegevoegd. Ook worden de filteringscriteria aangepast om de meerdere requestidentifiers te ondersteunen.

Wanneer de query is uitgevoerd moet het resultaat weer opgesplitst worden. Dit resultaat ziet er ongeveer zo uit:

Amount	**Response**	**RequestIdentifier**
574	1	129
663	2	129
524	3	129
583	1	132
476	2	132
702	3	132
672	1	135
456	2	135
633	3	135

Het opsplitsen van het resultaat wordt gedaan door de lijst oude DataRequests op te halen uit de gecombineerde DataRequest. Van deze oude DataRequests wordt opnieuw de RequestIdentifier vastgesteld. Vervolgens worden de tabelregels aan de hand van de kolom RequestIdentifier aan iedere DataRequest gekoppeld.

Syntactisch Batchen voor Blik op Werk

Het syntactische batchingsalgoritme waar Blik op Werk gebruik van maakt, heeft een veel bredere toepassing dan een enkele functie van MrIFunctions. Het syntactische batchingsalgoritme tracht om een SELECT n+1 probleem te verhelpen.

Een SELECT n+1 probleem is een probleem waarbij, voor het ophalen van alle benodigde informatie, n+1 queries gebruikt wordt. Hiervoor staat n voor het aantal objecten die opgehaald moeten worden. De 1 is de ene query die uitgevoerd wordt om de identificatiecodes van de benodigde objecten op te halen.

Een voorbeeld:
We hebben een gridvraag. Een gridvraag ziet er als volgt uit (invulling ter illustratie):

Geef aan welke attributen en/of stellingen U vindt passen bij de volgende producten.						
	Gezond	Lekker	Geeft me energie	Wil ik dagelijks eten	Goedkoop	Lang houdbaar
Fruit	☑	☐	☐	☐	☑	☐
Vlees	☑	☑	☐	☑	☐	☐
Snoep	☐	☑	☑	☑	☑	☑

Intern is deze vraag op ongeveer de volgende wijze ingericht:

V10_TEKST						
	V10_C1	V10_C2	V10_C3	V10_C4	V10_C5	V10_C6
V10_P1	V10C1P1	V10C2P1	V10C3P1	V10C4P1	V10C5P1	V10C6P1
V10_P2	V10C1P2	V10C2P2	V10C3P2	V10C4P2	V10C5P2	V10C6P2
V10_P3	V10C1P3	V10C2P3	V10C3P3	V10C4P3	V10C5P3	V10C6P3

Voor het beantwoorden van de vraag “Hoeveel respondenten vinden Fruit gezond?” moet dus uit de database gehaald worden hoe vaak V10C1P1 de waarde “1” heeft.

Met 1 SELECT-query haalt een rapportageapplicatie een lijst op van alle benodigde variabelen (V10C1P1 tot V10C6P3). Vervolgens wordt per mogelijke combinatie een DataRequest aangemaakt.

Zo ontstaat er een N (DataRequests) + 1 (de eerste query) probleem.

Intern heeft IBM Dimensions deze vraagcodes geabstraheerd. Er is een tabel “variabelen” in de database aangemaakt aan de hand van de vragenlijst. In deze tabel staat per vraagcode beschreven welke ID deze heeft.

Het semantische batchingsalgoritme tracht, aan de hand van een set querystructuren, te achterhalen welk onderdeel de vraagcode bevat. Als de rest van de querystructuren hetzelfde zijn, kunnen deze gecombineerd worden. Dit wordt gedaan door een nieuw DataRequest-object aan te maken, waar een lijst van IDs in is opgenomen.

Werking syntactisch batchingsalgoritme

Een syntactisch batchingsalgoritme is anders dan een semantisch batchingsalgoritme in dat er geen gebruik gemaakt wordt van domeinkennis. Er wordt puur naar de querystructuur gekeken. Toch maakt een syntactisch batchingsalgoritme enkele assumpties.

Deze assumpties zijn dat:

- Het niet uitmaakt welke queries uitgevoerd worden, als het resultaat maar klopt.
- Het uitvoeren van een query geen gevolgen heeft voor de database (zoals toevoegingen van data).
- De opbouw van de binnenkomende querystructuren valide is.

Deze assumpties zijn noodzakelijk. Zonder deze assumpties zijn er veel meer situaties mogelijk, waardoor het algoritme niet meer met al deze situaties rekening kan houden. Een flexibeler algoritme is hierbij ook een complexer algoritme.

Binnen MrIFunctions zijn een aantal querytypen gedefinieerd. Voor gesloten vragen bestaan er functies als “GetAverage”, “GetCount”, “GetMin”, etc. Deze functies maken queries die een enkele statistische functie uitvoeren op de resultaten van 1 vraag(onderdeel). Denk hierbij aan “GetCount(V10C5P2)” (aantal keer dat een respondent vlees het attribuut “Goedkoop” heeft gegeven). Dit syntactische batchingsalgoritme tracht om per statistische functie alle queries te combineren.

Een syntactisch batchingsalgoritme moet zelf querystructuren filteren. Het syntactische batchingsalgoritme voor Blik op Werk doet dit aan de hand van de gegevens die iedere query opvraagt.

Eerst controleert het syntactische batchingsalgoritme of de query combineerbaar is. Dit wordt gedaan door naar het SelectQueryPart te kijken. Uit het SelectQueryPart wordt een lijst Selectables gehaald. Vervolgens wordt geverifieerd of alle Selectables van het type AggregateFunction zijn. AggregateFunction representeert namelijk een statistische functie. Naast de controle voor AggregateFunctions, wordt er gekeken of er niet al een GroupByQueryPart is. Het is namelijk niet direct voorspelbaar hoe een query met een GROUP BY statement functioneert.

Nadat de querystructuren gefilterd zijn gaat het syntactische batchingsalgoritme proberen een set querystructuren te vinden die combineerbaar zijn. Dit werkt als volgt:

Het syntactische batchingsalgoritme pakt de eerste querystructuur van de lijst en vergelijkt deze met de rest van de lijst. Bij dit vergelijken worden een set controles gedaan. Als er aan een van deze controles niet voldaan wordt, gaat het syntactische batchingsalgoritme door met de volgende querystructuur.

Een voorbeeld uit de code:

```
FromQueryPart fqp1 = sq1.getFrom();
FromQueryPart fqp2 = sq2.getFrom();
if (fqp1 == null || fqp2 == null || fqp1.getJoins().size() !=
fqp2.getJoins().size()) {
    //Missing from part, or amount of joins don't add up - can't batch.
    return null;
}
RowSource base1 = fqp1.getBase();
RowSource base2 = fqp2.getBase();
if (!(base1 instanceof Table || base2 instanceof Table)) {
    //No support for dynamic rowsources.
    return null;
}
Table baseTable1 = (Table) base1;
Table baseTable2 = (Table) base2;
if
(!(baseTable1.getReferenceForFrom().contentEquals(baseTable2.getReferenceFo
rFrom()))) {
    //Both base tables need to be the same.
    return null;
}
WhereQueryPart wqp1 = sq1.getWhere();
WhereQueryPart wqp2 = sq2.getWhere();
if (wqp1 == null || wqp2 == null) {
    //If either of the where's is null, I can't match them.
    return null;
}
ConditionSet wcs1 = wqp1.getConditions();
ConditionSet wcs2 = wqp2.getConditions();
if (wcs1 == null || wcs2 == null || wcs1.getSetType() != wcs2.getSetType())
{
    //If the conditionsets are null, or they have a type mismatch then the
queries can't be batched.
    return null;
}
List<Condition> wcsl1 = wcs1.getConditions();
List<Condition> wcsl2 = wcs2.getConditions();
if (wcsl1.size() != wcsl2.size()) {
    //Mismatch in condition amount in where, so queries do something else.
    return null;
}
```

Hier is te zien hoe het syntactische batchingsalgoritme van twee querystructuren bepaalde aspecten controleert. In dit stuk code wordt gecontroleerd of beide querystructuren uit dezelfde basistabel gegevens ophalen.

Wanneer een paar querystructuren door de controle heenkomen, gaat het syntactisch batchingsalgoritme kijken hoe de twee querystructuren samengevoegd kunnen worden. Hierbij wordt, net zoals bij het semantische batchingsalgoritme, gezocht naar een requestidentifier. Als deze gevonden is, worden gegevens van de requestidentifier in een datastructuur opgeslagen. De twee querystructuren worden vervolgens gemarkeerd als “klaar voor batchen”.

Als er een match gevonden is, gaat het syntactische batchingsalgoritme gewoon door met het zoeken van verdere paren querystructuren die gebatcht kunnen worden.

Wanneer er andere paren gevonden worden, wordt gekeken of de requestidentifier bij alle paren op dezelfde plek geconstateerd is. Paren waarbij dit niet zo is worden uit de lijst gefilterd. Dit voorkomt dat een set queries worden gebatcht terwijl deze onderling niet batchbaar zijn. Wanneer het syntactische batchingsalgoritme de lijst doorlopen heeft, worden alle querystructuren die gemarkeerd zijn als “klaar voor batchen” verwijderd uit de lijst van querystructuren die nog gebatcht moeten worden.

Het combineren van de querystructuren begint met het maken van een kopie van de eerste querystructuur. Het maken van een kopie is normaliter vrij technisch complex (complexer dan dit syntactische batchingsalgoritme). Er wordt echter gebruik gemaakt van een trucje.

Het trucje komt vanuit het werken met netverbindingen. Over een UDP-kabel worden bytes verstuurd. Object georiënteerd programmeren betekent echter dat er overal in het geheugen verwijzingen staan naar andere locaties in het geheugen. Het maken van een exacte kopie werkt dus niet. Bij een exacte kopie verwijzen de verwijzingen namelijk naar de originele structuur.

Bij het serialiseren van objecten worden de objecten geconverteerd naar een sequentiële lijst van bytes. Hierin zijn de onderlinge wijzigingen geabstraheerd, zodat bij het opnieuw opbouwen van de objecten, alle verwijzingen naar de nieuwe objecten verwijzen.

Door middel van serialisatie is het dus mogelijk om een set objecten van de ene computer over een netwerkkabel naar de andere computer te sturen. Maar dat is in dit geval niet nodig. Het syntactische batchingsalgoritme serialiseert de querystructuur en bouwt hem direct weer op. Het resultaat is een kopie van de originele querystructuur, waarbij de verwijzingen van de objecten naar de nieuwe objecten verwijzen.

In de gekopieerde querystructuur worden enkele wijzigingen aangebracht. Aan de SelectQueryPart wordt de requestidentifier toegevoegd. Ook wordt er aan de querystructuur een GroupByQueryPart toegevoegd, waarin gegroepeerd wordt op de requestidentifier. Daarnaast wordt de WhereQueryPart aangepast zodat deze meerdere requestidentifiers ondersteunt.

Als laatste stap van het samenvoegen wordt er een QueryBatch-object aangemaakt. In dit object staat beschreven hoe het resultaat van de nieuwe querystructuur moet worden opgesplitst.

Hier volgt een illustratie van hoe de queryresultaten van de eerder getoonde gridvraag eruit zouden zien voor en na samenvoeging. Hierbij is ervoor gekozen om alleen de eerste drie queries te tonen – dit zou de situatie duidelijk genoeg moeten illustreren.

Voor het samenvoegen van de queries:

V10C1P1:

Amount
923

```
SELECT COUNT(response) as 'Amount'
FROM MO10002LC.dbo.responses2
(... filteringscriteria)
WHERE variableID = 77;
```

V10C1P2:

Amount
772

```
SELECT COUNT(response) as 'Amount'
FROM MO10002LC.dbo.responses2
(... filteringscriteria)
WHERE variableID = 78;
```

V10C1P3:

Amount
49

```
SELECT COUNT(response) as 'Amount'
FROM MO10002LC.dbo.responses2
(... filteringscriteria)
WHERE variableID = 79;
```

Na het samenvoegen van de queries:

Resultaat:

Amount	**QID**
923	77
772	78
49	79

```
SELECT COUNT(response) as 'Amount', variableID as 'QID'
FROM MO10002LC.dbo.responses2
(... filteringscriteria)
WHERE variableID IN (77, 78, 79)
GROUP BY variableID;
```

QueryBatch:

Query	**Oude Kolom**	**Nieuwe Kolom**	**Voorwaarde**
V10C1P1	Amount	Amount	QID = 77
V10C1P2	Amount	Amount	QID = 78
V10C1P3	Amount	Amount	QID = 79

Aan de hand van het QueryBatch-object kan later dus het resultaat van de gecombineerde query opgesplitst worden naar de individuele queries.

Wanneer een set queries op deze manier is gecombineerd, begint het syntactische batchingsalgoritme weer aan het begin van de lijst. Het syntactische batchingsalgoritme pakt de eerste querystructuur van de lijst en vergelijkt deze met de rest van de lijst.

Wanneer het niet lukt om een match te vinden, wordt een querystructuur van de lijst afgehaald en begint het syntactische batchingsalgoritme ook vanaf het begin van de lijst opnieuw.

Wanneer de lijst leeg is, is het syntactische batchingsalgoritme klaar.

In het volgende hoofdstuk wordt beschreven welke impact deze batchingsalgoritmes hebben op de performance van de rapportgeneratie.

4.15: Performancetesting

In dit hoofdstuk wordt beschreven hoe het gebruik van batchingsalgoritmes de performance van de rapportagegeneratie verbetert.

In het hoofdstuk Batchingsalgoritmes zijn twee batchingsalgoritmes beschreven. Dit waren een syntactisch batchingsalgoritme voor queries met statistische functies en een semantisch batchingsalgoritme voor SortCat. Deze batchingsalgoritmes zijn gedurende het performancetesten ontwikkeld.

Lokaal testen

Om te kunnen meten hoe groot de verbetering is van een aanpassing, moet eerst een standaardresultaat vastgesteld worden. Hiervoor wordt een rapportage gegenereerd zonder gebruik te maken van de Batcher. In de lokale testomgeving is dit resultaat 28 seconden voor het genereren van een rapportage.

Bij toepassing van het syntactische batchingsalgoritme wordt dit resultaat verbeterd tot 17 seconden. Deze verbetering is al aanzienlijk, maar er is ruimte voor verdere verbetering. SortCat maakt gebruik van een statistische functie, net als de GetAverage/GetCount/GetMax/GetMin functies van MrIFunctions. Omdat SortCat echter meerdere resultaten tegelijk opvraagt, kan deze niet door het syntactische batchingsalgoritme geoptimaliseerd worden.

Tijdens het overleg met dhr. Van der Storm geef ik aan dat met een semantisch batchingsalgoritme, SortCat ook geoptimaliseerd kan worden. Dhr. Van der Storm stelt een verdere performanceverbetering op prijs en stemt in met het ontwikkelen van een tweede batchingsalgoritme.

Het toepassen van het semantische batchingsalgoritme voor SortCat levert een verdere 4 seconden performancewinst op. Hiermee duurt het genereren van het rapport nog slechts 13 seconden.

Om meer inzicht te krijgen in verdere performanceoptimalisaties heb ik een analyse uitgevoerd. Dit inzicht wordt verkregen door de verdeling van de huidige executietijd in kaart te brengen. Uit mijn analyse blijkt het volgende:
Het initialiseren van MrIFunctions en de Batcher is in enkele milliseconden klaar. Vervolgens duurt het opstellen van de rapportagestructuur 3,5 seconden. Het toevoegen van DataRequests aan de Batcher kost 1 seconde. Het batchproces van de Batcher kost bijna 8,5 seconden. Het vervolgens uitpakken en verwerken van de ontvangen resultaten kost een tiende van een seconde.

Opvallend in deze verdeling is dat het opstellen van de rapportagestructuur 3,5 seconde in beslag neemt. Dit komt omdat bij het opstellen twee grote queries worden uitgevoerd. Deze twee queries kunnen niet gecombineerd worden, omdat deze niet van statistische functies gebruikmaken.

Verder blijkt uit mijn analyse dat het gebruik van de Batcher een halve seconde overhead met zich meebrengt. Het uitvoeren van de queries neemt iets meer dan 8 seconden in beslag.

Testen in productie

Omdat de afstudeerperiode bijna is afgelopen, wordt de Batcher in de productieomgeving getest zonder verder performanceverbeteringen.

De productieomgeving maakt gebruik van een oudere versie van de applicatieserver dan de versie die in de testomgeving gebruikt is. Om deze reden wordt bij de rapportagegeneratie een iets trager resultaat verwacht.

Voorafgaand aan de implementatie van de Batcher wordt een standaardmeting gedaan. In de productieomgeving duurt het ongeveer 38 seconden voor Blik op Werk om een rapport te genereren.

Na implementatie van de Batcher wordt de executietijd van het genereren van een rapport nogmaals gemeten. Het kost nu ongeveer 21 seconden voor Blik op Werk om een rapport te genereren. Dit is een vrij groot verschil met de testomgeving. Om dit verschil te verklaren, maak ik opnieuw een analyse van de verdeling van de executietijd.

Uit deze analyse blijkt dat in de productieomgeving, het batchproces ook slechts 8,5 seconden inneemt. Het opstellen van de rapportagestructuur is net als in de testomgeving in 3,5 seconden klaar.

In de productieomgeving wordt echter meer gedaan dan alleen een rapportage samenstellen. In tegenstelling tot de testomgeving, wordt in de productieomgeving ook een PDF-bestand gegenereerd. Uit een verdere meting blijkt dat het opbouwen van het PDF-bestand bijna 9 seconden in beslag neemt. Dit verklaart het verschil in de tijd voor het genereren van een rapportage tussen de productie- en testomgeving.

Korte probleemanalyse genereren PDF-bestand

Om erachter te komen waarom het genereren van het PDF-bestand 9 seconden duurt, besluit ik een korte probleemanalyse te maken.

Bij deze probleemanalyse heb ik de volgende hypotheses opgesteld:
Hypothese 1: Het PDF-bestand wordt stukje voor stukje opgebouwd, waarbij stringconcatenatie plaatsvindt. Wegens de werking van stringconcatenatie wordt tijdens het opbouwen van het rapport genoeg tekst gekopieerd voor wel 50[7] rapporten. Deze hypothese valt te bewijzen door de verdeling van de executietijd van het genereren te controleren. Performanceproblemen veroorzaakt door stringconcatenatie zijn namelijk exponentieel.
Hypothese 2: Bij het genereren van het PDF-bestand moet er gewacht worden op de beschikbaarheid van externe bronnen. Bij het bijvoorbeeld ophalen van een afbeelding moet een webpagina geopend worden, zodat de afbeelding van de website gedownload kan worden. Dit brengt een bepaalde vertraging met zich mee. Deze hypothese valt te bewijzen door het rapport te genereren, waarbij geen gebruik wordt gemaakt van externe middelen.
Hypothese 3: De functie voor het genereren van het PDF-bestand is over het algemeen traag. De verdeling van de executietijd van de generatiefunctie is vrijwel uniform. Deze hypothese valt te ontkrachten door de verdeling van de executietijd van het genereren te controleren. Als de verdeling van de executietijd niet uniform is, heeft de lange duur van het genereren een specifieke oorzaak.

Het genereren van het PDF-bestand door de rapportageapplicatie wordt als volgt uitgevoerd:
Eerst wordt er een stuk CSS opgebouwd, waarmee de stijl van het PDF-bestand wordt gedefinieerd.

[7] Gegeven 4 stukjes per pagina, heeft een rapport van 25 pagina's 100 stukjes. Door middel van stringconcatenatie wordt dus geheugen gebruikt voor (n*(n/2) + n/2) stukjes, of wel (100*(100/2) + 100/2) = (100*(50) + 50) = 5050 stukjes, en dus 50,5 rapporten.

Vervolgens wordt met behulp van de functie “cfdocument” het PDF-bestand gegenereerd. Dit wordt gedaan door met “cfdocumentsection” en “cfdocumentitem” onderdelen van het PDF-bestand te definiëren. Iedere pagina van het PDF-bestand bestaat uit een “cfdocumentsection”, met daarin een samenvoeging van stukken tekst, resultaten uit de rapportstructuur en een “cfdocumentitem” voor het paginanummer.

Om te controleren of stringconcatenatie iets met het probleem te maken heeft (hypothese 1), wordt de manier van opbouwen aangepast. Eerst wordt de inhoud van alle pagina’s samengesteld buiten de functie “cfdocument” om. Daarna worden deze pagina’s ingevoegd. Door deze aanpassing is het mogelijk om te meten hoe lang het duurt om de pagina’s op te bouwen.

Het blijkt slechts een tiende van een seconde te duren om alle pagina’s op te bouwen. Hiermee is de lange executietijd niet te verwijten aan stringconcatenatie binnen Blik op Werk. Het is echter nog wel mogelijk dat de functie “cfdocument” hier last van ondervindt.

Bij het genereren van een PDF-bestand met slechts een paar pagina’s blijkt dat dit 1,5 seconde duurt. Hiermee is het onmogelijk dat de verdeling van het genereren een exponentiële curve volgt. Dit geeft aan dat stringconcatenatie niet de hoofdoorzaak is van het probleem.

In deze paar pagina’s wordt geen gebruik gemaakt van externe verwijzingen. Hiermee is de hypothese dat er gewacht moet worden op externe bronnen ook ontkracht.

Er blijft een enkele hypothese over: “De functie voor het genereren van het PDF-bestand is over het algemeen traag”. Als we de assumptie maken dat deze hypothese klopt, leidt dit tot de volgende vraag: “Waarom”?

Op dit moment van de afstudeerstage heb ik niet de expertise, noch de tijd, om een antwoord te vinden op deze vraag. Dhr. Van der Storm kan ook niet een direct antwoord geven. Dhr. Van Velzen heeft echter een suggestie:

Bij het genereren van de rapportage wordt eerst in HTML een set rapportageonderdelen opgebouwd. Bij het genereren van het PDF-bestand worden deze rapportageonderdelen vervolgens met andere stukken HTML gecombineerd.

Het genereren van een rapportage wordt in HTML gedaan omdat eerdere klanten online de resultaten wilde bekijken. De rapportages worden daarvoor gegenereerd als webpagina’s. Bij Blik op Werk wil de klant echter de rapportage als PDF-bestand kunnen downloaden. Om dit te bewerkstelligen wordt zoals bij andere projecten de rapportage als webpagina’s gegenereerd. Vervolgens worden de webpagina’s via de functie “cfdocument” geconverteerd tot PDF-bestand.

Een PDF-bestand bestaat echter niet uit HTML. De functie “cfdocument” moet dus de HTML-code converteren naar de objecten waar een PDF-bestand uit bestaat. Hiervoor moet de HTML geïnterpreteerd worden, zoals een webbrowser dit moet doen bij het weergeven van een webpagina.

Dit zou mogelijk de oorzaak kunnen zijn waarom het genereren van het PDF-bestand lang duurt. Binnen de resterende tijd van mijn afstudeeropdracht is het echter niet meer mogelijk om een oplossing te vinden voor dit probleem. Wel is het mogelijk om dit probleem als mogelijke performanceverbetering te noteren. Door middel van een adviesrapport voor Panteia kan ik

vervolgens alle mogelijke performanceverbeteringen presenteren. Verder geef ik in het adviesrapport aan welke aanpassingen Panteia zou moeten toepassen ter verbetering van de performance.

In het volgende hoofdstuk beschrijf ik de laatste fase van mijn afstudeerproject: de overdracht van de Batcher en alle tussenproducten die ik tijdens de afstudeeropdracht gemaakt heb.

4.16: Overdracht

In dit hoofdstuk beschrijf ik hoe de overdracht van de gemaakte producten is verlopen. Gedurende het performancetesten heb ik geconstateerd dat in de productieomgeving het genereren van een rapport uitkwam op 21 seconden. Ik had dit resultaat niet verwacht. In de testomgeving duurde het genereren van een rapport 13 seconden. In de testomgeving was het genereren van het PDF-bestand echter niet meegerekend.

Aangezien het niet lukt om een directe oplossing te vinden voor het verbeteren van de performance van het genereren van een PDF-bestand, besluit ik een adviesrapport op te stellen.

Het adviesrapport bevat een set mogelijke aanpassingen die Panteia zou kunnen toepassen, om de performance van de rapportagegeneratie bij Blik op Werk verder te verbeteren. Uit deze set mogelijke aanpassingen wordt er een aantal geadviseerd. Dit advies is gebaseerd op de verwachte kosten en baten van het implementeren van een aanpassing.

Voor de website adviseer ik om feedback aan een gebruiker te geven. Dit zou gedaan kunnen worden door tijdens het genereren van een rapport aan te geven dat de server bezig is.

Voor de database adviseer ik om een nieuwere versie van MSSQL Server te gebruiken. Een nieuwere versie van MSSQL Server zou een verbetering in de performance leveren. Ook leveren nieuwere versies van MSSQL Server meer ondersteuning voor optimalisatie van queries en databaseconfiguratie.

Voor de Batcher adviseer ik om multithreading te gaan gebruiken voor het uitvoeren van queries. Dit is een complexe taak, maar deze aanpassing zou ongeveer de helft van de executietijd van de queries kunnen schelen.

Voor MrIFunctions adviseer ik om in de SortCat functie een filter toe te voegen. Dit filter controleert of onderdelen van de query dubbelop zijn. Dit zorgt ervoor dat de queries die op de database worden uitgevoerd geen overbodige complexiteit bevatten.

Voor de rapportageapplicatie van Blik op Werk adviseer ik om naar een andere manier te kijken voor het genereren van een PDF-bestand. Dit kost 8-9 seconden en lijkt daarmee de grootste bijdrage aan het performanceprobleem.

Dit adviesrapport is, samen met de andere documenten en de Batcher, die gemaakt zijn tijdens dit afstudeerproject, opgeleverd aan dhr. Van der Storm. Panteia neemt binnenkort op basis van het adviesrapport een beslissing.

Na mijn presentatie van het adviesrapport en op basis van het afgeleverde werk heeft Panteia mij gevraagd of ik voor hen zou willen komen werken. Zodra ik ben afgestudeerd wil Panteia samen met mij uitzoeken of Webdevelopment bij mij past en of ik in het webteam pas (waar dhr. Van der Storm en dhr. Van Velzen onderdeel van uitmaken).

5: Productevaluatie

In dit hoofdstuk vertel ik per product wat ik van het eindresultaat vind.

Probleemanalyse

De probleemanalyse is het resultaat van mijn gedachtegang gedurende de oriëntatie. Aanvankelijk heb ik de probleemanalyse alleen gemaakt omdat ik voor het afstuderen gestructureerd moet werken. Nadat ik de probleemanalyse had gemaakt, bleek echter dat dit een hele goede manier is om de situatie te documenteren. Daarnaast was de probleemanalyse goed bruikbaar in het faciliteren van communicatie tussen mij en dhr. Van der Storm. Een probleemanalyse kan in een korte tijd geschreven worden. Om deze reden denk ik dat ik in de toekomst een probleemanalyse vaker als documentatie zal gebruiken.

Plan van Aanpak

Aan de hand van de probleemanalyse heb ik een oplossingsrichting gedefinieerd. Deze oplossingsrichting diende als input voor het plan van aanpak. Binnen dit project heb ik het plan van aanpak gebruikt als communicatiemiddel en niet als een contract. Er is namelijk al een afstudeerplan, wat dient als een contract. Het plan van aanpak is niet bijgewerkt aan de hand van het verloop van het project. Dit vind ik achteraf jammer. Als ik naar de laatste versie van het plan van aanpak kijk, zie ik dat het wel meevalt. Er zijn vrijwel geen veranderingen geweest in het geplande verloop van het project en het daadwerkelijke verloop van het project. Enige aanpassingen zouden puur aanvullingen zijn op de informatie die al reeds in het plan van aanpak staat.

Architectuuranalyse

In het plan van aanpak heb ik een oplossingsrichting gedefinieerd. Om meer kennis te vergaren over de werking van de systemen die een rol spelen bij de rapportagegeneratie van Blik op Werk heb ik een deploymentdiagram gemaakt. De architectuuranalyse is een bestudering van dit deploymentdiagram. Achteraf vind ik dat ik het juiste abstractieniveau heb gebruikt. Er is geen sprake van irrelevante details, maar alle relevante systemen zijn wel gemodelleerd en beschreven. Van een enkel ding heb ik wel spijt: Waarom noemde ik een rapportageapplicatie een “rapportagemaker”?

Requirementsanalyse

In eerste instantie leek het maken van een requirementsanalyse overbodig. Het was vrij duidelijk wat er gedaan moest worden – de rapportagegeneratie van Blik op Werk duurt lang; maak het sneller. Het niet uitvoeren van een requirementsanalyse is echter een valkuil. Hoe kan je immers weten wat men wel of niet wil als je dit niet hebt uitgezocht?

Na uitvoeren van de requirementsanalyse bleek dat er geen directe eisen werden gesteld met betrekking tot de werking van een oplossing. Ik vind dat de tijd die ik aan de requirementsanalyse heb besteed grotendeels verspilde tijd is geweest. Natuurlijk moet je wel een requirementsanalyse uitvoeren, maar niet op de schaal waarop ik dat heb gedaan. Daar is normaliter geen tijd voor binnen een bedrijf.

Als ik “gewoon” bij Panteia werkte, had ik een simpelere requirementsanalyse uitgevoerd.

Functioneel Ontwerp

Aan de hand van de architectuuranalyse en requirementsanalyse heb ik een functioneel ontwerp gemaakt. De architectuuranalyse levert de context om het functioneel ontwerp binnen te plaatsen. Daar waar een requirementsanalyse de eisen, wensen en mogelijke voorkeuren beschrijft, beschrijft een functioneel ontwerp een invulling van deze requirements.

Ik moet hierbij terug denken aan het blok I-3. In I-3 heb ik een requirementsanalyse gemaakt. Deze requirementsanalyse was, volgens de expert voor mijn projectgroep, onvolledig. Als een applicatie bepaalde dingen moet gaan verrichten, moeten deze taken in de requirements terugkomen. Ik vond dat destijds moeilijk te begrijpen: Waarom zou een requirementsanalyse beschrijven hoe een applicatie intern dient te functioneren? Na deze afstudeerstage heb ik dit echter door.

Ik ben van mening dat een requirementsanalyse dient om de eisen van een opdrachtgever en andere stakeholders te beschrijven. Dat zijn de restricties waarbinnen je een oplossing moet vinden voor het probleem. Het functioneel ontwerp is een invulling van deze oplossing, waarbij rekening is gehouden met deze restricties. Zo wordt de werking van een systeem (bij dit project de Batcher) vastgesteld zonder inperkingen in de creativiteit van een oplossing.

Ik ben zeer tevreden met het functioneel ontwerp wat ik heb opgeleverd.

Technische Analyse

Bij de technische analyse is er gebruikgemaakt van een proof of concept. Bij een proof of concept zijn een aantal dingen van belang. Een proof of concept moet snel ontwikkeld worden en doelgericht zijn. Een proof of concept is als een maquette van een huis; bedoeld als hulpmiddel bij het ontwerpen van het uiteindelijke product.

Deze aspecten zijn ook teruggekomen bij het proof of concept wat ik gemaakt heb. De vragen die het proof of concept ging beantwoorden, zijn beantwoord. Daarmee ben ik tevreden met de technische analyse.

Technisch Ontwerp

Technisch ontwerp is implementatie ontwerp van functioneel ontwerp. Hier zijn een hoop verschillende versies van gemaakt. Na versie v1.4 van het diagram ben ik gestopt met het bijwerken van de beschrijving omdat deze niet gebruikt werd.

Het technisch ontwerp bestaat uit een klassendiagram en een document waarin dit klassendiagram wordt beschreven. Er zijn uiteindelijk 7 versies van het diagram geweest. De eerste 5 zijn getoond in dit document. De laatste twee versies, v1.5 en v1.6, omvatten slechts kleine aanpassingen.

Ik weet niet wat ik van mijn technisch ontwerp moet vinden. Ik ben van mening dat mijn ontwerp goed is. Wat ik echter in mijn ontwerp terug zie is een motief wat ik in meerdere van mijn ontwerpen zie; via een set abstractielagen tracht ik data te abstraheren naar objecten. Een voorbeeld uit het blok I-2:

Een van de praktijkopdrachten in het blok I-2 was het ontwerpen van een “tekstspel”. Dit spel bestaat uit kamers. Tussen deze kamers zitten verbindingen zodat een speler van de ene kamer naar een andere kamer kan. In een kamer kunnen zich meerdere objecten bevinden. Een object heeft een beschrijving, een naam en een gewicht.

Tot nu toe was deze opdracht vrij simpel. Er zijn dus blijkbaar kamers met verbindingsobjecten die weer naar andere kamers verwijzen. En in deze kamers kunnen zich objecten bevinden. Daarmee hebben we drie klassen: Kamer, Verbinding en Object.

Het tweede gedeelte van de opdracht bestond uit het toevoegen van een interactief object (bijvoorbeeld een zaklamp). Een speler kan dit oprapen en gebruiken in een andere kamer.

De speler kan iets oprapen, dus er komt een klasse bij die bijhoudt welke objecten de speler bij zich heeft. Het enige wat nu nog ontworpen moet worden, is het kunnen gebruiken van dit object. En daar gaat het bij mij mis. Ik heb destijds ervoor gekozen om een Actie-klasse aan te maken. Een object kan meerdere Acties hebben. Voor het implementeren van een zaklamp moet men subklassen maken van de klasse Actie om “ZaklampActie” toe te voegen. Dit heeft echter tot het gevolg dat ieder interactief object zijn eigen klasse krijgt. Hiermee kan een klassediagram zeer groot worden. Een alternatief is het aanmaken van een “Zaklamp”-klasse, wat een object is met speciale functies. Maar dat betekent dat er geen ondersteuning is voor andere interactieve objecten.

Bij het controleren van het huiswerk kreeg ik te horen dat ik “voor een interessante oplossing had gekozen”. Daar kon ik weinig mee; is dit goed, of is dit fout?

Bij de Batcher heb ik dezelfde oplossing toegepast. Het toevoegen van een nieuw querytype wordt gedaan door een nieuwe subklasse te maken van RequestConversionStrategy. Het toevoegen van een nieuw syntactisch of semantisch batchingsalgoritme wordt gedaan door een nieuwe subklasse te maken van Semantic- of SyntacticBatchingStrategy. Ik weet niet of dit goed of fout is.

Ik vermoed dat het voor de Batcher wel de juiste keuze is. Door het toepassen van deze abstractielaag hoeft een programmeur zich niet bezig te houden met de werking van de Batcher. Panteia heeft geen Java-programmeurs in dienst. Als iemand dus een nieuw querytype moet toevoegen aan de Batcher, hoeft hij geen directe kennis te hebben van de interne werking van de Batcher.

Batcher

De Batcher is de implementatie van het technisch ontwerp. Het idee om te batchen is voortgekomen uit het maken van technische ontwerpen voor andere projecten. Bij het schrijven van code of het ontwerpen van een applicatie is een van de richtlijnen die ik hanteer DRY. Don’t Repeat Yourself. Herhaal jezelf niet. Zodra drie of meer onderdelen van een systeem hetzelfde doen, moet dit geabstraheerd worden als dat mogelijk is.

DRY schoot door mijn hoofd toen ik naar de queries keek. Ze waren grotendeels hetzelfde – dus abstraheren waar mogelijk.

Toepassen van dit idee heeft goed uitgepakt. Ik ben trots op de Batcher zoals ik die gemaakt heb.

Wat ik minder vind, is hoe ik de batchingsalgoritmes heb geprogrammeerd. Normaliter bestaan mijn programmeerproblemen uit “hoe schrijf ik dit op een mooie manier op?”. Het laten werken van de code is geen probleem – binnen een minuut heb ik wel 10 manieren bedacht voor het programmeren van een functie. Maar welke manier levert mooie, leesbare code? Herbruikbare code? Onderhoudbare code? Omdat ik mij het grootste gedeelte van de tijd bezig hou met deze vragen worden deze aspecten goed verwerkt in mijn code.

Bij het programmeren van de batchingsalgoritmes bestonden mijn programmeerproblemen echter uit “hoe laat ik dit werken?”. Daardoor heb ik vrijwel geen tijd besteed aan het controleren van de leesbaarheid van de code. Ik vind dat dat te zien is.

Met behulp van cyclomatische complexiteit kan men objectief meten hoe complex een algoritme is. De cyclomatische complexiteit van een algoritme is het aantal individuele lineaire paden binnen het algoritme. Dit betekent dat de cyclomatische complexiteit een waarde heeft van 1, plus het aantal beslissingspunten in het algoritme. Meestal heeft een functie binnen een object een cyclomatische complexiteit van 1 tot 4. Het syntactische batchingsalgoritme komt op een totale cyclomatische complexiteit uit van meer dan 100. Het semantische batchingsalgoritme komt op een totale cyclomatische complexiteit uit van meer dan 75.

De complexiteit van de batchingsalgoritmes is verdeeld over slechts 3 functies. Dit zorgt ervoor dat de code lastig is om te begrijpen. Ook wordt de onderhoudbaarheid hiermee geschaad.

Aan de andere kant – ik heb niemand om mee samen te werken bij Panteia. Dhr. Van der Storm heeft niet genoeg kennis van Java om ondersteuning te bieden bij de implementatie van de batchingsalgoritmes. Ik heb uiteindelijk gedaan wat ik kon. Toen de batchingsalgoritmes op een late avond dan ook werkten, was ik blij.

Eindresultaat

Uit performancetests bleek dat de implementatie van de Batcher leidt tot een reductie in de executietijd van meer dan de helft. Ik was erg blij met dit resultaat. Van 28 seconden naar 13. 28 seconden is lang wachten. 28 seconden is de tijd die een snelle windows-computer nodig heeft om op te starten. 13 seconden is echter vrij snel. Zeker voor zo’n groot rapport.

De resultaten van de productieomgeving kwamen vrij hard aan bij mij. 21 seconden?! Hoe kan het dat ik in de testomgeving 13 seconden zie, en in de productieomgeving 21 seconden?

Om dit verschil uit te leggen heb ik een analyse uitgevoerd op de rapportagegeneratie in de productieomgeving. Daaruit bleek dat het genereren van het PDF-bestand 8-9 seconden in beslag neemt.

Ik heb geprobeerd het probleem op te lossen middels een probleemanalyse en het uitvoeren van proefondervindelijke tests. Ik heb daarbij geen oplossing voor het probleem gevonden en daar baal ik van.

Alhoewel ik trots ben op de Batcher had ik meer tijdswinst met het eindresultaat willen behalen. Het verschil in de productieomgeving vind ik niet groot genoeg. Als ik naar de beslissingen kijk die ik genomen heb zie ik niet waar ik een foute beslissing heb gemaakt. Misschien had het geholpen om eerder in de productieomgeving te gaan testen. Dit had echter niet veel geholpen in het geven van een voortijdige waarschuwing. De Batcher is namelijk een oplosser, niet een oplossing. De batchingsalgoritmes zijn de laatste onderdelen van de Batcher die ik ontwikkeld heb. Ik had mogelijk kunnen testen in de productieomgeving voor de implementatie van het semantische batchingsalgoritme. Dit had echter niet veel uitgemaakt – het implementeren van het semantische batchingsalgoritme vond 1 week voor de test in de productieomgeving plaats.

6: Procesevaluatie

In dit hoofdstuk vertel ik wat ik vind van de manier waarop ik tot het eindresultaat van iedere fase van het project ben gekomen. Ook stel ik mijzelf de vraag “Zou ik dit bij een ander project weer zo doen?”.

Oriëntatie op het probleem

Op de eerste dag van mijn afstudeerstage ben ik begonnen met uitzoeken wat er precies aan de hand was. Dit proces omvat het stellen van veel vragen en het uitproberen van alles wat er in mijn hoofd opkomt. Dit is niet gestructureerd werken. Deze ongestructureerde manier van werken levert echter wel een hoop informatie op.

Aan het eind van deze eerste dag had ik besloten dat ik de queries die door MrIFunctions worden gegenereerd wilde bekijken. Ik heb aan dhr. Van der Storm gevraagd of het mogelijk was om deze queries op het scherm af te drukken. Dat was niet mogelijk, omdat de queries direct naar de database gestuurd werden.

De volgende dag heb ik echter verdere reactie van dhr. Van der Storm. Hij is die avond tot laat bezig geweest om MrIFunctions aan te passen zodat ik alsnog de queries kon bekijken.

Hier werd ik op een prettige manier door verrast. Dit niveau van ondersteuning ben ik niet gewend. Gedurende het project zijn er meerdere gevallen geweest waarin ik dit niveau van ondersteuning heb ontvangen van dhr. Van der Storm. Ik merk dat dit mij motiveert om mijn werk te doen.

Na twee dagen op een ongestructureerde manier gezocht te hebben naar de hoofdoorzaak van het performanceprobleem ben ik overgestapt op gestructureerd werken. Dit heb ik gedaan door datgene wat ik heb gedaan te beschrijven in een probleemanalyse.

Zou ik dit bij een ander project weer zo doen? Bij projecten als dit wel. Ik moest mij oriënteren op het bedrijf, de opdracht en het probleem. Daarvoor is het stellen van veel vragen en het uitproberen van alles wat er in mijn hoofd opkomt een goed idee. Wanneer ik echter een project moet uitvoeren in een bekende omgeving, is er meer baat bij het gebruik van een gestructureerde aanpak vanaf het begin. Mocht ik toch gebruikmaken van een “ongestructureerde aanpak”, dan zou ik direct noteren wat ik heb uitgeprobeerd.

Het zoeken naar een mogelijke oplossing

Aan de hand van de probleemanalyse had ik een oplossingsrichting vastgesteld. Met behulp van een architectuuranalyse en een requirementsanalyse zou ik kunnen vaststellen welke vorm een oplossing zou hebben.

Ik heb deze aanpak beschreven in het plan van aanpak. Het plan van aanpak heeft samen met de architectuuranalyse gediend als communicatiemiddel met dhr. Van der Storm. Tijdens de oriëntatie heb ik een beeld geschetst over de opdracht en de bestaande architectuur. Met deze documenten beschrijf ik de opdracht en de huidige situatie zoals ik die zie. Als dhr. Van der Storm het met deze beschrijving eens is dan heb ik de situatie juist geïnterpreteerd. Deze werkwijze vond ik prettig werken.

Tijdens de oriëntatie had ik al een mogelijke oplossing bedacht. Door middel van het samenvoegen van queries kon de executietijd verminderd worden. Ik had deze oplossing graag direct

geïmplementeerd. Het direct implementeren van deze oplossing leidt echter tot een ongestructureerde aanpak. Achteraf denk ik dat het direct implementeren van de oplossing geen directe problemen voor mij tot gevolg zou hebben gehad. Wel zou dit het maken van enige keuzes bemoeilijken. Dit komt omdat door het maken van een architectuuranalyse en een requirementsanalyse, ik inzicht heb gekregen in hoe het systeem momenteel werkt en hoe Panteia wil dat het systeem in de toekomst werkt. Zonder dit inzicht is het niet mogelijk om keuzes te maken waarbij het belang van Panteia meegenomen wordt in de motivatie voor een keuze.

Kortom: alhoewel ik graag direct van start was gegaan met implementeren van een oplossing, ben ik van mening dat door het hanteren van een gestructureerde aanpak, ik betere keuzes kon maken in de latere fases van het project.

Zou ik dit bij een ander project weer zo doen? Ook hier ligt het aan het project. Dit is een intern project waarbij ik mij alleen maar naar dhr. Van der Storm hoef te verantwoorden. Hierdoor is focus op goede communicatie met dhr. Van der Storm voldoende. Bij een groter project moet de documentatie voor een meer algemene lezer geschreven worden. Qua hantering van een gestructureerde aanpak heb ik gedurende mijn opleiding geleerd dat het altijd veel simpeler lijkt dan dat het eigenlijk is. Als je zomaar een oplossing gaat implementeren loopt je project een mijnenveld in. Één fout in de requirements en BOEM – je bent af. Het is beter om veilig een gestructureerde aanpak te gebruiken. Bij tijdnood kan men dan nog altijd besluiten een kortere weg te gebruiken.

Het specificeren van de gevonden oplossing

De oplossing die ik tijdens de oriëntatie had bedacht bleek tot dusver nog de juiste oplossing te zijn. Tenminste, dat denk ik. Het is lastig om een open geest te hebben wanneer je al een oplossing heb bedacht.

Het functioneel ontwerp dient om te beschrijven wat een oplossing gaat doen. Hiermee kon ik mijn gevonden oplossing in detail beschrijven. Het maken van een functioneel ontwerp dwingt je om je oplossing uit te werken. Enige problemen komen daarmee al snel aan het licht. Dit geeft een vorm van zelfcontrole, wat ik heel handig vind. Bij het functioneel ontwerp besluit ik de oplossing de naam “Batcher” te geven.

Met behulp van een proof of concept kan ik kijken of mijn idee ook daadwerkelijk toepasbaar is. Ook kan ik zien of er enige problemen zijn die zouden kunnen voorkomen bij de implementatie van het probleem. In eerdere projecten heb ik problemen gehad met het vinden van de juiste scope voor een proof of concept. Een proof of concept wil al snel uitgroeien tot een “concept of product”. Deze keer heb ik echter eerst een set vragen bedacht. Deze vragen moeten door het proof of concept beantwoord worden. Dit geeft houvast bij het bepalen van de scope van het proof of concept, waardoor dit niet te groot wordt.

Aan de hand van het proof of concept heb ik een eerste opzet gemaakt van het technisch ontwerp. Toen ik dit aan dhr. Van der Storm presenteerde had hij moeite met het begrijpen van het klassendiagram. Dit vond ik vervelend. Tot op dit punt was het makkelijk om met behulp van documentatie de stand van het project te communiceren naar dhr. Van der Storm. Dat zou vanaf dit punt alleen maar lastiger gaan worden. Ik weet namelijk niet hoe je op een detailniveau de structuur van de code beschrijft… zonder gebruik te maken van een diagram.

Zou ik dit bij een ander project weer zo doen? Ja! Het functioneel ontwerp helpt bij het duidelijk maken van de werking van een oplossing. Het vaststellen van een aantal doelen (of vragen) zorgt ervoor dat een proof of concept niet te groot wordt. Voor het communiceren van de structuur van de code weet ik niet wat mijn alternatieven zijn. Het enige wat ik dan kan doen is alsnog het diagram maken en dit vervolgens mondeling uit te leggen.

Implementeren van de Batcher

Ik heb geen gebruik gemaakt van een systeemontwikkelingsmethode, omdat de voordelen van het gebruik van een systeemontwikkelingsmethode niet van toepassing waren bij dit project. Het niet gebruiken van een systeemontwikkelingsmethode betekent echter niet dat ik ongestructureerd aan de slag gegaan ben.

Mijn aanpak bestaat uit het aanpassen van het technische ontwerp. Vervolgens implementeer ik deze aanpassingen in de code. Als ik bij het implementeren van de aanpassingen problemen tegenkom, pas ik het ontwerp weer aan. Op deze manier verifieer ik mijn ontwerp.

Dhr. Van der Storm is niet in staat om mijn klassendiagrammen van het technische ontwerp te begrijpen. Met uitleg is het mogelijk om de betekenis over te dragen van de querystructuur, maar als ik een aanpassing presenteer moet ik alle aspecten daarvan uitleggen voordat dhr. Van der Storm dit begrijpt. Ik heb hierdoor niemand die mijn ontwerp kan valideren en dat maakt werken moeilijk. Er is namelijk geen controle mogelijk, dus ik weet niet of dat wat ik aan het doen ben goed is. Het enige bewijs waaruit blijkt dat ik goed bezig ben is een performanceverbetering bij de rapportagegeneratie. Goede resultaten spreken namelijk voor zichzelf.

Voor het controleren van de code gebruik ik Sonar. Ik heb Sonar al eerder gebruikt bij mijn stageopdracht. Sonar geeft aan waar ik mogelijk een fout heb gemaakt in de code. Zo heb ik tenminste nog een vorm van kwaliteitscontrole voor de code. Dit geeft me vertrouwen in de code die ik schrijf.

Zou ik dit bij een ander project weer zo doen? Dat ligt aan de grootte van dat project. Het gebruik van een systeemontwikkelingsmethode is bijna standaard. De enige reden dat ik het bij dit project niet heb toegepast is omdat ik voor een individueel project niet zo goed inzie wat de voordelen zijn. Voor het valideren van het ontwerp zou het mogelijk kunnen zijn om mensen in te huren. Op school heb ik dit ook gedaan – je gaat langs bij een ander groepje en laat daar je ontwerp zien. Er is echter een verschil tussen de situatie op school en de situatie in het bedrijfsleven. Op school doen we vaak dezelfde opdracht. Hierdoor is er domeinkennis aanwezig bij alle studenten. In het bedrijfsleven is deze domeinkennis niet aanwezig, omdat de mensen die je inschakelt niet met hetzelfde probleem bezig zijn.

Oplossen van het probleem

De laatste stap in het oplossen van het probleem is het implementeren van de batchingsalgoritmes en het vervolgens testen van de performance.

Gedurende mijn stage bij 42 heb ik gemerkt dat mijn motivatie om te werken enorm wordt vermindert als het eindproduct niet nuttig lijkt te zijn. Ik heb destijds gesteld dat wanneer ik geconfronteerd wordt met een situatie waarin iets af moet wat men echt nodig heeft, dit een enorme verbetering zou zijn voor mijn motivatie.

Na het schrijven van de batchingsalgoritmes kan ik zeggen dat deze stelling waar is. Toen het me niet lukte om voor vrijdag 6 uur het syntactische batchingsalgoritme te implementeren heb ik gevraagd of ik mocht overwerken. Toen het syntactische batchingsalgoritme nog steeds niet werkte om 9 uur ‘s avonds, ben ik naar huis gegaan. In het weekend heb ik geprobeerd in te loggen op mijn werkcomputer via een VPN-verbinding. Dit is niet gelukt. Ik was destijds van plan om in dat weekend te beginnen aan mijn afstudeerverslag. Dit lukte mij niet, omdat ik in mijn hoofd druk bezig was met het bedenken waarom het syntactische batchingsalgoritme niet werkte. De maandag daarop had ik binnen een half uur het probleem opgelost.

Eenzelfde situatie gebeurde bij het semantische batchingsalgoritme. Bij het semantische batchingsalgoritme was het echter zo dat deze wel werkte om 9 uur ‘s avonds. Hierdoor was ik wel in staat om de volgende dag normaal te functioneren.

Aan de ene kant vind ik het belangrijk dat je geeft om het product wat je maakt. Men zegt wel eens dat een gerecht zo lekker smaakt omdat het “met liefde” is klaargemaakt. De chef geeft om de gerechten die hij klaarmaakt. Dat is de reden dat deze gerechten van hoge kwaliteit zijn. Door te geven om het product wat je maakt zorg je ervoor dat de kwaliteit van het product goed is.

Aan de andere kant klopt er iets niet wanneer ik niet instaat ben om “normaal” een weekend te hebben, puur omdat mijn code niet werkt. Ik ben niet in staat om dit los te laten. Mijn code dient te werken – waarom werkt het niet?

Ik ben bang dat wanneer ik mezelf in het weekend losmaak van mijn werk, dit ook gebeurt op het werk zelf. Ik weet dus niet hoe ik hiermee om moet gaan – of hoe anderen hiermee omgaan.

Door meer samen te werken met anderen in projecten waarbij het eindresultaat er echt toedoet denk ik dat ik kan leren hoe ik hiermee om moet gaan. Totdat ik weet hoe ik hiermee moet omgaan kies ik ervoor om het werk belangrijker te laten zijn dan dat het misschien is. Zo blijft het uiteindelijke product tenminste iets waar ik trots op kan zijn.

Dus zou ik dit bij een ander project weer zo doen? Ja.

Het proces als geheel

Ik heb gemerkt dat mijn productiviteit nog steeds schommelt aan de hand van hoe druk ik het heb. Dit is waarschijnlijk het resultaat van zelfstandig werken – er is niemand waar ik direct verantwoording moet afleggen over wat ik heb gedaan in een dag. Langdurig zelfstandig werken zal voor mij altijd een probleem zijn.

Projecten verschillen teveel om hier de vraag “Zou ik dit bij een ander project weer zo doen” te beantwoorden. Wel kan ik zeggen dat ik geleerd heb dat het stellen van doelen voor een proof of concept helpt bij het waarborgen van de scope van het proof of concept. Ook is ongestructureerd werken een krachtige manier van werken die gewoon te gebruiken valt – mits je daarna verantwoordt wat je precies hebt gedaan.

7: Beroepscompetenties

In dit hoofdstuk beschrijf ik welke beroepscompetenties ik tijdens mijn afstudeerstage heb vervuld. De getallen die in de haakjes staan komen overeen met de getallen in de PDF van de beroepscompetenties.[8]

Ontwerpen, bouwen en bevragen van een database (2.2)

Voor deze beroepscompetentie heb ik mij bezig gehouden met het bevragen van een database.

MrIFunctions is verantwoordelijk voor het ophalen van benodigde gegevens voor het genereren van rapportages. Dit doet MrIFunctions door het procedureel genereren van queries, die vervolgens uitgevoerd worden op een database. Gedurende de afstudeeropdracht heb ik MrIFunctions bestudeerd. Onderdeel van het bestuderen van MrIFunctions was het bestuderen van de queries die MrIFunctions. Uit bestudering van deze queries dacht ik dat het mogelijk was om deze te combineren.

Gedurende de afstudeeropdracht is de Batcher ontwikkeld. De Batcher genereert en combineert queries aan de hand van de gegevens die een rapportageapplicatie aanlevert via MrIFunctions. Voor het combineren van queries wordt een querystructuur opgesteld. De querystructuren worden geanalyseerd en waar mogelijk samengevoegd. Deze samenvoeging wordt gedaan door een kopie van een querystructuur te maken en hier aanpassingen in aan te brengen. Vervolgens worden de querystructuren geconverteerd naar SQLqueries. Deze SQLqueries worden uitgevoerd op een database. Het resultaat wat de Batcher ontvangt wordt vervolgens opgesplitst aan de hand van de informatie die tijdens het samenvoegen is opgeslagen.

Voor het ontwikkelen van de Batcher was een uitgebreide kennis nodig van SQL en het bevragen van een database. Ik vind dat ik deze beroepscompetentie op niveau 3 heb behaald (Zelfstandig/Lastig). Mogelijk is deze beroepscompetentie zelfs op niveau 4 aangetoond (Zelfstandig/Complex). Performance speelde namelijk een rol in dit proces.

Ontwerpen Systeemdeel (3.2)

Voor deze beroepscompetentie heb ik mij bezig gehouden met het ontwerpen van de Batcher.

De Batcher is verantwoordelijk voor het opbouwen en combineren van queries aan de hand van de gegevens die een rapportageapplicatie aanlevert via MrIFunctions. Daarnaast moet de Batcher deze queries uitvoeren op een database en de resultaten terugsturen naar de rapportageapplicatie.

Dit betekent dat het ontwerpen van de Batcher het ontwerpen van een systeemdeel is wat met meerdere systeemdelen te maken heeft. Het ontwerpen zelf is gedaan middels een functioneel ontwerp en meerdere technische ontwerpen. Hierbij is gebruik gemaakt van UML. In het technische ontwerp is gebruik gemaakt van design patterns zoals het Strategy-pattern en het Proxyobject-pattern.

Hiermee vind ik dat ik deze beroepscompetentie op niveau 3 heb behaald (Zelfstandig/Lastig).

[8] Beroepstaken_Informatica_1.1.PDF, te vinden op https://blackboard.hhs.nl/courses/1/BI-Z-StI/content/_678961_1/Beroepstaken_Informatica_1.1.PDF voor studenten en docenten van de HHS.

Bouwen Applicatie (3.3)

Voor deze beroepscompetentie heb ik mij bezig gehouden met het ontwikkelen van de Batcher.

De Batcher dient om de performance te verbeteren voor het genereren van een rapportage bij Blik op Werk. Hiervoor genereert en combineert de Batcher queries aan de hand van de gegevens die de rapportageapplicatie van Blik op Werk aanlevert via MrIFunctions.

De Batcher is geschreven in Java. Voor het implementeren van de Batcher heb ik echter een rapportageapplicatie moeten aanpassen. De rapportageapplicaties zijn geschreven in ColdFusion. Gedurende de afstudeeropdracht heb ik geleerd hoe ik ColdFusion-code moet schrijven. Ik heb genoeg ervaring met ColdFusion opgedaan om systeemdelen te integreren.

Voor versiebeheer is gebruik gemaakt van SVN. Gedurende de afstudeeropdracht zijn er echter geen checkouts geweest van de Batcher.

Door deze handelingen te hebben verricht vind ik dat ik deze beroepscompetentie op niveau 4 heb behaald (Zelfstanding/Complex).

Initiëren en plannen van het testproces en rapporteren daarover (3.4/3.5)

Voor deze beroepscompetentie was het de bedoeling dat ik bij het ontwikkelen van de Batcher unittests zou gebruiken.

Ik heb echter niet uitgebreid gebruik gemaakt van unittests. Bij het ontwikkelen van de Batcher ben ik begonnen met de querystructuur. De querystructuur is niets anders dan een datastructuur in objecten. Het testen hiervan leek mij overbodig.

Testen op een hoger niveau is echter niet mogelijk met unittests. De Batcher is een oplosser, niet een oplossing. Dit betekent dat de oplossing die de Batcher produceert niet vast staat. Testen of de oplossing die de Batcher produceert goed is, wordt gedaan door te kijken of deze oplossing aan een aantal eisen voldoet. Deze eisen zijn dat het resultaat van het uitvoeren van de queries hetzelfde is en dat de gebatchte query een kortere executietijd heeft.

De performance valt echter niet accuraat te meten. Op de schaal van een unittest zou het testen van het batchen waarschijnlijk bestaan uit het samenvoegen van 3 queries. Als deze queries niet erg complex zijn, ligt de executietijd onder een tiende van een seconde. Bij deze tijden kan een enkele verstoring het resultaat zwaar beïnvloeden. Dit zou de betrouwbaarheid van een unittest ondermijnen – wat de unittest nutteloos maakt.

Testen van de opbouw van querystructuren aan de hand van de gegevens die een rapportageapplicatie aanlevert is vrijwel niet mogelijk binnen Java. Een van de delen die een rapportageapplicatie aanlevert is een requirementstructure. Dit is een “coldfusion.runtime.Struct”-object, wat een subklasse is van “java.util.Map”. Binnen Java heb ik niet de mogelijkheid om instanties van ColdFusion objecten aan te maken. Daarmee is het niet mogelijk om dit onderdeel van de Batcher te testen vanuit Java.

Als gevolg van deze problemen heb ik besloten af te zien van het gebruik van unittests om de kwaliteit van de Batcher te waarborgen. Er zijn wel unittests gemaakt voor specifieke situaties, maar

deze leveren niet een significante dekkingsgraad. Ook zijn er op acceptatieniveau tests uitgevoerd. Hiervoor worden met en zonder de Batcher meerdere rapportages gegenereerd. Uit deze tests blijkt dat het gebruik van de Batcher dezelfde rapportage oplevert als bij het genereren van een rapportage zonder de Batcher.

Ik ben van mening dat ik hiermee deze beroepscompetentie niet heb behaald.

8: Conclusie

Mijn afstudeeropdracht bestond uit het verbeteren van de performance van de rapportagegeneratie voor Blik op Werk. Door middel van een probleemanalyse heb ik geconstateerd dat de queries de hoofdoorzaak zijn van de huidige performance.

Op basis van een architectuuranalyse en requirementsanalyse heb ik een functioneel ontwerp gemaakt. Met behulp van een proof of concept heb ik een eerste opzet gemaakt van het technisch ontwerp van de oplossing. Ik heb besloten de oplossing Batcher te noemen, omdat de oplossing queries gaat batchen (samenvoegen).

Voor de Batcher zijn twee batchingsalgoritmes ontworpen. Door toepassing van de Batcher is de uiteindelijke performance van de rapportagegeneratie verbeterd van 38 naar 21 seconden. Voor het verder verbeteren van de performance is een adviesrapport gemaakt.

Ik vind dit resultaat goed, maar ik had meer willen halen. Het verschil in de executietijd tussen de productieomgeving en de testomgeving had ik niet verwacht. Ondanks dat ik niet geheel tevreden ben met het resultaat vind ik dat ik juist gehandeld heb.

Het blijft echter moeilijk voor mij om de juiste balans tussen werk en vrije tijd te vinden. Wanneer iets niet lukt vind ik het lastig om dit te laten rusten tot de volgende dag. Dit gevoel wordt versterkt door het idee dat niemand mijn diagrammen begrijpt. Ik hoop dat wanneer ik kan samenwerken met mensen het makkelijk wordt om dit verantwoordingsgevoel te delen en misschien los te laten in mijn vrije tijd.

Met deze afstudeerstage kan ik eindelijk twee vragen beantwoorden die al een hele tijd aan mij gesteld worden.

“Vind je dit nou leuk?” – Ja. Ik vind dit leuk werk. Niet het programmeren, niet het ontwerpen, niet het testen, niet het debuggen, niet de puzzeltjes, maar het geheel. Van begin tot eind – “U heeft een probleem – en ik de oplossing.”. Ik vind het leuk om dat te kunnen zeggen.

En een vraag die mijn moeder mij vanaf het begin van mijn opleiding al stelt:
“Kan iedereen dit nou, wat jij nu doet?” – Nee. Keer op keer heb ik gezien dat een medestudent moeite heeft met iets wat ik makkelijk vind. Bij mijn stage bij 42 was mijn opvolger niet in staat om mijn werk voort te zetten. Wanneer ik nu hetzelfde soort werk doe op een niveau wat ik lastig vind, terwijl ik dit normaliter makkelijk vind, kan ik zeggen dat nee, niet iedereen kan dit. Zelfs menig afstudeerder zou hier moeite mee hebben.

Gedurende de afstudeerstage heb ik kunnen zien hoe de kennis die ik geleerd heb tijdens mijn opleiding past in het bedrijfsleven.

Veel van de kennis die ik geleerd heb is niet toepasbaar. Dit betekent echter niet dat het volgen van de opleiding een verspilling van tijd is geweest. Door de opleiding te volgen heb ik mijn gereedschapskist uitgebreid. Met een uitgebreidere gereedschapskist kan ik beter beslissen welk middel geschikt is voor ieder probleem waar ik mee te maken ga krijgen.

Bijlagen

In dit hoofdstuk beschrijf ik de bijlagen die in dit afstudeerverslag zijn opgenomen. De bijlagen worden beschreven in de volgorde waarin ze bij dit afstudeerverslag zitten.

Termenlijst

De termenlijst dient om uitleg te geven over de termen die binnen dit verslag worden gebruikt.

Personenlijst

De personenlijst dient om uitleg te geven over de personen die binnen dit verslag worden vernoemd.

Tekstvak 1: Queries

Onderschrift: Twee queries uit de volledige lijst van 268 queries. Deze twee queries lijken verdacht veel op elkaar.

De volledige lijst queries is niet beschikbaar binnen het afstudeerdossier. Dit omdat deze lijst 155 pagina's groot is.

Figuur 1: Organogram van Panteia

Dit organogram is gebruikt in het hoofdstuk "Over Panteia". Mogelijk is dit organogram niet goed leesbaar wanneer dit zwart-wit wordt afgedrukt. Om deze reden is het organogram in de bijlagen opgenomen.

Figuur 2: Deploymentdiagram oude situatie

Dit deploymentdiagram is gebruikt in het hoofdstuk "Architectuuranalyse". Mogelijk is de tekst in het diagram niet goed leesbaar. Om deze reden is dit deploymentdiagram in de bijlagen opgenomen.

Figuur 4: Deploymentdiagram nieuwe situatie

Dit deploymentdiagram is gebruikt in het hoofdstuk "Functioneel Ontwerp". Mogelijk is de tekst in het diagram niet goed leesbaar. Om deze reden is dit deploymentdiagram in de bijlagen opgenomen.

Figuur 7: Klassendiagram

Dit klassendiagram is gebruikt in het hoofdstuk "Querystructuur". Vanwege het formaat van het diagram is dit in de bijlagen opgenomen.

Figuur 8: Klassendiagram

Dit klassendiagram is gebruikt in het hoofdstuk "Splitsen verantwoordelijkheden naar nieuwe klassen". Vanwege het formaat van het diagram is dit in de bijlagen opgenomen.

Figuur 9: Klassendiagram

Dit klassendiagram is gebruikt in het hoofdstuk "Implementatie Querystructuur". Vanwege het formaat van het diagram is dit in de bijlagen opgenomen.

Figuur 10: Klassendiagram

Dit klassendiagram is gebruikt in het hoofdstuk "Implementatie Batcher". Vanwege het formaat van het diagram is dit in de bijlagen opgenomen.

Termenlijst

Term	Betekenis	Type
Activity Diagram	Een UML diagram waarin een proces via activiteiten beschreven wordt.	UML
Batcher	De naam die aan de oplossing van het performanceprobleem gegeven is. De Batcher combineert queries, zodat deze minder lang duren om uit te voeren op de database.	Project
Blik op Werk	Een project van Panteia. Blik op Werk is een stichting die zich bezig houdt met de kwaliteit van diensten om mensen aan het werk te krijgen. Bij Blik op Werk kan een bedrijf een klanttevredenheidsonderzoek aanvragen.	Project
ColdFusion	Een programmeertaal waar Panteia uitsluitend mee werkt. Is gebaseerd op JAVA. http://nl.wikipedia.org/wiki/Adobe_ColdFusion	Programmeertaal
Deploymentdiagram	Een UML diagram waarin een architectuur beschreven wordt. Dit wordt gedaan door de fysieke systemen, de besturende software en de gestuurde software te modelleren. Ook staan hierin de relaties tussen de systemen gemodelleerd.	UML
IBM Dimensions	Onderdeel van IBM SPSS. Dit is een softwarepakket wat, aan de hand van een vragenlijst automatisch een website en database inricht. http://www.forgetdata.com/services/spssdc.html	Softwarepakket
Index Scan	Een taak die een database moet verrichten om gegevens voor een query op te halen. Bij een index scan wordt een index (“inhoudsopgave”) gescant voor bepaalde waardes. Het gebruik van een index scheelt in executietijd van een query, omdat niet alle data gelezen hoeft te worden.	SQL
JAVA	Een objectgeoriënteerde programmeertaal. Het grootste voordeel van JAVA is dat het cross-platform is – vrijwel alles draait JAVA.	Programmeertaal
JOIN	Een SQL statement om twee tabellen aan elkaar te koppelen.	SQL
Proxy-object	Een design pattern. Een proxy-object is een object wat zich voordoet als het “echte” object. Wanneer het proxy-object iets gevraagd wordt, tracht het proxy-object deze vraag direct te beantwoorden. Lukt dit niet, wordt de vraag doorgestuurd aan het echte object. Op deze manier is het mogelijk om een verwijzing te maken naar een object wat nog niet bestaat.	Design Pattern
Query Execution Plan	Een Query Execution Plan wordt door een database opgesteld wanneer een query uitgevoerd moet worden. Dit plan beschrijft hoe de database het ophalen van de gegevens gaat aanpakken. Door dit plan te bestuderen is het mogelijk om queryoptimalizatie toe te passen.	SQL
Sonar	Een tool die JAVA-bytecode analyseert. Aan de hand van deze analyse worden rapporten gegenereerd over mogelijke (stijl-)fouten en de mate waarin de code getest is. Deze tool is bruikbaar als een vorm van objectieve controle.	Tool
Stakeholders	Personen die “een aandeel hebben” in een systeem. Dit kunnen bijvoorbeeld gebruikers of ontwikkelaars zijn.	Requirements

Personenlijst

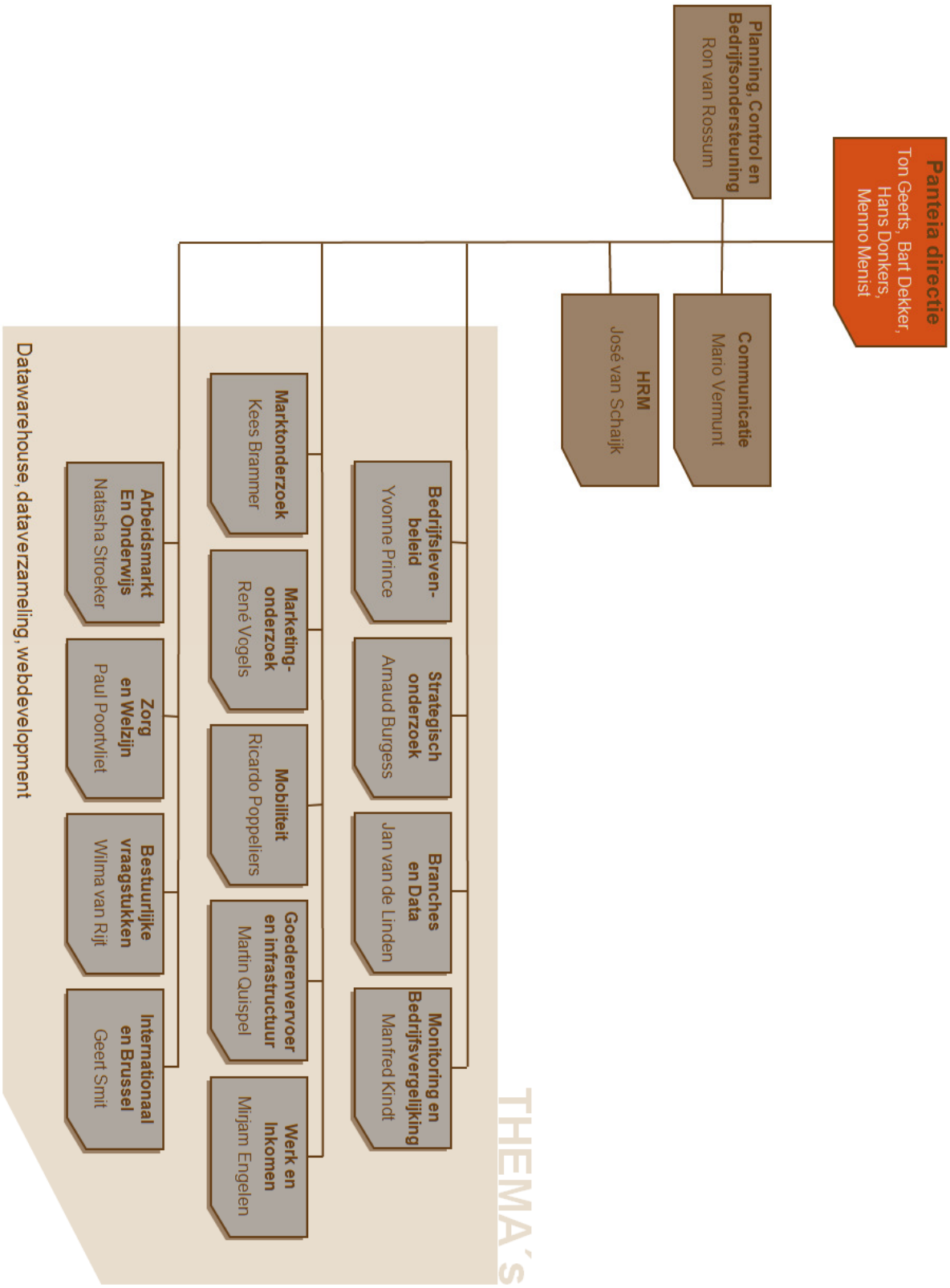
Naam	Rol	Beschrijving
Milan van der Storm	Stagebegeleider, ColdFusion ontwikkelaar	Heeft MrIFunctions ontwikkeld. Is mijn stagebegeleider.
Marvin van Velzen	ColdFusion ontwikkelaar	Is kamergenoot van Milan van der Storm. Levert opmerkingen over conversaties tussen mij en Milan.
Manfred Kindt	Opdrachtgever	Is themamanager. Geeft aan zelf weinig technische kennis over het huidige systeem te hebben.
Guus Leer	Databaseadmin	De databaseadministrator voor de afdeling Monitoring, Bedrijfsvergelijking en Webdevelopment.

Overige bijlagen (Figuren en tekstvakken)

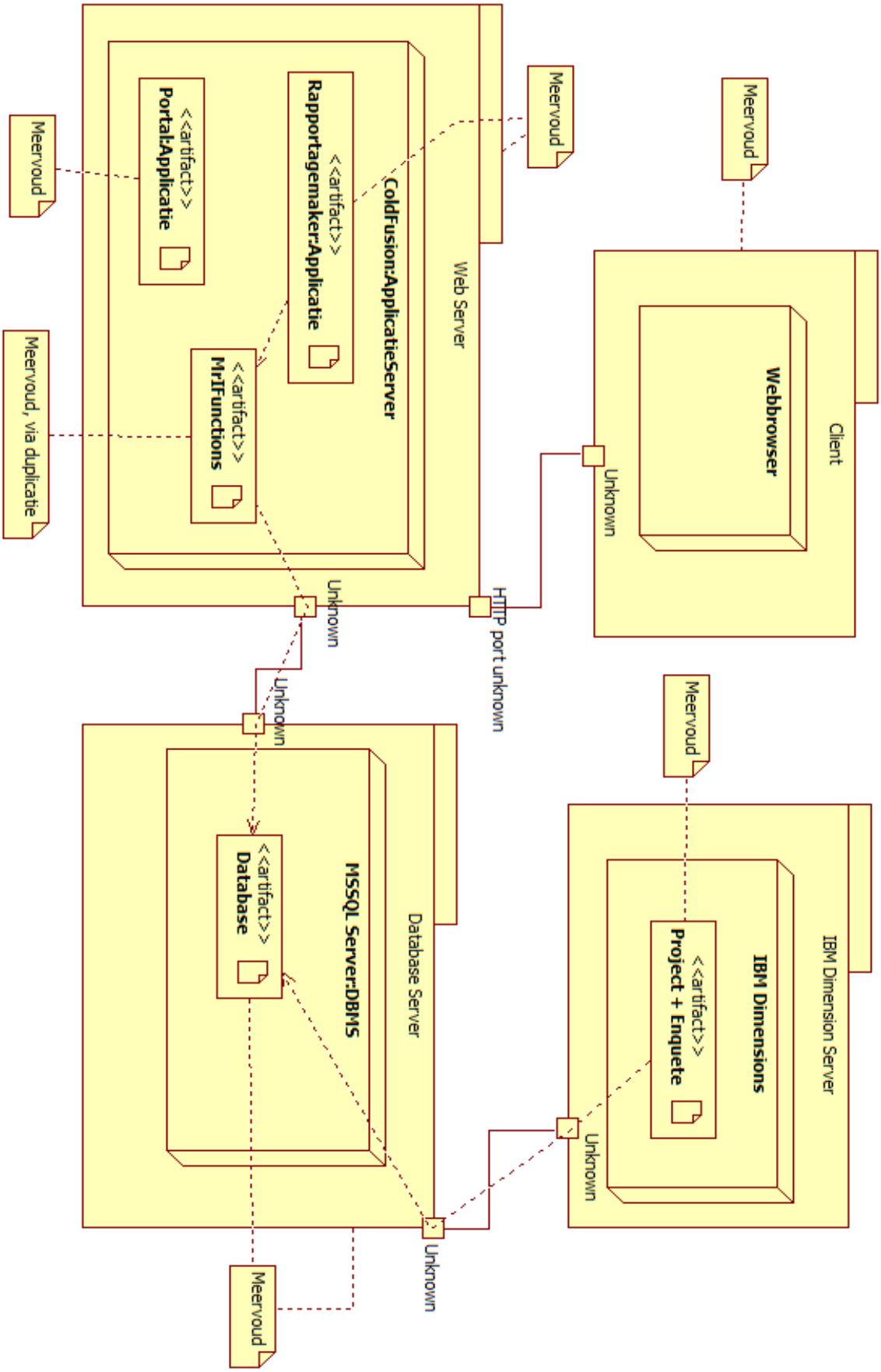
```
SELECT COUNT(DISTINCT Main.serial) as response --[V15{S12}.G]
FROM   [MO10002LO].dbo.otherData2 AS Main
       INNER JOIN [MO10002LO].dbo.Responses2 AS Status
               ON Status.Serial = Main.Serial
                  AND Status.VariableId = 6
                  AND ( Status.Response = 6
                         OR Status.Response IN ( 9, 10 )
                         OR Status.Response = 8 )
       INNER JOIN [MO10002LO].dbo.otherData2 AS AV_Datum
               ON AV_Datum.Serial = Main.Serial
                  AND AV_Datum.VariableId = 33
                  AND ( AV_Datum.dateVal < '2013-02-11 12:09:37.037' )
       INNER JOIN [MO10002LO].dbo.responses2 AS AV_opdrachtgeverType
               ON AV_opdrachtgeverType.Serial = Main.Serial
                  AND AV_opdrachtgeverType.VariableId = 37
                  AND ( AV_opdrachtgeverType.response <> 94 )
       INNER JOIN [MO10002LO].dbo.responses2 AS AV_Valid
               ON AV_Valid.Serial = Main.Serial
                  AND AV_Valid.VariableId = 36
                  AND ( AV_Valid.response = 1 )
       LEFT OUTER JOIN [MO10002LO].dbo.Responses2 AS Test
                 ON Test.Serial = Main.Serial
                    AND Test.VariableId = 6
                    AND Test.Response = 14
WHERE  Main.VariableID = 102
       AND Test.Response IS NULL;

SELECT COUNT(DISTINCT Main.serial) as response --[V15{S13}.G]
FROM   [MO10002LO].dbo.otherData2 AS Main
       INNER JOIN [MO10002LO].dbo.Responses2 AS Status
               ON Status.Serial = Main.Serial
                  AND Status.VariableId = 6
                  AND ( Status.Response = 6
                         OR Status.Response IN ( 9, 10 )
                         OR Status.Response = 8 )
       INNER JOIN [MO10002LO].dbo.otherData2 AS AV_Datum
               ON AV_Datum.Serial = Main.Serial
                  AND AV_Datum.VariableId = 33
                  AND ( AV_Datum.dateVal < '2013-02-11 12:09:37.037' )
       INNER JOIN [MO10002LO].dbo.responses2 AS AV_opdrachtgeverType
               ON AV_opdrachtgeverType.Serial = Main.Serial
                  AND AV_opdrachtgeverType.VariableId = 37
                  AND ( AV_opdrachtgeverType.response <> 94 )
       INNER JOIN [MO10002LO].dbo.responses2 AS AV_Valid
               ON AV_Valid.Serial = Main.Serial
                  AND AV_Valid.VariableId = 36
                  AND ( AV_Valid.response = 1 )
       LEFT OUTER JOIN [MO10002LO].dbo.Responses2 AS Test
                 ON Test.Serial = Main.Serial
                    AND Test.VariableId = 6
                    AND Test.Response = 14
WHERE  Main.VariableID = 103
       AND Test.Response IS NULL;
```

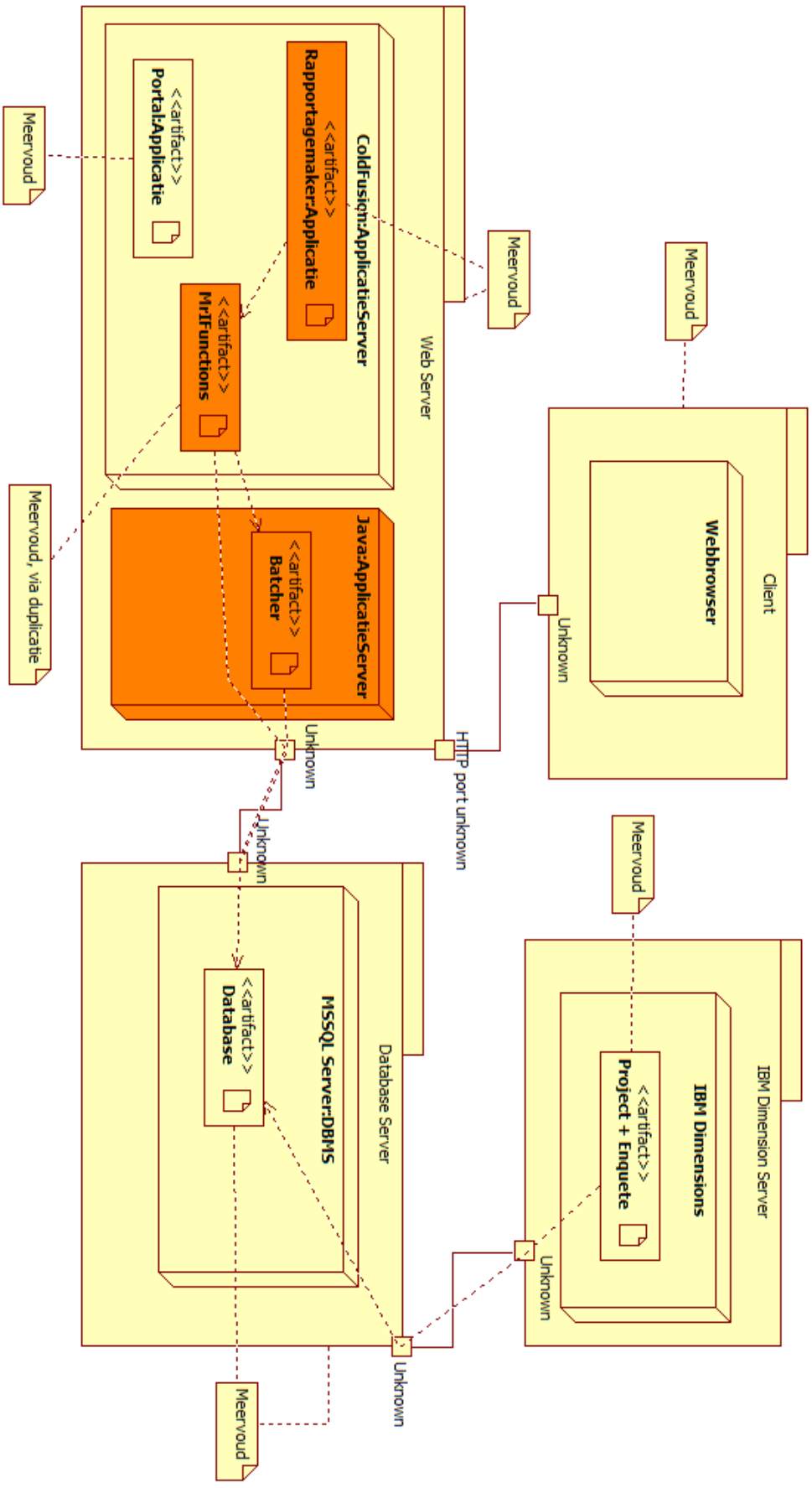
Tekstvak 1: Twee queries uit de volledige lijst van 268 queries. Deze twee queries lijken verdacht veel op elkaar.



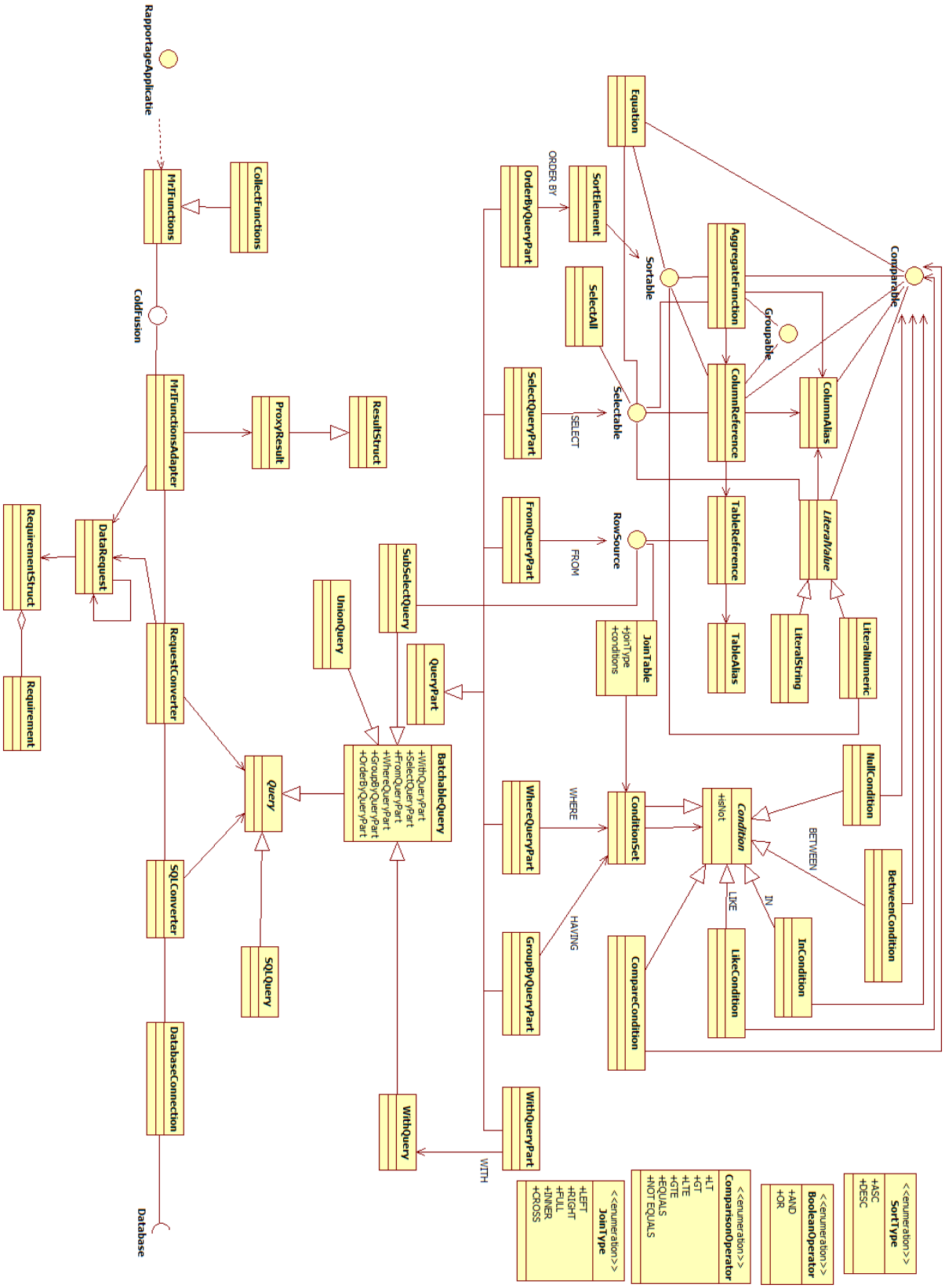
Figuur 1: Het organogram van Panteia.



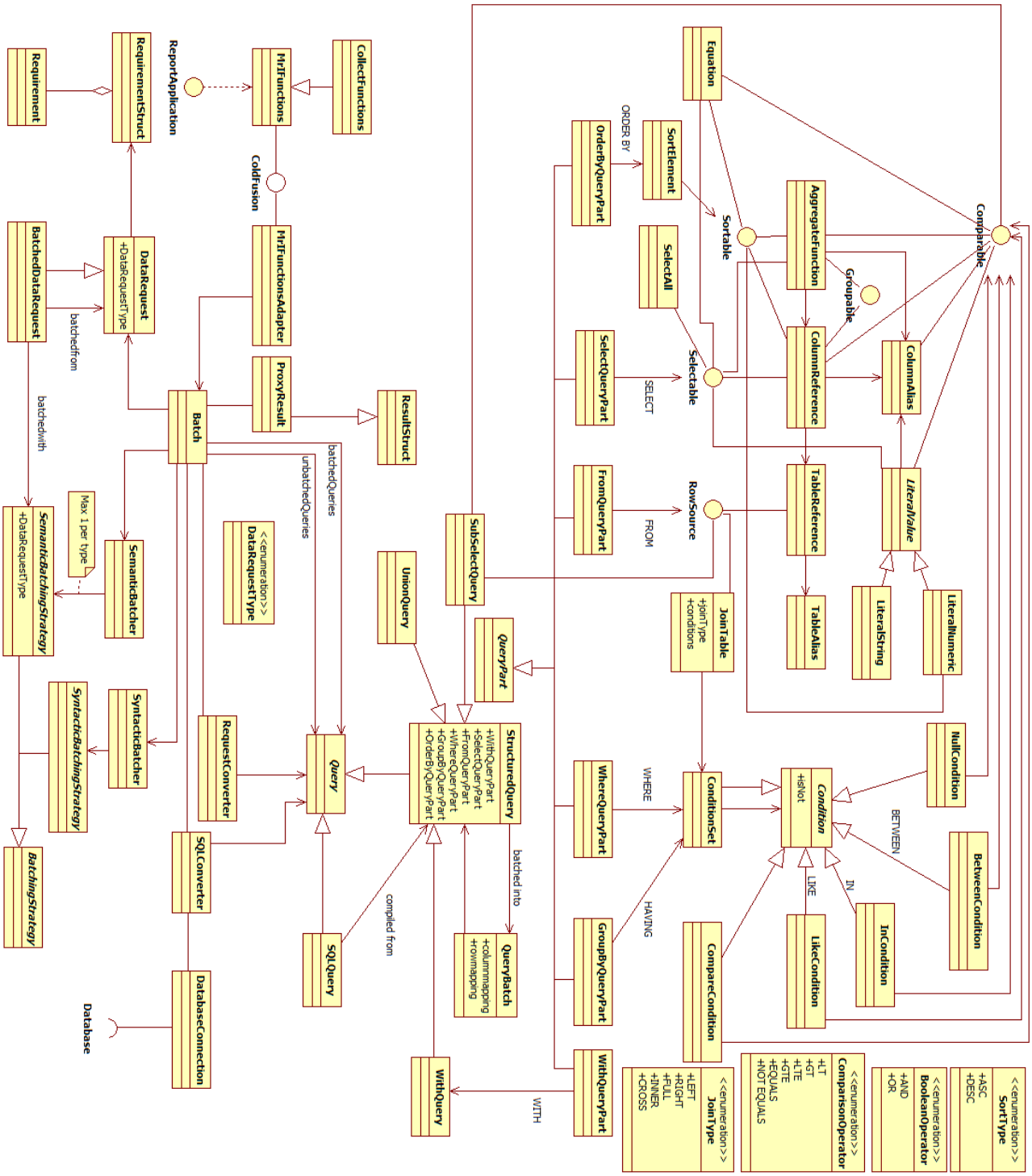
Figuur 2. Het deploymentdiagram van de oude situatie.



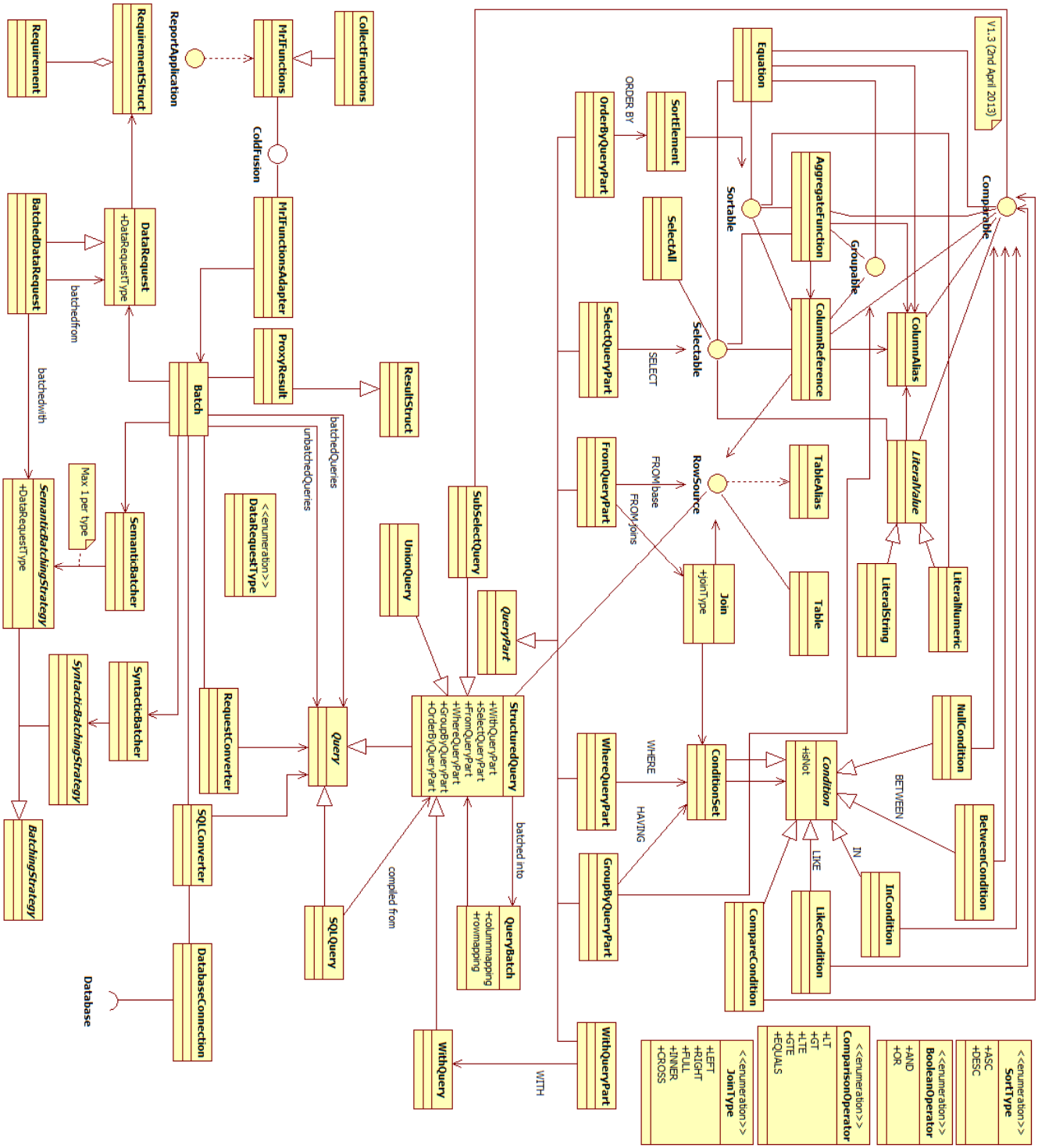
Figuur 4. Het Deploymentdiagram van de nieuwe situatie.



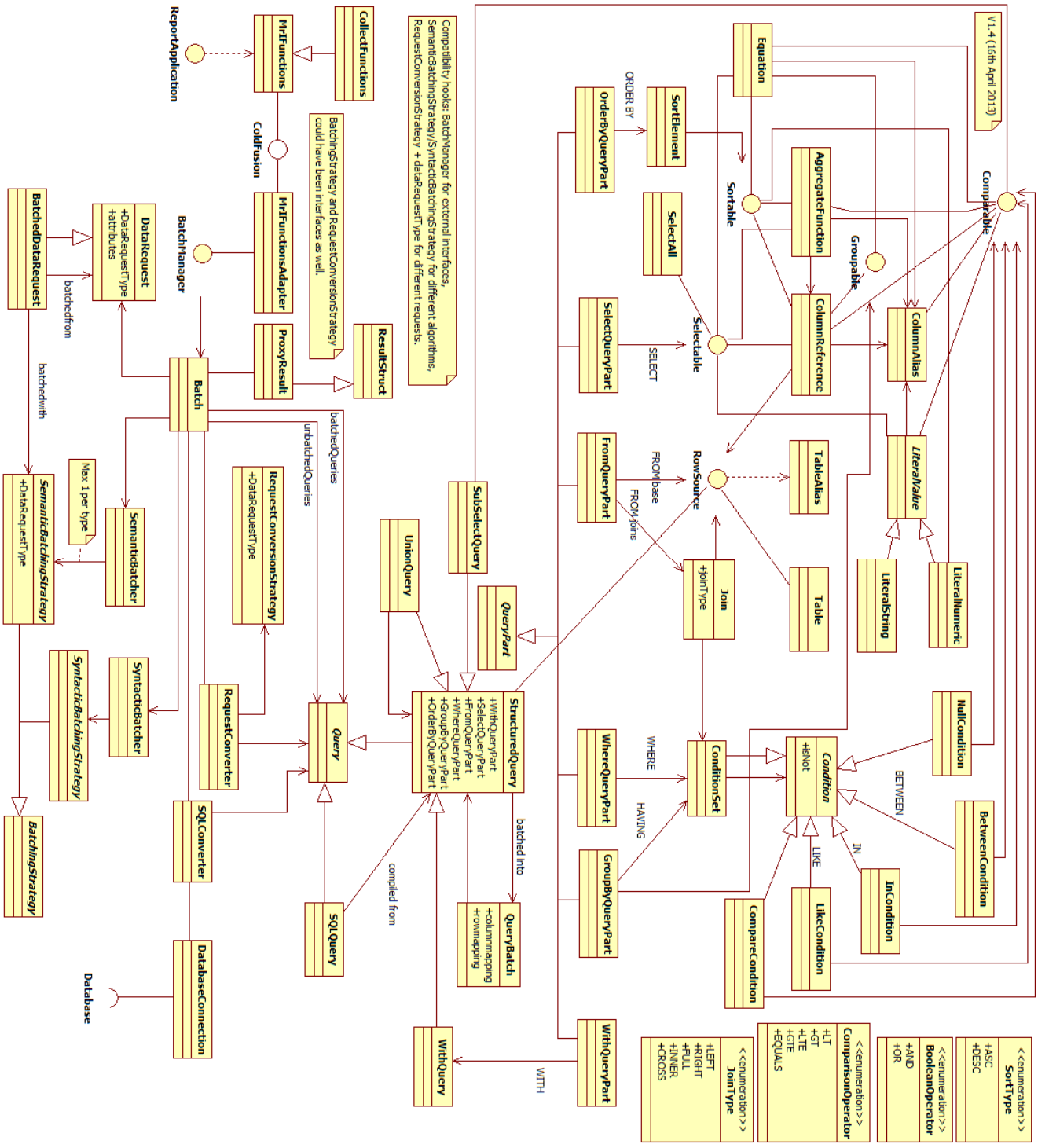
Figuur 7. Het Technisch Ontwerp, na uitbreiding querystructuur. Voor een grotere versie zie Main.png in de map Technisch Ontwerp van het afstudeerdossier.



Figuur 8. Het technisch ontwerp, na het splitsen van de dubbele verantwoordelijkheden. Voor een grotere versie, zie Main2.png in de map Technisch Ontwerp van het afstudeerdossier.



Figuur 9. Het Technisch Ontwerp, na verbetering querystructuur. Voor een grotere versie zie Main3.png in de map Technisch Ontwerp van het afstudeerdossier.



Figuur 10. Het Technisch Ontwerp, nadat de verbeteringen zijn toegepast die uit het implementeren van de Batcher zijn voortgekomen. Voor een grotere versie zie Main4.png in de map Technisch Ontwerp van het afstudeerdossier.

Afstudeerplan

Informatie afstudeerder en gastbedrijf (*structuur niet wijzigen*)

Afstudeerblok: 2013-1.1 (start uiterlijk 11 februari 2013)
Startdatum uitvoering afstudeeropdracht: 11 februari 2013
Inleverdatum afstudeerdossier volgens jaarrooster: 7 juni 2013

Studentnummer: 09033580
Achternaam: (dhr) van Dongen
Voorletters: P.J.
Roepnaam: Pim
Adres: van Rijswijckschans 58
Postcode: 2728 HG
Woonplaats: Zoetermeer
Telefoonnummer: 079-3319031
Mobiel nummer: 06-12386679
Privé emailadres: Pimgd@hotmail.com

Opleiding: Informatica
Locatie: Zoetermeer
Variant: voltijd

Naam studieloopbaanbegeleider: Arno Nederend
Naam begeleidend examinator: Tim Cocx
Naam tweede examinator: Nanny Jacobs

Naam bedrijf: Panteia
Afdeling bedrijf: Thema Monitoring & Bedrijfsvergelijking
Bezoekadres bedrijf: Bredewater 26
Postcode bezoekadres: 2715 CA Zoetermeer
Postbusnummer: 7001
Postcode postbusnummer: 2701 AA
Plaats: Zoetermeer
Telefoon bedrijf: 079-3222000
Telefax bedrijf: 079-3222101
Internetsite bedrijf: www.panteia.nl

Achternaam opdrachtgever: dhr. M.R.J. Kindt
Voorletters opdrachtgever:
Titulatuur opdrachtgever: ir.
Functie opdrachtgever: themamanager
Doorkiesnummer opdrachtgever: 079-3222415
Email opdrachtgever: m.kindt@panteia.nl

Achternaam bedrijfsmentor: dhr M.S. van der Storm
Voorletters bedrijfsmentor:
Titulatuur bedrijfsmentor: ing.
Functie bedrijfsmentor: webdeveloper
Doorkiesnummer bedrijfsmentor: 079-3222689
Email bedrijfsmentor: m.van.der.storm@panteia.nl
NB: bedrijfsmentor mag dezelfde zijn als de opdrachtgever

Doorkiesnummer afstudeerder:
Functie afstudeerder (deeltijd/duaal):

Titel afstudeeropdracht:

Ontwikkelen nieuwe database-interface in JAVA voor webapplicaties in ColdFusion bij Panteia.

Opdrachtomschrijving *(toelichtende tekst verwijderen)*

1. Bedrijf

Panteia:
Panteia is een allround onderzoeksbureau voor economisch en sociaal beleidsonderzoek, transportonderzoek en marktonderzoek. Ons werk omvat de volledige breedte van de maatschappij. Met meer dan 250 medewerkers is Panteia het grootste onderzoeksbureau van Nederland.

Thema “Monitoring, Bedrijfsvergelijking & webdevelopment”
Dit team bestaat uit 14 personen van wie de helft zich bezig houdt met automatisering. Het onderdeel webdevelopment is themaoverschrijdend en wordt ingezet bij projecten binnen geheel Panteia.

Projecten waar het webteam zich voornamelijk mee bezig houdt zijn online monitoringsportals (omzet/afzet van diverse branches) en online klant- of medewerkertevredenheidsystemen.

2. Probleemstelling

Panteia maakt gebruik van het softwarepakket IBM Dimensions, dit pakket verzorgt de gehele handling en dataopslag van online enquêtes. Dataopslag gebeurt op een onconventionele wijze in een MSSQL Database. De beschrijving van de metadata staat in een automatisch gegenereerd XML-bestand. Om de communicatie tussen enquetesoftware en online rapportage goed te laten verlopen is er enkele jaren geleden in Coldfusion een interface ontwikkeld. Gebleken is dat bij grote hoeveelheden data de performance van deze interface niet optimaal is.

3. Doelstelling van de afstudeeropdracht

Het doel van de opdracht voor het bedrijf is om de performance van de interface te verbeteren en mogelijke andere wensen en eisen (of wel knelpunten) aan te pakken.
Voor de afstudeerder is het een mogelijkheid om aan te tonen dat de vaardigheden die hij op de opleiding heeft verkregen bruikbaar zijn in het bedrijfsleven.

4. Resultaat

Wanneer de opdracht klaar is heeft is de Coldfusion interface vervangen door een Java interface. Deze java interface is aanzienlijk sneller en adresseert andere problemen die het bedrijf over de jaren heeft ondervonden. Een tweede doel is om het werk makkelijker te maken voor diegene die met de interface moeten werken zodat er meer tijd in het maken van het product beschikbaar is in plaats van de ondersteuning daarvan.

5. Uit te voeren werkzaamheden, inclusief een globale fasering, mijlpalen en bijbehorende activiteiten

- Plan van Aanpak en oriënteren omgeving (eerste week)
- Inventariseren en doorgronden backend IBM Dimensions (2 weken)
- Analyseren knelpunten van bestaande interface (2 weken)
- Functioneel ontwerp nieuwe interface (2 weken)
- Bouwen nieuwe interface en testen gedurende het bouwen (5-7 weken)
- Documenteren interface (1-2 weken)
- Eindpresentatie richting bedrijf voor gebruik nieuwe interface (in de 17de week)

6. Op te leveren (tussen)producten

- Knelpuntenanalyse
- Functioneel ontwerp
- Aanpak ontwikkelfase (definition of done, testplan)
- Interface
- Javadocs (in geval van Java)

7. Te demonstreren competenties en wijze waarop

2.2 Ontwerpen, bouwen en bevragen van een database.
Deze beroepstaak wordt uitgevoerd op niveau 3. Er is sprake van het procedureel genereren van bevragingen en dit zit een niveau boven het eenmalig oplossen van een lastig vraagstuk. (Zelfstandig/Lastig)

3.2 Ontwerpen systeemdeel
Deze beroepstaak wordt uitgevoerd op niveau 3. Aan de hand van de gestelde wensen en eisen moet een nieuw systeemdeel ontworpen worden. Dit systeemdeel moet gaan samenwerken met meerdere andere systeemdelen. (Zelfstandig/Lastig)

3.3 Bouwen systeemdeel
Deze beroepstaak wordt uitgevoerd op niveau 4. Het werk bestaat uit het bouwen van een systeemdeel die de brug maakt tussen twee systemen met geheel verschillende interne opbouw hebben. (Zelfstandig/Complex)

3.4/3.5 Initieren en Plannen van het testproces en het rapporteren daarover
Deze beroepstaak wordt uitgevoerd op niveau 2. Om de kwaliteit van het gemaakte werk te garanderen moet er getest worden. Waarschijnlijk zal het testen bij unittests blijven. (Zelfstandig/Simpel)

Probleemanalyse

Performance Inladen en Weergeven Onderzoeksresultaten

Datum: 12 Februari 2013
Auteur: Pim van Dongen
Versie: 1.0

Inleiding

Dit document gaat over de analyse van het probleem die in de huidige situatie bestaat. Dit document zal in eerste instantie gebruikt worden om te dienen als input voor een Plan van Aanpak. Later is dit document te gebruiken als documentatie waarom bepaalde keuzes gemaakt zijn.

In dit document worden enkele delen van de huidige situatie beschreven, evenals hun interactie onderling. Aan de hand van deze beschrijvingen zijn hypotheses opgesteld. Verder wordt beschreven middels welke tests deze hypotheses zijn getest en wat het resultaat hiervan is. Het laatste onderdeel van dit document bestaat uit de conclusie. In de conclusie wordt aan de hand van de geteste hypotheses een mogelijke oorzaak bepaald van het probleem.

Huidige Situatie

Binnen het Thema Monitoring & Bedrijfsvergelijking binnen Panteia worden klanttevredenheidonderzoeken voor andere thema's uitgevoerd. De resultaten van deze onderzoeken worden via een online applicatie toegankelijk gemaakt.

Wanneer een bedrijf naar haar resultaten wil kijken, kan zij een overzicht opvragen. Het opvragen van dit overzicht gaat momenteel via ColdFusion. Een ColdFusion functie communiceert met een MSSQL database, waarmee (via enkele andere modules) een overzicht wordt gegenereerd. Voor grote overzichten moet men echter erg lang wachten (~30 seconden). Panteia wil dit zien veranderen.

Componenten

In de huidige situatie worden enkele componenten gebruikt. Ieder van deze componenten heeft invloed op het eindresultaat en zou hierdoor de performance van het geheel kunnen beïnvloeden.

MicroSoft SQL Server
Ook wel MSSQL genoemd. Dit is de databaseserver, waar de verscheidende databases die in gebruik zijn door Panteia op draaien. Op deze databaseserver staan ook de databases die gebruikt worden voor de klanttevredenheidsonderzoeken.

ColdFusion
ColdFusion is de applicatieserver. De ontwikkelaars ontwikkelen de klanttevredenheidsonderzoeken met behulp van CFML en CFScript. Deze klanttevredenheidsonderzoeken draaien dan als applicaties op de applicatieserver. Ook is er een applicatieonderdeel ontwikkeld in CFML wat communiceert met de databaseserver. Via dit applicatieonderdeel kunnen rapportages over de klanttevredenheidsonderzoeken wordt gegenereerd.

IBM Dimensions
Een softwarepakket. Dit softwarepakket regelt de inrichting van de database voor de klanttevredenheidsonderzoeken. Dit softwarepakket beheert ook de interne werking van de vragenlijst. Het enige wat de ColdFusion ontwikkelaars hoeven te regelen is de vragenlijst als input en de communicatie van de output van het softwarepakket naar de gebruiker.

Hypotheses

Ieder van de componenten die in vorig hoofdstuk zijn besproken hebben invloed op het eindresultaat. Het probleem lijkt echter dusdanig groot te zijn dat er sprake is van een zogenaamde “bottleneck”; Een van de componenten binnen het geheel functioneert niet naar behoren. Het vinden en oplossen van deze bottleneck zou de performance van het geheel aanzienlijk moeten verbeteren.

Daarmee stellen we de volgende hoofdhypothese op:
Alhoewel alle onderdelen een zekere mate van invloed hebben, oefent 1 component veel meer invloed uit op het eindresultaat dan alle andere componenten. Aanpassen van dat ene component lost het probleem op.

De focus ligt hiermee dus op het vinden van 1 enkele boosdoener. Er zal waarschijnlijk op meerdere plaatsen winst te behalen zijn, maar de focus ligt op het vinden en behalen van de grootste winst.

Hypotheses per component

Per component zijn een aantal hypotheses opgesteld. Iedere hypothese bestaat uit een aanname, een beschrijving en een mogelijke oplossing. Het testen van deze hypotheses zal gebeuren door te controleren of ieder van deze onderdelen toepasbaar is op de huidige situatie. Er wordt dus eerst gekeken of de aanname juist is. Daarna wordt er gekeken of de beschrijving van de situatie overeenkomt met de huidige situatie. Als dit zo is, wordt er gekeken of het toepassen van een versimpelde versie van de oplossing een positief effect heeft op het eindresultaat. Als dit alle drie het geval is, weten we waar de grootste winst gehaald kan worden en is het mogelijk om een Plan van Aanpak op te stellen.
Format: Hypothese #: Aanname, beschrijving, oplossing.

MicroSoft SQL Server
Hypothese 1: De database is te traag. Dit is te zien doordat het uitvoeren van meerdere taken op de database te gelijk een enorme vertraging oplevert. De oplossing zou hierbij liggen in het kijken naar de inrichting van de database, zowel op configuratie als op hardwareniveau.
Hypothese 2: De database is verkeerd geconfigureerd. De meeste handelingen dit uitgevoerd worden op de database worden binnen een normale tijd uitgevoerd. Wanneer men echter een handeling onderneemt die niet gezien wordt als standaard, ondervindt de database problemen. Hiervoor zou naar de configuratie van de database gekeken moeten worden.
Hypothese 3: De database kan de hoeveelheid werk niet aan die vanuit het applicatieonderdeel binnenkomt. Dit is te zien doordat andere systemen die intensief gebruikmaken van de databaseserver geen last ondervinden van deze problemen. Daarnaast is dit zichtbaar doordat de database gedurende het genereren druk bezig is, terwijl de applicatie aan het wachten is.

ColdFusion
Hypothese 4: De applicatieserver is te traag. Tijdens normaal gebruik van de applicaties op de applicatieserver zijn er al aanwijzingen van problemen. Wanneer men dan een complex taak uitvoert, kan de applicatieserver de hoeveelheid werk niet goed verwerken. Om dit op te lossen zou er meer hardware ingezet moeten worden. Daarnaast zou er gekeken moeten worden of de applicatieserver

wel de recentste versie is en of nieuwere versies het probleem zouden verhelpen.
Hypothese 5: De applicaties zijn niet goed ontwikkeld en zijn daardoor ruim trager dan zou moeten. Dit is te zien doordat taken die niet met andere componenten van het systeem communiceren ook dezelfde problemen ondervinden. Om dit te verhelpen zouden de applicaties aangepast moeten worden.
Hypothese 6: Het applicatieonderdeel wat met de databaseserver communiceert is inefficiënt. Dit is te zien doordat terwijl het genereren van het rapport bezig is, de database niet tot licht belast is, terwijl de applicatie druk bezig is. Dit is op te lossen door het applicatieonderdeel wat met de databaseserver communiceert te vervangen of aan te passen.

IBM Dimensions
Omdat ik na slechts twee dagen nog geen voldoende kennis heb vergaard over de werking van IBM Dimensions kan ik hier geen hypotheses voor bedenken.

Hypothese 7: De verwachtingen van de opdrachtgever zijn onrealistisch. Alles lijkt in orde te zijn en er is geen sprake van enige opvallende pieken binnen het gehele systeem. Dit betekent dat het gehele systeem aangepast zou moeten worden (opschalen), of dat de opdrachtgever genoegen moet nemen met het bestaan van het probleem.

Uitgevoerde tests en resultaten

In dit hoofdstuk wordt per hypothese besproken welke tests hiervoor zijn uitgevoerd, welke resultaten er waren en welke conclusies hieruit getrokken kunnen worden. Dit hoofdstuk wordt samengevat in het hoofdstuk Conclusie.

Hypothese 1: De database is te traag.
Hiervoor hebben we getest wat er gebeurt als de database meerdere taken tegelijk moet uitvoeren. Het laten genereren van meerdere rapportages (2-3) duurt niet merkbaar langer dan het laten genereren van een enkele rapportage. Dit geeft aan dat de database niet te traag is en dat het probleem elders ligt.

Hypothese 2: De database is verkeerd geconfigureerd.
Hiervoor heb ik gekeken naar de Query Execution Plans die MSSQL Server biedt. De Query Execution Plans toonde een probleem aan. De databaseserver was namelijk voor het grootste gedeelte bezig met het doorzoeken van een enkele index. Daarna was de rest van de query in een keer klaar.
Het toepassen van andere indexen zoals deze aangeraden werden door MSSQL Server zorgden echter niet voor een aanzienlijke verbetering. Wel is de workload beter verdeeld door de query volgens de Execution Plans.

Hypothese 3: De database kan de hoeveelheid werk niet aan die vanuit het applicatieonderdeel binnenkomt.
Hiervoor heb ik getest hoe lang de database erover doet om de SQL queries uit te voeren die het applicatieonderdeel verstuurd bij het genereren van het rapport. Tijdens deze test was de applicatieserver niet actief. De database had maar liefs 30 seconden nodig om alle queries uit te voeren. Dit lijkt het probleem te zijn. Wanneer ik naar de afzonderlijke tijd kijk van iedere query, is deze best redelijk. Onder de 0,2 seconden voor een drie of zelfs vierdubbele join. Er worden echter zoveel queries uitgevoerd dat de database heel lang bezig is.
Deze hypothese lijkt dus een stap in de juiste richting te zijn.

Hypothese 4: De applicatieserver is te traag. Dit lijkt niet het geval te zijn. Wanneer er namelijk een runtimefout aanwezig is in de code van de applicatie, wordt dit heel snel gerapporteerd. Het laten genereren van een complex rapport leidt bovendien tot een timeout op een databaseverbinding in plaats van een timeout op de uitvoering van de code.

Hypothese 5: De applicaties zijn niet goed ontwikkeld en zijn daardoor ruim trager dan zou moeten. Hier heb ik nog geen tests uitgevoerd. Ik heb echter wel gezien dat het genereren van een overzicht van 5 enquêtevragen zo'n anderhalve seconde duurt. Per enquêtevraag zijn twee queries vereist. Ieder van deze queries duurt 0.1 tot 0.2 seconden. Dit laat weinig tijd over voor de applicaties om traag te zijn. Hierdoor denk ik dat dit niet de hoofdoorzaak van het probleem is.

Hypothese 6: Het applicatieonderdeel wat met de databaseserver communiceert is inefficiënt. Ook dit is niet het geval. Wanneer ik logging op het applicatieonderdeel aanbreng is de log veel eerder compleet dan het uiteindelijke resultaat. Zodra de log compleet is, is de applicatie in principe op de

database aan het wachten. Zolang de applicatie ruimschoots klaar is voordat de database dat is, is de interne werking van het applicatieonderdeel niet de hoofdoorzaak van het probleem.

Hypothese 7: De verwachtingen van de opdrachtgever zijn onrealistisch. Ik heb al enkele problemen geconstateerd met het uitvoeren van queries op de database. Dit betekent dat er niet niets aan de hand is en dat er wel degelijk een probleem is.

Aan de hand van deze uitgevoerde tests ben ik verder gaan kijken naar de queries die naar de database gestuurd worden door het applicatieonderdeel. Veel van deze queries lijken op elkaar. Ik vind tussen de lijst queries enkele queries die wel heel erg op hetzelfde lijken en slaag erin om deze te combineren. Het uitvoeren van deze 12 queries op de oude wijze kost 1.2 seconden, terwijl mijn gecombineerde query klaar is in 0.19 seconden. Het valt me op dat er nog meer queries zijn die op hetzelfde lijken. Hierom lijkt het mij dat de hoeveelheid queries die naar de database gestuurd worden door het applicatieonderdeel de hoofdoorzaak van het probleem is.

Conclusie

Ik heb een aantal hypotheses opgesteld. Deze hypotheses betreffen de verscheidende componenten van het systeem.

Ik heb niet de kennis om mogelijke problemen binnen het IBM Dimensions softwarepakket aan te wijzen. De applicaties op de applicatieserver reageren snel. Wanneer we naar de database kijken zien we dat deze standaard door het IBM Dimensions softwarepakket wordt ingericht. Hier zijn enkele mogelijkheden tot verbetering, maar deze leiden slechts tot zeer kleine verbeteringen.

Tijdens het genereren van het rapport is de database echter wel de hele tijd bezig. Wanneer ik dan de andere stappen van het gehele proces oversla en de stap van het proces waarbij de database bevraagd wordt isoleer, blijkt deze stap 30 seconden te duren.

De database handelt queries met een adequate snelheid af. Ook lijkt de database geen problemen te hebben met complexere queries. Toch komt de database niet snel door zijn workload heen.

Wanneer ik de queries voor de database bekijk, valt het mij op dat deze veelal hetzelfde zijn. Wanneer ik erin slaag om enkele van deze queries te combineren, neemt de tijd om de gehele lijst queries uit te voeren af met een seconde. Dit lijkt weinig, maar wanneer ik zie dat er dit gaat om een vermindering van 1 seconde voor het combineren van 12 queries uit een lijst van 268 queries, denk ik dat dit de hoofdoorzaak van het probleem is.

Plan van Aanpak

Toepassen Batching op Queries voor PDF-Rapportages

Datum: 13 Februari 2013
Auteur: Pim van Dongen
Versie: 1.0

Inleiding

Dit document gaat over het eerste plan voor het uitvoeren van de gehele opdracht. Het plan van aanpak is niet het absolute woord en dient eerder als een richtlijn. Hiervoor worden dingen als het doel van het project en de ondervonden problematiek beschreven. Dit zodat het mogelijk is om hier op terug te vallen wanneer er keuzes gemaakt worden; Het doel is namelijk niet het ontwikkelen van het beschreven product, maar het oplossen van het ondervonden probleem.

In eerste instantie wordt dit document gebruikt om de voorlopige toekomst te beschrijven. Dit document beschrijft de huidig ondervonden problematiek, de gekozen oplossingsrichting en hoe deze oplossing gedefinieerd gaat worden. Ook wordt beschreven welke middelen hierbij worden gebruikt, zodat hier tijdig rekening mee gehouden kan worden.

Later zou dit document kunnen dienen als documentatie over de eerste stappen binnen het project. Gezamenlijk met andere verslagen die gedurende het project gemaakt zullen worden, worden alle keuzes en beslissingen die tijdens het project worden gemaakt vastgelegd.

Probleembeschrijving

Het volgende stuk tekst komt uit de probleemanalyse.

Binnen het Thema Monitoring & Bedrijfsvergelijking binnen Panteia worden klanttevredenheidonderzoeken voor andere thema's uitgevoerd. De resultaten van deze onderzoeken worden via een online applicatie toegankelijk gemaakt.

Wanneer een bedrijf naar haar resultaten wil kijken, kan zij een overzicht opvragen. Het opvragen van dit overzicht gaat momenteel via ColdFusion. Een ColdFusion functie communiceert met een MSSQL database, waarmee (via enkele andere modules) een overzicht wordt gegenereerd. Voor grote overzichten moet men echter erg lang wachten (~30 seconden). Panteia wil dit zien veranderen.

Uit de probleemanalyse is gebleken dat het uitvoeren van alle queries die nodig zijn om het overzicht te genereren de hoofdoorzaak is voor de lange wachttijd. Verder is tijdens de probleemanalyse aangetoond dat dit probleem te verhelpen is door queries te combineren.

Doel

Het hoofddoel van dit project is het verhelpen van de performance problemen die ondervonden worden bij het opvragen van een overzicht. Daarnaast wordt er tijdens het project gekeken of er andere belangen zijn die mogelijk een rol spelen binnen het systeem.

Afbakening

Gedurende het project kunnen andere wensen of eisen naar boven komen. Alhoewel er getracht zal worden om deze wensen te vervullen, ligt de focus van het project op het hoofddoel.

Tijdens de probleemanalyse is geconstateerd dat meerdere componenten van het gehele systeem een invloed hebben op het eindresultaat. Dit project zal zich eerst richten op het oplossen van de grootste invloedfactor(spelling?). Mogelijkheden tot andere verbeteringen worden wel gerapporteerd, maar niet direct binnen dit project uitgevoerd. Deze verbeteringen zouden echter wel uitgevoerd kunnen worden als extra wensen of eisen, maar ook hier ligt de focus eerst op het hoofddoel.

In principe beperkt het project zich tot de oplossingsrichting die beschreven is in de probleemanalyse.

Opdrachtomschrijving

De opdracht die tijdens dit project uitgevoerd zal worden bestaat uit het batchen van de databasequeries die gebruikt worden voor het genereren van het overzicht. Als eerste onderdeel van deze opdracht, moet de exacte oplossing gedefinieerd worden.

De oplossingsrichting is in de probleemanalyse gedefinieerd als het combineren van de queries ("Batching") om de performance te verbeteren. Via een analyse zal gekeken worden wat de beste aanpak is voor het combineren van deze queries.

Aan de hand van deze analyse wordt dan vastgesteld "wat" de oplossing omvat. Deze oplossing wordt dan op component niveau beschreven zoals dit ook gebeurd is in de probleemanalyse – de exacte invulling van de oplossing is dan nog niet bekend.

Met de oplossingsrichting en de positie van het oplossende component binnen de systeem bekend moeten voorwaarden opgesteld worden waar dit oplossende component aan moet voldoen. Deze voorwaarden kunnen zowel eisen, wensen als voorkeuren zijn. Deze voorwaarden zullen worden geïdentificeerd en verzameld gedurende de requirementsanalyse.

Aan de hand van de requirementsanalyse kan dan een ontwerp voor de oplossing gemaakt worden. Dit ontwerp beschrijft de werking en het gebruik van de oplossing in detail. Aan de hand van dit ontwerp zal dan een verdere planning gemaakt worden voor het implementeren van deze oplossing.

Omdat momenteel de exacte definitie van de oplossing nog niet bekend is, is de verdere loop van de opdracht nog niet te voorspellen. Naarmate het project vordert, zullen nieuwe versies van dit Plan van Aanpak gemaakt worden. In deze versies wordt dan de verdere loop van het project beschreven.

Dit betekent ook dat alle versies van het Plan van Aanpak gezamenlijk het volledige Plan van Aanpak vormen.

Producten

In dit hoofdstuk worden de producten die tijdens dit project opgeleverd worden beschreven. Ook hier geldt dat latere versies van het Plan van Aanpak deze lijst kunnen wijzigen door de resultaten van de uitgevoerde analyses.

Probleemanalyse

In de probleemanalyse wordt de oplossingsrichting gedefinieerd. Aan de hand van de probleemanalyse kan een specifieker Plan van Aanpak opgesteld worden. Ook kan met behulp van de probleemanalyse de opdracht afgebakend worden tot het oplossen van het probleem.

Plan van Aanpak

In het Plan van Aanpak staat in grote lijnen beschreven hoe het project zal verlopen. Van het project worden de aspecten Producten, Planning en Benodigdheden beschreven. Het Plan van Aanpak dient als hoofddocument voor het project – alle andere onderdelen van het project zijn gebaseerd op het Plan van Aanpak.

Architectuuranalyse Huidige Situatie

In de architectuuranalyse van de huidige situatie wordt gekeken welke onderdelen van het huidige systeem betrokken zijn bij het probleem wat in de probleemanalyse is geconstateerd. Met behulp van deze analyse kan gespecificeerd worden waar de oplossing binnen het huidige systeem toegepast moet gaan worden.

Requirementsanalyse

Zodra duidelijk is waar de oplossing toegepast gaat worden binnen het huidige systeem is het ook duidelijk met welke onderdelen van het huidige systeem de oplossing rekening moet houden. Met behulp van een requirementsanalyse wordt vastgesteld hoe de oplossing met deze onderdelen rekening moet houden. Ook is er binnen de requirementsanalyse ruimte voor het beschrijven van de overige eisen of wensen die men heeft met betrekking van de opdracht.

Functioneel Ontwerp

In het functioneel ontwerp wordt de functionaliteit van de oplossing beschreven. Dit is in principe “wat” de oplossing gaat doen. “Hoe” de oplossing deze functionaliteit levert wordt niet in het functioneel ontwerp beschreven. Het functioneel ontwerp dient als een middel om aan een niet-technische opdrachtgever te kunnen communiceren wat de oplossing gaat doen.

Technische Analyse

In de technische analyse wordt gekeken naar de mogelijkheden die er zijn binnen de implementatie van de oplossing. Aan de hand van de technische analyse kan men op de interne architectuur van de oplossing bepalen.

Technisch Ontwerp

In het technisch ontwerp wordt de interne architectuur van de oplossing uitgewerkt tot een volledige beschrijving van de oplossing. Aan de hand van het technische ontwerp is het mogelijk om een planning te maken voor het ontwikkelen van de oplossingsimplementatie.

Oplossing – Eindproduct

Als eindproduct voor het project moet er een implementatie van de oplossing opgeleverd worden. Omdat de oplossing nog niet gedefinieerd is, kunnen er momenteel geen eisen aan deze oplossing gesteld worden. De voorwaarden waar deze oplossing aan moet voldoen worden opgesteld worden vastgesteld gedurende de requirementsanalyse.

Planning

In dit hoofdstuk wordt een globale planning beschreven voor de verloop van het project. De planning is opgedeeld per product.

Product	Startweek	Duur (eindweek)
Probleemanalyse	Week 1	< 1 Week (Week 1)
Plan van Aanpak (v1.0)	Week 1	< 1 Week (Week 1)
Architectuuranalyse Huidige situatie	Week 1	1 Week (Week 1-2)
Requirementsanalyse	Week 1-2	1 Week (Week 2-3)
Functioneel Ontwerp	Week 2-3	1 Week (Week 3)
Technische Analyse	Week 3	1 Week (Week 4)

Het is nog niet mogelijk om de overige producten in te plannen gezien de huidige situatie.

Ondersteunende mensen en middelen

Voor de succesvolle uitvoering van het project is samenwerking met andere medewerkers binnen het bedrijf Panteia noodzakelijk. Ook zijn er bepaalde materialen nodig voor het produceren en opleveren van de producten zoals deze in het hoofdstuk Producten beschreven staan.
Deze lijsten zijn niet volledig: Het is mogelijk dat er in latere fases van het project meer kennis nodig is van mensen. Ook zijn een aantal “standaard” materialen (zoals een computer met tekstverwerker, een werkplek, pen en papier) niet in de lijst opgenomen omdat deze dan onoverzichtelijk wordt.

Mensen

Webdeveloper/Stagebegeleider - Milan van der Storm
Als stagebegeleider heeft Milan een direct leidinggevende rol. Dit betekent dat enige keuzes op non-operationeel niveau overlegd moeten worden met Milan.

Gebruikers huidige systeemdeel
Voor de requirementsanalyse moeten de gerelateerde mensen geïdentificeerd en ondervraagd worden.

Materialen

Software:

Rechten tot huidige systeem
Deze rechten zijn benodigd om toegang te krijgen tot het huidige systeem. Zonder deze rechten is het niet mogelijk om een analyse uit te voeren op het huidige systeem.

Ontwikkelomgeving ColdFusion
Voor het analyseren van het applicatieonderdeel wat met de database communiceert. Dit applicatieonderdeel is in CFML geschreven en het analyseren hiervan zou gemakkelijker gaan wanneer er een ontwikkelomgeving voor ColdFusion aanwezig is.

Modelleersoftware (UML tool)
Voor het modelleren van het functioneel en technisch ontwerp is modelleersoftware vereist. Dit zal waarschijnlijk een UML tool worden.

Fysiek:

Whiteboard met stiften en wisser
Het maken van een functioneel en technisch ontwerp is makkelijker wanneer er een whiteboard beschikbaar is. Dit leidt tot een eerder en een beter resultaat.

Architectuuranalyse

Verantwoordelijkheden componenten huidige systeem

Datum: 20 Februari 2013
Auteur: Pim van Dongen
Versie: 1.2

Inleiding

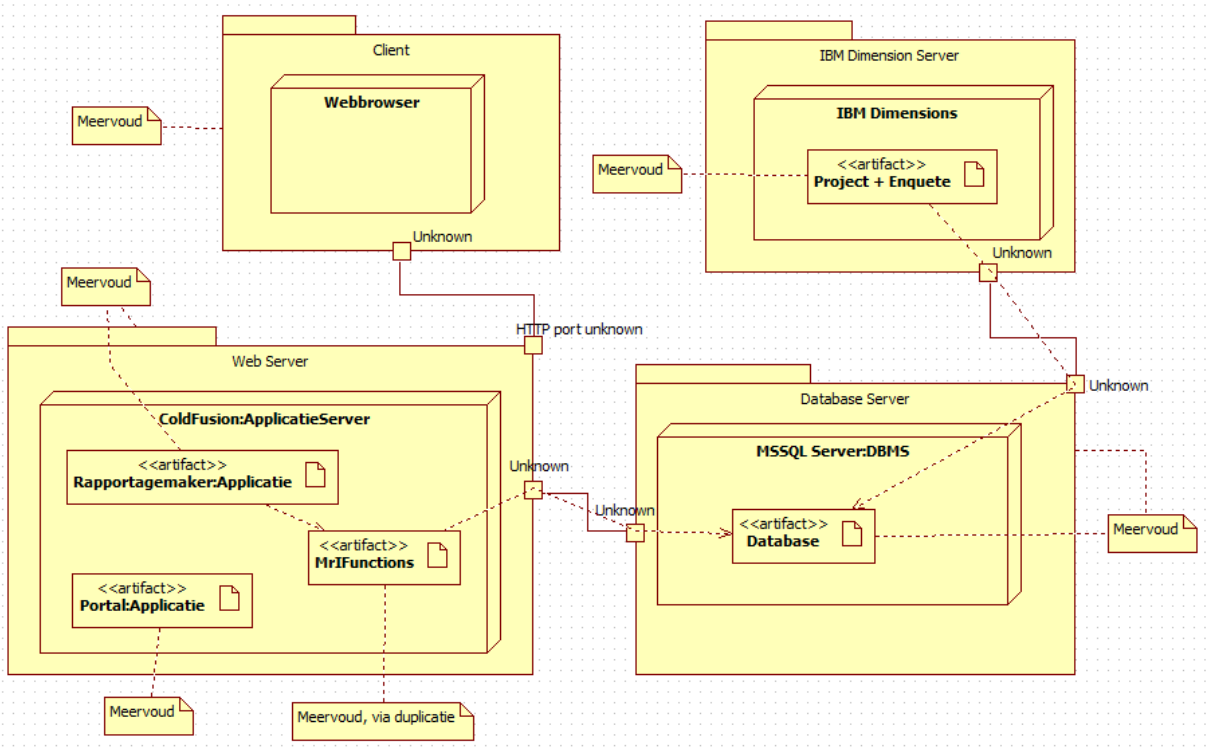
Dit document gaat over de architectuur van het systeem dat in de huidige situatie gebruikt wordt. Hierbij is “het systeem” het geheel van browser tot data binnen de processen die benodigd zijn om tot een overzicht te komen van het uitgevoerde onderzoek.

Dit document zal in eerste instantie gebruikt worden om op een gestructureerde wijze de oplossing voor het probleem verder te definiëren. Later is dit document te gebruiken als documentatie voor de plaatsing van de geïmplementeerde oplossing binnen het systeem.

In dit document zal eerst de architectuur van het systeem in de huidige situatie worden beschreven. Hierbij moet ik aangeven dat de beschreven architectuur slechts een abstractie is; het beschrijven van het systeem in zijn volledigheid zou meer tijd innemen dan wenselijk.

Componenten

In het onderstaande deployment diagram (UML) zijn de relevante delen van het bestaande systeem gemodelleerd.



In dit hoofdstuk worden alle componenten die gemodelleerd staan in het bovenstaande diagram beschreven. Per component wordt beschreven welke functie deze heeft en hoe deze (mogelijk) bijdraagt aan het probleem zoals deze nu bestaat.

In het geval dat het lastig is om een beeld te krijgen van het geheel, raad ik aan om eerst het hoofdstuk Procesbeschrijving te lezen. In dat hoofdstuk wordt via een procesbeschrijving geïllustreerd hoe componenten hun functie uitvoeren en hoe dit tot het probleem leidt.

Client

De client is de gebruiker. Omdat ik niet bekend ben met de vele manieren waarop een overzicht opgevraagd kan worden is hier sprake van een abstractie. De client zou bijvoorbeeld een computer of een mobiele telefoon kunnen zijn.

Webbrowser

Via de webbrowser kan de client een verzoek om het overzicht versturen en later het overzicht ontvangen. De webbrowser staat zelf buiten het proces van het genereren van de rapportages – het enige wat de webbrowser doet is de webserver vragen om het overzicht (en het mogelijk downloaden/weergeven van het overzicht).

Webserver

De webserver is de host van de website waar de client een overzicht kan opvragen. De webserver ontvangt requests van de client via een HTTP poort en verwerkt deze.

ColdFusion - Applicatieserver

Op de webserver draait een applicatieserver. Binnen het huidige systeem is deze applicatieserver ColdFusion. De ColdFusion applicatieserver host applicaties die geschreven zijn in ColdFusion. In principe dient de applicatieserver als een laag om te communiceren tussen de applicaties en de webserver (en daarmee mogelijk de databaseserver of client).

Portalapplicatie

De website van de enquete. Via de portal kunnen deelnemers van de enquete zich inloggen om de enquete te doen. Ook kan er via de portal ingelogd worden om een overzicht te laten genereren.

Rapportageapplicatie

Een abstractie van een onderdeel van het proces. De rapportageapplicatie representeert de applicaties die gebruikt worden om een rapportage of overzicht te genereren voor de client. Deze applicaties vragen gegevens uit de database op, om deze dan vervolgens in een bepaalde structuur te plaatsen en deze naar de client te sturen. De rapportageapplicatie zou mogelijk aan het probleem kunnen bijdragen doordat deze op een niet optimale wijze gebouwd is. Hierdoor duurt het langer om tot het gewenste resultaat te komen.

MrIFunctions

Een applicatieonderdeel wat gebruikt wordt in rapportageapplicaties. Dit applicatieonderdeel is verantwoordelijk voor het converteren van een bepaalde wens voor gegevens van een applicatie naar een query voor de database. Deze query kan dan vervolgens uitgevoerd worden op de database, waarna dit applicatieonderdeel de gegevens ontvangt vanuit de database. Deze gegevens worden waar nodig anders gestructureerd, waarna zij worden doorgestuurd naar de applicaties die deze gegevens nodig hebben. Dit applicatieonderdeel zou mogelijk aan het probleem kunnen bijdragen door inefficiënt te zijn (zowel in interne werking als communicatie met de database).

IBM Dimensions Server

De IBM Dimensions Server host het IBM Dimensions softwarepakket.

IBM Dimensions

Een softwarepakket voor het genereren van een database voor een enquête. Dit softwarepakket kan met behulp van een vragenlijst en een stijlbestand een enquêteerwebsite hosten.

Project (+ Enquete)

IBM Dimensions werkt met projecten. Ieder project heeft zijn eigen database op de databaseserver. Ieder project kan een vragenlijst en stijlbestand hebben.

Databaseserver

De databaseserver is verantwoordelijk voor het hosten van het databasemanagementsysteem (DBMS).

MSSQL Server – DBMS

Op de databaseserver draait een DBMS. Binnen het huidige systeem is dit DBMS MSSQL Server. Dit DBMS host verscheidende databases voor verscheidende projecten.

Database

De database bevat de gegevens die verkregen worden gedurende onderzoeken en die benodigd zijn voor het maken van een overzicht of rapportage. Om de gegevens uit de database te verkrijgen moeten er queries aan de database gesteld worden. De database zou mogelijk aan het probleem kunnen bijdragen door een ontbreking van optimalisaties in structuur.

Procesbeschrijving

In dit hoofdstuk wordt aan de hand van een procesbeschrijving geïllustreerd hoe ieder component zijn functie uitvoert en hoe dit uiteindelijk tot het probleem leidt.

Een aantal bedrijven willen graag weten hoe het volgens de medewerkers van de bedrijven met de bedrijven gaat. Ook willen deze bedrijven hun resultaten vergelijken met andere bedrijven.

Hiervoor gaat Panteia een onderzoek uitvoeren. Er wordt gebruik gemaakt van een online enquête, omdat de automatisering een enorme besparing van kosten is. Ook werkt dit makkelijker voor de medewerkers van de bedrijven.

Met behulp van IBM Dimensions wordt een project aangemaakt. Hierbij wordt een database opgezet, een vragenlijst aangemaakt en een applicatie om deze vragenlijst mee te presenteren aan de medewerkers. Wanneer dit klaar is kunnen de medewerkers de enquête online invullen.

Om de enquête in te vullen, gaat een medewerker (bijvoorbeeld) met zijn webbrowser naar de website (portal) waar hij de enquête kan invullen. Zijn webbrowser vraagt dan aan de webserver om de pagina waar de enquête zich op bevindt. Met behulp van de enquêteringsapplicatie wordt deze pagina gegenereerd.

De medewerker vult de enquête in. Tijdens het invullen worden de resultaten opgeslagen in de database.

Op een gegeven moment zijn de medewerkers klaar met invullen van de enquête en wil het bedrijf een overzicht van de resultaten hebben. Hiervoor gaat (bijvoorbeeld) een manager naar een overzichtspagina via dezelfde portal die de enquête gebruikt. Nadat hij daar heeft ingelogd kan hij een overzicht opvragen.

Wanneer hij een overzicht opvraagt gaat de rapportageapplicatie op de applicatieserver een rapportage genereren. De rapportageapplicaties vraagt aan het MrIFunctions applicatieonderdeel om de gegevens op te halen. Dit MrIFunctions applicatieonderdeel communiceert dan met de databaseserver, die via het DBMS de gegevens uit de database ophaalt en terug stuurt. Op basis van de dan verkregen gegevens stelt de rapportageapplicatie het rapport op en toont dit aan de manager.

Het ophalen van de gegevens uit de database duurt echter zeer lang. Dit komt doordat de rapportageapplicatie voor ieder deel van zijn rapport een vraag aan het MrIFunctions applicatieonderdeel stelt. Dit applicatieonderdeel vertaalt deze vraag in een of meerdere queries, die dan vervolgens aan de database worden gesteld. Deze interactie tussen de rapportageapplicatie en het MrIFunctions applicatieonderdeel leidt tot meer dan 250+ queries die aan de database gesteld worden.

De database is snel in het verwerken van deze queries, maar niet snel genoeg. Met een verwerkingstijd van 0.1 tot 0.2 seconden per query is het al snel duidelijk wat de oorzaak is van het lange ophalen.

Mogelijke oplossingen

In dit hoofdstuk worden de mogelijke oplossingen beschreven. De exacte implementatie is hierbij nog niet bekend en deze analyse beperkt zich tot een kosten-batenanalyse aan de hand van de ingeschatte hoeveelheid werk en resultaat.

Voor iedere oplossing wordt beschreven hoe deze in het huidige systeem staat en welke functies deze zou uitvoeren. Ook wordt beschreven wat de mogelijke kosten en baten van deze oplossing is. Aan het eind van dit hoofdstuk worden de oplossingen met elkaar vergeleken.

Verbeteren Rapportageapplicatie

Deze oplossing is gebaseerd op het idee dat iedere vraag die de rapportageapplicatie stelt leidt tot meer werk voor de database. De rapportageapplicatie wordt aangepast om vragen voor gegevens uit de database meer te bundelen, zodat de vraag voor gegevens de database niet te veel belast.
Deze oplossing zou zeer hoge kosten met zich mee brengen. Deze zeer hoge kosten komen doordat iedere rapportageapplicatie aangepast zou moeten worden. Het aanpassen van een enkele rapportageapplicatie zou lage kosten met zich meebrengen. Er zijn verscheidende rapportageapplicaties in gebruik en deze moeten ieder los aangepast worden. Het resultaat zou zijn dat het performanceprobleem voor iedere behandelde rapportageapplicatie deels opgelost is. Nieuwe rapportageapplicaties zouden echter weer problemen kunnen ondervinden.

Vervangen MrIFunctions applicatieonderdeel

Deze oplossing is gebaseerd op het idee dat het applicatieonderdeel de vragen op een verkeerde manier stelt aan de database. Er zijn maar een beperkt aantal types van queries die dit applicatieonderdeel aan de database kan stellen. Hierdoor worden veel queries van hetzelfde type uitgevoerd – de database is vaak bezig met hetzelfde uit te voeren! Deze oplossing bestaat uit het aanpassen of vervangen van het MrIFunctions applicatieonderdeel. Het vervangende applicatieonderdeel zou proberen om queries samen te voegen om de druk op de database te verminderen.
Deze oplossing zou lage kosten met zich meebrengen. De oplossing betreft het vervangen van een enkel applicatieonderdeel waarvan de werking reeds bekend is. Het resultaat zou een applicatieonderdeel zijn wat het performanceprobleem oplost. Wegens de structuur van het huidige MrIFunctions applicatieonderdeel zullen ook de rapportageapplicaties aangepast moeten worden om gebruik te kunnen maken van het vervangende applicatieonderdeel.

Toevoegen Query-combiner applicatieonderdeel

Deze oplossing is gebaseerd op het idee dat het MrIFunctions applicatieonderdeel de vragen op een verkeerde manier stelt aan de database. Het was echter lastig om destijds dit applicatieonderdeel te ontwikkelen en in principe werkt het applicatieonderdeel naar behoren. Het is alleen te traag.
Deze oplossing bestaat uit het toevoegen van een “Query-combiner” applicatieonderdeel. Dit applicatieonderdeel zou de queries die naar de database gestuurd worden onderscheppen, waarna deze geanalyseerd worden. Vervolgens worden de queries gecombineerd waar mogelijk, waarna deze uitgevoerd worden.
Deze oplossing zou hoge kosten met zich meebrengen. Alhoewel er maar op een enkele plek binnen het systeem een aanpassing gedaan moet worden omvat deze aanpassing het analyseren van tekst.

Tekst analyseren is een complexe taak. Dit applicatieonderdeel zou wel in te zetten zijn bij ieder project wat van het huidige MrIFunctions applicatieonderdeel gebruik maakt.

Relatief gezien zou het aanpassen van de rapportageapplicatie waarschijnlijk de snelste aanpak zijn. Deze aanpak is echter niet toekomstbestendig. Ook is die aanpak niet schaalbaar naar de andere projecten die momenteel het performanceprobleem ondervinden.

De oplossing met de minste impact op het systeem is de query-combiner. Het is echter nog maar de vraag of deze aanpak technisch haalbaar is. Mogelijk zouden er tijdens het implementeren toch nog dingen aangepast moeten worden, wat het grootste voordeel van deze oplossing doet laten verdwijnen.

Dan resteert nog de oplossing waarbij het MrIFunctions applicatieonderdeel wordt vervangen. Deze aanpak heeft geen last van enige onzekerheden. De werking van het onderdeel wat vervangen wordt is te meten door te kijken naar de bestaande situatie. Zolang de rapportageapplicatie dezelfde resultaten terug krijgt is de exacte inrichting van het applicatieonderdeel niet belangrijk. Dit geeft een hoop technische vrijheid – speling, als het ware. Het is mogelijk dat de rapportageapplicaties aangepast moeten worden gebruik te maken van het vervangende applicatieonderdeel.

Conclusie

Ik heb de componenten van het huidige systeem beschreven en gemodelleerd. Aan de hand van deze beschrijving, heb ik een aantal mogelijke oplossingen opgesteld.

In de probleemanalyse was geconcludeerd dat de oorzaak van het performanceprobleem lag bij de hoeveelheid queries die de database moet verwerken bij het genereren van een rapport. Dit beperkt de mogelijke onderdelen van aanpassing tot de rapportageapplicatie, de database en de laag daartussen (MrIFunctions).

Het aanpassen van de database leidt niet tot merkbare resultaten, zoals eerder geconstateerd bij de probleemanalyse. Hierdoor verschuift de focus naar het MrIFunctions applicatieonderdeel. De drie geformuleerde oplossingen komen in principe op het volgende neer: Samenvoegen van de queries voordat ze omgezet worden in SQL, samenvoegen van de queries tijdens het omzetten naar SQL en samenvoegen van de queries na het omzetten in SQL.

Samenvoegen van de queries voordat ze omgezet worden in SQL vereist dat de rapportageapplicaties aangepast worden. Dit is tijdrovend werk en de oplossing is niet gemakkelijk over te brengen naar een volgende applicatie.

Samenvoegen van de queries nadat ze omgezet zijn in SQL vereist tekstanalyse van de SQL-queries. Dit is zeer lastig en mogelijk niet gemakkelijk uit te breiden voor het toevoegen van andere functionaliteit.

Samenvoegen van de queries tijdens het omzetten ervan naar SQL is daarmee de makkelijkste oplossing. Een oplossing op die plaats binnen het systeem heeft de mogelijkheid zowel de SQL queries als het resultaat aan te passen. Dit is nodig, omdat het aanpassen van de SQL queries tot een iets ander resultaat leidt (namelijk een lijst van resultaten in plaats van per SQL query een enkel resultaat).

Hiermee kom ik dus tot de volgende conclusie:
Binnen de huidige architectuur is er het meeste resultaat te bereiken door het MrIFunctions applicatieonderdeel aan te passen. De oplossing bestaat uit het aanpassen of vervangen van het MrIFunctions applicatieonderdeel zodat deze queries voor de database combineert en het resultaat weer uitsplitst voor de rapportageapplicaties.

Requirementsanalyse

Toepassen Batching op Queries voor PDF-Rapportages

Datum: 28 Februari 2013
Auteur: Pim van Dongen
Versie: 1.0

Inleiding

Dit document gaat over de wensen en eisen die gesteld worden aan de oplossing voor het probleem wat dit project gaat verhelpen. In dit document zal eerst de ruwe vorm van de oplossing beschreven worden. Daarna wordt per component van de oplossing beschreven welke wensen en eisen hierop van toepassing zijn.

In eerste instantie dient dit document als een naslagwerk van de wensen en eisen die aan het begin van het project gevonden zijn. Later kan dit document gebruikt worden als een soort “checklist”, om te kijken of de geïmplementeerde oplossing alle wensen en eisen dekt.

Oplossingsbeschrijving

In dit hoofdstuk wordt de ruwe vorm van de oplossing voor het performance probleem beschreven.

Aan de hand van de architectuuranalyse is geconcludeerd dat zowel de rapportageapplicaties als de interface tussen de database en de rapportageapplicaties gewijzigd zullen moeten worden. De oplossing bestaat dus uit 3 componenten:

Rapportageapplicaties

De rapportageapplicaties moeten aangepast worden zodat zij alle benodigde data in een keer opvragen alvorens deze data gebruikt wordt. Anders is het niet mogelijk om batching toe te passen.

Database interface

De database interface moet aangepast worden om de queries niet direct uit te voeren en om te kunnen gaan met gebatchte queries.

Batcher

Als laatste component moet er een batcher gemaakt worden. De batcher is verantwoordelijk voor het samenvoegen (batchen) van de queries. Waar de andere twee componenten aanpassingen zijn van bestaande onderdelen zal de batcher in zijn geheel nieuw gemaakt moeten worden.

Requirements

In dit hoofdstuk zijn de wensen, eisen en enige voorkeuren per oplossingscomponent beschreven.

Rapportageapplicaties

De rapportageapplicaties dienen grotendeels hetzelfde te blijven.

1: Alleen de rapportageapplicaties met ernstige performance problemen moeten aangepast worden.

Dit houdt in dat niet alle rapportageapplicaties aangepast zullen worden om met de batcher te werken. Enige toekomstige rapportageapplicaties zullen waarschijnlijk wel standaard met de batcher werken. De exacte definitie van “ernstige performance problemen” zal later gedurende het project bekend worden. In ieder geval valt de Blik op Werk rapportageapplicatie binnen de categorie “ernstige performance problemen”.

Database interface

De functielijst van de database interface zal volledig ondersteund moeten worden zodat het converteren van werkwijze en oude applicaties weinig tijd in neemt.

2: De volledige functielijst zoals die momenteel binnen CollectFunctions en MrIFunctions bestaat moet ondersteund worden.

Dit zodat rapportageapplicaties theoretisch compatibel zijn en het aanpassen van de rapportageapplicaties soepeler verloopt.

3: VOORKEUR: De database interface moet gedocumenteerd worden, zowel in structuur als in werking.

De huidige database interface is niet gedocumenteerd. Hierdoor is het lastig om onderhoud te verrichten aan de database interface of andere aanpassingen te maken.

Batcher

De batcher is een nieuw component en zal in zijn geheel ontwikkeld moeten worden.

4: De batcher dient het performance probleem te verminderen met 75%.

Dat wil zeggen dat er van de 30 seconden die momenteel geconstateerd zijn bij het genereren van het rapport, er door het gebruik van de batcher nog slechts 7.5 of minder seconden resteren.

5: De batcher is verantwoordelijk voor het aanpassen van het resultaat zodat deze conform de databaseinterface is.

Op deze manier zijn de database interface en de batcher minder afhankelijk van elkaar, wat een beter design en product oplevert.

6: De batcher moet zonder enige aanpassing toegevoegd kunnen worden aan een nieuw project, mits deze gebruik maakt van de database interface die aangepast is aan de batcher.

Dit zodat men bij nieuwe projecten dezelfde stijl kan hanteren als de stijl die momenteel gehanteerd wordt. Momenteel kopieert men het database gedeelte van project tot project, zonder enige

aanpassingen. Dit betekent dat als er een fout gevonden wordt, de verbetering van deze fout snel toegepast kan worden binnen alle projecten.

Functioneel Ontwerp

Toepassen Batching op Queries voor PDF-Rapportages

Datum: 10 Mei 2013
Auteur: Pim van Dongen
Versie: 1.1

Inleiding

Dit document gaat over het functioneel ontwerp van de oplossing voor het performance probleem. In het functioneel ontwerp staat beschreven “wat” de oplossing gaat doen om het probleem te verhelpen. Ook wordt uitgelegd hoe de oplossing binnen de huidige architectuur geplaatst wordt.

In eerste instantie dient dit document als een rapport richting de leidinggevende over de werking en vorm van de gevonden oplossing. Later is dit document te gebruiken als documentatie voor de werking van de gekozen oplossing, en hoe deze oplossing binnen de bestaande architectuur past.

Functionele Beschrijving

In dit hoofdstuk wordt de werking van de oplossing op functioneel niveau uitgelegd. Dat wil zeggen dat alle verantwoordelijkheden en taken die de oplossing uit gaat voeren vastgelegd worden.

Dit hoofdstuk is opgesplitst in drie delen; ieder deel gaat over een apart component van de oplossing, zoals deze splitsing al eerder is gemaakt in de Requirementsanalyse (waar het gedaan werd op basis van de Architectuuranalyse).

Rapportageapplicaties

De rapportageapplicaties die “ernstige performance problemen” ondervinden worden aangepast om met de batcher om te kunnen gaan. Deze aanpassing bestaat uit het opsplitsen van de programmaonderdelen waarin er om data voor rapportages gevraagd wordt en de programmaonderdelen waar de daadwerkelijke rapportage wordt gegenereerd. Door middel van deze opsplitsing is het mogelijk om andere methoden en technieken (zoals batching) toe te passen bij het ophalen van de data.

Database interface

De database interface, momenteel bekend als de ColdFusion applicatiecomponenten CollectFunctions en MrIFunctions moeten aangepast worden om goed met de batcher samen te kunnen werken. Momenteel stellen zij hun vragen direct aan de database en wachten zij op een respons. De aanpassing bestaat uit het doorsturen van de vraag voor gegevens naar de batcher en op basis van een functieaanroep gegevens te accepteren. Dit zorgt ervoor dat de rapportageapplicaties andere taken kunnen uitvoeren terwijl zij wachten op de gevraagde gegevens (bijvoorbeeld berekenen welke andere gegevens nodig zijn en de basis structuur van het rapport opzetten). Daarnaast is de aanpassing nodig om te voldoen aan de niet-functionele requirements over de onderhoudbaarheid van de database interface.

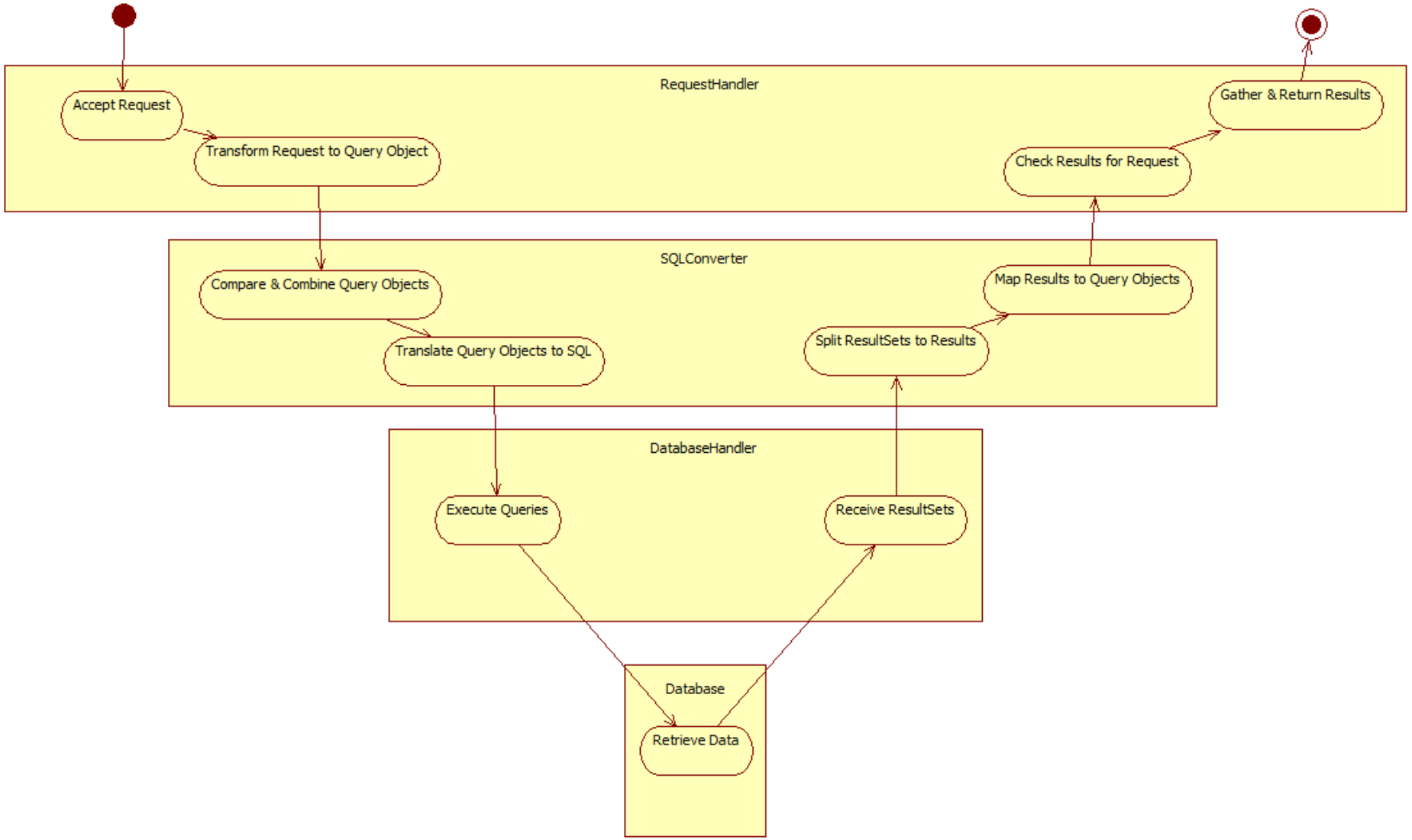
Batcher

De batcher is een nieuw onderdeel van het systeem. De verantwoordelijkheden van de batcher bestaan uit het analyseren van de binnenkomende requests voor data uit de database en deze vervolgens samen te voegen via het batching process. Daarnaast is de batcher verantwoordelijk voor het versturen van de gebatchte queries naar de database en is de batcher verantwoordelijk voor het opsplitsen van het verkregen resultaat zodat er per request een antwoord terug gegeven kan worden.

De batcher bestaat intern ook weer uit een aantal onderdelen. Omdat dit document over het functionele ontwerp gaat is de exacte interne constructie nog niet bekend. De interne werking van de batcher is echter ongeveer als volgt:

De batcher bestaat uit een requesthandler, die de requests voor gegevens omzet in een query object, een SQLConverter, die een of meerdere query objecten in een echte SQL query omzet, en een databasehandler, die de verbinding regelt met de database en de SQL queries via deze verbinding uitvoert.

Het proces is gemodelleerd in het model op de volgende pagina. Aan de hand van dit model ga ik dieper in op de verantwoordelijkheden van ieder onderdeel van de batcher.


RequestHandler
Accept Request
Transform Request to Query Object
Gather & Return Results
Check Results for Request
SQLConverter
Compare & Combine Query Objects
Translate Query Objects to SQL
Map Results to Query Objects
Split ResultSets to Results
DatabaseHandler
Execute Queries
Receive ResultSets
Database
Retrieve Data

RequestHandler

De requesthandler is verantwoordelijk voor het accepteren van requests voor data uit de database. De requesthandler vertaalt deze requests naar Query objecten voor de SQLConverter en houdt in de gaten of een request vervuld is. Als een request vervuld is stuurt hij het antwoord terug. De RequestHandler is hiermee verantwoordelijk voor Requests en Query objecten.

SQLConverter

De SQLConverter is verantwoordelijk voor het combineren (batchen) van Query objecten en het vertalen van de gecombineerde Query objecten naar SQL. Vertaalde queries kunnen uitgevoerd worden op de database en daarmee kan de data opgehaald worden. De SQLConverter is daarnaast verantwoordelijk voor het uitsplitsen van de opgehaalde data en het vervolgens koppelen van een set resultaten aan het Query object wat om die resultaten vroeg. Dan kan de RequestHandler de juiste resultaten voor ieder request terug sturen. De SQLConverter is hiermee verantwoordelijk voor Query objecten en de resultaten die hiervoor terug gegeven worden door de database.

DatabaseHandler

De DatabaseHandler is verantwoordelijk voor het beheren van de databaseconnectie. Dit houdt in dat de DatabaseHandler omgaat met het maken van de verbinding en foutafhandeling voor deze verbinding. Via de databaseverbinding kunnen de vertaalde queries uitgevoerd worden op de database. De resultaten van deze queries worden dan vervolgens door de database terug gestuurd over deze verbinding, waarna de DatabaseHandler deze terug stuurt naar de SQLConverter. De DatabaseHandler is hiermee verantwoordelijk voor de databaseverbinding en het uitvoeren van de queries op de database.

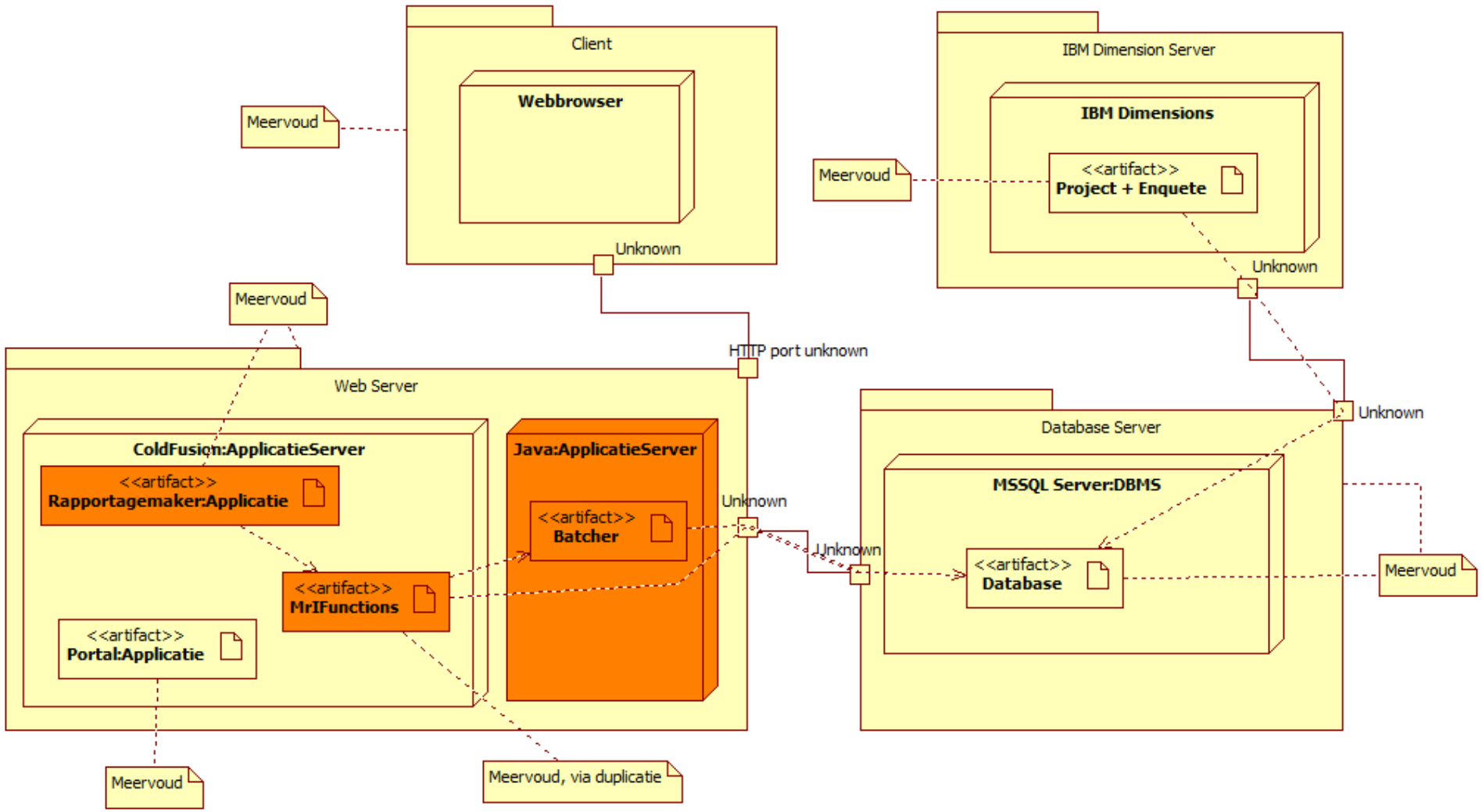
Architectuur

In dit hoofdstuk wordt beschreven hoe de oplossing geplaatst wordt in de huidige architectuur. Deze architectuur staat gemodelleerd op de volgende pagina. De onderdelen die een oranje kleur hebben worden hierbij aangepast voor de implementatie van de oplossing.

Het meeste is al uitgelegd in het hoofdstuk Functionele Omschrijving. 1 ding is echter nieuw in het diagram: De Batcher en de Java ApplicatieServer waar de Batcher op draait.

De batcher wordt ontwikkeld in Java en heeft dus een Java applicatieserver nodig om te functioneren. Daar waar het applicatieonderdeel MrIFunctions eerst direct met de database communiceerde, gaat dit nu hoofdzakelijk via de batcher. Dat is een van de redenen dat het MrIFunctions applicatieonderdeel aangepast moet worden. Voor queries die geen performance verbetering kunnen ondervinden van batching is het nog steeds mogelijk om direct op de database uitgevoerd te worden.

De reden waarom de rapportageapplicaties aangepast moeten worden is dat deze anders moeten gaan communiceren met het applicatieonderdeel MrIFunctions. Dit zodat er ook daadwerkelijk gebruik gemaakt kan worden van batching.


Client
Webbrowser
Meervoud
Unknown
IBM Dimension Server
IBM Dimensions
<<artifact>>
Project + Enquete
Meervoud
Unknown
Meervoud
HTTP port unknown
Web Server
ColdFusion:ApplicatieServer
<<artifact>>
Rapportagemaker:Applicatie
<<artifact>>
MrIFunctions
<<artifact>>
Portal:Applicatie
Java:ApplicatieServer
<<artifact>>
Batcher
Unknown
Unknown
Database Server
Unknown
MSSQL Server:DBMS
<<artifact>>
Database
Meervoud
Meervoud
Meervoud, via duplicatie

Technische Analyse

Toepassen Batching op Queries voor PDF-Rapportages

Datum: 19 Maart 2013
Auteur: Pim van Dongen
Versie: 1.2

Inleiding

Dit document gaat over de technische problemen die zouden kunnen voorkomen bij het ontwikkelen van de batcher. Ook beschrijft dit document oplossingen voor deze problemen waar mogelijk.

De technische problemen die in dit document zijn beschreven zijn aan het licht gekomen door het ontwikkelen van een proof of concept. Dit document beschrijft ook hoe dit proof of concept is opgezet.

Het doel van dit document is om mogelijke technische problemen die kunnen voorkomen bij het ontwikkelen van de batcher in kaart te brengen. Dan kan er in het technische ontwerp rekening gehouden worden met deze mogelijke problemen.

Proof of Concept

In dit hoofdstuk wordt beschreven hoe het proof of concept is opgezet.

Het proof of concept is opgezet om enkele vragen te beantwoorden. Deze vragen waren als volgt:

- Hoe kan er verbinding gemaakt worden met de database?
- Hoe kan een Java applicatie een ResultSet omvormen tot een datastructuur binnen ColdFusion?
- Hoe zou het algoritme van query-batching eruit zien?
- Is het mogelijk om met proxy-objecten te werken?

Per vraag wordt nu beschreven waarom deze vraag beantwoordt moest worden. Daarnaast wordt per vraag ook een kort samengevat antwoord gegeven, wat later in andere hoofdstukken van dit document wordt uitgewerkt.

Verbinding met de database

ColdFusion regelt databaseverbindingen via serverconfiguratie. Ik ben niet bekend met hoe dit in Java zou werken, waardoor er waarschijnlijk naar een alternatief gekeken moet worden.

Met behulp van JDBC drivers van Microsoft is het mogelijk om met de database te verbinden. Het is echter niet makkelijk om een JDBC driver voor MSSQL Server te vinden die Windows authenticatie ondersteunt, dus er zal gebruik gemaakt moeten worden van “normale” authenticatie voor de database.

Binnen Java een ResultSet omvormen tot een ColdFusion datastructuur

Alhoewel ColdFusion gebaseerd is op Java maken ze gebruik van andere typen datastructuren. Hierdoor moet er een conversie tussen datatypes gebeuren wanneer er gecommuniceerd wordt tussen ColdFusion en Java.

Het blijkt dat van Java naar ColdFusion lastig is. Java heeft echter geen problemen met ColdFusion objecten. Als er vanuit ColdFusion een datastructuur naar Java gestuurd wordt, kan Java deze zonder enige conversie gebruiken. Wanneer deze datastructuur dan weer naar ColdFusion terug gestuurd wordt, is er geen conversie nodig.

Algoritme query-batching

De batcher werkt door “queries met elkaar te vergelijken en deze dan te combineren”. De manier waarop dit gebeurt is echter nog niet eerder exact beschreven, waardoor het nog onzeker is of het ook echt daadwerkelijk mogelijk is.

Door middel van het abstraheren van de query tot de query structuur is het mogelijk om het aantal mogelijke verschillende queries terug te brengen naar een overzienbaar aantal. Met behulp van waarheidstabellen kunnen vervolgens regels vastgesteld worden. Deze regels kunnen vervolgens gebruikt worden om te bepalen of het mogelijk is om twee query structuren te combineren.

Omdat dit onderwerp een zeer belangrijk onderdeel is van het project, wordt dit onderwerp verder uitgewerkt in een apart hoofdstuk.

Proxy-objecten

De eigenlijke vraag is “hoe kan men binnen ColdFusion a-synchroon resultaat ontvangen van queries”? Een van deze manieren is via proxy-objecten. Proxy-objecten dienen als wrappers om het daadwerkelijke resultaat. Wanneer de applicatie gegevens nodig heeft vraagt hij het proxy-object om deze gegevens. Als deze gegevens beschikbaar zijn worden deze direct geretourneerd, zo niet, dan moet er door de applicatie gewacht worden tot dat de gegevens beschikbaar zijn.

Het is gebleken dat het mogelijk is om proxy-objecten te gebruiken. Wel betekent dit dat mogelijk eerst het resultaat uit de wrapper gehaald moet worden voordat hier mee gewerkt kan worden – er is dus een extra stap vereist binnen de rapportageapplicaties.

Technische Problemen

Dit hoofdstuk beschrijft de technische problemen die ondervonden zijn gedurende het ontwikkelen van het proof of concept. De problemen zijn verdeeld per categorie.

Database

Windows Authenticatie

Niet veel JDBC drivers ondersteunen de Windows Authenticatie die gebruikt wordt door MSSQL Server. Er zal dus gebruik gemaakt moeten worden van een “standaard” database account om verbinding te maken met de database.

Applicatie gaat hangen bij uitvoeren query

ColdFusion is gebaseerd op Java. Voor het gebruik van ColdFusion moet daarvoor dus een Java Runtime Environment (JRE) geïnstalleerd zijn. Deze zit standaard bij de applicatieserver. De versie van de aangeleverde JRE kan echter 1.6.0_29 zijn. Deze versie van de JRE heeft een bug waardoor de Java-applicatie hangt wanneer je (onder andere) een query uitvoert.

Om dit op te lossen raad ik aan dat de JRE van 1.6.0_30 gedownload wordt. Vervolgens moet binnen de 1.6.0_29 JRE het volgende bestand vervangen worden: ../lib/jsse.jar. Dit bestand moet vervangen worden met hetzelfde bestand uit JRE 1.6.0_30. Het installeren van andere JRE versies is natuurlijk ook mogelijk, maar deze oplossing lijkt geen neveneffecten te hebben. Toen ik probeerde JRE 1.6.0_30 in zijn geheel te installeren voor de ColdFusion applicatieserver, wou deze niet meer opstarten. Daarom raad ik aan JRE 1.6.0_29 met een jsse.jar uit 1.6.0_30 te gebruiken.

Communicatie tussen ColdFusion en Java

Datastructuren

Om ervoor te zorgen dat huidige rapportageapplicaties makkelijk te converteren zijn om met de batcher te gaan werken, moet de batcher de resultaten van de queries terug geven in het format van ColdFusion’s datastructuren.

Alhoewel ColdFusion automatisch objecten van bepaalde datatypes converteert naar ColdFusion datatypes, werkt dit niet helemaal naar behoren. Zo wordt een HashMap (java.util.HashMap, extends java.util.Map) keurig omgezet naar een Struct (coldfusion.runtime.struct). Echter wanneer je een HashMap hebt met HashMaps daarin, lukt het ColdFusion niet om deze objecten te converteren.

Gelukkig is een struct een subclass van java.util.Map. Door vanuit ColdFusion een lege struct mee te geven aan de batcher, kan de batcher via de class van de struct nieuwe structs maken. Wanneer de datastructuur dan vervolgens teruggestuurd wordt naar ColdFusion, is het niet nodig om de datastructuur te converteren.

Proxy-objecten

Binnen de huidige rapportageapplicaties wordt eerst de structuur van het rapport opgebouwd. Daarna wordt bepaald welke gegevens nodig zijn om het rapport te vullen. Vervolgens wordt voor ieder rapportonderdeel om de gegevens gevraagd, waarna deze direct in het rapport gezet worden.

Hierdoor moet de rapportageapplicatie in de huidige situatie per rapportageonderdeel wachten totdat de gegevens opgehaald zijn uit de database.

Door middel van proxy-objecten is het mogelijk om een tijdelijk resultaat terug te geven. Een proxy-object functioneert als een dichte doos – in de achtergrond worden de dozen gevuld met de benodigde gegevens. Ondertussen kan de rapportageapplicatie verder met het opvragen van gegevens.

Eerst alle gegevens opvragen en daarna pas gebruiken is slechts een deel van de oplossing. Omdat batching de volgorde aanpast waarin de queries worden uitgevoerd kan er nog een extra performance winst behaald worden door te kijken welke queries al klaar zijn, en daar dan de resultaten van te verwerken. Mogelijk is hierbij multi-threading vereist, maar dit is mogelijk.

Query-batching

Dit hoofdstuk beschrijft de verscheidende onderdelen van de batcher en de technische problemen die bij ontwikkeling daarvan zouden kunnen ontstaan.

Query-batching algoritme

Voordat queries gebatcht kunnen worden moet er eerst gekeken of ze compatibel zijn met elkaar. Queries zijn compatibel als batching geen invloed heeft op de resultaten.

Query-batching kan op twee niveaus gedaan worden. Semantisch, of syntactisch.
Semantisch batchen bestaat uit het samenvoegen van vragen voordat deze vertaald worden naar SQL. Een voorbeeld: gegeven tabellen “Diersoort”, “Gebied” en “Diersoort_In_Gebied”, worden de volgende vragen gesteld: “Geef het aantal diersoorten wat in gebied 1 voorkomt” en “Geef het aantal diersoorten wat in gebied 2 voorkomt”. Deze vragen worden dan omgezet tot “Geef per gebied het aantal voorkomende diersoorten, waarbij het gebied ‘gebied 1’ of ‘gebied 2’ is.”
Batchen op dit niveau is makkelijker dan batchen op het syntactische niveau, maar voor iedere vraagtype moet er opnieuw met de hand beschreven worden hoe het batchen werkt. Omdat dit met de hand moet worden complexere batchmogelijkheden mogelijk gemist.

Syntactisch batchen bestaat uit het samenvoegen van queries op basis van de querystructuur. Hiervoor is het niet nodig om per vraagtype aan te geven of en hoe batchen mogelijk is – er hoeft alleen maar per querystructuurcombinatie beschreven te worden hoe het batchen werkt. Dit brengt echter een zekere mate van complexiteit met zich mee.
Binnen het semantisch batchen kan men domeinkennis toepassen. Er zou bijvoorbeeld een extra join toegevoegd kunnen worden om eerst op andere criteria te filteren. Binnen syntactisch batchen is het toepassen van domeinkennis niet mogelijk omdat je “alle” syntactische situaties moet afdekken. Aan de andere kant is door dit volledige afdekken van het syntactische gedeelte ook gelijk de volledigheid van semantische constructies ondersteund (“diersoorten in gebied”, “werknemers bij project”, “producten in aanbieding”).

Binnen de batcher zal gebruik gemaakt worden van syntactisch batchen. Om dit te kunnen doen worden binnenkomende vragen geconverteerd naar een querystructuur. Wanneer alle querystructuren van de batch binnen zijn, worden deze aan elkaar vergeleken. Binnen het proof of concept is gebruikgemaakt van het combinatie patroon “alles is hetzelfde, op 1 ‘where attribute = value’ na”. Een voorbeeld:
SELECT hetzelfde (met een statistische functie)
FROM hetzelfde
WHERE hetzelfde, op 1 attribuut na wat alleen een andere value heeft
naar
SELECT hetzelfde, attribuut
FROM hetzelfde
WHERE hetzelfde, attribuut IN (values)
GROUP BY attribuut

Als we dit voorbeeld invullen voor “Diersoort_In_Gebied”, (“geef het aantal diersoorten wat geen insect is in gebied 1” + “geef het aantal diersoorten wat geen insect is in gebied 2”:

```
SELECT COUNT(*) as amt
FROM Diersoort_In_Gebied dig JOIN Diersoort d ON dig.diersoort = d.pkey
WHERE d.type != “insect” AND dig.gebied = 1;
+
SELECT COUNT(*) as amt
FROM Diersoort_In_Gebied dig JOIN Diersoort d ON dig.diersoort = d.pkey
WHERE d.type != “insect”  AND dig.gebied = 2;
naar
SELECT COUNT(*) as amt, dig.gebied
FROM Diersoort_In_Gebied dig JOIN Diersoort d ON dig.diersoort = d.pkey
WHERE d.type != “insect” AND dig.gebied IN (1,2)
GROUP BY dig.gebied;
```

Andere combinatiepatronen kunnen de mogelijkheden van de batcher uitbreiden. Deze zullen echter gedurende het project bedacht moeten worden aan de hand van de queries die uitgevoerd moeten worden.

Nadat de gebatchte queries uitgevoerd zijn moeten de resultaten weer opgesplitst worden. Dit gaat via een query identifier. In het vorige combinatiepatroon is het attribuut waarop gebatcht wordt de query identifier. In principe werkt een query identifier als volgt:

```
SELECT <columns>
FROM results
WHERE queryidentifier = value;
```

Functioneel bewijs

Per combinatiepatroon moet aangegeven worden met behulp van een volledig dekkend bewijs dat de gestelde querycombinatie in alle gevallen dezelfde resultaten retourneert.

Dit is het bewijs voor de werking voor het eerder getoonde combinatie patroon:
Dit bewijs maakt de assumptie dat observatie geen invloed heeft op het resultaat. Dat wil zeggen dat het niet uitmaakt hoe vaak er een SELECT query wordt uitgevoerd omdat dit geen invloed heeft op het eindresultaat. Daarnaast is het voor batchen zo dat de volgorde van de queries niet uitmaakt. Het batchalgoritme gaat hier dan ook van uit.

Alle SELECT statements die een statistische functie gebruiken, waarbij geen GROUP BY aanwezig is, retourneren slechts 1 regel. Deze regel bevat de waarde die berekend is, of NULL als het niet mogelijk is om een waarde te berekenen omdat er geen regels zijn die aan de condities in de where voldoen.

COUNT is hier een uitzondering op: COUNT geeft 0 terug als er geen matchende regels zijn. Dit betekent dat het type van de gebruikte statistische functie bepaalt hoe er met ontbrekende regels omgegaan moet worden.

De queries die gebatcht moeten worden variëren maar op 1 punt: de waarde van een bepaald attribuut. Op dit attribuut met de varierende waarde gaan we batchen. Dit gebeurt via WHERE IN en

GROUP BY. Met behulp van WHERE IN kunnen we de toegestane variaties van de waarde voor het attribuut vastleggen. Met behulp van GROUP BY splitsen we vervolgens het resultaat van de statistische functies weer op. Daarnaast voegen we het attribuut in de SELECT toe. Het attribuut dient als query identifier – omdat we op het attribuut hebben gebatcht, kunnen we aan de hand van de waarde van het attribuut bepalen bij welke query het resultaat hoort.

Omdat GROUP BY voor de SELECT gaat, is er geen verandering in de hoeveelheid geobserveerde data. Dus is er per groep geen verandering in het resultaat van de statistische functies. Wel is er verschil mogelijk in het aantal groepen omdat GROUP BY geen groepen toestaat van 0 regels.

Hier moeten we dus zelf mee rekening houden. Bij het verkrijgen van het resultaat verdelen we eerst de ResultSet aan de hand van de query identifier. Voor de rest van de queries bouwen we het resultaat op aan de hand van het type van de gebruikte statistische functies. Enige onderdelen uit de SELECT die beginnen met een “ krijgen de gehele string terug volgens het patroon (“<string>”[ ][as <kolomnaam>]). Alle andere onderdelen worden als NULL geretourneerd.

Conclusie

Gedurende het ontwikkelen van het proof of concept zijn enkele problemen aan het licht gekomen. Deze problemen zijn zowel technisch als design-gerelateerd.

De technische problemen hebben vooral te maken met de manier waarop de communicatie tussen bepaalde systemen verloopt. De communicatieproblemen tussen de rapportageapplicaties en de batcher blijken oplosbaar via een kleine aanpassing in het design. De communicatieproblemen tussen de batcher en de database zijn echter niet oplosbaar in het design.

Voor ColdFusion applicatieservers die gebruik maken van JRE 1.6.0_29, zal ../lib/jsse.jar vervangen moeten worden met de ../lib/jsse.jar van JRE 1.6.0_30. Dit omdat de databaseconnectie er anders voor zorgt dat de batcher gaat hangen. Ook is het gebruik van Windows authenticatie niet mogelijk.

Op het design gebied is geconstateerd dat er twee stijlen zijn van batchen. Deze zijn Semantisch batchen en Syntactisch batchen. Semantisch batchen heeft het voordeel van domeinkennis en controle, terwijl syntactisch batchen het voordeel heeft van flexibiliteit en portabiliteit.

Alhoewel er in dit document veel gezegd is over iedere stijl van batchen raad ik aan om beide stijlen toe te passen in de batcher. Semantisch batchen verandert namelijk niets aan de vorm van de data die aangeleverd wordt voor de Syntactische batcher. Met behulp van Syntactisch batchen kunnen 80% (een schatting) van de gevallen worden geholpen – de overige 20% zou oplosbaar zijn via het Semantisch batchen. Men heeft namelijk alleen last van de negatieve aspecten van de stijlen wanneer deze stijlen veel toegepast moeten worden – de complexiteit van Syntactisch batchen maakt het bijna onmogelijk om alle situaties te dekken, en het specifieke van Semantisch batchen levert veel werk op wanneer dit moet worden toegepast voor alle situaties.

Technisch Ontwerp

Batcher

Datum: 15 April 2013
Auteur: Pim van Dongen
Versie: 1.2

Inleiding

Dit document dient als uitleg over het Technisch Ontwerp design. In dit document wordt er gerefereerd aan versie 1.4 van het Technisch Ontwerp. Omdat het diagram van het Technisch Ontwerp design vrij groot is, is het niet in dit document opgenomen.

Dit document dient in eerste instantie als uitleg over het Technisch Ontwerp. Dit document zal echter niet gedurende het project bijgehouden worden zodra er meer details bekend zijn over de implementatie. Alleen bij wijzigingen in het design wordt dit document aangepast. De documentatie over de implementatie van dit design zal uiteindelijk uit javadoc bestaan.

Om het onderhoud van het systeem te vergemakkelijken is het hoofdstuk waarin het technisch ontwerp per klasse wordt besproken in het Engels geschreven. De code, javadoc en ander commentaar in de code zijn in het Engels geschreven. Om dezelfde redenen dat de code en de javadoc in het Engels is wordt dit ook toegepast op het technisch ontwerp – technische documentatie is voor diegene die ermee moet werken – project documentatie is voor diegene die het project overziet.

Onderdelen

De werking van de batcher is in het functioneel ontwerp samengevat tot een aantal processtappen. In dit hoofdstuk wordt uitgelegd welk onderdeel van de batcher betrokken is bij ieder van deze processtappen.

Accepteren binnenkomende requests

Vanuit de rapportageapplicatie (gerepresenteerd als een externe interface in het diagram) komen requests voor gegevens binnen. Deze worden in de klasses MrIFunctions en CollectFunctions opgevangen. Dit gebeurt in ColdFusion. Vervolgens worden deze requests door gestuurd naar de MrIFunctionsAdapter, in Java.

Omzetten requests in query objecten

Nadat een request geaccepteerd is moet deze omgezet worden in een Query object. Dit wordt echter niet direct gedaan wegens de mogelijkheid tot semantisch batchen. De MrIFunctionsAdapter voegt de request toe aan de huidige batch. Wanneer het dan tijd is om te gaan batchen, wordt eerst met behulp van de SemanticBatcher een deel van deze requests samengevoegd. Daarna worden alle DataRequests geconverteerd in Query objecten via de RequestConverter.

Vergelijken en combineren query objecten

Vervolgens worden de query objecten met elkaar vergeleken en gecombineerd waar mogelijk. Dit wordt gedaan door de SyntacticBatcher.

Vertalen Query objecten naar SQL

Daarna moeten de query objecten vertaald worden naar SQL. Dit wordt gedaan door de SQLConverter.

Uitvoeren queries en ontvangen resultsets

Via de DatabaseConnection worden de queries vervolgens uitgevoerd en de ResultSets daarvan ontvangen.

ResultSets splitsen tot resultaten en deze mappen op queries

Van de gebatchte queries worden de ResultSets opgesplitst en aan de originele Query objecten gemapt. Dit wordt gedaan door de SQLConverter, aan de hand van de informatie die opgeslagen is tijdens het batchen zelf.

Queries terug mappen naar requests

Nadat de ResultSets gemapt zijn op de queries moeten de queries op de DataRequests gemapt worden. In tegenstelling tot het syntactisch batchen zijn er hier mogelijk veel meer trucs gebruikt bij het batchen – hierdoor wordt het uitsplitsen van de resultaten overgelaten aan de SemanticBatcher (die weet immers hoe het samengevoegd is).

Gegevens terug sturen

Wanneer alle resultaten vervolgens bekend zijn worden de resultaten in de ResultStructs geplaatst. Daarna wordt de batch functie beëindigd.

Classes

This chapter describes each element of the technical design. For each class or interface, the name, responsibilities and attributes will be described.

Classes are divided in packages – these packages aren't included in the diagram in order to maintain a clear view of the big picture. All given qualified class names are custom, unless designated with "native". When references are made to objects in code, Capitalization is used. When references are made to objects as concepts, no capitalization is used. That way, it's possible to distinguish between "queries" and "Query objects".

ColdFusion – ColdFusion Package

MrIFunctions – ColdFusion Class

MrIFunctions is responsible for acting as the original MrIFunctions database interface. Instead of directly executing queries on the database like the old MrIFunctions interface did, this one will communicate with the batcher.

CollectFunctions extends MrIFunctions – ColdFusion Class

CollectFunctions is an extension of MrIFunctions. It is preserved for compatibility rather than function. Whether it will serve as a façade, like MrIFunctions, or attempt some type of batching on its own is unclear at this moment.

ColdFusionInterface – Java Package

MrIFunctionsAdapter – Java Class

MrIFunctionsAdapter serves as the Java part of the ColdFusion-Java bridge. It accepts the data requests made by MrIFunctions (coldfusion.MrIFunctions), and stores them in a DataRequest object (request.DataRequest). It then adds the DataRequest to a Batch object (batcher.Batch). To allow the reporting application to continue making requests, a proxy object is sent back. This proxy object will be filled with the requested data once the batch is executed, or the proxy object is accessed prior to being filled with the requested data. MrIFunctionsAdapter can also receive function calls for executing already generated SQL queries.

ResultStruct – Java Class

ResultStruct holds the data, columnlist and typelist of an executed query. ResultStruct has two main functions: getStruct and getQuery. Each of these returns the stored data in the format of the requested datastructure (native coldfusion.runtime.Struct and native coldfusion.sql.QueryTable).
Attributes:

- List<String> (native java.util.List<native java.lang.String>) columns – An ordered list of columns that are part of the resultset stored in this object.
- List<String> (native java.util.List<native java.lang.String>) types – An ordered list of types that correspond to the columns attribute.
- Map<String, List> (native java.util.Map<native java.lang.String, native java.util.List>) data – The raw results retrieved from the database.

ProxyResult extends ResultStruct – Java Class

ProxyResult is a proxy object for ResultStruct. It allows MrIFunctionAdapter to send an object containing the results back straight away without needing the results first – via a reference, Batch can add the results later. If the results are needed but the batch hasn't started yet, ProxyResult can initiate the execution process of the batch.

Request – Java Package

DataRequest – Java Class

A model representation of a request for data made by the reporting application. Contains the details of the request, as well as a reference to the requirements (domain term, "restrictions" seems to be the proper English word for it) that are placed upon the requested data. DataRequest also has a DataRequestType (request.DataRequestType), to allow the use of semantic batching.

BatchedDataRequest extends DataRequest – Java Class

When semantic batching is used, multiple DataRequests are combined into one. At a later point in time (when results have been retrieved and need to be returned in the appropriate format), the BatchedDataRequest is used by the semantic batcher in order to map the results to each DataRequest. For this reason, the BatchedDataRequest also has a reference to the Strategy that was used to create this object with – SemanticBatchingStrategies (batcher.SemanticBatchingStrategy) are responsible for both batching DataRequests and splitting the ResultSets to the proper DataRequests.

Batcher – Java Package

Batch – Java Class

The Controller in the whole process, the Batch class contains a reference to all the objects needed for batching. These are DataRequests, Queries, the RequestConverter, the Semantic and Syntactic Batcher and the SQLConverter. Acting as the central communication point, Batch represents both the contents and the packaging of a batch of DataRequests. The Batch class initiates the batching, execution and splitting steps of the batch-execution process.

BatchManager – Java Interface

The BatchManager is an interface that represents the ability for a class to be able to "manage" Batch objects. Managing Batch objects entails that the BatchManager is responsible for them – creation, updating and eventually deleting them as well. Within this project, the BatchManager interface has been added to reduce coupling between MrIFunctions and the Batcher. This interface is intended to facilitate the possibility of the batcher being used in a different project.

SemanticBatcher – Java Class

The semantic batcher is responsible for batching DataRequests together in order to reduce the amount of queries pre-emptively prior to the syntactic batching process. The semantic batcher only provides the tools; through the use of SemanticBatchingStrategies, the actual batching is performed. SemanticBatchingStrategies are attached to the SemanticBatcher by the Batch.

SyntacticBatcher – Java Class

The second part of the batcher, the syntactic batcher runs each of its attached

SyntacticBatchingStrategies. The SyntacticBatcher only provides the tools for both query analysis and query batching; the actual analysis and combination process is done via SyntacticBatchingStrategies.

BatchingStrategy – Abstract Java Class

In order for the Batcher to be modular, the batcher makes use of BatchingStrategies. Prior to starting a Batch, BatchingStrategies are added to the Batch. This allows for customization without recompilation and allows the use of the batcher in a larger amount of environments – query analysis algoritms that take 5 minutes are not worth it when generating a report, but when you need to generate ALL reports in one go…

Exact implementation of base class is yet unknown – it's not yet known which aspects are shared between semantic and syntactic batching.

SyntacticBatchingStrategy extends BatchingStrategy – Abstract Java Class

A SyntacticBatchingStrategy is a BatchingStrategy that can be used by a SyntacticBatcher. A SyntacticBatchingStrategy is responsible for both determining how a SyntacticBatcher picks the queries that it will combine, as well the batching itself. SyntacticBatchingStrategies should keep performance in mind as deep query analysis is not lucrative – batches will contain 200+ queries, and as each batch operation reduces the possible gains of any future batching operations, the margin for performance increases grows smaller.

SemanticBatchingStrategy extends BatchingStrategy – Abstract Java Class

A SemanticBatchingStrategy is a BatchingStrategy that can be used by a SemanticBatcher. A SemanticBatchingStrategy is responsible for determining how the batching process of DataRequests of a certain DataRequestType will occur. The SemanticBatchingStrategy is also responsible for splitting the ResultSet to each DataRequest when execution of the Batch is completed. SemanticBatchingStrategies can be used to easily batch what would be very complex to do in an SyntacticBatcher. This is due to the fact that each SemanticBatchingStrategy may only be applied to one DataRequestType – and as such, the algorithm has prior knowledge of the eventual query structure.

RequestConverter – Java Class

The operating force behind the Batchers, the RequestConverter converts DataRequests into Query objects. The RequestConverter does not know how to do this, so the RequestConverter only implements methods for building up segments of the query. RequestConversionStrategies are responsible for providing the algorithm for converting the request into a Query.

RequestConversionStrategy – Abstract Java Class

The RequestConversionStrategy is an abstract class that serves as the base for implementing the means of converting DataRequests into Query objects. Unlike BatchingStrategies, the RequestConversionStrategy is not optional. The Batcher cannot execute queries that are not in query form – so for each type of DataRequest, a RequestConversionStrategy should be present.

QueryStructure – Java Package

Query – Abstract Java Class

The base class of all queries. Its responsibilities are yet unknown (it's not fully clear yet just what functionality is actually shared between Query classes).

StructuredQuery extends Query implements RowSource – Java Class

A StructuredQuery is a Query that consists of objects representing parts of the Query. StructuredQueries tend to be batchable because it is easy for an SyntacticBatchingStrategy to "browse" the structure of a StructuredQuery and then derive a possible combination of two StructuredQueries. StructuredQuery can have a reference to a QueryBatch object if it has been batched before. StructuredQuery objects can get batched as many times as is needed. StructuredQuery implements RowSource because it technically results in rows. For basic queries, this has no proper use, but in complex queries, the implementing of RowSource allows querytypes such as subselects and with-queries to make use of the structure of StructuredQuery.

SQLQuery extends Query – Java Class

Contrary to a StructuredQuery, SQLQuery objects cannot be batched at all. They represent queries in their SQL state – as a string. SQLQueries can be passed directly into the batcher to allow the execution of queries that would act differently when batching (custom T-SQL functions, for instance). SQLQueries are generated by the SQLConverter prior to Query execution.

UnionQuery extends StructuredQuery – Java Class

UnionQuery is a StructuredQuery that has been batched via a BatchingStrategy that UNION'ed two or more Queries together. Generally, UnionQueries cannot be batched any further.

WithQuery extends StructuredQuery – Java Class

WithQuery is a StructuredQuery, but is never part of the main batch. WithQuery represents the WITH statement in T-SQL, where a query can be pre-executed to serve as a RowSource for later queries.

SubSelectQuery extends StructuredQuery implements Comparable – Java Class

SubSelectQuery is a StructuredQuery that represents the usage of a sub-SELECT in a query. Because subselects can be used in "IN" conditionals, SubSelectQuery implements Comparable.

QueryBatch – Java Class

QueryBatch is a "log" of some sort created by the SyntacticBatcher when two or more StructuredQueries are batched. QueryBatch describes which queries were batched and how the resulting resultset should be mapped to these queries.
Attributes:

- Map<Selectable, Map<StructuredQuery, Selectable>> (native java.util.Map<querystructure.Selectable, native java.util.Map<querystructure.StructuredQuery, querystructure.Selectable>) columnmapping – A map containing the columns for the resultset. Each column has a list of queries that require a copy of the column for their resultset, with a reference to the column the data should be put in.

- Map<StructuredQuery, ConditionSet> (native java.util.Map<querystructure.StructuredQuery, querystructure.ConditionSet>) – A list of queries that require the data to be filtered prior to copying. This is used in case a batched query uses an UNION or GROUP BY, for instance. The ConditionSet can determine whether a row belongs to the query or not.

Comparable – Java Interface

The Comparable interface indicates that an implementing object can be compared to in a query. Different from native java.lang.Comparable, this Comparable interface indicates that an object can be part of a condition, on either side. Comparables can be used in the WHERE, ON and HAVING statements.

Selectable – Java Interface

The Selectable interface indicates that an implementing object can be used in the SELECT statement of a query.

Sortable – Java Interface

The Sortable interface indicates that an implementing object can be used in the ORDER BY statement of a query.

Groupable – Java Interface

The Groupable interface indicates that an implementing object can be used in the GROUP BY statement of a query.

RowSource – Java Interface

The RowSource interface indicates that an implementing object can be used in the FROM statement of a query, and be referred to as a table. RowSources can have a TableAlias, and RowSources must be able to provide either a TableAlias or some other identifier (as native java.lang.String) to allow references to the RowSource in SQL.

QueryPart – Abstract Java Class

QueryPart is an abstract class. Its responsibilities are yet unknown (it's not fully clear yet just what functionality exactly is shared between all the query parts).

SelectQueryPart extends QueryPart – Java Class

The SelectQueryPart represents the SELECT statement in a query. It contains a list of Selectables.

FromQueryPart extends QueryPart – Java Class

The FromQueryPart represents the FROM statement in a query. It contains a base RowSource, and a list of Joins to apply to this RowSource.

WhereQueryPart extends QueryPart – Java Class

The WhereQueryPart represents the WHERE statement in a query. It contains a ConditionSet. If no ConditionSet is present, the WHERE statement is omitted from the generated query.

GroupByQueryPart extends QueryPart – Java Class

The GroupByQueryPart represents the GROUP BY and HAVING statements in a query. It contains a list of Groupables and a ConditionSet. If no ConditionSet is present, the HAVING statement is omitted from the generated query.

OrderByQueryPart extends QueryPart – Java Class

The OrderByQueryPart represents the ORDER BY statement in a query. It contains a list of SortElements. Each SortElement contains a list of Sortables, together with a modifier ASC or DESC.

WithQueryPart extends QueryPart – Java Class

The WithQueryPart represents the WITH statements in a T-SQL query. It contains one or more WithQueries.

ColumnAlias implements Comparable – Java Class

A ColumnAlias is used to represent an "AS" statement in SQL for a Selectable other than the wildcard (*).

TableAlias – Java Class

A TableAlias is used to represent an "AS" statement in SQL for a RowSource.

LiteralValue implements Comparable, Selectable – Abstract Java Class

A LiteralValue is for comparing or selecting with "literals". It means that there is no interpretation required – values will be inserted into queries "as is".

LiteralString extends LiteralValue – Java Class

LiteralString is the textual variety of LiteralValue. It can be used to compare strings and dates.

LiteralNumeric extends LiteralValue implements Sortable – Java Class

LiteralNumeric is the numerical variant of LiteralValue. It can be used to compare integers or floats. Because even 64-bit floats are not perfect, values are stored in a string until they need to be included in the query.

ColumnReference implements Comparable, Selectable, Sortable, Groupable – Java Class

ColumnReferences are used to allow SyntacticBatchers to see that multiple points in the same query refer to the same column. ColumnReferences can have a ColumnAlias as well, which affects the output of the SQLConverter. To prevent ambiguity errors when multiple tables are involved, a ColumnReference can also have a reference to a RowSource.

AggregateFunction implements Comparable, Selectable, Sortable – Java Class

AggregateFunctions are used to represent the use of an aggregate function in a query. This can also include custom made aggregate functions. Prior to using an Aggregate function, it has to be registered at the Batch first. This is because grouping may cause rows to disappear, and as a direct result of that, the default value is unknown.

Equation implements Comparable, Sortable, Selectable, Groupable – Java Class

Equations are used for "special" expressions in SQL. Use of an expression makes a StructuredQuery partially if not wholly unbatchable as it is unknown what the expression actually does.

SelectAll implements Selectable – Java Class

SelectAll represents the wildcard (*) in a SELECT query. It is not possible to batch a query that contains SelectAll.

SortElement – Java Class

A SortElement contains a single Sortable and a SortType.

Condition – Abstract Java Class

A condition is a restriction placed on either a Join, a Row, or a Grouping (in the ON, WHERE and HAVING clauses respectively). Conditions can be included in ConditionSets to create the full conditional statement.

NullCondition extends Condition – Java Class

An “IS NULL” condition. Uses one comparable as a left operand, in the format “comparable IS NULL”.

BetweenCondition extends Condition – Java Class

A “BETWEEN X AND Y” condition. Uses three comparables – the first is the value to be compared, the second is the low end value and the third is the high end value. Format “comparable BETWEEN lowend AND highend”.

InCondition extends Condition – Java Class

An “IN” condition. Uses two comparables, one as the value, the other as the collection. Format “comparable IN collection”.

LikeCondition extends Condition – Java Class

A “LIKE” condition. Uses two comparables, one as the value, the other as search string. Format “comparable LIKE searchstring”.

CompareCondition extends Condition – Java Class

A mathematical condition. Uses two comparables and a ComparisonOperator. Format “comparable1 comparisonoperator comparable2”. ComparisonOperator can be one of the following: >, <, =, >=, <=, !=.

ConditionSet extends Condition – Java Class

A ConditionSet is a set of Conditions (actually internally stored as a List). To prevent any AND/OR ambiguity, a ConditionSet can only link conditions together by OR, or by AND. To allow “(x and y) or z” constructions, a ConditionSet inherits from Condition as well.

Join – Java Class

A Join represents an SQL Join. It has a reference to a RowSource (to allow joining to Tables, subselectqueries, temporary tables, etc.), as well as a JoinType and a ConditionSet to represent the “ON” conditions.

Table implements RowSource – Java Class

A Table object represents a basic Table or View in SQL.

JoinType – Java Enumeration

JoinType is used to indicate what type of SQL Join is to be used for a Join. The enumeration consists of the following values: “LEFT”, “RIGHT”, “FULL”, “INNER” and “CROSS”.

BooleanOperator – Java Enumeration
BooleanOperator is used to indicate what kind of Boolean operator should be applied to a ConditionSet. The enumeration consists of the following values: “AND” and “OR”.

SortType – Java Enumeration
SortType is used to indicate in what direction a column should be sorted in a resultset. The enumeration consists of the following values: “ASC” and “DESC”, meaning “Ascending” and “Descending” respectively.

ComparisonOperator – Java Enumeration
ComparisonOperator is used to indicate what kind of comparison should be made between two Comparables in a CompareCondition. The enumeration consists of the following values: “LT”(<), ”GT”(>), “LTE”(>=), “GTE”(<=), and “EQUALS”(=).

Database – Java Package

SQLConverter – Java Class
The SQLConverter is responsible for converting StructuredQueries to SQLQueries. The reason why an SQLConverter is needed is because certain objects act differently in different circumstances – for instance, it is impossible to group by a column alias, but you can make comparisons with a column alias.

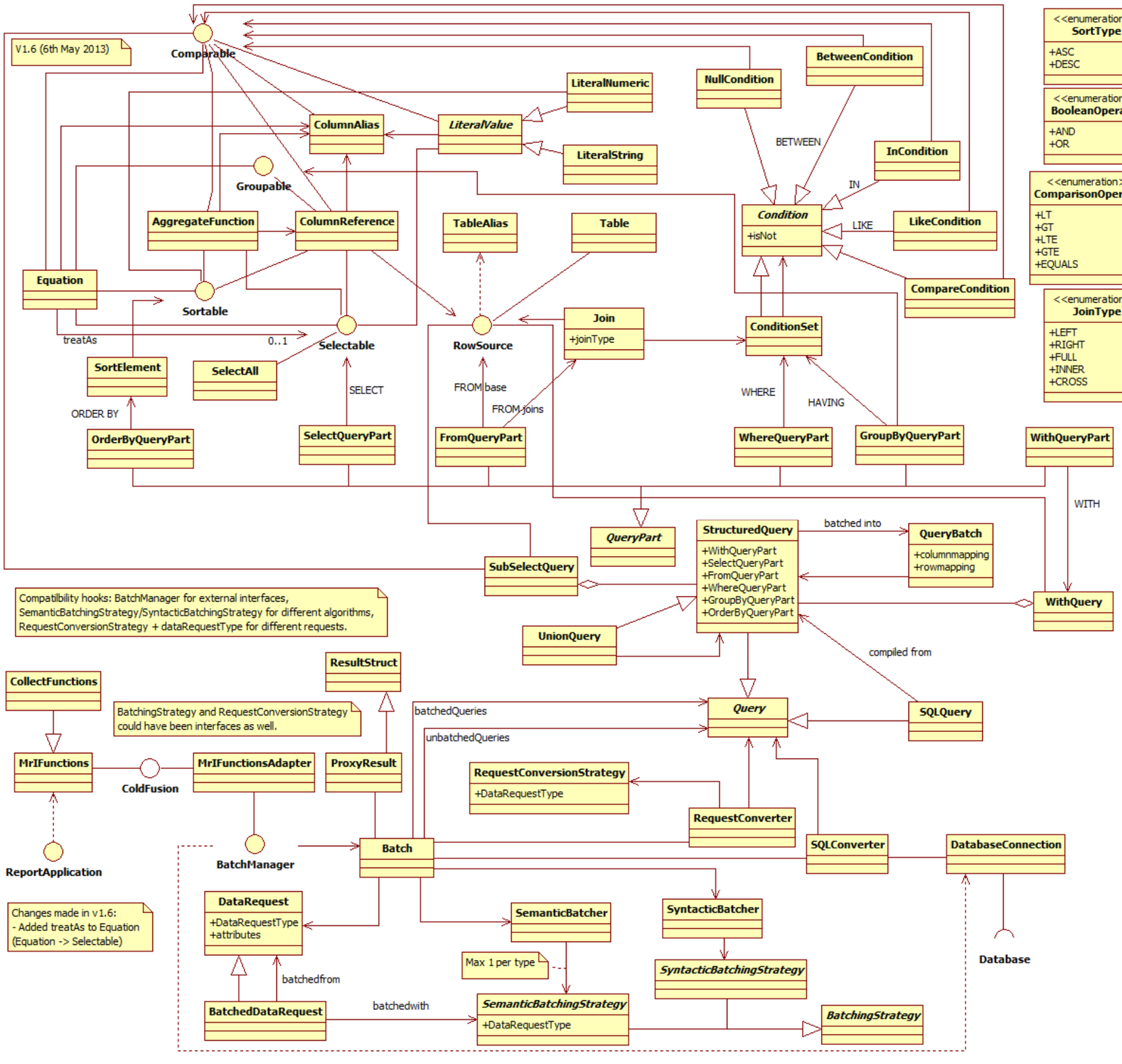
DatabaseConnection – Java Class
The DatabaseConnection is responsible for executing the queries on the database and receiving the ResultSets from the Database.

Other

ReportApplication – External Interface
The application that makes use of the batcher. It makes requests for data through (coldfusion.MrIFunctions).

ColdFusion – Language Interface
There exists a barrier between Java and ColdFusion. This interface represents this barrier.

Database – External Interface
The database that the data is stored in. Data can be requested from the database by sending queries for execution via the DatabaseConnection (database.DatabaseConnection).


V1.6 (6th May 2013)
Comparable
LiteralNumeric
NullCondition
BetweenCondition
ColumnAlias
LiteralValue
LiteralString
BETWEEN
InCondition
Groupable
IN
AggregateFunction
ColumnReference
TableAlias
Table
Condition
+isNot
LIKE
LikeCondition
Equation
CompareCondition
Sortable
Join
+joinType
ConditionSet
treatAs
0..1
Selectable
RowSource
SortElement
SelectAll
SELECT
FROM base
FROM-joins
WHERE
HAVING
ORDER BY
OrderByQueryPart
SelectQueryPart
FromQueryPart
WhereQueryPart
GroupByQueryPart
WithQueryPart
WITH
QueryPart
StructuredQuery
+WithQueryPart
+SelectQueryPart
+FromQueryPart
+WhereQueryPart
+GroupByQueryPart
+OrderByQueryPart
batched into
QueryBatch
+columnmapping
+rowmapping
SubSelectQuery
WithQuery
Compatibility hooks: BatchManager for external interfaces,
SemanticBatchingStrategy/SyntacticBatchingStrategy for different algorithms,
RequestConversionStrategy + dataRequestType for different requests.
UnionQuery
compiled from
ResultStruct
CollectFunctions
BatchingStrategy and RequestConversionStrategy
could have been interfaces as well.
batchedQueries
Query
SQLQuery
unbatchedQueries
MrIFunctions
ColdFusion
MrIFunctionsAdapter
ProxyResult
RequestConversionStrategy
+DataRequestType
RequestConverter
ReportApplication
BatchManager
Batch
SQLConverter
DatabaseConnection
Changes made in v1.6:
- Added treatAs to Equation
(Equation -> Selectable)
DataRequest
+DataRequestType
+attributes
SemanticBatcher
SyntacticBatcher
Database
Max 1 per type
batchedfrom
SyntacticBatchingStrategy
BatchedDataRequest
batchedwith
SemanticBatchingStrategy
+DataRequestType
BatchingStrategy
<<enumeration>>
SortType
+ASC
+DESC
<<enumeration>>
BooleanOperator
+AND
+OR
<<enumeration>>
ComparisonOperator
+LT
+GT
+LTE
+GTE
+EQUALS
<<enumeration>>
JoinType
+LEFT
+RIGHT
+FULL
+INNER
+CROSS

Adviesrapport

Performance Inladen en Weergeven Onderzoeksresultaten

Datum: 29 Mei 2013
Auteur: Pim van Dongen
Versie: 1.3

Inleiding

Dit document gaat over de verdere handelingen die Panteia kan ondernemen betreffende de performance van het inladen en genereren van een rapport van onderzoeksresultaten. Ook beschrijft dit document een advies over welke van deze handelingen uitgevoerd zouden moeten worden.

De handelingen die in dit document zijn beschreven zijn aan het licht gekomen gedurende de uitvoering van het project.

Het doel van dit document is het in kaart brengen van de mogelijkheden die Panteia heeft om het performanceprobleem verder te verminderen. Daarnaast wordt een set van deze mogelijkheden aangeraden middels een advies.

Huidige Situatie

In dit hoofdstuk wordt de verdeling van de executietijd binnen de huidige situatie beschreven.

Oude Situatie

In de oude situatie duurde het genereren van een rapportage ongeveer 33 seconden.

Het genereren van een rapportage bestaat uit een aantal stappen. Als eerste wordt de rapportagestructuur opgesteld. Vervolgens wordt de vragenlijst in het geheugen geladen. Uit de vragenlijst worden de antwoordcodes gehaald. Deze worden in de rapportagestructuur geplaatst. Vervolgens wordt per onderdeel van de rapportagestructuur aan MrIFunctions gevraagd om de gegevens op te halen. MrIFunctions bouwt hiervoor queries op. Deze queries worden naar de database gestuurd, waar ze uitgevoerd worden. MrIFunctions ontvangt van de database het resultaat van de queries. Dit resultaat wordt teruggegeven aan de rapportageapplicatie.
De rapportageapplicatie geeft het resultaat door aan een ander applicatieonderdeel, OutputFunctions. OutputFunctions converteert het resultaat naar een datastructuur. Ook maakt OutputFunctions HTML voor grafieken. OutputFunctions stuurt de datastructuur met HTML terug naar de rapportageapplicatie. Wanneer alle onderdelen van de rapportagestructuur door dit proces zijn geweest, wordt de rapportagestructuur naar een PDF-maker gestuurd. Deze genereert verdere HTML om het rapport volledig in HTML op te bouwen. Het HTML-rapport wordt vervolgens in de ColdFusion functie “cfdocument” gestopt. Deze functie converteert het HTML-rapport naar een PDF bestand. Vervolgens wordt het PDF-bestand naar de webbrowser van de Client gestuurd.

Het opbouwen van de rapportagestructuur duurt ongeveer 3,5 seconden. Dit komt omdat er queries door de rapportageapplicatie worden uitgevoerd die niet door MrIFunctions kunnen worden gegenereerd. Deze queries kosten 3,5 seconden om uit te voeren. De opbouw van de rapportagestructuur zonder het uitvoeren van queries is binnen 10 milliseconden klaar.

Het ophalen van de gegevens en het converteren naar HTML per onderdeel van de rapportagestructuur duurt ongeveer 21 seconden. Dit is niet afzonderlijk te meten omdat de cyclus per onderdeel van de rapportagestructuur wordt doorlopen.

Het converteren van het HTML-rapport naar een PDF-bestand duurt ongeveer 8,5 seconden. Detailmeting geeft aan dat de volledige opbouw van het HTML-rapport ongeveer een tiende van een seconde in beslag neemt. Uitvoeren van de functie cfdocument neemt de rest van de tijd in.

Huidige Situatie

In de huidige situatie is als performanceverbetering de Batcher geïmplementeerd. De Batcher neemt een deel van het genereren van queries over van MrIFunctions om de tijd die dit inneemt te verminderen.

Het opbouwen van de rapportagestructuur duurt nog steeds ongeveer 3,5 seconden. De queries die de rapportageapplicatie uitvoert zijn niet aangepast.

Het ophalen van de gegevens en het converteren naar HTML per onderdeel van de rapportagestructuur is opgesplitst in drie delen. Als eerste stap worden de vragen voor de benodigde

gegevens in de Batcher gezet. Dit kost 1 seconde. Deze enkele seconde komt omdat voor sommige vraagtypen een query moet worden uitgevoerd om te achterhalen hoeveel onderdelen een vraag heeft.
De tweede stap is het genereren van querystructuren, batchen, genereren van de queries en uitvoeren van de gegenereerde queries. Het genereren van querystructuren duurt ongeveer een tiende van een seconde. Het batchen van de querystructuren neemt ongeveer 0,3 seconden in beslag. Het genereren van de queries aan de hand van de querystructuren neemt ongeveer een tiende van een seconde in beslag. Het vervolgens uitvoeren van de queries en retourneren van het resultaat ligt iets onder de 8 seconden. Daarmee kost deze hele stap nu in zijn geheel ongeveer 8,5 seconden.

De derde stap is het uitpakken van de resultaten die de Batcher terugstuurt en het verwerken van dit resultaat in OutputFunctions. Het uitpakken en verwerken kost ongeveer een tiende van een seconde. Daarmee kost het geheel ongeveer 9,5 seconden, in tegenstelling tot de 21 seconden in de oude situatie.

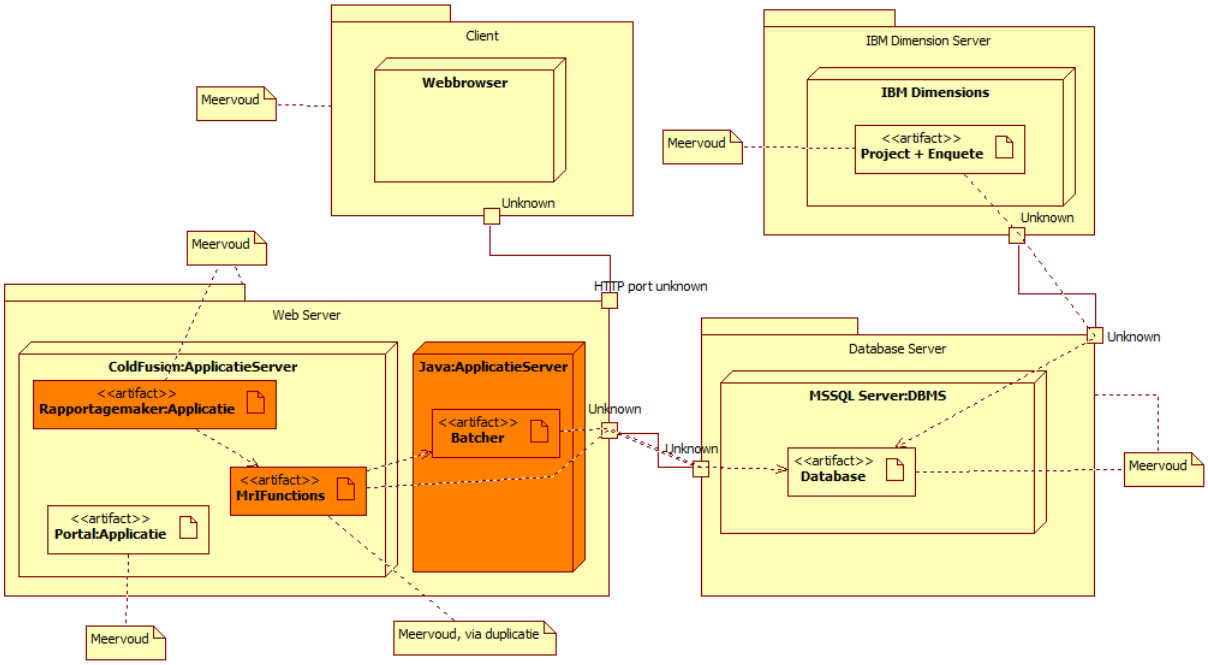
Aan het converteren van het HTML-rapport naar een PDF-bestand is niets gewijzigd. Dit neemt nog steeds ongeveer 8,5 seconden in beslag.

In de huidige situatie is door implementatie van de Batcher de totale tijd dus van ongeveer 33 seconden naar ongeveer 21 seconden teruggebracht. Dit is al een hele verbetering, maar er kan nog meer tijdwinst worden behaald. In het volgende hoofdstuk wordt besproken hoe onderdelen van het proces aangepast kunnen worden voor performance verbetering.

Mogelijke aanpassingen

In dit hoofdstuk worden de mogelijke aanpassingen voor het verder verbeteren van de performance besproken. Deze aanpassingen zijn slechts ideeën – het is mogelijk dat de voorgestelde aanpassing geen effect, of zelfs een negatief effect heeft. Voordat een aanpassing wordt toegepast in de productieserver dient er een performancetest gedaan te worden in een testomgeving.

De mogelijke aanpassingen zijn opgedeeld aan de hand van het systeemonderdeel waar zij betrekking op hebben.



Deze afbeelding geeft de systeemonderdelen weer.

Portal

De webbrowser kan niet bewerkt worden door Panteia omdat deze op de computer van de Client staat. Het is echter wel mogelijk om de inhoud van de website aan te passen.

Feedback

Het is belangrijk om goed met gebruikers te communiceren. Een melding weergeven dat het rapport gegenereerd wordt zou helpen om klachten te verminderen. In de huidige situatie is het zo dat een gebruiker op de knop drukt voor het downloaden, waarna de webpagina voor een lange tijd bezig is met “iets”.

Database

Gebruik van nieuwere versie DBMS

Momenteel maakt Blik op Werk, samen met vele andere projecten die gebruikmaken van IBM Dimensions, gebruik van een database op een MSSQL Server. De versie van deze DBMS is “MSSQL Server 2005”. Er zou onderzoek uitgevoerd kunnen worden met het doel om te kijken of het mogelijk is om gebruik te maken van een nieuwere versie van MSSQL Server. Nieuwere versies van MSSQL Server zouden beter in staat zijn om queryanalyse uit te voeren. Ook zouden nieuwere versies van MSSQL Server over het algemeen meer geoptimaliseerd moeten werken.

Gebruik van geïndexeerde views

MrIFunctions maakt gebruik van een aantal typen queries. Voor sommige van deze querytypen zou het gebruik van een geïndexeerde view kunnen helpen. Een deel van de query is dan al geprecompileerd, wat zou kunnen schelen in de executietijd. Deze aanpassing omvat het implementeren van geïndexeerde views voor enkele querytypen die door MrIFunctions worden gebruikt.

Batcher

Aanpassen batchingsalgoritmes

Binnen de huidige batchingsalgoritmes wordt naar een bepaald patroon gekeken. Als dit patroon geïdentificeerd is binnen meerdere querystructuren, worden deze gecombineerd. Een van de mogelijke aanpassingen is het aanpassen van het Syntactische batchingsalgoritme “AggregateFunctionSyntaxBatch”. Momenteel wordt binnen dat algoritme gecontroleerd of meerdere querystructuren dezelfde statistische functies gebruiken over dezelfde dataset met dezelfde restricties, op een ID code na. Op deze manier wordt een SELECT n+1 probleem verholpen. Daar waar eerst voor ieder onderdeel van een vraag een query wordt gesteld, worden nu in 1 query alle onderdelen van een vraag beantwoordt. In de rapportages die momenteel worden gegenereerd komt het vaak voor dat een rapportageonderdeel bestaat uit twee delen. Het eerste deel is het resultaat van een statistisch functie. Het tweede deel is het aantal resultaten wat in die statistisch functie is verwerkt. Het batchingsalgoritme “AggregateFunctionSyntaxBatch” is nog niet in staat om dit patroon te herkennen. Hierdoor worden er in twee queries dezelfde dataset behandeld. De ene keer voor het uitvoeren van de statistische functie. De andere keer voor het tellen van het aantal resultaten. Aanpassen van het batchingsalgoritme zodat het probeert om dit probleem te verhelpen zou een seconde kunnen schelen in de executietijd van de queries.

Logging configureerbaarmaken

Momenteel is het in de Batcher zo dat bepaalde gegevens worden weggeschreven naar de hardeschijf. Deze gegevens zijn bedoeld voor de ontwikkelaar om te kunnen zien wat er tijdens de executie van de Batcher is gebeurd. Standaard staat het printen van deze informatie altijd aan. Dit is in principe een overhead. Door middel van het configureerbaar maken van het loggen kan deze overhead gedeactiveerd worden als deze niet benodigd is.

Multithreading voor uitvoeren queries

In de probleemanalyse is gebleken dat het genereren van meerdere rapportages tegelijk (bijna) geen extra tijd kost. De Batcher maakt momenteel gebruik van een enkele databaseverbinding. Met multithreading is het mogelijk om te doen alsof er meerdere rapportages tegelijk opgebouwd worden. In de werkelijkheid zijn het echter twee onderdelen van dezelfde rapportage. Door de implementatie van ondersteuning voor werken met meerdere databaseverbindingen met multithreading, kan deze aanpassing toegepast worden.

MrIFunctions & OutputFunctions

SortCat aanpassen om overbodige joins te vermijden met zeroExcluded

Een van de querytypes in MrIFunctions is SortCat. Het is mogelijk om bij deze functie een filter aan te geven dat de waarde “0” niet meegerekend wordt in statistische functies als een berekening van een gemiddelde. Dit wordt gedaan door bij de functie het argument “zeroExcluded” de waarde “true” te geven.

Om de gegevens uit de database te halen bouwt MrIFunctions een query op. In deze query wordt aan de hand van een reqstruct (een datastructuur met “requirements”) filters toegevoegd. Momenteel is het mogelijk dat er via de reqstruct een filter wordt toegevoegd wat controleert of de waarde “0” niet meegerekend wordt in statistische functies. Dit gebeurt ook in de praktijk bij het genereren van rapportages. Deze filters zijn in de gevallen waar “zeroExcluded” op “true” staat, overbodig. In MrIFunctions controleren of deze situatie zou ontstaan kan voorkomen dat de database dubbel werk aan het uitvoeren is.

StringBuilder gebruiken in OutputFunctions

Stringconcatenatie is relatief duur in tijd, wanneer dit vergeleken wordt met het optellen van getallen. Dit komt omdat een String onaanpasbaar is. Bij het concateneren van twee Strings moet dus een nieuw String object aangemaakt worden. Gegeven een constante lengte van een String, kost het O(n*(n/2) + n/2) om deze te concateneren. O (N) is een functie van tijd, beter bekend als de Grote-O-notatie.[9] StringBuilder maakt echter een lijst van alle Strings die geconcateneerd moeten worden en concateneert deze pas wanneer dit nodig is. Hierdoor levert StringBuilder O(n) performance op, wat veel sneller is.

Hiermee zou OutputFunctions sneller de resultaten van de queries naar HTML (of iets anders) kunnen converteren.

[9] Zie de bijlagen voor een uitleg over “Grote O notatie”.

Rapportageapplicatie

Veranderen wijze opbouwen PDF

De huidige manier waarop het PDF-bestand wordt opgebouwd is erg omslachtig. Eerst worden de resultaten tot HTML geconverteerd. Daarna gaat een ColdFusion functie proberen de HTML te interpreteren en te converteren tot een PDF-bestand. Daarnaast lijkt de functie cfdocument niet erg snel te zijn. Er zou onderzoek gedaan kunnen worden naar alternatieve wijzen om een PDF bestand op te bouwen.

StringBuilder gebruiken in het samenstellen van de PDF

Ook bij het samenstellen van het PDF-bestand vindt veel String concatenatie plaats. Door het gebruik van StringBuilder zou ook hier een performanceverbetering te behalen zijn. Deze aanpassing bestaat daarmee dus uit het aanpassen van de code voor het opbouwen van de inhoud van het PDF-bestand zodat deze gebruik maakt van StringBuilder.

Applicatieserver

Gebruik van nieuwere versie ColdFusion

Momenteel wordt er gebruik gemaakt van ColdFusion 9 voor de rapportageapplicaties. Het zou mogelijk kunnen zijn om gebruik te gaan maken van ColdFusion 10. De nieuwe versie van ColdFusion zou sneller kunnen zijn. Dit komt mede doordat een nieuwere versie van ColdFusion met een nieuwere versie van een JVM (Java Virtual Machine) kan werken. Deze aanpassing bestaat hiermee uit het vervangen van ColdFusion 9 met ColdFusion 10.

Advies

In dit hoofdstuk wordt per aanpassing die in het vorige hoofdstuk is behandeld een advies gegeven. Dit advies wordt beargumenteerd aan de hand van een schatting in de kosten en baten van een mogelijke aanpassing.

Portal

Feedback

Dit zou een goede toevoeging zijn voor de website. Veel moderne websites en mobiele applicaties maken gebruik van deze functionaliteit. Deze functionaliteit bestaat uit het weergeven van een melding dat het rapport wordt gegenereerd en dat dit ongeveer 20 seconden kan duren. Deze melding wordt tijdens het genereren van het rapport weergegeven. Alhoewel het toepassen van deze aanpassing de performance zelf niet verbeterd zou dit wel de impact van het probleem bij klanten moeten verminderen.

Database

Gebruik van nieuwere versie DBMS

Deze aanpassing wordt aangeraden. Voor toepassing hiervan zal uitgezocht moeten worden of het gebruikte pakket compatibel is met een nieuwere versie van MSSQL Server. Het voordeel is echter van toepassing op alle rapportageapplicaties die gebruik maken van IBM Dimensions en MrIFunctions en niet alleen Blik op Werk. Ook leveren nieuwere versies van MSSQL Server betere ondersteuning voor verdere performancetests.

Gebruik van geïndexeerde views

Deze aanpassing wordt niet geadviseerd. De Batcher vervormt queries waardoor deze mogelijk niet gebruik kunnen maken van een geïndexeerde view. Daarnaast zou deze aanpassing per project doorgevoerd moeten worden.

Batcher

Aanpassen batchingsalgoritmes

Deze aanpassing wordt niet direct geadviseerd. Bij het uitvoeren van tests heb ik geconcludeerd dat het combineren van queries met verschillende statistisch functies een performanceverbetering zou kunnen opleveren van ongeveer 1 tot 1,5 seconden. Het werk wat hiervoor verricht moet worden is echter vrij complex. Als er meerdere grote aanpassingen aan de Batcher gedaan gaan worden zou deze aanpassing rendabel kunnen zijn. Anders zijn de kosten van het doorgronden van de werking van de Batcher te groot. Dit komt omdat Panteia momenteel geen werknemers in dienst heeft die kennis hebben van de interne werking van de Batcher.

Logging configureerbaarmaken

Deze aanpassing wordt niet geadviseerd. Het is al reeds mogelijk om de logging te deactiveren. Dit wordt gedaan door in de code van de Batcher dit aan te passen en vervolgens de Batcher opnieuw te

compileren. Er is geen verschil in de performance van het configureerbaar logging en niet-configureerbaar logging wanneer deze beide gedeactiveerd zijn.

Multithreading voor uitvoeren queries
Deze aanpassing wordt geadviseerd. Toepassen van deze aanpassing zou de executietijd van de queries mogelijk halveren. De implementatie kan technisch complex zijn, maar deze complexiteit is niet gerelateerd aan de Batcher. Dit betekent dat alleen ervaring met Java benodigd is voor het implementeren van deze aanpassing.

MrIFunctions & OutputFunctions

SortCat aanpassen om overbodige joins te vermijden met zeroExcluded
Deze aanpassing wordt geadviseerd. Bij Blik op Werk wordt de functie SortCat onder andere gebruikt om het aantal onderdelen van een categorische vraag op te halen. Dit betekent dat de queries hiervoor niet gebatcht kunnen worden. Deze aanpassing heeft dus nog steeds impact, ook al neemt de Batcher een deel van de queries over. Daarnaast is het zo dat deze aanpassing invloed heeft op alle rapportageapplicaties die gebruik maken van SortCat.

StringBuilder gebruiken in OutputFunctions
Deze aanpassing wordt niet direct geadviseerd. De impact van OutputFunctions op de executietijd van de code is zeer laag (< 40 ms). Dit is zeer weinig wanneer dit vergeleken wordt met de 21 seconden die Blik op Werk momenteel nodig heeft voor het genereren van een rapport. Wanneer Panteia echter geïnteresseerd is om de performance van kleinere online rapportages te verbeteren zou deze aanpassing wel toegepast kunnen worden.

Rapportageapplicatie

Veranderen wijze opbouwen PDF
Deze aanpassing wordt geadviseerd. Voor Blik op Werk is dit het onderdeel waar theoretisch de grootste performanceverbetering behaald kan worden. Dit is echter wel een aanpassing waarvoor Panteia mogelijk meerdere onderdelen van Blik op Werk voor zou moeten aanpassen. Dit betekent dat deze aanpassing ook de grootste hoeveelheid werk omvat.

StringBuilder gebruiken in het samenstellen van de PDF
Voor deze aanpassing geldt hetzelfde advies als bij de aanpassing voor het gebruik van StringBuilder in OutputFunctions. Deze aanpassing wordt niet geadviseerd, tenzij meerdere projecten baat hebben bij implementatie van deze aanpassing.

Applicatieserver

Gebruik van nieuwere versie ColdFusion
Voor deze aanpassing wordt geen direct advies gegeven. Het is voor de adviseur onbekend of het vervangen van ColdFusion 9 met ColdFusion 10 financiële kosten met zich meebrengt. Ook is de mogelijke impact op de executietijd niet duidelijk. Het uiteindelijke advies bestaat uit het vergelijken van de performance van applicaties die op ColdFusion 9-servers draaien met de performance van applicaties die op ColdFusion 10-servers draaien. Aan de hand van deze vergelijking zou ik adviseren Panteia een keuze te maken.

Conclusie

In dit adviesrapport zijn een aantal mogelijke aanpassingen opgesteld. Deze aanpassingen richten zich op het verbeteren van de performance van het genereren van rapportages voor Blik op Werk.

Van deze mogelijke aanpassingen worden er een aantal geadviseerd om te implementeren. Deze selectie is gedaan aan de hand van de verwachte kosten en baten van iedere aanpassing.

Voor de website wordt aangeraden om feedback aan een gebruiker te geven. Dit zou gedaan worden door tijdens het genereren van een rapportage aan te geven dat de server bezig met het genereren van het rapport.

Voor de database wordt aangeraden om een nieuwere versie van MSSQL Server te gebruiken. Een nieuwere versie van MSSQL Server zou een verbetering in performance leveren. Ook leveren nieuwere versies van MSSQL Server meer ondersteuning voor optimalisatie van queries en databaseconfiguratie.

Voor de Batcher wordt aangeraden om multithreading te gaan gebruiken voor het uitvoeren van queries. Dit is een complexe taak, maar deze aanpassing zou ongeveer de helft van de executietijd van de queries kunnen schelen.

Voor MrIFunctions wordt aangeraden om in de SortCat functie een filter toe te voegen. Dit filter controleert of onderdelen van de query dubbelop zijn. Dit zorgt ervoor dat de queries die op de database worden uitgevoerd geen overbodige complexiteit bevatten.

Voor de rapportageapplicatie van Blik op Werk wordt aangeraden om naar een andere manier te kijken voor het genereren van een PDF-bestand. Dit kost ruim 8 seconden en lijkt daarmee de grootste bijdrage aan het performanceprobleem.

Deze vijf aanpassingen zijn het meest rendabel. Daarom adviseer ik dat deze ze toegepast moeten worden.

Bijlagen

Nederlandse Wikipedia over Grote O Notatie: http://nl.wikipedia.org/wiki/Grote-O-notatie

Grote-O-notatie

De **Grote-O-notatie** of *Big-O-notatie* wordt in de informatica gebruikt om de (tijds)complexiteitsgraad van algoritmen uit te drukken. Deze notatie is een 'slechtste-geval' maat. Ze geeft enkel informatie over het maximaal aantal elementaire bewerkingen dat een algoritme bij gegeven invoergrootte zal uitvoeren. De notatie is machine-onafhankelijk.

De grote-O notatie heeft de volgende vorm:

$$A = O(f(n))$$

Informeel komt het erop neer dat een algoritme A bij gegeven invoergrootte n (groot genoeg) steeds minder dan f(n) elementaire bewerkingen zal uitvoeren. Formeel is de betekenis van de grote-O notatie als volgt gedefinieerd:

$$A = O(g(n)) \Leftrightarrow \exists_{c>0} \exists_{N \in \mathbb{N}} \forall_{n \geq N} : 0 \leq A(n) \leq c \cdot g(n)$$

Het bovenstaande wil zeggen: vanaf een bepaalde invoergrootte N wordt A van boven af begrensd door functie g, maal een constante factor c. Oftewel: vanaf probleemgrootte N is de waarde g(n) altijd groter dan het aantal stappen dat A nodig heeft om een probleem met invoergrootte n op te lossen. Daarmee is g een bovengrens voor de complexiteit van A.

Merk op dat (volgens de formele definitie) O(n), $O(n^2)$ impliceert. Immers als het algoritme trager of gelijk aan n groeit, dan zeker trager dan n^2. Dit is algemeen geldig in de Grote-O-notatie: een lagere complexiteit impliceert een hogere complexiteit.

Veel voorkomende O-functies zijn (in volgorde van stijgende complexiteit):

- *O(1)*; (*constant*: tijd onafhankelijk van de grootte van het probleem; preciezer: de tijd is nooit meer dan een bepaalde bovengrens, ook al is het probleem nog zo groot)
- *O(log n)*, (*logaritmisch*: evenredig aan logaritme van *n*, als *n* een grootte-orde toeneemt neemt de tijd met een constante toe)
- *O(n)*, (*lineair*: tijd evenredig aan *n*)
- *O(n log n)*, (*linearitmisch*: product van de vorige twee - dit is de complexiteit van de beste sorteeralgoritmen die bekend zijn)
- $O(n^2)$ (*kwadratisch*: tijd neemt kwadratische toe met de grootte van het probleem)
- $O(2^n)$ (*exponentieel*: tijd neemt exponentieel toe met de grootte van het probleem)

Algoritmen met een complexiteit $O(n^k)$ met $k \in \mathbb{N}$ (en n de invoergrootte) noemen we polynomiaal. Algoritmen met een complexiteit $O(k^n)$ met $k \in \mathbb{N}$ noemen we exponentieel. Merk opnieuw op dat exponentiële algoritmen polynomiaal zijn, maar omgekeerd niet.